Alma Mater Studiorum - University of Bologna

DEIS - DEPARTMENT OF ELECTRONICS, COMPUTER SCIENCE AND SYSTEMS
PhD Course in Electronics, Computer Science and Telecommunications

XXII CYCLE – SCIENTIFIC-DISCIPLINARY SECTOR ING/INF05

Integration of symbolic and connectionist AI techniques in the development of Decision Support Systems applied to biochemical processes

Candidate:
Davide Sottara

Supervisor:
Prof. Paola Mello

PhD Course Coordinator:
Prof. Paola Mello

Final Examination Year 2010

To my family

ABSTRACT

There exist complex systems, such as bio-chemical plants, which require a constant management to be kept in optimal operating conditions. To this end, automation is the only feasible option: the tasks involved, ranging from fault detection to diagnosis to control, can hardly be performed by traditional model-based controllers alone, but instead require the additional application of artificial intelligence-based techniques.

AI is a vast field, covering many technologies which can be roughly classified in "Hard" or "Soft" techniques: the former typically use a symbolic representation of the data and elaborate it by logic reasoning; the latter, instead, process information at a sub-symbolic level, exploiting the interactions of many simple elaboration units - hence they are also known as connectionist techniques.

Since different techniques are more suitable for different problems, a complex management infrastructure is likely to include more than one module. These modules should be able to interact in order to exploit the mutual potentialities when processing the information coming from the managed system. Moreover, this information is likely to be imperfect - vague and/or uncertain and/or incomplete - so the modules should be able to deal with it appropriately.

In this dissertation, we claim not only that taking imperfection into account is a necessary feature of an "intelligent" module interfaced to a real-world system, but also that extending formal logical reasoning with imperfection allows to obtain a deeper integration of "Hard" and "Soft" computing techniques than simply using them together, in cascade or in parallel, obtaining strongly hybrid modules which can be simpler and yet more robust than their pure counterparts.

Moreover, it will be shown that such hybrid modules can be deployed effectively within an infrastructure which combines the concepts of service, agent and event in a natural way.

SOMMARIO

Esistono sistemi complessi, quali i reattori bio-chimici, che hanno bisogno di un monitoraggio costante per essere mantenuti nelle condizioni operative ottimali. Per far ció, l'automazione é l'unica strada percorribile: le funzionalitá necessarie, che vanno dalla diagnosi al controllo, possono difficilmente essere svolte soltanto dai controllori tradizionali, basati sui modelli, ma richiedono, in aggiunta, l'applicazione di tecniche di intelligenza artificiale.

L'AI é un dominio molto vasto, che copre diverse tecnologie che possono essere divise, con buona approssimazione, in tecniche di "Hard" e "Soft" Computing. Le prime usano solitamente una rappresentazione simbolica delle informazioni e le elaborano per mezzo di ragionamenti logici; le seconde, invece, processano le informazioni ad un livello sub-simbolico, sfruttando le interazioni di molte unitá di elaborazione semplici - motivo per cui sono anche note come tecniche "connessioniste".

Dato che tecnologie diverse sono piú adatte per problemi diversi, una infrastruttura di gestione complessa includerá probabilmente piú di un modulo al suo interno. Tali moduli dovrebbero essere in grado di interagire per sfruttare al meglio le rispettive potenzialitá nel processare le informazioni provenienti dal sistema gestito. Inoltre, questa informazione é spesso imperfetta - vaga e/o incerta e/o incompleta - pertanto i moduli dovrebbero poterla gestire in modo appropriato.

In questa dissertazione, si sostengono due tesi: non solo che gestire l'imperfezione é una caratteristica necessaria di un modulo "intelligente" interfacciato ad un sistema reale, ma anche che introdurre l'imperfezione nel ragionamento formale permette di ottenere un livello di integrazione tra tecnologie di "Hard" e "Soft" Computing piú profondo del semplice usarle insieme, in cascata o in parallelo, ottenendo cosí dei moduli fortemente ibridi che possono essere piú semplici e allo stesso tempo piú robusti delle loro controparti "pure".

Inoltre, si mostra che tali moduli ibridi possono essere usati efficacemente all'interno di una infrastruttura che combina i concetti di servizio, agente ed evento in modo naturale.

PUBLICATIONS OF THE AUTHOR

[1] G. L. Bragadin, G. Colombini, L. Luccarini, M. Mancini, P. Mello, M. Montali, and D. Sottara. Formal verification of wastewater treatment processes using events detected from continuous signals by means of artificial neural networks. Case study: SBR plant. *ENVIRONMENTAL MODELLING AND SOFTWARE*. ISSN 1364-8152. doi: 10.1016/j.envsoft.2009.05.013. URL http://dx.medra.org/10.1016/j.envsoft.2009.05.013. Article in Press. (Cited on pages 204, 207, 223, 237, 247, and 255.)

[2] Sottara D., P.Mello, L.Luccarini, and G.Colombini. Controllo e gestione intelligente degli impianti di depurazione. In *Europa del Recupero : le ricerche, le tecnologie, gli strumenti e i casi studio per una cultura della responsabilitÃ ambientale*, pages 156 – 161, S.Arcangelo di Romagna (RN) – ITA, 5-8 Novembre 2008. Maggioli Editore (ITALY).

[3] D.Sottara, L.Luccarini, and P.Mello. Strumenti di IA per il controllo e la diagnosi dei processi biologici negli impianti a fanghi attivi. In *Europa del recupero : le ricerche, le tecnologie, gli strumenti e i casi studio per una cultura della responsabilitÃ ambientale*, pages 150 – 155, S.Arcangelo di Romagna (RN) – ITA, 5-8 Novembre 2008. Maggioli Editore (ITALY).

[4] L. Luccarini, P. Mello, D. Sottara, and A. Spagni. Artificial Intelligence based rules for event recognition and control applied to SBR systems. In *Conference Proceedings of the 4th Sequencing Batch Reactor Conference*, pages 155 – 158, ROMA – ITA, 7-10 April, 2008. s.n. (Cited on pages 203, 239, and 247.)

[5] P. Mello, M. Proctor, and D. Sottara. A configurable RETE-OO engine for reasoning with different types of imperfect information. *IEEE Transactions on Knowledge and Data Engineering (TKDE) - Special Issue on Rule Representation, Interchange and Reasoning in Distributed, Heterogeneous Environments*, 2010. Article in Press. (Cited on page 118.)

[6] M. Nickles and D. Sottara. Approaches to Uncertain or Imprecise Rules - A survey. In G. Governatori, J. Hall, and A. Paschke, editors, *Rule Interchange and Applications, International Symposium, RuleML 2009, Las Vegas, Nevada, USA, November 5-7, 2009. Proceedings*, volume 5858 of *Lecture Notes in Computer Science*, pages 323–336. Springer, 2009. ISBN 978-3-642-04984-2. (Cited on page 103.)

[7] D. Sottara and P. Mello. Modelling radial basis functions with rational logic rules. In E. Corchado, A. Abraham, and W. Pedrycz,

editors, *Hybrid Artificial Intelligence Systems, Third International Workshop, HAIS 2008, Burgos, Spain, September 24-26, 2008. Proceedings*, volume 5271 of *Lecture Notes in Computer Science*, pages 337–344. Springer, 2008.

[8] D. Sottara, L. Luccarini, P. Mello, S. Grilli, M. Mancini, and G.L. Bragadin. Tecniche di intelligenza artificiale per la gestione e il controllo di impianti di depurazione. caso di studio: SBR in scala pilota alimentato con refluo reale. In Luciano Morselli, editor, *Ambiente: tecnologie, controlli e certificazioni per il recupero e la valorizzazione di materiali ed energie. ECOMONDO X Fiera Internazionale del Recupero di Materia ed Energia e dello Sviluppo Sostenibile. Rimini. 8-11 novembre 2006*, volume 1, pages 106 – 111. Maggioli Editore (ITALY), 2006. ISBN 88-387-3887-1.

[9] D. Sottara, L. Luccarini, and P. Mello. AI techniques for Waste Water Treatment Plant control. Case study: Denitrification in a pilot-scale SBR. In B. Apolloni, R. J. Howlett, and L. C. Jain, editors, *Knowledge-Based Intelligent Information and Engineering Systems, 11th International Conference, KES 2007, XVII Italian Workshop on Neural Networks, Vietri sul Mare, Italy, September 12-14, 2007. Proceedings, Part I*, volume 4692 of *Lecture Notes in Computer Science*, pages 639–646. Springer, 2007. ISBN 978-3-540-74817-5. (Cited on pages 203 and 239.)

[10] D. Sottara, P. Mello, and M. Proctor. Adding uncertainty to a RETE-OO inference engine. In N. Bassiliades, G. Governatori, and A. Paschke, editors, *Rule Representation, Interchange and Reasoning on the Web, International Symposium, RuleML 2008, Orlando, FL, USA, October 30-31, 2008. Proceedings*, volume 5321 of *Lecture Notes in Computer Science*, pages 104–118. Springer, 2008. ISBN 978-3-540-88807-9.

[11] D. Sottara, G. Colombini, L. Luccarini, and P. Mello. A Pool of Experts to evaluate the evolution of biological processes in SBR plants. In E. Corchado, X. Wu, E. Oja, Á. Herrero, and B. Baruque, editors, *Hybrid Artificial Intelligence Systems, 4th International Conference, HAIS 2009, Salamanca, Spain, June 10-12, 2009. Proceedings*, volume 5572 of *Lecture Notes in Computer Science*, pages 368–375. Springer, 2009. ISBN 978-3-642-02318-7. (Cited on pages 203, 205, 207, and 247.)

[12] D. Sottara, G. Colombini, L. Luccarini, and P. Mello. A wavelet based heuristic to dimension neural networks for simple signal approximation. In Bruno Apolloni, Simone Bassis, and Carlo F. Morabito, editors, *Proceeding of the 2009 conference on Neural Nets, WIRN 2009, Vietri sul Mare (SA), Italy, May 28-30, 2009*, pages 337–344. IOS Press, 2009. ISBN 978-1-60750-072-8. (Cited on page 204.)

[13] D. Sottara, L. Luccarini, G.L. Bragadin, M.L. Mancini, P. Mello, and M. Montali. Process quality assessment in automatic manage-

ment of wastewater treatment plants using formal verification. In *International Symposium on Sanitary and Environmental Engineering-SIDISA 08 -Proceedings*, volume 1, pages 152/1 – 152/8, ROMA – ITA, 24-27 june 2008 2009. ANDIS.

[14] D. Sottara, A. Manservisi, P. Mello, G. Colombini, and L. Luccarini. A CEP-based SOA for the management of wastewater treatment plants. In *EESMS 2009. IEEE Workshop on Environmental, Energy, and Structural Monitoring Systems, 2009. Proceedings*, pages 58–65, 25/09/ 2009. doi: 10.1109/EESMS.2009.5341314. URL http://dx.medra.org/10.1109/EESMS.2009.5341314.

[15] D. Sottara, P. Mello, L. Luccarini, G. Colombini, and A. Manservisi. Controllo intelligente in linea per una gestione efficiente e sostenibile degli impianti di trattamento reflui. Caso di studio: SBR in scala pilota. In *Ecodesign per il pianeta: soluzioni per un ambiente pulito e per una nuova economia*, pages 655 – 660, S.Arcangelo di Romagna (RN) – ITA, 28-31 Ottobre 2009. Maggioli Editore (ITALY).

[16] D. Sottara, P. Mello, and M. Proctor. Towards modelling defeasible reasoning with imperfection in production rule systems. In G. Governatori, J. Hall, and A. Paschke, editors, *Rule Interchange and Applications, International Symposium, RuleML 2009, Las Vegas, Nevada, USA, November 5-7, 2009. Proceedings*, volume 5858 of *Lecture Notes in Computer Science*, pages 345–352. Springer, 2009. ISBN 978-3-642-04984-2. (Cited on pages 170 and 171.)

[17] N. Wulff and D. Sottara. Fuzzy reasoning with a RETE-OO Rule Engine. In G. Governatori, J. Hall, and A. Paschke, editors, *Rule Interchange and Applications, International Symposium, RuleML 2009, Las Vegas, Nevada, USA, November 5-7, 2009. Proceedings*, volume 5858 of *Lecture Notes in Computer Science*, pages 337–344. Springer, 2009. ISBN 978-3-642-04984-2. (Cited on page 179.)

ACKNOWLEDGMENTS

I wish to thank all the people who have supported, encouraged and helped me during this PhD. I would like to begin with my supervisor, Prof. Paola Mello, for all the support, the advice and the guidance during these years. I would also like to thank Luca Luccarini for the trust placed in me, and because without his vision this project would never have started, nor would have been carried on despite all the difficulties. Much credit also goes to Mark Proctor, for all the enlightening discussions at the most improbable times and places, and to Gabriele and Alberto for their help in hard coding and bug fighting.

More than a simple thank you goes to all the people I've had the pleasure to work with: at DEIS, at ENEA, at Newcastle and the DROOLS team all over the world. The list would simply be too long to appear here, so excuse me if I don't cite every one of you.

I would like to acknowledge the role of the companies who supported and sponsored this reasearch activity. Other than the University of Bologna and the National Agency ENEA, first and foremost credit goes to HERA s.p.a. for all the resources, both monetary and material they provided. My thanks also go to JBoss and SPES for the technological and logistic support. Part of the project has also been supported by the Italian MIUR PRIN 2007 project No. 20077WWCR8.

Al di lá della importante esperienza culturale e lavorativa, devo ringraziare di cuore tante persone sul piano umano, perché credo che quanto ho guadagnato sul piano personale in questi anni, grazie a tutti voi, valga immensamente di piú. Paola, Luca, Mark, voi siete tra i primi anche da questo punto di vista. Eppure, il primo ringraziamento va alla mia famiglia, che mi ha supportato e sopportato, per non avermi fatto mai mancare nulla, in primis il vostro affetto. Devo ringraziare tutti gli amici, vecchi e nuovi. Coloro che conosco da una vita, perché negli anni il rapporto si é rinsaldato invece che dissolversi. Tutti i ragazzi e le ragazze che nel sono passati dai laboratori di via dei Fornaciai, perché non siamo stati solo colleghi. Gli amici dell'universitá, trovati, talvolta persi di vista e poi ritrovati. La compagnia di Newcastle: I miss you lads and lasses! I ragazzi del Judo, la cui amicizia supera i muri della palestra - e grazie di tutto anche a te, maestro Giorgio, sei sempre nel mio cuore. E onore alle PaC, immancabili compagni d'avventura. Chiudo con una menzione speciale per un paio di persone speciali. Non faccio nomi, che non mi piace fare preferenze tra le tante persone con cui posso dire di avere un rapporto di amicizia nel senso piú vero e profondo del termine. In cuor vostro sapete a chi mi riferisco. Tuttavia... - Ale, Fante, non me ne vogliate, ma grazie per esserci sempre stati, fosse per divertirsi o per confidarsi, e soprattutto grazie perché se sono qui é anche merito vostro.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

bma basic mass assignment

AI	Artificial Intelligence
API	Application Programming Interface
AR	Associative Rules
ASM	Activated Sludge Model
AST	Abstract Syntax Tree
BDI	Belief - Desire - Intention
BLIP	Business Logic Integration Platform
BLP	Bayesian Logic Program
BN	Bayesian Network
BNN	Bayesian Neural Network
BNR	Biological Nutrient Removal Plant
BPMN	Business Process Modelling Notation
BRMS	Business Rule Management System
CART	Classification and Regression Tree
CBR	Content-Based Router
CF	Certainty Factor
CL	Clustering and Classification Algorithm
CBR	Case-Based Reasoning
CE	DRL Conditional Element
CEP	Complex Event Processing
DB	Data Base
DO	Dissolved Oxygen
DSS	Decision Support System
ECA	Event, Condition, Action
EDA	Event-Driven Architecture
EDSS	Environmental Decision Support System
EM	Expectation-Maximization
EPA	Event Processing Agent
EPN	Event Processing Network
ES	Expert System
ESB	Enterprise Service Bus
FE	Fuzzy Engine
FF-NN	Feed Forward Neural Network
FLP	Fuzzy Logic Programming

FOL	First Order Logic
FS	Fuzzy System
GIS	Geographic Information Systems
GPM	Gaussian Process Modelling
HC	Hard Computing
HS	Hybrid System
ICA	Instrumentation, Control and Automation
IDM	Imprecise Dirichlet Model
KB	Knowledge Base
KBS	Knowledge-Based System
LHS	Left-Hand Side
LP	Logic Programming
LTI	Linear, Time-Invariant
MAS	Multi-Agent System
MBR	Membrane Bio-Reactor
MP	Modus Ponens
MSE	Mean Square Error
MVL	Many-Valued Logic
N/C	Name/Category
NFS	Neuro-Fuzzy System
NLN	Neural Logic Network
NN	Neural Network
NAF	Negation As Failure
ORP	Oxidation / Reduction Potential
P-DEN	Pre-Denitrification Treatment Plant
PID	Proportional Integral Derivative
POJO	Plain Old Java Object
PLC	Programmable Logic Controller
PRS	Production Rule System
RBF	Radial Basis Function
RBS	Rule-Based System
RHS	Right-Hand Side
RTC	Real-Time Control
SC	Soft Computing

SCADA	Supervisory Control And Data Acquisition
SBR	Sequencing Batch Reactor
SOA	Service Oriented Architecture
SOM	Self-Organizing Map
TBM	Transferable Belief Model
TMS	Truth Maintenance System
WM	Working Memory
WME	Working Memory Element
WS	Web Service
WWTP	Waste Water Treatment Plant

1

INTRODUCTION

Contents

The main thesis defended in this dissertation is that while there exists a dicothomy between "Soft" and "Hard" Artificial Intelligence Computing Techniques, they are not alternatives, but complementary, and they can not only be combined, but *integrated* - i.e. blended into a functioning and unified whole[1] - at different levels to develop *hybrid* systems which outperform their single components when dealing with complex problems.

Historically, different AI-based technologies have had periods of great diffusion, followed by other periods in which they were almost forgotten and replaced by other more popular ones. Just to cite some, one can think of the rise of Neural Networks in the 40s, their decline in the 70s and their new popularity in the 80s. Similar considerations apply to expert systems, which possibly lived their golden age in the 70s and the 80s, or fuzzy systems, which were the object of many controversies, until only recently have become widely accepted. The problem is that there exist many classes of tools claiming to be "intelligent", so that they can solve efficiently and effectively complex problems for which standard algorithmic or numeric techniques are not sufficient, and every class contains dozens of variants. This poses a problem for devel-

1 Merriam-Webster Dictionary

opers and users alike: the worst choice an engineer can do to solve a problem is to choose a tool because of its popularity, or because it has solved *many other* problems.

When one looks at them in greater detail, it turns out that each technology has benefits and drawbacks, making it more suitable for certain classes of problems and less suitable for others. Many real-world problems, however, are *complex*: they can be decomposed in a set of sub-problems which are intercorrelated, so that they can be solved independently only in rare cases. To give an idea of the main case study addressed in this dissertation, consider the problem of "optimizing a bio-chemical plant": to do so, it is first necessary to define what *optimizing* means, but then the solution will likely involve different tasks such as estimation (of the unobservable variables defining the process state), prediction (of the evolution in time of the process), diagnosis (of malfunctionings), control (of manipulable variables), learning with adaptation (to variable environmental conditions) and so on, all of which involve different types of knowledge on the domain and appropriate policies.

When dealing with such problems, it is almost impossible that there exists one single tool capable of dealing appropriately with all the issues: *complex solutions*, then, are often *hybrid*, as they integrate different technologies to cope with different challenges. Many of the existing hybrid systems, however, are more properly hybrid tools: they are combinations of hardly more than two basic tools, and still designed for very specific problems. Moreover, the level of integration varies from full integration, where data and structures are shared, to (more often) the simple application of individual modules, cascaded or connected in parallel.

What seems to be lacking, instead, is a framework for large scale hybridization, where tools can interact, integrate or even *emulate* each other as needed. With perhaps a bit of ambition, the ultimate goal remains the human intelligence. Humans are capable of performing quite different intelligent tasks, to the point that there exist several definitions of intelligence. Some of them - reasoning, for example - are "conscious", i.e. they require awareness and involve an explicit representation of the information processed and the actions performed, to the point that they can be communicated or described. Some others, like learning or recognizing shapes, are "unconscious": they take place at lower levels, relying on mechanisms that are not evident, but instead seem almost automatic. There is a strong analogy between this distinction and the separation of AI techniques in "Hard" and "Soft" computing, with the former being conscious and the latter unconscious. In humans, however, there is a strong feedback between conscious and unconscious processes, to the point that the flow of "control" often shifts seamlessly from one level to the other; moreover, the boundary is not neat, as there are tasks that can be performed at both levels, and the levels can sometimes emulate each other. Consider, for example, the problem of multiplying two numbers. The product of simple num-

bers is usually performed associatively (i.e. in an unconscious manner); when complexity increases, people usually shift to an explicit approach, but children tend to do so even for simple cases, while there exist people capable of performing unconscious multiplications of (very) large numbers!

This background motivates the research of *strongly hybrid* architectures, where different AI techniques can not only interact, but integrate each other at different levels of abstraction, matching the complexity of the problems they have to solve. This work, then, will discuss in detail the already cited complex problem of the management of a biochemical process plant, in particular studying the case of activated sludge waste water treatment plants, showing the benefits of developing and applying a hybrid architecture to it.

1.1 BACKGROUND

The project that will be discussed in this dissertation was born out of the cooperation of different parties with different goals and interests. The detrimental effects of water pollution, together with the strict normatives on water discharge, have increased the strategic relevance of water treatment plants. Water treatment, however, is an expensive and fault-prone process which could require constant monitoring and management to be kept at full efficiency. Tese reasons motivated the start of a research project on plant optimization and control within the PROT-IDR section of ENEA[2], the (former) National Agency for the Energy and Environment. At first, the participation in two European Projects (TELEMAC[3] and EOLI[4]) led to the definition of some analytical techniques and management policies, tested on pilot- and laboratory-scale plants, in addition to the development of a remote data acquisition and control interface tool, which now has become a product commercialized by SPES[5]. Successively, the studies have been carried out in cooperation with HERA[6], the local multiutility for water, energy and environmental services and owner of several full-scale treatment plants.

While much had been done from the point of view of data acquisition, and several criteria had been found which could potentially exploit the data coming from the plants, these policies were not effectively translated into an automatic management system, but rather into a *remote* one, with little support for "intelligent" automation and, even then, with practically no infrastructure for the interaction between modules.

My work, then, has been focused two goals: the use of AI for the development of modules implementing the theoretical control policies

2 www.enea.it
3 www.ercim.eu/telemac/
4 www.inma.ucl.ac.be/EOLI/
5 www.spesonline.com
6 www.gruppohera.it

and the development of an adequate management infrastructure. The two are not independent, since to achieve the desired level of integration between AI techniques it was necessary to condition the structure of the architecture, which in turn exploits some features of an intelligent system. In designing and developing the systems, a few additional constraints have been taken into account: the real-world finality of the application suggested the use of mainstream, yet open-source softwares, to ensure a good degree of reliabilty and support. In particular, the choice has fallen on JBoss'[7] products: the JBossESB communication middleware and the Drools business logic integration platform, both of which will be discussed extensively. The former has been customized in some parts and used for the infrastructure; the second, instead, has been used as the core logic and integration component for the development of hybrid AI modules. This required a significant enhancement of its core engine, part of which was done during a stay at the University of Newcastle, where the components are developed. This extension, which is now implemented in a test prototype, will soon likely be integrated as an optional module in the main release.

1.2 CONTRIBUTIONS OF THE DISSERTATION

In greater detail, the innovations introduced by this dissertation are:

1.2.1 *Creating an (imperfect) bridge between symbolic and sub-symbolic systems*

Analyzing the concrete problem domain, it turns out that most of - in not all - the information to be processed is affected by some form of imperfection, be it uncertainty, vagueness, confidence or a combination thereof. After defining the concepts, it will be shown that imperfection can even be beneficial in terms of consistency robustness, provided that it is dealt with in a coherent manner, using the appropriate technique.

After that, several AI techniques, which are commonly applied to deal with problems like the ones emerging from automatic (plant) managemenet will be described, showing their applicability as well as their limits, and analyzed using functional criteria such as interpretability and flexibility. This will show that soft and hard computing are complementary, both in the classes of problems they can solve and in the properties they exhibit in doing so. Moreover, it will be discussed that imperfection is a fundamental factor in choosing the appropriate tool to solve a problem, so it should be taken into account both at design time and at run time.

Most importantly, it will be argued that imperfection has a crucial role in integrating symbolic and connectionist tools. In fact, the boundary between the two classes becomes less strict, provided that a sym-

7 www.jboss.org

bolic system is enhanced with the support for imperfect reasoning, and that a sub-symbolic system is given a logic interpretation, which, due to the nature of this class of tools, can't but be imperfect.

1.2.2 *Enhancing the RETE algorithm with imperfection*

Despite its usefulness, especially when trying to achieve *strong* hybridization, symbolic reasoning systems do not support imperfection natively (in fact, many optimization they exploit are based on the opposite, unrealistic assumption of perfect information). Analyzing the most widely used algorithm for the development of forward-chain rule engines, RETE, it was found that it actually can execute imperfect reasonings in a native way, provided that:

- The underlying inference mechanism is generalized appropriately.
- The rule language is extended to include new features, both to increase the expressiveness of the language and to specify configuration meta-data to condition the behavior of the engine.
- The engine itself is enhanced, extending the default structure with new functionalities and adding some new ones altogether.

A RETE-based engine has many advantages, both in terms of performance and expressiveness. Drools itself uses RETE for its core, but comes with a set of additional features and components which are shared only by main commercial equivalent tools. The possibility of enhancing the core of Drools allowed to exploit the novel features while still having all the additional benefits at disposal.

1.2.3 *Developing a complex, strongly hybrid management system*

The flexibility of the engine will be demonstrated using it to implement many different examples, which are representative of some common problems solved using AI. The engine will be used both as a standalone component and in a hybrid version with other sub-symbolic tools at different levels of integration, from cascading to full emulation. In order to show its real usefulness, however, a more realistic application will be implemented as well.

Modern complex systems use architectural patterns which are becoming standard best practices: in particular, many complex problems nowadays are solved using (Web) Service-Oriented Architectures. Enterprise applications are commonly built using an enterprise service bus and a rule engine, leveraging advantages such as flexibility, reliability, reusability and expandability. Some systems also introduce additional concepts, such as *process* or *event*; in fact, Drools supports them natively. Actually, all these concepts can be conveniently applied in the development of a management system for a water treatment

plant, but it will also be shown that imperfection-aware, strongly hybrid, AI-based components can be a fundamental added value.

1.3 ORGANIZATION OF THE DISSERTATION

The dissertation is divided in four parts, as follows:

1. Artificial Intelligence Techniques
2. Monitoring and Control from an AI perspective
3. A Hybrid Rule Engine
4. Case Study: a Hybrid Environmental Decision Support System

The first two parts are more survey-oriented and give an overview of the state of the art in different fields, albeit from a point of view functional to the development of the system presented in the third and fourth part of the work, which constitutes the concrete applicative contribution of this work. In particular, the first part introduces the concept of imperfect information and shows how it can be managed appropriately using AI techniques, stressing the differences between the various types, but also the points of contact between them. This is first done from a theoretical point of view, but then the analysis is applied to concrete technologies, emphasizing their imperfect nature and showing how it can be exploited. The second part introduces the problem of the control and management of complex systems, systems for which the canonical model-based control schemas are not applicable, so advanced AI-based architectures are required instead. The third part describes the properties of the hybrid rule engine, showing the new language features and the revised internal structure. Finally, the last part will discuss how to apply imperfect reasoning to a concrete case study - the implementation of a complex management architecture -, outlining the advantages in doing so.

The rationale behind this sequence is as follows: handling imperfection is the key to develop strong hybrid systems, so it is defined in the first place. The evaluation of the benefits and the drawbacks of the presented AI tools, in fact, depends also on their capacity (or lack thereof) of handling imperfection. Knowing the properties of AI technologies is necessary to understand the structure of the control infrastructures commonly applied to complex (water treatment) systems. From an analysis of the most advanced existing solutions, it turns out that there is still a great margin of improvement: while in other fields architectures such as the ones based on services and/or events have been succesfully adopted, applications to water treatment are scarcer; moreover, none supports imperfection in a comprehensive way[8]. From a brief survey of the mainstream middleware components, both open source and commercial - it turns out that this lack of support is structural, so the only option was to add it explicitly, enhancing an existing

8 fact which seems to hint that AI tools are not always used at their full potentialities

engine. The enhancements greatly expand the potentialities of the engine, hinted at using several example applications and implementing a realistic one, applied to a water treatment plant, which significantly improves the structure and functionalities of the existing ones.

The contents of each chapter are given in detail below.

1.3.1 *Artificial Intelligence Techniques*

- Chapter 2 defines the concept of imperfect information, discussing the differences between uncertainty, vagueness and inconsistency. It also introduces the canonical techniques used to deal with imperfection, namely the probabilistic and possibilistic/fuzzy approaches, showing that their adoption can be much more beneficial than a naive approach which ignores the imperfection altogether.
- Chapter 3 introduces several AI techniques, both from the field of hard and soft computing, which will be applied in the realization of the case study architecture. For each different approach, its capability (or lack thereof) of handling imperfect information is outlined.
- Chapter 4 defines the concept of hybrid AI system, giving some criteria to classify them. It also lists the most common hybrid architectures which can be found in literature and real-world implementations.

1.3.2 *Monitoring and Control from an AI perspective*

- Chapter 5 presents the vision of an integrated architecture for the automatic management of a complex system, introducing - and at the same time delimiting the context to - the main case study: waste-water treatment plants. The opportunities and the challenges for the automation of a plant are discussed, followed by an overview of the possible architectures which outlines the benefits and drawbacks of each one. The overview starts from the simpler and more focused techniques, such as PID controllers implemented using PLC hardware, moving to full-fledged Decision Support Systems. It turns out that an ideal DSS, to be sufficiently scalable, flexible and maintainable, should not be built as a monolithic application, but using a combination of service-, event- and agent-based architectures.

1.3.3 *A Hybrid Rule Engine*

- Chapter 6 analyzes the existing BRMS, the candidate technology to form the main building block of a complex management architecture. The relevant functionalities include the support for

workflows, events and imperfection: while the former are common in commercial systems, the latter is not (with a few limited exceptions).

- Chapter 7 shows the modifications made to an existing RETE-based rule engine, Drools, necessary to have it support different forms of imperfect reasoning. To this end, a generalized and highly configurable inference process has been implemented and embedded in the underlying RETE network.
- Chapter 8 shows the potentialities of the hybrid engine, presenting several examples of possible applications. They are meant to be "patterns" more than real applications, serving as guidelines in the development of hybrid systems. They will be used to discuss what can be done at the symbolic level, using rules, and what needs to be done (or is better done) at a lower level, using other connectionist techniques. The examples include "pure" logic extensions of rules to the non-boolean case, the introduction of non-monotonic reasoning through induction or exceptions, the integration - up to the emulation - and usage of other soft-computing techniques.

1.3.4 *Case Study: a Hybrid EDSS*

- Chapter 9 presents a specific WWTP class, the Sequencing Batch Reactors, which are particularly suitable for control and monitoring purposes. To support this claim, the results of the application of several AI techniques to the offline analysis of its signals are shown, many of which were proposed by ourselves.
- Chapter 10 proposes a generic, hybrid, distributed management system based on rule agents who implement services and/or handle events. Thanks to the imperfect reasoning engine, the agents can handle imperfection at rule-level natively. This architecture is applied to the *online* control of a SBR plant: to this end, some specialized, hybrid agents are designed combining different AI techniques.

Part I

ARTIFICIAL INTELLIGENCE TECHNIQUES

2

DEALING WITH IMPERFECT INFORMATION

Contents

The ideal piece of information coming from the real world is *perfect* [277]: it is *precise* and *certain*, so that its values are neither ambiguous nor inaccurate and there is no reason to doubt their validity. *Imperfection*

In practice, this is just an ideal situation: the quality of the data collected in a realistic scenario is hardly so high; moreover, an information processing system is usually just a model, though as realistic as possible, which is likely to introduce some additional error during the computations.

Thus, it is unrealistic to think that the results of a computation are perfect. One can still build a system and ignore the imperfections, but the validity of the output would have to be questioned every time: this could be relatively easy whenever the results are not the expected ones, but could be impossible and even dangerous in certain situations (imagine for example an undetected alarm condition).

A much better option is to build a system that can recognize the quality of the information it processes, deal with it and possibly evaluate the quality of its responses. Obviously, the "garbage-in garbage-out" principle still holds, but there is a continuum of situations between pure noise and perfect information that can still be exploited. In fact, despite the negative acception of the term, an imperfect representation of knowledge may be more concise, robust and less expensive to obtain than its perfect version. For example, the use of an interval - e.g. the one given by confidence bounds - in place of a single number is an inexpensive, yet more expressive way of reporting a measurement. Consider also the age-related version of the *Sorite Paradox*: if a person is young on one day, they will also be young the day after, until the

day when they will be old. This is a paradox in classical logic unless a different adjective is defined for every day in the life of a person, but is perfectly acceptable in an imperfect logic, where a single property, "young", has a truth degree that varies continuously with age.

The drawback is that imperfection may appear in more than one way, due to different causes, even at the same time. This (lack of) information has to be processed in a coherent way, because it is part of the data. The goal of this Chapter, thus, is to give a brief description of some of the most common sources of imperfection and to introduce their main theoretical and mathematical models. The AI techniques discussed in the following Chapters will then be analysed according to their capacity to deal with which types of imperfect data: a clear distinction is essential not to make serious mistakes. Data, in fact, may be characterized by two or more types of imperfection, which should not be confused. Moreover, when complex systems are built from simpler modules, such as in the main case study presented in this dissertation, the output of one block may become the input of another, so imperfection has to be propagated along with the information.

2.1 PROPERTIES OF IMPERFECT INFORMATION

Several attempts to describe and classify the various types of imperfection in a standard framework exist in literature: among them, the already cited survey by Smets [277] but also, for example [147] and [170]. More recently, the W3C *Incubator Group on Uncertainty*[1] *Reasoning for the Web* has defined an ontology (see [3]) for the representation of imperfect information on the Web. The classification presented in this chapter derives principally from a combination of the first and last work: an adapted version of the original ontology is shown in Figure 1.

2.1.1 *Sources of Imperfection*

Before it is possible to model imperfection, one has to take into account its origin. At a very abstract level two distinctions can be made:

ALEATORY VS EPISTEMIC Sometimes the imperfection is inherent in the state of the world, other times it is due to a limited knowledge of the observer. This distinction could lead to a philosophical debate between idealism and determinism: would it possible to determine the outcome, say, of a coin toss for an observer with complete knowledge of the state of the universe? From a practical point of view, instead, it is more important to distinguish between knowledge that *could* be completed and information that *can't* be acquired.

1 Notice that here the term Uncertainty is used in the wider sense of Imperfection

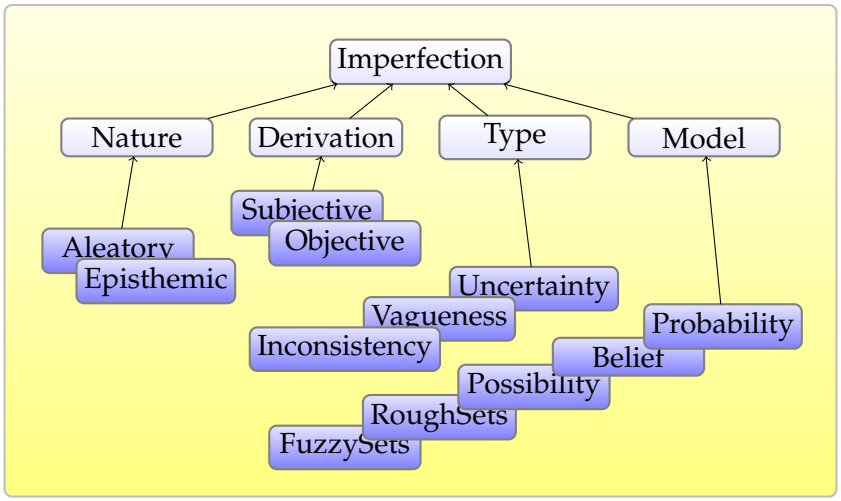

Figure 1: An ontology for the representation of Imperfect information

OBJECTIVE VS SUBJECTIVE Another property relevant in practice is the repeatability of the conditions that led to the generation of the imperfect information. An objective event can be repeated and thus described and predicted, so it is generally more tractable than one where the information depends only on the judgement of an observer.

2.1.2 *Types and Causes of Imperfection*

Despite much philosophical debate, it is generally accepted that imperfection can take on the following major forms (among others):

UNCERTAINTY properly called derives from the lack of knowledge about the actual state of the world. A fact or an event, be it past, present or future, may be true or false, but the available information does not allow to decide which is the case. More generally, the real value of a variable is known to lie in a set of alternatives, but it can't be identified uniquely.

Objective Uncertainty

Aleatory uncertainty is often related to *randomness*, the impossibility to predict exactly the outcome of an event before it takes place, such as the number resulting from casting a die. The degree of uncertainty is reflected by the likelihood of the event. When dealing with epistemic uncertainty, a subjective judgement has to be adopted: the degree of uncertainty is expressed by the *degree of belief*, which in turn can be related to the necessity and the possibility of an event. A typical example is the outcome of a football match. Notice that the subjective belief can be supported by an objective likelihood, while the converse is normally not acceptable. When belief is used as a second-order uncertainty measure, for example to state the quality of an obser-

Subjective Uncertainty

Confidence

vation together with the analysis of its degree of imperfection, it can also be called *confidence*.

Incompleteness

Another source of uncertainty is *incompleteness*: when a piece of information is missing, one can only speculate on its possible state, even if sometimes the deficiency may be irrelevant for the purpose of the elaborations at hand.

VAGUENESS arises when knowledge is as complete as it can be, but the terms used to denote it do not allow to identify the entities which are being referred in a precise and unequivocal way. When vagueness is due to *ambiguity* or *approximation*, there is more than one possible interpretation. The first more properly defines situations where it's difficult to decide between two different alternatives, while the second term is more appropriate for situations where many similar states are collapsed into a unique class: for example, consider a temperature which is known to lie somewhere in between 20 and 25 degrees. Sometimes, instead, a single concept, such as "age", takes a specific value, but the boundaries of its definition are relaxed - usually *fuzzified* - in some way: for example, "young".

Fuzziness

INCONSISTENCY is a property of a set of facts with *conflicting* information, such that there is no possible world it can describe (e.g. a person is reputed to be younger than 15 but has a driving license). Conflicts have to be resolved, usually removing, ignoring or modifying part or all the conflicting information. Inconsistency may be a symptom of *incorrect* or noisy information, even if sometimes erroneous information does not lead to any (apparent) inconsistency. When the errors are small and not relevant, however, the information is better defined *inaccurate*.

Incorrect vs Inaccurate

2.1.3 *Models of Imperfection*

In literature there exist several theories dedicated to the modelling of imperfection, but each one usually deals with one specific type of imperfection. Since a complete discussion is not among the goals of this work, this section will just give a brief introduction of the most relevant ones, outlining the purpose and the domain in which they are applicable. The next Section, instead, will discuss the similarities, the differences and the points of contact of different theories, hinting at possible ways to combine them.

Probabilistic Approaches to Uncertainty

The canonical model to deal with uncertainty is the theory of probability. The field is actually split in two branches, one focused on objective, random variables, the other dedicated to the study of subjective belief.

PROBABILITY THEORY Probability is a way to measure, i.e. to assign a quantitative description, to the uncertainty about the state of the world. It can be used by an agent who has limited knowledge about an event, either because it has not been observed precisely, and thus some data are missing or incomplete, or because it has not happened yet. In particular, a random variable X models a property which value is not known with certainty, but which can take any one of the values in a set $\Omega \subseteq \mathcal{U}$ included in a "universe of the discourse". An event causes the variable to be assigned a specific value at a certain point in time (e.g. the outcome of a coin toss may assign heads or tails to the variable which models the side the coin landed on), but obviously until the event takes place *and* has been observed, the value of the variable can't be known. The agent, however, can assign a probability to each possible outcome which models the different degrees of belief at which a values is considered (or expected) to be the "real" one. In fact, the set Ω is associated to a *probability distribution*, a non-negative, normalized function $f : \Omega \mapsto [0, 1]$ which assigns a probability to each candidate outcome. Notice that one is not always necessarily interested in atomic events (i.e. $p(X = x_i)$), but it is possible to estimate more complex situations such as $p(X \in A \subseteq \Omega)$. Since for mathematical convenience a random variable can only take real values, in many cases an encoding in necessary (e.g. mapping heads to 1 and tails to 0 in the coin example).

Random Variables

Probability Distribution

The laws of probability are traditionally governed by Kolmogorov's axioms. Given two outcomes A and $B \in \Omega$[2]:

Probability Axioms

$$
\begin{aligned}
&p(A) \in [0, 1] \\
&p(\emptyset) = 0 \\
&p(\Omega) = 1 \\
&p(A \cup B) = p(A) + p(B) \Leftrightarrow A \cap B = \emptyset
\end{aligned}
$$

from which one can derive the product rule for the conditional probability $p(A \cap B) = p(A) \cdot p(B|A)$ which is at the core of Bayes' theorem

Bayes' theorem

$$
p(A|B) = \frac{p(B|A) \cdot p(A)}{p(B)} \tag{2.1}
$$

A random variable is usually described by its *moments*: the most common are the expected value $E[X] = \sum_{i:x_i \in \Omega} x_i \cdot p(x_i)$ and the variance $E[(X - E[X])^2] = \sum_{i:x_i \in \Omega} (x_i - E[X])^2 \cdot p(x_i)$. The first gives an idea of the value which the variable will take on average, while the second is necessary to estimate the dispersion of the outcomes around this value. The expected value alone, in fact, may not have a physical meaning, like in the case of the coin toss, where the expected value of 0.5 does not correspond to any possible outcome. Moreover, a variable with low variance is much stabler and thus more predictable than one with high variance.

Mean and Variance

2 when the variable is obvious, it will be omitted

Other than these factual points, there is actually much debate on what probability is [269] and especially how it should be estimated [289]. In fact, probabilities can be interpreted in terms of:

- **Frequency**: when the uncertainty is due to randomness in an aleatory but repeatable process, the probabilities can be measured objectively in terms of relative frequencies. In particular, $p(A)$ is taken to be the number of time the outcome has turned out to be compatible with A divided by the total number of times the experiment was repeated. While rigorous, this approach obviously fails when the events are not repeatable (and even then, one must pay attention to carry out the experiments in similar conditions).

  Repeated Trials

  Moreover, the estimate becomes effectively reliable as the number of trials goes to infinity. If one models the experiments using random *characteristic* variables, i.e. $X_j = 1$ if the j-th experiment was compatible with A, 0 otherwise, the probability $p(A)$ is the expected value of the sum $\sum_j X_j$. The law of large numbers then guarantees that the convergence is *almost sure*, i.e. that the probability that computed values is not correct is negligible.

- **Betting Behaviour**: this opposite approach is more suitable for non-repeatable events, where the subjective disposition of the observer is crucial. The underlying metaphor is the one of a betting game, where one has to spend a reasonable amount of *utility* (often measured in terms of currency), knowing that a unit will be won if event A takes place, but the original amount will be lost otherwise. A rational agent would not bet an amount superior to the equivalent of the probability $p(A)$ itself. On the converse, one can take the subjective probability to be equal to the maximum acceptable bet, so the probability assignment is totally a matter of personal preference.

  Subjective Disposition

- **Evidence**: in many cases, however, the subjective belief can be supported and/or modified by some available, objective evidence. At the limit, the result of a sequence of repeated trials can be considered the optimal piece of evidence when trying to assign a probability to an event. Such an observer implicitly accepts the frequency principle proposed in [143].

Bayesian probability

Some authors stress the difference between subjective and objective probability, at the point that the former is more properly called *belief* and the notation $\mathrm{bel}(A)$ and $p(A)$ is used to distinguish between the two types. Obviously, where appropriate, $\mathrm{bel}(A) = p(A)$. Belief is at the center of the Bayesian approach (see [57] for a comprehensive presentation with several applicative cases). The actual state of knowledge is encoded by some parameter θ, so every prediction is conditioned by their values: $\mathrm{bel}(A) = \mathrm{bel}(A \mid \theta)$. As observations ω are made, the knowledge is updated using Bayes' theorem: the *posterior* $\mathrm{bel}(\theta|\omega)$ depends on the product of the *prior* $\mathrm{bel}(\theta)$ and the conditional $\mathrm{bel}(\omega \mid \theta)$.

For this reason, in Bayesian frameworks great importance is held by *conjugate* distributions, families of parametric distributions closed with respect to belief update, i.e. that retain the same form after being combined using Bayes' theorem. One of the most widely used is the Gaussian, but many others exist [57]:

$$\mathrm{bel}(x|\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{||x-\mu||^2}{2\sigma^2}}$$

IMPRECISE PROBABILITIES AND THE IMPRECISE DIRICHLET MODEL Probabilities can be used to measure uncertainty on the outcome of an event, but there are situations where it is not possible to estimate a distribution with precision, so probabilities become uncertain themselves. A possible way to deal with this uncertainty is to use interval bounds in place of precise real values, so $p(x)$ is not determined unequivocally, but assumed to lie in a range $[p_l(x), p_u(x)]$.

The underlying theory was developed by P. Walley [294], stressing the subjective, behavioural aspect of assigning probabilities to events. In this framework, a random variable X is associated to a *gamble*, a function $g(X) : \Omega \mapsto \mathfrak{R}$ which assigns a gain, measured in terms of utility units (a generalization of currency), to each possible outcome. A betting agent will earn (or lose) an amount equal to $g(A)$, assuming that event A takes place, i.e. the variable X actually assumes one of the values in A. Thus, the *lower prevision* $P_l(g_X)$ is defined as the maximum price the agent would be willing to pay to accept the bet avoiding sure loss. For example, suppose that $\Omega = \{a, b, c\}$ and $g(a) = 10, g(b) = 2, g(c) = -5$: an acceptable price to "enter the game" would obviously depend on the agent's belief regarding the possible outcome of the gamble. If an agent believed that a would surely happen, they could be willing to pay up to $P_u(g_X)$ is the maximum acceptable selling price. In the example, an agent could even be willing to sell the gamble at negative rates (i.e. pay up to 5) according to the expectation concerning c[3].

Betting behaviour: previsions

The definition of avoiding sure loss leads to that of coherence [293], which does not allow the existence of a combination of gambles making it acceptable to buy another gamble at a price greater than its lower prevision.

The notion of coherent prevision is more general than that of probability, and in fact includes it. It is sufficient to consider the *characteristic gamble* of an event A, where one wins 1 unit if A takes place and loses the bet utility otherwise: the probability $p(A)$ is exactly the *fair* price one would pay to accept the gamble if the odds were known with precision. When this is not the case, the upper and lower prevision (which, in this case, are effectively the upper and lower probabilities of A) show the agent's disposition, induced by their subjective belief, towards the outcome of the event. More generally, even the belief/plausibility and

Probabilities as gambles

3 so beware of extremely convenient gambles!

necessity/possibility pairs introduced in the following sections can be considered specific cases of coherent previsions.

Imprecise Dirichlet Model

An interesting case study, useful to define probability bounds for multinomial data while still using a Bayesian approach, is the Imprecise Dirichlet Model (IDM, [292],[54]). A Dirichlet model is appropriate to describe events with a finite number of possible outcomes (i.e. $|\Omega| = N < +\text{inf}$. The Dirichlet distribution is a second-order distribution, which gives the likelihood $p(\mathbf{p}|\beta)$ of any probability distribution $\mathbf{p} = \{p(\omega_1), \ldots, p(\omega_N)\}$ on Ω, given the values of a vector of parameters β modelling the belief of the observer:

$$\text{Dirichlet}_N(\mathbf{p}|\beta) = \frac{\Gamma\left(\sum_{j=0}^{N-1} \beta_j + N\right)}{\prod_{j=0}^{N-1} \Gamma\left(\beta_j + 1\right)} \prod_{j=0}^{N-1} p_j^{\beta_j} \tag{2.2}$$

The parameters β_j have a precise frequentist interpretation: they are the actual number of observations of the j^{th} event over a total number of observations $B_{tot} = \sum_{j=0}^{N-1} \beta_j$. If the parameters are be fixed on a subjective basis, one can speak of "observation-equivalents". The unique maximum of distribution 2.2 yields the maximum likelihood distribution $\mathbf{p}^{ml}$, which coincides with the frequentist one as the number of observations grows, since it can be shown that the overall variance along the N coordinates is bounded by a constant which tends to 0 as B_{tot} goes to infinity:

$$\mathbf{p}^{ml} = \frac{\beta}{B_{tot}} \qquad \sum_{j=0}^{N-1} \sigma_j^2 \leqslant \frac{N(N-1)}{N^2(B_{tot} + N + 1)}$$

As observations are made, the conditioning of the posterior distribution just requires an update of the counters β_j. One usually starts with a prior number of observation-equivalents B_D and a prior probability distribution d_j (for example, a uniform distribution such that $d_j = 1/N$). The total number of observations B_{tot} is given by the sum of B_D and the number of samples B_O. Defined δ_j the fraction of real observations assigned to category j, one can define a probability distribution and its *empirical* bounds:

$$p_l(\omega_j) = \frac{\delta(j)B_O}{B_O + B_D} \leqslant \frac{\delta(j)B_O + d_j B_D}{B_O + B_D} \leqslant \frac{\delta(j)B_O + B_D}{B_O + B_D} = p_u(\omega_j)$$

The lower bound is based purely on real observations, while the upper bound assigns admits that all the prior belief could have been assigned to any one class indifferently. As B_O increases, the effect of the priors is diminished and the bounds are tightened.

DEMPSTER-SHAFER THEORY AND TBM A probability distribution assigns a probability to each element $\omega \in \Omega$. Sometimes this

kind of assignment fails to capture the belief of an observer, since it does not allow to assign a belief to a set of possible outcomes without making any distinction between them. This scenario, instead, is quite common when observations are uncertain and don't allow to distinguish clearly between elements. Historically, this issue has been studied by Dempster-Shafer's theory of belief and plausibility functions [268]. In this framework, a *basic mass assignment* m (bma), is defined on the power-set 2^Ω. This function has all the characteristics of a probability distribution; moreover, the sets A for which $m(A) > 0$ are called *focal elements*. The value of $m(A)$ measures the belief that a random variable X has one of the values in A, without saying anything about the belief about any of them individually. In particular, assigning mass to Ω itself models a condition of total ignorance, while assigning mass to $\emptyset$ is usually a symptom of inconsistency in the available information since X can't take any value in Ω (a contradiction). An example is shown in Figure 2, where each node of the lattice is annotated with its mass, while *bel* and *pl* are in the lower and upper part of the circles respectively. Given a bma, it is possible to define two quantities, *belief* $bel(A)$ and *plausibility* $pl(A)$, which may understood as the lower and upper bounds, respectively, of a subjective probability $p(A)$:

Mass assignments, Belief and Plausibility

$$bel(A) = \sum_{\emptyset \neq B \subseteq A} m(B) \quad pl(A) = \sum_{B \cap A \neq \emptyset} m(B)$$

In fact, bel takes into account only the mass which *certainly* supports an event A, while pl takes into account any event which *might* possibly involve A. The relation, valid for previsions in general, holds: $bel(A) = 1 - pl(\neg A)$. Notice that the functions are strictly interrelated, since, given bel, it is possible to reconstruct the underlying bma: $m(A) = \sum_{B \subseteq A} (-1)^{|A-B|} bel(B)$.

A basic mass assignment m_0 can be used to model the uncertain belief of an agent, summarizing all his available knowledge. When new information is acquired, it updates the agent's belief. The only requirement is that this additional knowledge m_1 is encoded using the bma it alone would induce on the agent. Two basic mass assignments can be merged using the well-known Dempster-Shafer's combination rule:

Belief combination

$$m(A) = \frac{1}{1 - \sum_{B \cap C = \emptyset} m_0(B) \star m_1(C)} \sum_{B \cap C = A \neq \emptyset} m_0(B) \star m_1(C)$$

In the original rule, $\star$ is the product, but it is not the only possible operation. In fact, several combination rules exist, some of them including a "discounting" pre-processing which takes into account the reliability of the sources: a discussion of the benefits and drawbacks of other, different rules can be found, for example, in [267] and [276].

Smets' Transferable Belief Model [275] extends this scenario: information coming from different sources is performed at credal level using bmas, but eventually decisions are taken at probabilistic level. This

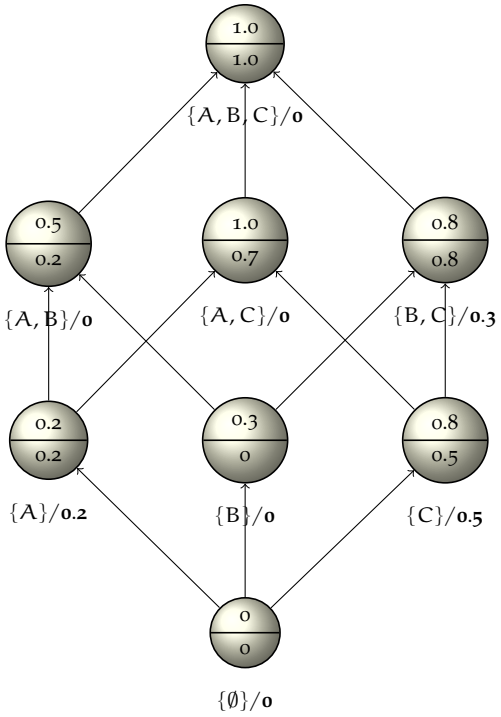

Figure 2: Example: bma, *bel* and *pl* on $2^{\Omega=\{A,B,C\}}$

requires the conversion of a generic bma into a Bayesian one, i.e. a probability distribution on Ω. The original model used the pignistic transformation, according to which a mass is shared equally between the elements of a set:

$$m_B(\omega) = \sum_{A \ni \omega} \frac{m(A)}{|A|}$$

IDM and TBM

To conclude this introduction, notice that the Imprecise Dirichlet model can be extended to take into account belief and plausibility. So far, it has been assumed that an experiment returned a definite outcome, so that it was always possible to count the number of occurrences of each element ω_j during the repeated trials. An uncertain observation could not allow to identify a precise outcome ω, but if a subset $J \in 2^\Omega$ can be identified such that it surely includes the real outcome (at worst, Ω itself), the TBM can be applied. In practice, one can define the belief, pignistic and plausibility functions, κ, δ and λ respectively, which assign a different weight to each element ω_j depending on J :

$$\kappa_j(J) = \begin{cases} 1 & \text{if } J = \{j\} \\ 0 & \text{else} \end{cases} \qquad \lambda_j(J) = \begin{cases} 1 & \text{if } j \in J \\ 0 & \text{else} \end{cases}$$

$$\delta_j(J) = \begin{cases} \frac{1}{|J|} & \text{if } j \in J \\ 0 & \text{else} \end{cases}$$

These in turn can be used to define a pignistic probability and its upper and lower bounds in case of uncertain observations:

$$\frac{\sum_{t=0}^{T} \kappa_j(J(t))B_O}{B_O + B_D} \leqslant \frac{\sum_{t=0}^{T} \delta_j(J(t))B_O + d_j B_D}{B_O + B_D} \leqslant \frac{\sum_{t=0}^{T} \lambda_j(J(t))B_O + B_D}{B_O + B_D}$$

CONFIDENCE The intuitive, but vague, notion of *confidence* does not correspond to a well defined theoretical model. Instead, it has sometimes been used naively in place of more formal frameworks, like in early expert systems. Its proper role, however, is that of a second-order measure of uncertainty [296], used together with another more structured approach. Confidence is usually measured using a real value $\chi \in [0, 1]$ and can be assigned like a weight to any piece of information. It becomes useful when knowledge coming from different sources has to be merged: in particular, when the data are inconsistent, confidence can be used to solve the conflicts. Examples of this appear in non-monotonic reasoning (e.g. [242]), but also in the TBM, where the fusion of bmas can include a discounting step in which the the discount coefficient can be determined by confidence itself. Moreover, the problem of evaluating the degrees of confidence is not trivial, and may vary on a case-by-case basis. Notice, for example, that the IDM includes a natural candidate definition of confidence: the ratio $\frac{B_O}{B_O+B_D}$ is a non decreasing value which increases with the number of observations. Moreover, the variance of the estimated probability distributions tends to 0 as $\chi \to 1$. In fact, the approach in [296] can be considered a binomial IDM.

Possibilistic Approaches to Vagueness

While probabilistic techniques deal with uncertainty, the methods used to treat vagueness are based on the theory of fuzzy sets and its developments.

FUZZY SETS Traditional sets can be defined extensionally, by numbering their elements, or intensionally, by providing a boolean *characteristic function* which evaluates to 1 if an element is member of the set or to 0 otherwise.

Fuzzy sets [309] generalize this notion. A fuzzy set S on a universe U is composed of elements $x \in U$ which have a gradual degree of membership $\varepsilon \in [0, 1]$ in the set. If the set is defined intensionally, the characteristic function is called more properly *membership function* and usually denoted by $\mu_S : U \mapsto [0, 1]$. A higher value means a higher degree of compatibility between x and any ideal member (or "prototype") of the set. In particular, the set of full members $S \supseteq C_S = \{x \mid \mu_S(x) = 1\}$

Fuzzy membership

is called the *core* of S, while the *support* of S is the complementary of the set of full non-members, i.e. $\{x \mid \mu_S(x) > 0\}$.

α-cuts

A fuzzy set can be cast to a traditional, crisp set by taking its α-cut : $S_\alpha = \{x \mid \mu_S(x) > \alpha\}$. As the parameter α increases, one obtains a family of nested subsets of the support of S. The operation is invertible, as one can define the membership of an element by taking $\mu_S(x) = \sup_\alpha : x \in S_\alpha$.

Operations on fuzzy sets

The theory of crisp sets can be generalized to fuzzy sets by defining the fuzzy counterparts of the operations of negation $\neg$, intersection $\cap$ and union cup. In particular, when A and B are fuzzy sets:

$$\begin{aligned} \mu(\neg A) &= 1 - \mu(A) \\ \mu(A \cap B) &= \min\{\mu(A), \mu(B)\} \\ \mu(A \cup B) &= \max\{\mu(A), \mu(B)\} \end{aligned}$$

The definitions above preserve commutativity, distributivity, idem-potency and other desirable properties of an algebraic structure, including De Morgan's laws, but do not respect the law of the excluded middle, but this is perfectly acceptable in a context where a person, for example, can be at the same time young and not young (e.g. a 25-years old student), even if only up to a limited degree, since $\mu(A \cap \neg A) < 0.5$.

Semantics of fuzzy sets

Fuzzy sets are a convenient way to define concepts expressed using vague linguistic descriptions, such as "Old", "Tall" and "High", and reason with and over them. Often, the evaluation of the membership function associated to a set relies on some quantitative feature of the object to be tested. For example, the function $\mu_{Tall}(\text{Person } x)$ is likely to rely on the height of the people it tests, at the point that it should be more explicitly written $\mu_{Tall}(x \mid x.height)$. Notice that the features are assumed to be known with certainty, so that μ can be evaluated precisely. In fact, fuzzy sets deal with vaguely-bounded concepts by assigning a graded, partial degree of truth to a statement such as "x is Tall", but once x is known and "Tall" is defined, no more imprecision actually exists. This situation is actually quite different from a scenario where one does not know the height (or, more generally, the value of the necessary features) with precision, and even more from the case, apparently similar, where one only knows that a person is tall in some degree and needs to estimate the value of its height. Fuzzy sets can be based on other semantics than similarity ([108]): in fact, the latter case is handled by the theory of possibility introduced in the next paragraph, while the former requires more complex representations than a simple real-valued membership degree, some of which will be presented in Section 2.2.

Boolean Possibility Theory

POSSIBILITY THEORY Possibility [111] is another - possible - approach to dealing with imperfection, in particular with incomplete knowledge, and can be derived from a (modal) logic point of view, at least in its original form. While probability may be used to quantify their belief on the actual state of a partially known world, *possibility* may instead be used to assess its definite provability. This theory is

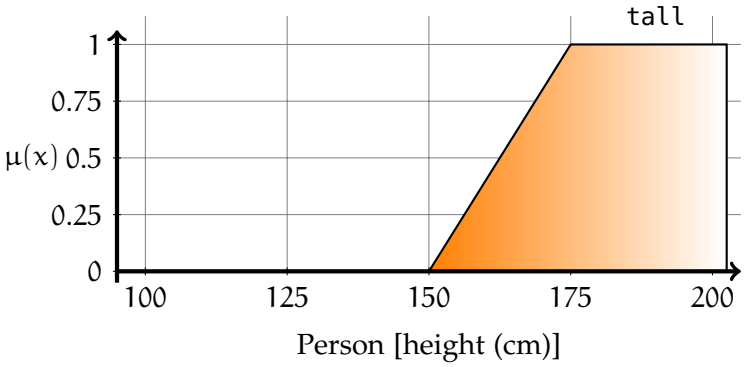

Figure 3: The fuzzy set `Tall`

founded on two concepts, *Necessity* $N(A)$ and *Possibility* $\Pi(A)$: the former describes events that **must** happen, the latter event that **may** do so. A necessary event is one that is expected to happen, since its truth can be proved according to the knowledge in possess of an agent; a possible one is not expected not to happen, since its falsity can't be proved with certainty.

From this point of view, necessity and possibility are boolean variables such that, given a statement or unitary event x, $N(x) = 1 \Rightarrow \Pi(x) = 1$ and $N(x) = 1 - \Pi(\neg x)$ (hence also $\Pi(x) = 0 \Rightarrow N(x) = 0$): intuitively, a necessary fact is also possible, but if a fact is necessary then its negation is not possible. Hence, four states are possible:

- $N = 1, P = 1$: an agent is certain of the truth of a fact
- $N = 0, P = 0$: an agent is certain of the falsity of a fact
- $N = 0, P = 1$: an agent is uncertain about a fact, which *could* be true but needs not to
- $N = 1, P = 0$: an agent's knowledge is inconsistent, since an impossible event is expected to be true.

The definitions can be generalized from atomic facts to events by taking $\Pi(A) = \sup_{x \in A} \Pi(x)$ together with $\Pi(\Omega) = 1$. This definition leads to the following composition rules, from which it results that one can't generally obtain definite values for N and Π, but only lower and upper bounds:

$$N(A \cup B) \geqslant \max\{N(A), N(B)\} \quad \Pi(A \cup B) = \max\{\Pi(A), \Pi(B)\}$$
$$N(A \cap B) = \min\{N(A), N(B)\} \quad \Pi(A \cap B) \leqslant \min\{\Pi(A), \Pi(B)\}$$

Fuzzy Possibility Theory

On the other hand, given a set Ω, the evaluation of $\Pi(x \in \Omega)$ for each gives a *Possibility distribution*. If one substitutes the original definition of impossibility to prove the contrary with a fuzzier notion of how acceptable is for a value x to be the actual one in a given situation, one gets a fuzzy-based possibility theory as proposed by Zadeh

in [306] and furthered by Dubois and Prade [117] (the dual notion of necessity can be related to a fuzzy degree of surprise in *not* seeing an event happen). In this context, a fuzzy set S can be considered a possibility distribution on its domain, given the fact that the actual (unknown) value belongs to S set in some degree. For example, one could know that a man x is tall in a degree greater or equal than some value α: his actual height is not known with certainty, but S (or, more properly, its α-cut) defines a possibility distribution $\pi_{Tall}(h|x)$ stating, for each candidate value h, how acceptable it would be, should it be chosen as the real value. Notice that this is the dual case of the fuzzy membership evaluation, where the quantitative feature (e.g. the height) is known with precision and the membership has to be computed: the implicit equivalence $\mu_S(x|h) = \pi_S(h|x)$ is stated by Zadeh's equivalence principle.

2.2 RELATIONS BETWEEN DIFFERENT TYPES OF IMPERFECTION

The goal of this Section is to remark the differences as well as the points of contact between the main models of imperfection, namely fuzziness, probability/belief and possibility. As usual, only a few key points will be discussed, since whole works exist dedicated to the specific topic, such as [110] and [111].

2.2.1 *A Comparison of Imperfection Types*

Fuzziness models partial grades of truth, evaluated with full knowledge of the necessary information. The only uncertainty can arise when trying to decide whether a fuzzy property can be considered true in a boolean sense, since a threshold has to be chosen, but otherwise the state of the world is certain.

Probability is applied in conditions of uncertainty, where an agent lacks information about the state of truth of some fact, which is boolean and not gradual: it measures their disposition in believing the world to be in a specific state, despite the inability to know it for sure.

Possibility, in a way, measures the "prejudice" - or lack thereof - of an agent towards any candidate actual situation, given his present state of incomplete knowledge, so, like probability, is not defined at a different conceptual level than truth degrees.

Probability vs Fuzziness

Two quite famous examples can be used to explain the difference between the three concepts. In the first ([56]), a thirsty person is offered two bottles, a fuzzy one and a probabilistic one. The former is a bottle of fresh water with truth degree 0.9, for example because it is not exactly fresh water, but a mixture made for 9 parts of water and 1 of something else; the latter is a bottle full of fresh water with probability 90%, or, with 10% probability, a bottle full of something else. While it is true that the expected amount of non-water from the second choice is equal to the amount present in the first, in one case the person is

certain to drink but a few drops of that something else, while in the other they will either drink none (a likely case) or all.

Probability vs Possibility

The distinction between possibility and probability is shown clearly by Zadeh in [306], considering the number of eggs that an agent, Hans, could eat for breakfast. A probability distribution assigns a weight to each value, but what is effectively important is the ratio of these values (the odds), in order to distinguish the more from the less likely when trying to identify the actual value. Possibility, instead, is concerned with the individual values. Indeed, what is impossible (i.e. $p = 0$) must remain impossible (i.e. $\Pi = 0$), but a possible event may be improbable, as stated by Zadeh's possibility/probability consistency principle.

Possibility vs Fuzziness

As already noted, instead, fuzziness and possibility have a dual role: the former gives a qualitative label (but evaluated using a quantitative truth degree) to a quantitative feature; the latter tries to limit the set of values compatible with a vague definition.

DEGREES More confusion could be induced by the use of the term "degree". In fact, the same word can be applied to quite different concepts, all of which share the property of being gradual:

- Degree of **truth**: measures the compatibility of an entity with a prototype defined using some criterion.
- Degree of **probability**: measures the ratio of favourable events over a total number of cases.
- Degree of **belief**: measures an agent's opinion in assuming a property to be true.
- Degree of **possibility**: measures an agent's disposition towards accepting a situation to be true.
- Degree of **confidence**: measures the strength of an agent's rely upon a statement.

To make things worse, the same basic model, a real number in $L = [0, 1]$ is suitable for any of the types of degree just listed. However, a real number is appropriate only if the value of the degree is known with certainty. When this is not the case, one has to model an imperfect piece of information on $\Omega = L$, so any imperfect (meta) model could theoretically be used.

- **Real numbers** $\varepsilon \in L$: the standard representation, suitable to model any one degree.
- **Intervals** $[\tau, 1 - \varphi] \in L \times L$: interval ranges are obviously used for upper and lower previsions, so imprecise probabilities as well as belief/plausibility and necessity/possibility pairs rely on this representation, even if with different semantics. Intervals can also be used to model fuzzy degrees of truth: moreover, they emerge naturally when type-II fuzzy sets are used [310], [201]. A type-II fuzzy set is defined by an imperfect membership function $\mu : \Omega \mapsto L \times L$ which evaluates to an interval instead of a precise degree for any given object.

- **Fuzzy numbers**: A fuzzy number is a possibility distribution on (a subset of) $\mathfrak{R}$ (e.g. see [118]). A fuzzy number models the concept of "approximately" ε, admitting more than one possible value (at different degrees). A fuzzy interval can be considered a special case of fuzzy number with uniform possibility equal to 1 for all values between the lower and upper bound.
- **Fuzzy degrees**: A fuzzy degree is one of the values of a linguistic variable used to model the concept of partial truth, such as `true`, `approximately true` or `almost false`. A fuzzy degree is connected to a fuzzy set, which in turn can be considered a fuzzy number.
- **Belief structures**: in complex cases, where the actual value of a degree has to be decided using evidential information, the TBM could be used, defining a belief structure on 2^L.

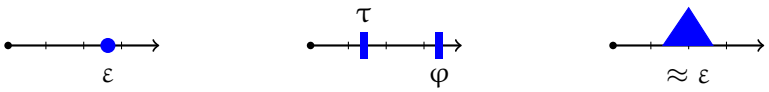

Figure 4: Simple Degrees

Remarkably, these complex implementations can model any class of degrees, so for example a fuzzy degree can be used to describe an ill-known probability value. This allows to build higher-order imperfection models, such as the already discussed imprecise probabilities, which use intervals instead of precise values. The membership function of a type-II fuzzy set is a fuzzy set itself, but the idea can be generalized recursively to type-n sets [263]. Another relevant example is given in [97], where a TBM is implemented using imperfect degrees.

2.2.2 *Reconciling the differences*

As shown in the previous Section, the different types of imperfection can be combined, even in complex ways, but hardly subsume each other, unless very strong and specific assumptions are made. Actually, there exist several works in literature where the topic is debated (e.g. [182], [110], [116], [86], [212], [50], in addition to the already cited works of Smets and Dubois and Prade), some more well-founded than others, and possibly the question is not closed to this date.

The discussion exists at two levels, a "strong" and a "weak" one: the former is more theoretical, and arguments to what degree one model of imperfection can be encompassed by another; the latter, instead, is more practical and tries to determine which model is more appropriate for different classes of problems.[4]

Fuzzy Probability

For example, it is possible to define the fuzzy set `Probable` over

4 In my humble opinion, the former remains open for discussion.

the domain of events: the membership of an event A, then, can be assumed to be equal to $p(A)$, but this is just a qualitative description of a property based on a well-known, quantitative feature, i.e. the degree of probability, pretty much like the set `Tall` is defined over the height of people. A general way to transform a partial degree of truth into a probability could be expressed by the predicate logic formula[5]:

$$\forall X, P \exists Y : \texttt{Holds}(P, Y) \wedge \texttt{Similar}(X, Y) \Rightarrow \texttt{Probable}(\texttt{Holds}(P, X))$$

`Similar` is a place-holder for a generic fuzzy property which evaluates to a partial truth according to the compatibility between an object X and a prototype Y; `Holds` is a boolean property stating that a certain property P is true for Y. When the latter is true, the degree of similarity between X and Y is taken to be equal to the membership of $\texttt{Holds}(P, X)$ in the set of probable events, i.e. to be equal to the probability that P is true for X. This axiom is arbitrary (albeit in some contexts it could be considered reasonable) because it is based on the strong assumption that the limit of similarity, equality, is a sufficient condition to ensure that a property holds, while only identity guarantees it. Moreover, assuming that probability increases with similarity also implies that probability decreases the less compatible X and Y are: but even so, it could be possible (albeit improbable) for P to hold fully even for an object X which is very different from Y.

At the same time, the attempts to define a membership degree in term of a frequentist probability, i.e. to consider $\varepsilon = \mu_S(x)$ equal to the percentage of people who would accept x to be member of a crisp version of S', have been strongly criticized (e.g. see [56]). To obtain a crisp set S', an α-cut is required: for n people out of N, with $\varepsilon = n/N$, to assign x to S', it is necessary that n people choose a threshold `alpha` lesser or equal than ε, while the other $N - n$ choose a value greater than ε. This means that, for any person, $p(\alpha \leqslant \varepsilon) = \varepsilon$, i.e. that the prior distribution for α is uniform or, in other words, that there is not preferable threshold for mapping a partial truth degree onto an absolute one.

Belief in Fuzzy Degrees

Things may be different if one considers *subjective* probabilities, i.e. an agent's belief, instead of frequentist ones, i.e. chance. While the truth degree of a fuzzy property S can be evaluated unequivocally when the membership function μ and the argument x are known, there remains the question of how μ was determined in the first place. In a recent work ([86]), Coletti and Scozzafava show that the membership $\mu_S(x)$ could be interpreted as an agent's partial belief in the truth of x having property S. It could be argued that for an agent who accepts the definition of S (i.e. the membership function), their belief is induced by μ, but for an agent who is defining μ, it is this one which is induced by the agent's disposition. Nevertheless, in this second case the membership $\mu(x)$ behaves like a subjective belief (i.e. a probability) *conditioned* by the knowledge of x.

5 A comprehensive discussion of the relation between logic and imperfection will be the topic of the second part of this work

Despite the theoretical controversies, however, the problem of "weak" reconciliation is trivial: since imperfection may manifest in different forms, and different techniques have been *optimized* to deal with each one, the design and implementation of an efficient information elaboration system robust with respect to imperfection should always apply the most appropriate technique. Most practical cases can be assimilated to the computation of some function $\phi : X \mapsto Y$, so that, given an input $x \in X$ it necessary to find a value $y \in Y$ such that $y = \phi(x)$. Candidate ϕ may include characteristic functions $\chi_S(x)$, membership functions $\mu_S(x)$, real-valued functions $f(x)$ and (conditional) probability distributions $p(y|x)$, so this model is general enough to evaluate any type of imperfect information. In turn, the input x may be imperfect in some way: in fact, it could be specified using a possibility distribution (fuzzy or not) $\pi_X(x)$ or by a probability distribution $p_X(x)$[6]. Thus, the evaluation of ϕ requires a generalized composition principle which takes into account both the imperfection associated to the input and the one introduced by the function:

Imperfect Function Evaluation

$$y = \phi(X) \star (X = x)$$

Depending on the combination, the composition principle becomes:

PERFECT INPUT

When x is perfectly known, the output is precisely determined, regardless of its semantics.

POSSIBILISTIC INPUT VS CHARACTERISTIC FUNCTION

The possibility distribution entails both

$$\Pi(x \in S) = \sup_{x|\chi(x)=1} \{\Pi(x)\}$$

and

$$\Pi(x \notin S) = \sup_{x|\chi(x)=0} \{\Pi(x)\}$$

i.e. both the possibility that x belongs to S and the possibility that x does not belong to S. The two values are sufficient, since $N(x \in S) = 1 - \Pi(x \notin S)$ (resp. for $N(x \notin S)$).

POSSIBILISTIC INPUT VS MEMBERSHIP FUNCTION

The composition operation $\star$ is equivalent to a logical conjunction: the input has to be equal to x `and` x must be a member of S. The possibility distribution over the input effectively limits the domain to a fuzzy subset of X, so one gets a truth interval degree bounded by

$$N_S = \inf_x \min(\mu(x), 1 - \pi(x))$$

and by

$$\Pi_S = \sup_x \min(\mu(x), \pi(x))$$

.

6 for simplicity, only continuous variables will be considered

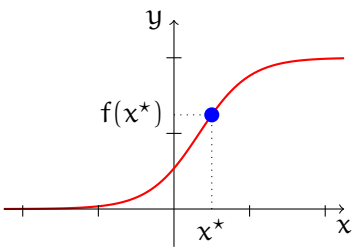

Figure 5: Real-valued function of a *perfect* input

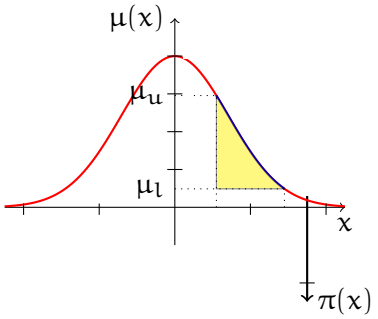

Figure 6: Membership function of a possibilistic input

POSSIBILISTIC INPUT VS REAL-VALUED FUNCTION

The distribution $\pi(x)$ induces a possibility distribution in output. For y to be possible, there must exist a possible x such that $y = f(x)$, so :

$$\Pi(y) = \sup_{x|y=f(x)} \{\pi(x)\}$$

. The dual notion of necessity is associated to the impossibility that the input is mapped onto a value different from y, which translates into:

$$N(y) = 1 - \inf_{x|y\neq f(x)} \{\pi(x)\}$$

POSSIBILISTIC INPUT VS PROBABILISTIC OUTPUT

For each value y, one gets a second-order possibility distribution on $[0, 1$, i.e. the possibility that any value $alpha \in [0, 1]$ is the actual value of $p(y)$. This value

$$\Pi(\alpha) = \Pi(p(y) = \alpha) = \sup_{x|p(y|x)=\alpha} \{\pi(x)\}$$

. As usual when possibility is involved,

$$N(\alpha) = N(p(y) = \alpha) = \inf_{x|p(y|x)\neq\alpha} \{1 - \pi(x)\}$$

.

PROBABILISTIC INPUT VS CHARACTERISTIC FUNCTION

In this case, it is possible to compute the probability that x belongs to S, integrating the probability distribution over the part of space that overlaps with the domain of S:

$$p(x \in S) = \int_{x \mid \chi(x)=1} p(x)dx$$

PROBABILISTIC INPUT VS MEMBERSHIP FUNCTION

When x is uncertain, it is impossible to determine a unique membership degree, but a lower and an upper bound can be obtained considering only the possible input values

$$\mu_l = \inf_{x|p(x)>0} \mu(x)$$

and

$$\mu_u = \sup_{x|p(x)>0} \mu(x)$$

. The expected membership value can also be computed: $E[\mu(x)] = \int \mu(x) \cdot p(x)dx$. Considering μ a generic real-valued function, the following case also applies.

PROBABILISTIC INPUT VS REAL-VALUED FUNCTION

A real-valued function of a random variable induces a probability distribution on the output space. Assuming that p_X is the probability density on X, the corresponding p_Y on Y can be found, under some mild constraints, computing the inverse of f and using the well-known formula:

$$p_Y(y) = p_X(f^{-1}(y)) \left| \frac{df^{-1}(y)}{dy} \right|$$

PROBABILISTIC INPUT VS PROBABILISTIC OUTPUT

When the input and joint distribution are known, it is possible to compute the general output distribution

$$p(y) = \int_X p(y|x) \cdot p(x)dx$$

Table 1 summarizes the results:

	$\chi(x)$	$\mu(x)$	$f(x)$	$p(y\|x)$
x	$\chi(x)$	$\mu(x)$	$f(x)$	$p(y\|x)$
$\pi(x)$	$\Pi(x \in S)$ $\Pi(x \notin S)$	$[N_S, \Pi_S]$	$[N(y), \Pi(y)]$	$[N(p(y)),$ $\Pi(p(y))]$
$p(x)$	$p(x \in S)$	$[\mu_l, \mu_u]$ $E[\mu]$	$p(y)$	$p(y)$

Table 1: Effects of Imperfection in I/O

2.3 CONCLUSIONS

While far from being a comprehensive list of all the existing literature, this Chapter tried to give an overall picture of the different types of imperfection which can affect information, which at the greatest level of abstraction can be assimilated to vagueness and/or uncertainty. This is not necessarily a negative quality of information (and in most real-world systems it can't be avoided anyway), but the worst thing an information processing system can do is ignore it. Instead, it should be considered an alternative type of information, albeit less desirable than the pure one, and processed in a coherent way.

There exist several models of the different types of imperfection, some of which are alternative, while others are complementary. However, each model has been studied within a theoretical framework which provides the techniques necessary to elaborate and propagate the imperfection. These theories have important differences, but also several points of contact which allow to use them in a way that is not necessarily mutually exclusive, depending on the contingent necessities of the applicative context.

In conclusion, it is reasonable to assume that imperfection is an intrinsic feature of the real world, which also pervades the way humans think and act. So, artificial intelligence-inspired information processing systems should not treat their data as if they were perfect. In the next chapters, it will be shown that most AI tools can handle at least one type of imperfection natively, or can be extended to do so. Eventually, it will be shown how the combination of different techniques, empowered with imperfection-handling capabilities, actually *facilitates* the development of complex yet robust "intelligent" systems.

3

AI TECHNIQUES

Contents

This chapter is dedicated to introducing some of the most popular AI techniques, both from the field of Soft and Hard Computing. Since the topic is vast and a comprehensive description is not a goal of this thesis, for each technology only a brief and partial introduction will be given. Instead, the discussion will be focused on how each technique models uncertainty and/or vagueness. The accuracy of the analysis will also be greater for the ones which have actually been applied to the problems discussed in the following chapters.

3.1 HARD COMPUTING

Hard Computing is a term coined for the "traditional" approaches in AI, as opposed to the class of *Soft* algorithms described later in this Chapter. To give a positive, essential definition, Hard Computing denotes all *perfect* Artificial Intelligence techniques. It is based on the ideas of *correctness* and *completeness*: given a problem, an intelligent solver should be able to find all and only the feasible solutions in a finite time [1].

Symbolic Systems

HC systems tend to prefer a *symbolic* approach: an abstract model of the problem is built assigning a symbol to each real world entity, then solutions are searched by manipulation of the symbols according to some predetermined rules. The search process is based on boolean logical reasoning: the problems which have traditionally been addressed

1 a requirement which could be criticized for being neither sufficient nor necessary to consider an algorithm "intelligent", but only "smart"

include theorem proving, planning, natural language processing and games [258]. The main advantages of HC systems are their declarative and modular approach: they knowledge they model applies to various domains on a case-by-case basis, but is seldom procedural; moreover they are usually *monotonic*, so that knowledge can be expanded as new information is acquired without backward compatibility issues.

3.1.1 *Premise - Formal Logic*

Most HC techniques are based on *formal logic*, a special case of formal language. A formal language is a class of *expressions*, sequences of symbols belonging to an *alphabet*, which must respect some *grammatical* constraints. In formal logic, the expressions - or *formulas* - model statements about the world: the language, then, is extended with *inference* rules which allow, given a set of formulas subject to some specific constraints, to obtain new formulas of the language by mere syntactical manipulation. These transformations are meant to model the logical[2] *reasoning* a human would perform. In particular, **First-Order Predicate Logic** (FOL) is a formal logic which can be used to reason on the properties of the world, but not on logic itself.

The "world" an intelligent being would reason about is a set of entities $\Omega = \{x_1 \dots x_N\}$, which can be biunivocally mapped on a set of symbolic constants K such that each $c \in K$ refers to one and only one individual $x \in \Omega$ [144]. The association between a constant and its entity is called *evaluation* or *interpretation* and will be denoted by $\|\cdot\|$, so that $x_j = \|c_j\|$. Whenever and entity can't be identified univocally, a *variable* X or a list of variables $\mathbf{X}$ will be used: a variable still refers to an entity, i.e. $\|X\| = x_\star$, but this entity may change every time the evaluation is performed. An entity can also be referred through a function $f(\cdot)$, a construct which takes n *terms* as arguments and uses them to identify the entity to refer. Constants, variables and functions can be terms, so $x_f = \|f(t_1, \dots, t_n)\| = f(\|t_1\|, \dots, \|t_n\|)$.

Evaluation

FOL LANGUAGE Terms t_i can be used to build *well-formed* formulas according to the grammatical constraints:

- **Predicates** : $p(t_1, \dots, t_n)$ is an (atomic) formula, where p is an n-ary predicate symbol.

  A predicate expresses a relation that may hold or not between its term arguments: $\|p(t_1, \dots, t_n)\| = p(\|t_1\|, \dots, \|t_n\|) \in \mathtt{true}, \mathtt{false}$
- **Equality** : $t_j == t_k$ is a formula. $==$ is actually a special predicate, such that if $==(t_j, t_k)$ holds, then $x = \|t_j\| = \|t_k\|$.
- **Negation** : if φ is a formula, then $\neg\varphi$ is a formula. The logical negation inverts the truth degree of a formula, so whenever $\|\varphi\| = \mathtt{true}$ then $\|\neg\varphi\| = \mathtt{false}$ and vice versa.

2 in the sense of Aristoteles

- **Connectives** : if φ and γ are formulas, then $\varphi \vee \gamma$ is a formula. The semantics of $\vee$ is that of a logical disjunction, so its evaluation corresponds to the well-known truth table of the "`or`" operator (i.e. `true` when either operand is `true`).

  The other traditional connectives can be defined from $\neg$ and $\vee$:
  - Conjunction : $\varphi \wedge \gamma \doteq \neg(\neg\varphi \vee \neg\gamma)$
  - Equivalence : $\varphi \equiv \gamma \doteq (\varphi \wedge \gamma) \vee (\neg\varphi \wedge \neg\gamma)$
  - Exclusive disjunction : $\varphi \neq \gamma \doteq \neg(\varphi \equiv \gamma)$
  - Implication : $\varphi \rightarrow \gamma \doteq \neg\varphi \vee \gamma$

  A logical connective can be considered a truth-functional predicate, e.g. $\vee(\varphi, \gamma)$, which evaluation is a function of the evaluations of its operands.

- **Quantifiers** : if X is a variable and $\varphi(X)$ a formula containing X, then $\forall X : \varphi(X)$ is a formula. The interpretation of this new formula is obtained by testing φ using all its possible values: $\min_{X \setminus c \in K} \|\varphi(c)\|$.

  The dual quantifier $\exists$ is defined as $\neg\forall\neg$.

INFERENCE Of all formulas, particular relevance is given to the universally quantified implications: $\forall \mathbf{X}, \mathbf{Y} : \varphi(\mathbf{X}) \rightarrow \gamma(\mathbf{Y})$, which is the basic component of the canonical deductive inference rule, *modus ponens*:

$$\frac{\varphi_P(\mathbf{A}) \ , \ \forall \mathbf{X}, \mathbf{Y} : \varphi(\mathbf{X}) \rightarrow \gamma(\mathbf{Y})}{\gamma_C(\mathbf{B})} \tag{3.1}$$

Deduction is not the only inference process allowed in FOL: others such as abduction, induction, chaining, merging and reductio [158] can be used. In general, the inference rules operate on a *theory* T, a collection of formulas, eliciting new knowledge in the form of new formulas. Properly, T *entails* a *valid* formula φ if any evaluation $\|\cdot\|$ under which T is true also gives $\|\varphi = \texttt{true}\|$. The inference process is *correct* if no invalid formula is entailed and *complete* if all valid formulas can be entailed. Moreover, a formula is *decidable* if either φ or $\neg\varphi$ can be entailed [138]. The actual "reasoning", i.e. the application of the inference rules to a theory, is performed by a dedicated, general purpose component called *inference engine*. The separation between knowledge and reasoning is shown in Figure 7: knowledge is then roughly divided in rules and facts, while the engine relies on two memories, a short term and a long term one. While there is not always a biunivocal correspondence, facts may be contingent and be meaningful only for a limited time, whereas the validity of rules and other facts spans over several reasoning sessions, so the engine may treat them differently to improve performance.

Inference Engine

The different choices of language and engine define the various HC approaches in building an intelligent application. Roughly speaking, systems can be divided in two main classes, Logic Programming and Rule-Based Systems, but only the latter will be discussed. In any case,

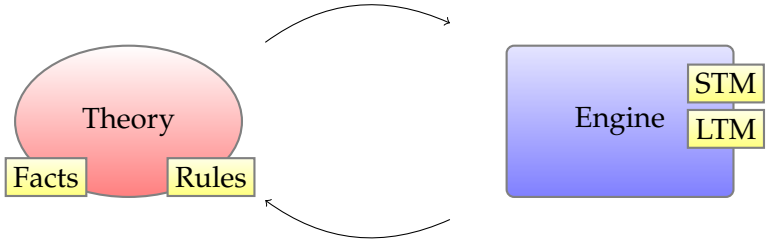

Figure 7: Separation between Knowledge and Inference

notice that the provided definition of FOL is *perfect*. The interpretation of a formula is univocal since every term identifies one and only one entity, and a formula is either true or false, so there are neither degrees of uncertainty, nor of partial truth. Hence, no HC application supports imperfection *natively* - even if imperfection-aware reasoning can be built to reason *over* it, but not *with* it.

3.1.2 *Rule-Based Systems*

In a wide sense, "Rule-Based System" (RBS) is a label which can be applied to many logic-inspired artificial systems. However, it is often used to denote those system which should be more properly called *Production Rule Systems* (PRS), to the point that the two are used interchangeably.

Rules in PRS

A PRS considers an implication $\varphi \rightarrow \gamma$ from a different point of view. Assuming modus ponens is a valid inference rule, the knowledge of a formula φ' matching with φ allows to conclude γ', a formula which depends on γ and the unification of φ' and φ. This act can be encoded by a (production) rule, a syntactic construct which can be used to guide the reasoning process of an appropriate engine. A rule is usually written in any one of the forms "`if` φ `then` γ", "`when` φ `then` γ" or, in a compact way,

$$\varphi \Rightarrow \gamma$$

notations which stress the role of φ as *premise* and γ as conclusion.

Chaining in PRS

PRSs are usually implemented using *forward-chaining* engines: at the opposite of logic programming, inference is data-driven and reactive. This is not always true, since PRS are compatible with backward chaining - or, better, with hybrid chaining systems where forward and backward chaining are both used. Using **BC!**, rules are analysed and only the ones capable of producing the current goal are considered. Using **FC!**, instead, a new fact φ' is matched with the premise of all the available rules: if the match is successful, that rule becomes *active* and is put into an *agenda*. A *conflict-resolution* strategy is then applied to choose one of the active rules and *fire* it, applying the consequences defined by the rules itself. The consequences may include the generation of

Conflict Resolution

new facts which could activate other rules: they are likewise put in the agenda to be fired subsequently. Common conflict-resolution strategies rely on factors such as *refractoriness*, *recency*, *specificity* and *salience* [158].

Consequences and Side Effects

Production Rule engines do not impose strict constraints on the form of the rules, so they usually support full first order logic, including equality and function interpretation. Actually, in PRS often the consequence is not even a logical formula, but a sequence of generic actions, which may non necessarily include the generation of new facts. In this case, the role of the logical implication in a rule is diminished if not forgotten at all: in "pure" PRS the act of writing a rule itself is equivalent to stating the truth of the underlying, universally quantified implication. In "pragmatic" PRS, instead, rules may have a more general semantics, to the point that several types of rules have been identified (see [117] and Section 8.1.3).

The possibility of introducing non-logical side effects among the consequence of a rule may spoil the declarative nature of the rule-based approach, but is nevertheless an advantage in the realization of real-world applications, which may require to implement concrete actions as well as theoretical reasoning.

The RETE Algorithm

There are two possible approaches for the implementation of a rule engine: it can be designed to be a rule language interpreter or a compiler. The popular RETE algorithm [134] and its improved versions [104] use the latter method to optimize the rule evaluation process in presence of large theories where many rules share common parts in the premises. To do so, a theory is compiled into a network - a directed, acyclic graph - whose nodes evaluate predicates (also called *constraints*) and cache information locally: each node evaluates only one predicate, but nodes can be shared between rules. The network topology and structure itself holds the long-term knowledge of the system; the facts, instead, are *insert*ed in the engine and stored in the *working memory*, a term which defines the node memories as a whole. For this reason, facts in this context are also called *working memory elements* (WME).

Working Memory

The formulas RETE can process are conjunctive normal forms of binary predicates, but modern RETE-based systems use object-oriented versions of the algorithm where WME are actually POJOs, so a premise has a form similar to:

$$
\begin{aligned}
&== (\text{X.class}, \text{Type}_1) \wedge \cdots \wedge p_j(\text{X.field}_j, c) \wedge \ldots \\
&\wedge \\
&== (\text{Y.class}, \text{Type}_2) \wedge \cdots \wedge q_k(\text{Y.field}_k, \text{X.field}_z) \wedge \ldots \\
&\wedge \ldots
\end{aligned}
$$

Patterns

Simple constraints expressed on the same type of objects define a *pattern*: the first is always a type check, followed by constraints on the

individual fields. So, for example, ⟨`Persons with age` $>$ 18⟩ and ⟨`Cars with color = red and price`$> 10^5$⟩ are patterns matched by adult people and expensive red cars. Each of these binary predicates, which involve one of the fields of an object and either a literal or another field of the object, is evaluated by an α-node: all the α-nodes defining the same pattern, in turn, are linked together in a chain ending in a local α-memory which stores all the objects matching that pattern. When a rule involves more than one pattern, *join* nodes perform all the possible combinations, creating lists of objects - called *tuples* - and filtering them through β-nodes. The β-nodes evaluate constraints which involve two objects, typically because the value of the field of one is compared to the value of a field of the other. The accepted, partially formed tuples are stored in β-memories: the join nodes, in fact, assemble the tuples by adding an object taken from their "right" α-memory to a tuple extracted from their "left" β-memory. Eventually, a full tuple which matching all the constraints in the premise of a rule reaches a terminal node, where an activation is generated and sent to the agenda to be scheduled for firing. To make things clearer, the network resulting from the application of the algorithm to the abstract formula above is drawn in Figure 8. The structure, which is affine to a SELECT-JOIN sequence of operations in a relational database, determines the efficiency of the algorithm: in fact, when a new object reaches an α-memory, it can be joined with all possible partial tuples without having to re-evaluate them from the beginning (resp. whenever a partial tuple reaches a β-memory). The main drawback of this approach is the increased memory occupation, which increases exponentially with the maximum number of patterns in a rule.

α *Network : Select*

β *Network : Join*

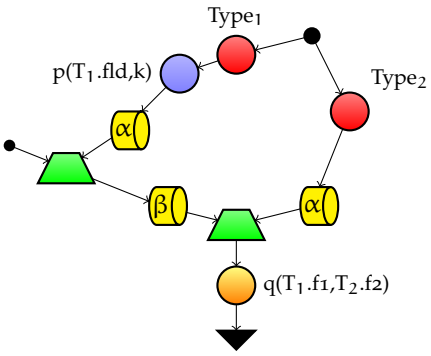

Figure 8: RETE Example

The properties of PRS and the RETE algorithm will be analysed and discussed in much greater detail in Chapter 7.

3.1.3 *Case-Based Reasoning*

Case Based Reasoning (CBR) is a problem-solving approach based on a simple consideration: certain problems are recurrent, so the solution

for a problem should be remembered and, should the same problem be faced again, recalled and used, instead of rebuilding it again from the beginning, according to the conceptual schema shown in Figure 9. A description of CBR and a review of its main applications can be found in [95].

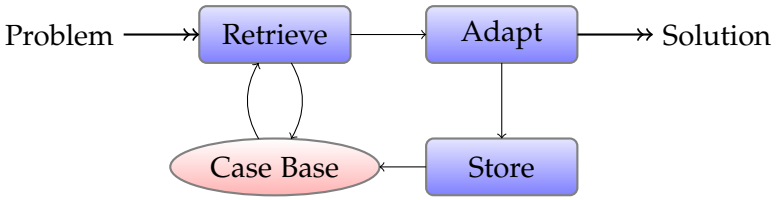

Figure 9: Case-Based Reasoning

3.2 SOFT COMPUTING

"Soft Computing" (SC) is a general term that encompasses a wide range of AI techniques, from neural networks (NN) to fuzzy logic (FS), from probabilistic reasoning to genetic algorithms, and more. According to [311], such techniques are more tolerant and robust than their hard computing counterparts in presence of any of the possible kinds of imperfection (see Chapter 2) which may appear in the information they have to process. In fact, SC systems encode data in a quantitative, non symbolic way which is also closer and more suitable to represent uncertainty and vagueness.

3.2.1 *Neural Networks*

Artificial Neural Networks (NN) have been created with the goal to emulate the behaviour of the human brain in terms of memory and reasoning. The brain is assumed to be composed of billions of elementary cells, the neurons, connected by links which propagate information using electrical pulses.

Artificial Neurons

Using a simplified version of this model, the artificial neuron shown in Figure 10 was created by McCulloch and Pitts in 1943 [199]. The "*dendrites*", each one associated to a parameter $w_{j:1..N}$, propagate a weighted version of the N inputs, $x_{j:1..N}$, to the neuron, where they are combined by a *transfer* function. This function usually performs a simple sum of the weighed inputs, i.e. $z = \sum_{j=0}^{N} w_j \cdot x_j$, adding the contribution of a $(N+1)$-th input, called *bias*. The output y of the neuron, then, is given by an *activation* function, σ, such that:

$$y = \sigma(z) = \sigma\left(\sum_{j=1}^{N} w_j \cdot x_j + w_0\right)$$

In natural neurons, the output is propagated along the "*axon*", which, through connections called "*synapses*[3]", are linked to the dendrites of other neurons. The original activation function σ is a step function with a threshold controlled by the bias (i.e. $y = 1 \leftrightarrow z > 0, 0$ otherwise), so neurons are said to *fire* when they are sufficiently *excited* by the input stimuli.

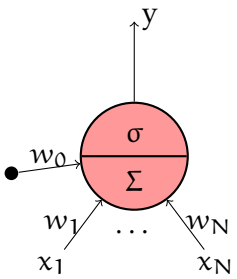

Figure 10: Generic Artificial Neuron

Artificial Neural Networks

Given its elementary structure, a single neuron, either artificial or natural, has very limited computational capabilities. To perform complex (and meaningful) tasks, a neural network requires a large number of interconnected neurons working in parallel. While the capabilities of natural neural networks are far from being well understood, it is known that an artificial neural network with an appropriate size and topology can approximate an arbitrary, non-linear function $f : \mathfrak{R}^N \mapsto \mathfrak{R}^M$, provided that the weights, globally denoted by $\mathbf{w}$, are set accordingly. In fact, N dedicated input neurons are used to sample the inputs $\mathbf{x} \in \mathfrak{R}^N$ from the environment. The input is then processed by the other neurons in the network to return the output $\mathbf{y} \in \mathfrak{R}^M$. A relevant advantage of NNs is that the optimal values of the parameters $\mathbf{w}$ can be learned from the data themselves without having an explicit knowledge of the underlying process, i.e. the function f to be approximated.

NN Applications

This makes NNs a convenient tool for problems such as regression, classification, recognition, control and prediction. However, the construction and training of a neural network is a delicate process since the designer has many degrees of freedom, which range from the choice of the form to encode input and output with, the type of activation functions (often linear and sigmoidal functions), the number of neurons and their interconnection topology and the training procedure.

NN Design

Several types of networks with different architectures and configurations exist in literature, often specialized and optimized for some particular task. This section will focus in greater detail on the ones that have been applied and whose results will be found in Chapters 8 and 10; a very detailed description can instead be found in [149].

3 for this reason, the weights are sometimes called *synaptic* weights

Feed-Forward Networks

In Feed-forward Net works (FF-NN), neurons are arranged in three or more layers: each layer (except the last one) is fully connected to next one. FF-NNs are universal function approximators, so they can be used to model any generic function $f : \mathfrak{R}^N \mapsto \mathfrak{R}^M$. In most cases, three layers are sufficient: the N input neurons in the first layer are connected to the H neurons in the hidden layer. These, in turn, are connected to M output neurons, which return the aggregate values:

$$y_m(\mathbf{x}) = g\left(\sum_{h=1}^{H} w_{m,h} \cdot \sigma\left(\sum_{j=1}^{N} w_{h,j} \cdot x_j + w_{h,0}\right) + w_{m,0}\right) \quad (3.2)$$

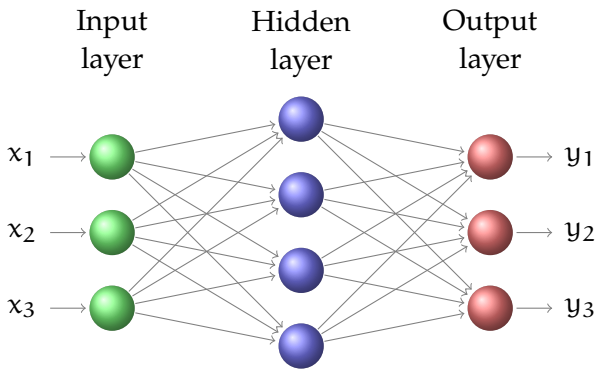

Figure 11: Simple 3-4-3 Feed-Forward Network

σ is usually a sigmoidal activation function, while g is typically linear or sigmoidal: the former is preferred if the network is used to solve a regression problem, while the latter is used for classification-oriented ones. Like the size of the layers, the functions are fixed at design time; the model parameters - the weights $\mathbf{w}$ - are instead adapted to minimize the training error, which is usually the mean square error (MSE) computed on the training set $T = \langle \mathbf{x_i}, \mathbf{t_i} \rangle_{i:1..|T|}$:

$$\varepsilon(T) = \frac{1}{|T|} \sum_{i=1}^{|T|} \|\mathbf{y}(\mathbf{x_i}) - \mathbf{t_i}\|^2 \quad (3.3)$$

The goal of the training procedure is to find the optimal a-posteriori weights $\mathbf{w}_{opt}$ minimizing the error $\varepsilon(T)$. In particular, provided that enough data are available, the training set can be partitioned into three subsets: the *training* set proper, the *validation* set and the *test* set. The error expression computed on the first set is used, differentiated, by many training algorithms, such as back-propagation or conjugate gradient descent [59], which are iterated until the MSE on the validation set falls below a chosen threshold or remains stable. Eventually, the MSE on the test set is assumed to be a performance indicator of the trained network.

Bayesian FF-NN

The training of FF-NNs, however, has also been analysed from a Bayesian point of view [189]. The data T can be assumed to be samples from an unknown function f with additive Gaussian noise $\gamma = \nu(0, \beta)$, so the probability $p(y|\mathbf{x}, \mathbf{w})$ is Gaussian with mean $y(\mathbf{x})$. The likelihood of the training data $p(T|\mathbf{w})$ is then a product of Gaussian functions: if the prior distribution of the weights $p(\mathbf{w})$ is taken to be Gaussian with mean zero and variance α, the posterior $p(\mathbf{w}|T)$ becomes Gaussian as well. It can be shown (see for example [57]) that the maximum likelihood weights $\mathbf{w}_{opt}$ minimize the error expression 3.4.

$$\sum_{i=1}^{|T|} (y(\mathbf{x}_i, \mathbf{w}) - \mathbf{t}_i)^2 + \frac{\alpha}{2} \cdot \sum_{j \in \mathbf{w}} w_j^2 \tag{3.4}$$

The weights $\mathbf{w}_{opt}$ obtained by Bayesian analysis minimize the MSE, with the addition of a component, derived from the prior on the weights, corresponding to the "regularization term"which is commonly used to smooth the output of the network [149].

ELMAN NETWORKS Feed-forward networks are stateless combinatorial machines: they can approximate arbitrary non-linear functions, but are inadequate to deal with even the simplest dynamic systems. Such systems are usually described using state-space equations: one possible model is

$$\begin{aligned} \mathbf{s}(t+1) &= f(\mathbf{x}(t), \mathbf{s}(t)) \\ \mathbf{y}(t) &= C * \mathbf{s}(t) \end{aligned}$$

Simple Recurrent Networks

The output depends on the internal state, which in turn is the result of a combination of the input and the previous value of the state itself. Elman networks can be used to approximate this class of models: their structure is similar to that of a FF-NN, but the hidden layer is fully connected to itself using a time-delay link. The output $\mathbf{s}$ of the neurons in the hidden layer is equivalent to the internal state of a dynamic system, in fact the sigmoidal functions in the hidden layer approximate its transfer function, which can be non-linear. The weights of the synapses connecting the hidden layer to the linear output neurons, instead, form the observation matrix C.

Radial Basis Functions Networks

A Radial Basis Function (RBF) Network is an *interpolation* technique which can be implemented with a neural model. Even if its purpose is less general than other types of networks, its advantages include:

RBF Properties

- **Regularity**: the resulting function is continuous (i.e. $\forall x, y, \delta : \|x - y\| < \delta \Rightarrow \exists \delta : \|f(x) - f(y)\| < \varepsilon(\delta)$) and smooth (i.e. ε/δ is not large)
- **Determinism**: the training algorithm leads to a global optimum

Such networks model an unknown function $f : \mathfrak{R}^n \mapsto \mathfrak{R}^m$, which is known only through some *distinct* samples $\langle \mathbf{x}_j, \mathbf{y}_j \rangle_{j:1..J}$. A RBF uses a combination of radial functions ϕ_j, functions with a center $\mu_j \in \mathfrak{R}^n$ such that their value in a given point depends only on its distance from the center: $\phi_j(\mathbf{x}) = \phi(\|\mathbf{x} - \mu_j\|)$. Notice that, even if the euclidean distance $\|\cdot\|$ has been used, any distance measure could be adopted.

RBF Network Architecture

The network uses three layers: an interface input layer with n neurons, a hidden layer with J neurons, one for each training pair, and a linear output layer made of m neurons. The activation functions of the hidden neurons are radial, with the inputs of the training pairs being used as centres (i.e. $\mu_j = \mathbf{x}_j$). The interpolating model $f^\star$, then, can be expressed by equation 3.5:

$$f^\star(\mathbf{x}) = \sum_{j=1}^{J} w_j \cdot \phi(\|\mathbf{x} - \mu_j\|) \tag{3.5}$$

Given the training set, J equations of type 3.5 can be written: the solution of the resulting linear system gives the optimum hidden-ouput weight vector $\mathbf{w_{opt}}$ (when $m > 1$ a linear is solved for each component).

GENERALIZED RBF The RBF is an interpolator, which means that the learned function $f^\star$ passes through all the training points: when the data are many and noisy, this level of over-fitting is not a desirable property. In such cases, generalized RBF are simpler, smoother and thus more adequate models. A generalized RBF uses $K \ll J$ functions which are not constrained to be centred on the training data. The functions are usually - but not necessarily - *localized*, i.e. they have a compact support: $\forall\epsilon\exists\rho : \|\mathbf{x} - \mu\| > \rho \Rightarrow \phi(\mathbf{x}) < \epsilon$, so the input space is effectively partitioned between the functions. This property, while not required, is useful to give an interpretation to the output of the network.

In literature, several classes of function have been used used [71], such as thin-plate splines, multiquadrics and inverse multiquadrics, but most implementations use multivariate Gaussian basic functions :

Gaussian Basis Functions

$$\phi_j(\mathbf{x}) \propto e^{-(\mathbf{x}-\mu)^T \Sigma_j^{-1} (\mathbf{x}-\mu)}$$

Such functions introduce an additional parameter, the *spread* or *scope* matrix which controls the topology of the support of each function. This is effectively changes the definition of distance in the input space, adding more parameters.

In fact, when less than J functions are used, even the position of the centres are no longer determined and must be chosen during the training procedure . Several approaches exist, but the most common are:

RBF Training

TWO-PHASE TRAINING : With this strategy, the centres are chosen first and fixed; during a second phase, the weights are optimized. The centres can be selected at random from the training data or, better, chosen using a clustering algorithm or even

a SOM (see Section 3.2.1 and [260]). When the functions have been positioned, equation 3.5 can be used to create an underdetermined linear system. It solution, which can be obtained by pseudo-inversion of the coefficient matrix, yields a weight vector which is optimal in the least squares set [149].

GRADIENT DESCENT : This supervised approach expresses the quadratic approximation error (see Eq. 3.3) as a function of the network parameters, μ,$\mathbf{w}$ and possibly Σ, then uses the gradient descent technique to optimize them all at the same time.

BAYESIAN RBF. Interpolation problems can be considered a subclass of regression problems, which are usually formalized and then solved from a Bayesian point of view. Like with feed-forward networks, where the optimization and probabilistic approaches lead to similar results, even RBF networks have a Bayesian justification [149],[57].

In a regression problem, the input data are noisy samples $\mathbf{y}_j = \mathbf{x}_j + \ni_j$ of an unknown function $f : \mathfrak{R}^n \mapsto \mathfrak{R}^m$. The solution is given by a function $f^\star$ which takes into account all the available information, returning the expected output for a given input: $f^\star(\mathbf{x}) = E[\mathbf{y}|\mathbf{x}]$. Considering one output dimension at a time, applying the definition of expectation and Bayes' theorem, one gets:

$$f^\star(\mathbf{x})^{(m)} = \frac{\int_{-\infty}^{+\infty} y^{(m)} \cdot f_{\mathbf{X},Y}(\mathbf{x}, y^{(m)}) dy}{f_{\mathbf{X}}(\mathbf{x})}$$

Kernel estimation

To be computed, $f^\star$ requires the single and joint probability density functions $f_{\mathbf{X}}$ and $f_{\mathbf{X},Y}$, which are unknown but can be estimated from the data points using kernel functions . A kernel $K(\mathbf{x})$ is a continuous, symmetric, bounded function with a single maximum in the origin that satisfies $\int_{\mathfrak{R}^n} K(\mathbf{x}) d\mathbf{x}$. Using kernels, the estimator becomes[4]:

$$f^\star(\mathbf{x})^{(m)} = \frac{\sum_{j=1}^{J} y_j^{(m)} \cdot K\left(\frac{\mathbf{x}-\mathbf{x}_j}{h}\right)}{\sum_{k=1}^{J} K\left(\frac{\mathbf{x}-\mathbf{x}_k}{h}\right)} \tag{3.6}$$

Equation 3.6 has a dual interpretation: the output y is a weighted average of the training data but, assuming that the kernels are *spherical* (i.e. $K(\mathbf{x}) = K(||\mathbf{x}||)$, the weights themselves can be considered (normalized) radial basis functions. Notice that, like for generalized RBF, less than J kernels can be used, provided that the centres are chosen appropriately.

4 The additional parameter h is a smoothing parameter that controls the scope of the kernel

Self-Organizing Maps

Unlike other networks, where neurons cooperate to memorize an input-output relation, Self-Organizing Maps (SOM) [172] are based on *competitive* learning . All the neurons are fed with the same input, which is usually a D-dimensional vector of numerical features. Through its D synaptical weights, each neuron memorizes a reference value, represented by a D-dimensional vector as well: the neuron's activation depends on the distance between the input value and the stored value. The distance, defined on the input space, can be any symmetric, non-negative function preserving the triangular inequality: often the euclidean distance is chosen, but others such as l-norms or the Mahalanobis distance [191] can be used. The activation, in particular, is inversely proportional to the distance, so a neuron is the more active the more its stored pattern is similar to the actual input.

Competitive Learning

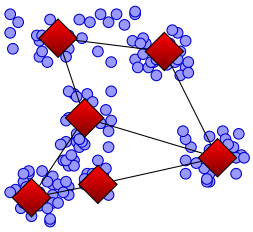

Figure 12: 2d Self-Organizing Map in a 2D space

The neuron with the highest activation is called *winner* neuron and, if the so-called *winner-takes-all* approach is adopted, it inhibits the activation of the other neurons using appropriate synaptic links. The SOM has a single main layer (plus the input layer), fully connected by these inhibitory synapses, but the network relies on another, more important structure . The neurons are placed at the nodes of a *lattice graph*, an ordered d-dimensional structure ($d < D$) which edges are modelled by additional links. Typically, $d = 1$ or $d = 2$, so the network is more properly a *chain* or a *grid*. A neuron, then, can have *spatial* neighbours, determined by the similarity between the stored patterns, and *topological* neighbours, depending on the structure fixed at creation time.

SOM Structure

A SOM , thus, can be used to cluster a set of data, selecting an appropriate set of prototypes which approximate the underlying distribution of the data by positioning on the principal curves of the data set ([99]). For this reason, SOMs are used in many contexts for data quantization and recognition. Unfortunately, like with FF-NNs, the initial choice of the number of neurons and the value of the training parameters are critical to ensure the accuracy and the regularity of the final representation. Several extensions have been proposed to overcome these limitations, including but not limited to Vi-SOM [302], Neural Gas [195] and Regularized SOM [140].

SOM Applications

Neural Networks and Imperfect Information

Given the variety of neural-inspired architectures, there are several problems that can be solved choosing an appropriate network. Like other soft computing techniques, they can deal with several types of imperfect information in a robust and coherent way: in fact, they are necessarily imperfect. First of all, the inputs are seldom certain and precise - this is usually one of the conditions which leads to choosing a neural network in place, for example, of rules or mathematical models - but usually affected by noise which can't always be cancelled; secondly, the models they learn are actually approximations rather than accurate representations.

However, when carefully designed, neural networks are graceful: the effects on the output due to the imperfection of the input are limited and can usually be quantified, provided that an analysis of the structure and the parameters of the network itself is performed. Unfortunately, this additional knowledge is sometimes discarded or ignored: while this could be justifiable when a definite, perfect answer is to be given to the client (be it another software module or a human user), the end-user should be able to access all the available information and decide what do with it.

Consider, for example, a classification problem such as character recognition: given the description of an object through a set of N features, it must be assigned to one of M categories. This is a function approximation problem - the functions to be learned are the *characteristic* functions of the M sets - so it can be solved by an adequately sized $N - H - M$ feed-forward network with sigmoidal activation functions in the output layer and trained with feature vectors paired to characteristic vectors, i.e. vectors where all components are 0 except for one, corresponding to the desired class, set to 1. The output $\mathbf{y}(\mathbf{x})$ of the network for a new input $\mathbf{x}$ will hardly be a characteristic vector, but it can be converted into one using the softmax activation function in the third layer (for example, see [196]), or simply by setting to 1 the j-th component such that $j = \arg\max_k\{\mathbf{y}[k]\}$. The output of such a network, instead, can be interpreted as the posterior probabilities of each class given the input features: while assigning the object to the class with the greatest (probability) value is a strategy optimal in a Bayesian sense (see [312], which also includes a survey of several applications), the actual value of the probability is an useful piece of information worthy propagating.

Similar situations arise when using Self-Organizing Maps: the winner neuron can be used to infer some properties of the input data, but being the closest neuron according to some distance function δ is only a qualitative information which does not guarantee that the pattern and the neuron are actually similar. To avoid such situations, a quantitative *activation* degree can be associated to the neurons, for example using a Gaussian function of the pattern-neuron distance, which better models and explains the association.

In other cases, neural networks are used as predictors: NN are not good extrapolators, but prediction tasks usually require to analyse data with different characteristics than the ones the network have been trained on. In this case, a prediction should always be accompanied by a confidence estimate in order to judge the quality of the estimate, especially if an important decision is to be made: stock market and weather forecasts are some of the *less* critical examples. Bayesian Neural Networks ([57]) do not just return a punctual value, but a probability distribution for the output which is function of the network parameters, in turn conditioned on the training set and some additional information such as noise variance. Other researchers, like [270] and [121], have instead proposed methods to estimate confidence intervals for the output.

Thus, neural networks can give additional information beyond their primary outputs: such knowledge should not be lost, but propagated, especially when a network is used as a building block of more complex systems (see Chapter 4). Chapters 9 and 10 will show some case studies where the uncertainty produced by NN is actually exploited to build more robust systems.

3.2.2 *Clustering Algorithms*

Clustering

The term "*Clustering*" can be applied to several data mining techniques [159] used to identify an internal structure within a large set $\mathbf{X}$ composed of J data $\mathbf{x}_{j:1..J}$ (also called patterns, or samples, or feature vectors, depending on the context). While they have many applications [248], their description is not relevant here and will be omitted, even if the SOM can be considered an example of this category as well as a NN.

In a nutshell, the goal of a clustering algorithm is to partition the data set in N clusters, subsets $C_{k:1..N} \subseteq X$ such that, given a similarity measure $S(\cdot,\cdot)$ between two patterns, maximizes $S(\mathbf{x}_{j1}, \mathbf{x}_{j2})$ if the two belong to the same cluster and minimizes it otherwise.

Clusters can be defined in terms *prototypes* $\mathbf{c}_K$, a pattern which features approximate the average values for the members of that cluster; in alternative rules or constraints on the values of the features can be used to define their borders (e.g. like in classification trees). Instead, it is relevant to notice that there exist several imperfect versions of this idea. Defined a numeric variable $u_{j,k} \in [0,1]$ that expresses whether $\mathbf{x}_j$ belongs to C_k:

Clustering Modalities

- **Hard Clustering**: a pattern can belong to one single cluster

$$\forall\, j \;:\; \left(\exists!\; k^\star : u_{j,k^\star} = 1 \;\wedge\; \sum_k u_{j,k} = 1 \right)$$

- **Fuzzy Clustering**: a pattern can be partially member of a cluster, meaning that it does not completely match the prototype of the

ideal member of the cluster, or that it lies near the (fuzzy) border between two or more clusters.

$$\forall j : \sum_k u_{j,k} = 1 \qquad u_{j,k} = \mu_{C_k}(\mathbf{x}_j)$$

- **Probabilistic Clustering**: a pattern belongs to a single cluster, but it is not possible to decide which one with certainty, so a probability is assigned to each cluster. The numerical constraint is equal to the fuzzy case, but with a different meaning.

$$\forall j : \sum_k u_{j,k} = 1 \qquad u_{j,k} = p(\mathbf{x}_j \in C_k)$$

- **Possibilistic Clustering**: the algorithm evaluates the possibility that a pattern belongs to a cluster, given the value of its features. It must be possible for it to belong to at least one cluster.

$$\forall j : \exists k : u_{j,k} > 0 \qquad u_{j,k} = \pi(C_k|\mathbf{x}_j)$$

3.2.3 *Bayesian Networks*

Bayesian Networks (BN) [237] are probabilistic graphical models which can be used to encode the relations of mutual (in)dependence between an ordered set of random variables $\mathbf{X}_{j:1..N}$. In general, the joint distribution $p(\mathbf{X})$ depends on the conditional distributions:

$$p(\mathbf{X}) = \prod_{j=1}^{N} p(X_j|X_{k:1..(j-1)}) \tag{3.7}$$

BNs, instead, exploit the notion of *statistical independence*: two variables X_i and X_j are independent if and only if $p(X_i|X_j) = p(X_i)$, meaning that knowledge on the actual state of X_j does not alter the available knowledge on X_i. Moreover, the related notion of *conditional independence* can be defined and used: two variables X_i and X_j are conditionally independent given a third variable X_k if and only if $p(X_i|X_j, X_k) = p(X_i|X_k)$.

Construction of Bayesian Networks

In concrete applications, then, Equation 3.7 usually takes a simpler form, since not all possible mutual influences are relevant and need to be considered. Unfortunately, deciding which variable depends on which is a problem that depends on the context being modelled, but once the dependencies have been established (usually with the help of an human expert or learned from data [150]), it is possible to build a directed acyclic graph $G = \langle E, V \rangle$ to encode such structure. First, an ordering on the variables is imposed such that if X_i depends on X_k, then $i > k$; then the vertex set V is built assigning each variable X_j to a node V_j; finally, for each pair of variable X_j and X_k such that X_k is not independent on X_j, an edge $e_{j,k}$ is created to connect V_j and V_k. In

the resulting graph, then, a variable depends only on its node's parent nodes' variables:

$$p(\mathbf{X}) = \prod_{j=1}^{N} p(X_j | e_{j,k} \in E) \tag{3.8}$$

Moreover, a node's parents, its children and the parents of its children form its *Markov Blanket*: a variable is conditionally independent of any other variable given the state of the variables in its Markov Blanket.

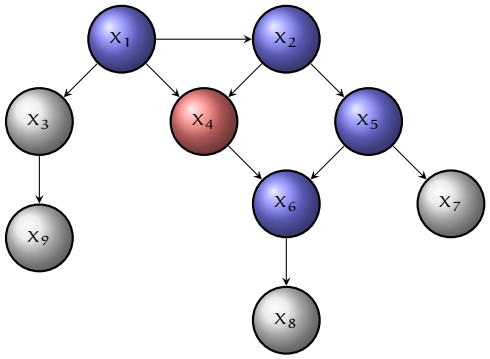

Figure 13: Bayesian Network: Markov Blanket (in blue) for X_2

Once the structure of a BN has been established, it is necessary to fix the values of its parameters. If the random variables are discrete, such as in the common case where they are actually boolean, each node V_j is associated to a conditional probability table stating the probability distribution of X_j for any possible combination of the values of tis parents. This causes the number of the entries of the tables to increase exponentially with the number of parents, so alternatives have been proposed to use more compact representations.

Specialized Bayesian Network

One possible approach is *parameter sharing*: some conditional probabilities are constrained to have the same value, thus reducing their effective number. The *noisy-OR* gate introduced in [237] assumes that the variables are binary and that the edges model not just probabilistic influences, but causal relations. Moreover, only one cause is allowed to be true at any given time, so the conditional probability table reduces to : $p(X_j = 1 | X_k = 1, X_{i|e_{i,j} \in E, i \neq k} = 0)$. When all the variables are Gaussian, uni- or multi-variate, a linear-Gaussian model can be adopted [57]: in this case, the mean and variance are given by a linear combination of the states of the parent nodes.

Applications of Bayesian Networks

Despite the difficulties in learning their structure, BN are widely used for many tasks [151] in fields such as medicine, law, image processing, finance and more. Just to cite some more specific classes, in probabilistic classification networks [136], the output categories are mapped onto nodes which depend on the input features, but the network also allows to model probabilistic influences between the inputs themselves. BN are also often used for for regression/prediction and

diagnosis (e.g. see [30],[215]), especially in the medical field, where the capability of a network propagate information in both directions makes them a convenient deductive and abductive tool.

Inference in Bayesian Networks

Inference in BN is the process of propagating the *belief* consistently in all the network's nodes: that is, to update the conditional probability distribution of each variable according to the state of the ones it is depending on. The underlying principle is stated by Bayes' theorem:

$$p(X_j \mid X_k) = \frac{p(X_k \mid X_j) \cdot p(X_j)}{p(X_k)} \tag{3.9}$$

The *posterior* probability of X_j conditioned by the knowledge on X_k can be computed multiplying the *prior* probability $p(X_j)$ by the *likelihood* of X_k given X_j and then normalizing the result. Notice that the roles of X_j and X_k can be exchanged so, even if a preferential influence direction has been established, for example by deciding that one random variable is the cause and the other is the effect, even the "subordinate" variable can modify the belief on the state of the principal one. In Bayesian Networks, this justifies the fact that information can be passed along the edges in both directions, slightly complicating the task of propagating the available knowledge.

While there exist algorithms that solve this problem, a particularly efficient solution can be found if the network has a simple structure with no loops: in fact, some general purpose algorithms first try to eliminate the loops in the network, then apply the more specific procedure [237], [57].

Message Passing

If the network has at worst the structure of a poly-tree, the propagation of information in a BN can be implemented using a message-passing model: each node stores the current probability distribution $p(X_j)$ of its variable and informs its parents and children whenever it changes. In particular, it sends a message λ_{X_j} to the former and a message π_{X_j} to the latter. Thus, the computation does not involve the whole network at a single time, but is a combination of local processing and interaction between the nodes. In order to map one variable's distribution to another, the conditional are used, denoted using the symbol $\Phi_{j,K} = p(X_j \mid X_{k \in K})$, so $p(X_j) = \Phi_{j,K} \star p(X_K)$. Φ and $\star$ are assumed to have a dual nature, so that $p(X_{k \in K}) = p(X_j) \star \Phi_{j,k}$. If the variables are discrete, π and λ can be encoded using (column) vectors, while Φ becomes a matrix such that $\Phi_{j,k}[a, b] = p(X_j = X_j[a] \mid X_k = X_k[b])$[5] and $\star$ is the standard row-by-column matrix product, but extracting $\Phi_{j,k}$ from $\Phi_{j,K}$ for a given k is not immediate (see 8.2.2).

Eventually, the propagation algorithm in poly-trees will be presented. Propagation in chains and trees is a specialization of the more general

5 $[\cdot]$ is the vector indexing notation

case, so it will be omitted. The algorithm, instead, is cited to better understand the particular implementation given in 8.2.2.

POLY-TREES PROPAGATION A node can receive messages both from parents and from children: whenever a message is received, the local belief is computed using the information in the local conditional $\Phi_{j,j-1}$ (the merge operation $\cdot$ is the standard element-by-element multiplication, followed by division for an appropriate normalization constant).

$$p(X_j) = \left(\prod_{k|e_{j,k}\in E} \lambda_{X_k} \right) \cdot \left(\Phi_{j,j-1} \star \prod_{k|e_{k,j}\in E} \pi_{X_k} \right) \tag{3.10}$$

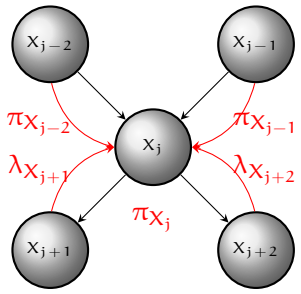

Figure 14: Propagation in polytrees

Afterwards, update messages are computed using a simple general principle: the local information $p(X_j)$ is propagated to both parents and children, after having been "discounted" of the contribution coming from the target node. Hence, the forward message sent to the children becomes:

$$\pi_{X_j}(k) = \frac{p(X_j)}{\lambda_{X_k}} = \left(\prod_{i|e_{j,i}\in E, i\neq k} \lambda_{X_i} \right) \cdot \left(\Phi_{j,j-1} \star \prod_{k|e_{k,j}\in E} \pi_{X_k} \right)$$

Likewise, the backward message for parents depends on the destination as well:

$$\lambda_{X_j}(k) = \prod_{i|e_{j,i}\in E} \lambda_{X_i} \star \left(\Phi_{j,j-1} \star \prod_{i|e_{i,j}\in E, i\neq k} \pi_{X_i} \right)$$

3.2.4 *Fuzzy (Logic) Systems*

The concept of fuzziness and the role of fuzzy sets in dealing with vagueness have already been introduced in Chapter 2. This section, instead, will focus on Fuzzy Sytems (FS), a specific soft-computing tech-

nique which exploits fuzzy sets and fuzzy logic to model complex non-linear relations in a robust and yet intuitive way. According to [216], the fuzzification of classical logic has the goal of

> *...addressing the vagueness phenomenon ..., modelling it with truth degrees taken from an ordered scale ..., preserving as many properties of classical logic as possible.*

In practice, there are two possible approaches: one, more mathematically oriented, can be called fuzzy logic "in a narrow sense" and studies the properties of graded, many valued logics from a formal point of view. The other, fuzzy logic "in a broader sense", is closer to L. Zadeh's original definition and uses a softer approach. The first will be discussed in greater detail in Section 8.1, while this section will be dedicated to the latter (even if examples of both types will be presented in Chapter 8).

Like the other Soft Computing techniques introduced in this Chapter, FS are versatile function approximators: for this reason, they can be used in applications varying from automated control to image processing and pattern recognition [47]. Fuzzy controllers alone, in particular, have become a relevant research field, because they can be applied to non-linear systems, where the well-known techniques for linear, time-invariant systems can't be used. Their wide diffusion can possibly be ascribed to their relative simplicity (even if a good FS requires a careful design, involving the tuning of several sensitive parameters) and robustness, coupled with rules which are easy to understand.

FUZZY VARIABLES AND PARTITIONS A many-input function maps a multi-variate vector $\mathbf{x}_{i:1..N}$ of some domain $\mathbf{X} = X_1 \times \cdots \times X_N$ to a unique output $\mathbf{y} \in Y_{out}$: in the specific case of real-valued functions, $\mathbf{X} \subseteq \mathfrak{R}$. Fuzzy logic systems, instead, map vague concepts, such as `tall`, `fast` and `low`, onto other vague concepts.

Linguistic Variables

In [310], Zadeh introduced the concept of *fuzzy linguistic variable*. A linguistic variable λ has a finite domain Λ composed by terms, represented using the natural language, which are suitable to qualitatively describe the value of an implicit, correlated quantitative variable x with domain X. For example, the linguistic variable `Age` could have a domain $\Lambda = \{$`young`, `mature`, `old`$\}$. A suitable hidden variable x, in this case, would be the age measured in years, so $X_{age} = [0, 100]$.

Each possible linguistic value $\lambda_j \in \Lambda$ is associated to a fuzzy set A_j. Together, the sets define a fuzzy partition of Δ:

$$\Delta \ni \forall x : \sum_j \mu_{A_j}(x) = 1$$

Fuzzification

For example, a 25-year old person could be `young` with degree 0.7 and `mature` with degree 0.3; another example of a more generic fuzzy partition is shown in Figure 15. The sets usually cover all the domain ($\forall x : \exists j^\star : A_{j^\star} > 0$), but in most practical cases a point is covered by no more than two different sets.

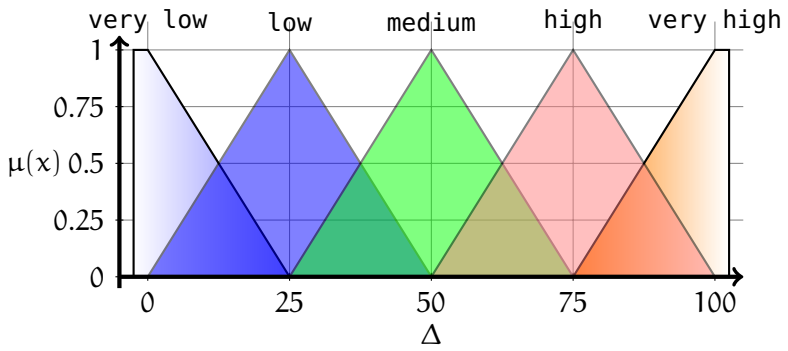

Figure 15: Fuzzy Partition Example

When a domain $\mathcal{U}$ is multi-variate, linguistic variables, each one with its domain and fuzzy partition, can be combined to describe a vector $\mathbf{x}$ using a conjunctive sentence such as "x_1 is $A_j^{(1)}$ and ... and x_N is $A_j^{(N)}$" or, in a more compact form, $\bigwedge_i A_j^{(i)}(x_i)$.

(De)fuzzification

Such a sentence has a dual interpretation [307]: the first is the evaluation of the compatibility of a vector $\mathbf{x}$ with the composite fuzzy set given by the Cartesian product of the values of the linguistic variables. This operation, known as fuzzification, returns a fuzzy membership degree in $[0, 1]$.

The second yields a *possibility* distribution π over $\mathbf{X}$, conditioned by the knowledge that the coordinates of a vector $\mathbf{x}$ can't be identified with certainty, but belong to a given fuzzy set in some degree ε. The possibility degree of each value $\mathbf{x}$ is equal to its membership in the set, reshaped using ε: if a precise value is to be chosen, a defuzzification operation g has to be applied. Common choices for g include computing the center of gravity of the distribution, its center of mass, or its mean: a comprehensive discussion can be found in [179] and [261].

Fuzzification	Defuzzification
$\varepsilon = \mu_{\prod_i A_j^{(i)}}(\mathbf{x})$	$\mathbf{x} = g(\pi(\mathbf{x}\|\varepsilon_{\prod_i A_j^{(i)}}))$

FUZZY RULES A function $f : \mathfrak{R}^N \mapsto \mathfrak{R}$ can be approximated by first fuzzifying its domain and its range, then matching the resulting input and output sets using rule constructs. The concept of "fuzzy rule" has many interpretations [117], but most implementations of fuzzy controllers and similar systems use Mamdani's inference schema, which is the one recalled here (see also [106]). Mamdani's approach uses conjunctive rules, which test the compatibility of some numeric input x_i with the value $A_j^{(i)}$ of a linguistic variable. The individual checks are joined to form the premise of a rule using the logical connective `and` ($\wedge$):

Mamdani Fuzzy Systems

$$r : \bigwedge_i (x_i \texttt{ is } A_j^{(i)}) \Rightarrow Y \texttt{ is } B$$

Taking $P = \prod_i A_j^{(i)}$, the rules can be written in the more compact form: $P(\mathbf{X}) \Rightarrow B(Y)$. A rule has different meanings: first of all, it gives a qualitative description of the local behaviour of the function f. Then, any vector $\mathbf{x}_P$ with full membership in P has the possibility of being mapped by f (which is unknown) onto any range value y with degree $B(y)$. Finally, values in the neighbourhood of, and thus similar to, $\mathbf{x}_P$ will be mapped onto values similar to $y_P = f(\mathbf{x}_P)$.

Premise

Assuming that $k_{i:1..N}$ is the cardinality of Δ_i, the fuzzy partition of the i^{th} coordinate, up to $\prod_{i:1..N} k_i$ rules can written. An input vector $\mathbf{x}$ is fed in parallel to all the rules: each one of its coordinates is fuzzified using the various fuzzy sets associated to the linguistic values and the resulting membership degrees are composed using an operator which usually (but not necessarily) returns the minimum of the degrees, so that $\varepsilon_P = \min_{i:1..N} \mu_i$. It is implicitly assumed that the different components are *non-interactive* ([310]), which is a reasonable assumption in many practical cases. After that, generalized modus ponens is applied, composing the premise with the implication to obtain the conclusion:

$$\frac{P^\star(\mathbf{x}), P(\mathbf{X}) \rightarrow B(Y)}{B^\star(Y)} \tag{3.11}$$

Notice that the inputs may not only be the quantitative values $\mathbf{x}$ (e.g. age = 25), but also their qualitative, linguistic counterpart $P^\star$ (e.g. age = `young`), so, in order to apply the inference rule 3.11 one has to find the generalized degree of compatibility between the input and the premise of a rule. In the first case, it is the already cited evaluation of the membership function, in the latter the two fuzzy sets have to be intersected, so that the resulting degree is the maximum of the intersection of their possibility distributions. So, for each rule r:

$$\varepsilon_r = \pi_r(\mathbf{x}) = \sup_{\mathbf{x}} \min_i \left[\min_{x_i} \{ A_j^{\star(i)}(x_i), A_j^{(i)}(x_i) \} \right]$$

Implication

After the premise degree ε_r has been computed, it is combined with a degree $\varepsilon_\rightarrow$ associated to the implication itself. This value is taken to be $B_r(Y)$ (the fuzzy set appearing on the right side of rule r), so that, applying modus ponens, one gets the contribution to the possibility distribution of Y given by rule r:

$$\pi_r(Y) = \min\{\varepsilon_r, B(Y)\}$$

Conclusion

The rules, then, return a degree for each linguistic value B_j associated to the output range: if more than one rule entails the same value, the degrees are combined using an `or`-like operator. The final degrees are used to cut the fuzzy sets, then the distributions are joined so that:

$$B^\star(Y) = \max_r \{\pi_r(Y)\}$$

Fuzzy Additive Systems

The distribution $\pi(Y)$ can be used as an input to other chained rules, or defuzzified to obtain a precise value $y(\mathbf{x})$. However, there exists a specific class of fuzzy systems, called Fuzzy Additive Systems, which

bypasses this step by allowing quantitative conclusions in the rules, i.e. $P(\mathbf{x}) \Rightarrow (Y = b_r)$. In this case, $B^\star(Y)$ is computed by taking a linear combination of the range values b_r, weighted using the rules' activation degrees ε_r. A function g, usually a sigmoidal one, can optionally be used to normalize the output.

3.3 CONCLUSIONS

The term AI is a label which can be applied to a vast class of algorithms, each one with different properties. Even though most of them can ultimately be interpreted in terms of function approximation or search in a state space, the specific properties of each one make them more appropriate for a class of problems rather than another. Thus, the knowledge of the benefits and drawbacks of each tool is essential when choosing which one to use for a given problem.

		Output		
		Vague	**Perfect**	**Uncertain**
Input	**Vague**	FS NN		
	Perfect	NN CL	RBS CL CBR	NN CL
	Uncertain			BN

Table 2: AI techniques and Imperfection management

Rule-Based Systems are better used when perfection is a requirement, while Bayesian Networks and Fuzzy Systems are better options when the data are affected by uncertainty or vagueness, respectively. While they are symbolic or, at least, can have a symbolic component, Neural Networks are essentially sub-symbolic techniques with a higher degree of flexibility and can be used for various purposes with various types of information. It must be credited, however, that the term "Neural Network" actually applies to a wide class of systems. Nevertheless, the tools presented in this Chapter remain focused and tailored for specific tasks. In the next Chapter, it will shown that more complex problems can be solved more effectively and efficiently by combining two or more techniques.

4

HYBRID TECHNIQUES

Contents

The algorithms presented in Chapter 3, taken individually, have been successfully applied in several contexts, but also suffer from some drawbacks. The fact itself that so many techniques exist and are widely used proves that an approach can be more efficient and/or effective than others for a given problem.

Nevertheless, in many cases the combination of two or more methods can further improve the performance of a solution: the idea is obviously to overcome the limitations of the single modules while exploiting the advantages as much as possible. Such systems, called *Hybrid*, are more complex than standard ones and require a careful design, but usually achieve better performances. The literature is full of examples: various examples can be found in [211] and [272], and others, specifically applied to bio-chemical processes, will be discussed in Chapter 5.

Hybrid systems

This Chapter, instead, is divided in two parts. The first is focused on analysing the desirable properties of an "intelligent" algorithm, discussing whether and how the techniques previously introduced possess them, while the second part is dedicated to the characteristics of an hybrid architecture. Such criteria will then be used in Chapter 8 to analyse the applications that can be built using the extended rule-based engine, itself presented in Chapter 7.

4.1 FEATURES OF PURE AI TOOLS

In literature there are few standard criteria to analyse and compare the various AI algorithms and even fewer objective ways to evaluate them.

Comparisons, then, are often based on experience and open to much debate.

4.1.1 *Relevant Properties*

A comprehensive set of features is discussed in [211], where *Adaptation*, *Discovery*, *Explanation*, *Flexibility* and *Learning* are chosen. Additionally, *Gracefulness* will be considered.

Learning

Learning measures the effectiveness of an algorithm in extracting a desired, generalized relation from a set of training data. In many cases an intelligent system can't be instructed (i.e. programmed) directly with the knowledge necessary to perform a required task, either because it is not available to the programmer or because some relevant parameters have to be tuned case by case. Some systems, however, can be trained to improve their performance, using the raw data themselves as input. Learning can be *supervised* or not, depending on whether a specific target output is given for every input during the training phase; moreover, the procedure can be *reinforced* by the presence of an external entity which evaluates the output of the system and conditions the training accordingly. Typical problems that involve learning include, and are not limited to, pattern matching and classification [119], regression and model approximation.

The main problem of learning is the validation of the acquired knowledge, especially if a completely unsupervised procedure is used: the learned model, in fact, is not guaranteed to be correct, if meaningful at all, and must be tested appropriately. (The completeness, instead, usually depends on how much the training data cover the possible input space: notice that the required amount of data grows exponentially with the size of the input space, a phenomenon known as Curse of Dimensionality [52]).

Discovery

A system is effective at discovery if it is capable of extracting sound models from the inputs it is trained on. The learning of an undesired model can be due to several reasons: the training samples may be insufficient, or noisy, or may be biased in some way and not be representative of the average inputs. Sometimes, moreover, the training may be eccessively focused on the training data (*overfitting*): in any case, the performance of the system is likely to be poor since the internal representation is not correct.

Flexibility

An intelligent system, instead, is flexible if it has discovered a model capable of returning acceptable, if not optimal, results even in presence of inputs that are somewhat different from the ones it has been trained on. The data may be novel, but often the difference is due to some imperfect, imprecise or incomplete values, as discussed in Chapter 2. In order to be robust, a flexible system requires good generalization capabilities.

Adaptation

An adaptive system, instead, is even capable of revising its own internal knowledge as the input data it processes change significantly in time (e.g. the process it monitors is nonstationary). Such systems

usually perform some kind of continual training, even during their normal functioning.

Explanation

The correctness of the output itself, however, is not always the only requirement: efficiency apart, some intelligent system are also self-explanatory. Such systems provide not only a result, but also some kind of justification on how that result was obtained. This feature usually increases the complexity, but provides a significant additional amount of information. Moreover, explanation is a convenient way to ensure that the learning process has led to a meaningful model, since it does not require the analysis of the internal structure of the tool.

Gracefulness

Eventually, one can consider the level of gracefulness of a system. A sound system is expected to return a correct result, at least for inputs similar to the training data; a flexible one could return a correct result in presence of totally novel inputs; but no system can return a sensible result when it processes totally corrupt or meaningless data. A graceful system, however, returns and possibly explains a result whose degree of uncertainty or vagueness is function of the imperfection of the input. This ensures that the quality of the answers does not drop abruptly to unacceptable levels, but degrades in a predictable way.

4.1.2 *A Comparison of some Algorithms*

The features described in subsection 4.1.1 can be used to analyse the AI techniques discussed in Chapter 3.

RULE-BASED SYSTEMS are built using rules, which encode knowledge in a human-readable way, so their results can be easily understood and explained to a human user: even MYCIN [70], the first rule-based expert system, included a basic explanation module which showed the chain of derivations used to entail a given conclusion. This basic approach has successively evolved into more complex explanation systems, which do not limit to paraphrase the rules involved in the inference, but provide additional detail, including corollary information, and allow interaction with the user [49]. More recently, explanation sub-systems have started to be enriched with argumentation systems [208], which allow to take into account both supporting and contrasting elements for a given statement [55]. RBSs excel at explanation, but on the other hand, they are quite weak from the other points of view. While rules can be learned from data, for example using ILP! [], and measures such as coverage and support [] can be used to estimate the quality of the discovered relations, they are static logic constructs, so they are neither adaptive nor graceful. Rules can gain some flexibility by relaxing the number and type of conditions, usually at the expense of some specificity, but eventually a system has to decide whether to apply a rule or not. If comparisons and/or thresholds are involved, even a minimal variation in the input data may cause the system to behave in totally different ways.

CASE-BASED REASONING involves dedicated modules for case matching, retrieval, adaptation and storage: in order to analyse one such system, one should analyse the individual models. CBR systems are, by definition, adaptive and explanatory, since new situations are dealt with tuning the strategies adopted for past ones, which can be used to justify the responses. A case base, in fact, is not learned from large amounts of data, but programmed explicitly and then expanded integrating new cases constructed at run-time. A CBR tool, then, has basic learning capabilities (at least with respect to other tools): the main problem of the storage module is that it could focus on *possible* rather than *probable* or *useful* cases. This seriously impairs the system's discovery ability: remedies include ranking to retrieve more relevant cases first and deletion strategies to remove unnecessary cases [278]. The degrees of flexibility and gracefulness, instead, depend on the matching and retrieval strategies, which use different indexing techniques (e.g. see [27]) to find the appropriate cases, even in presence of partial matches.

NEURAL NETWORKS are possibly the most effective learning tool: even a network with a basic topology, such as the 3-layer feed-forward network trained with back-propagation, can approximate an arbitrary model between two numeric domains. For more complex problems, several other configurations exist, such as time-delay networks, networks with feedback, completely connected networks, ... Provided they are dimensioned and trained appropriately (not always an easy task [206]) FF-NN are excellent interpolators: their flexibility is limited only by their poor performance at extrapolation. Likewise, techniques such as regularization can be used to control the smoothness of the learned approximating function, which implies a high level of gracefulness. Adaptiveness, instead, can be achieved using a sequential training mode, allowing new data to be learned as they appear. FF-NNs, however, are "black-box" models: in general, such networks have almost no explanation capabilities, so there is no way to justify a result. Even worse, it is impossible to ensure that the learned model corresponds to the correct, or at least to a meaningful one. In fact, the performance of a network is typically evaluated on a subset of the same raw data from which the training set has been extracted.

RADIAL BASIS FUNCTIONS are another type of universal function approximator. The particular type of kernel functions and the fixed topology always[1] allows to find the best set of parameters that solves the training problem. RBFs, thus, are graceful and flexible interpolators (and poor extrapolators). With respect to other, more general NNs, however, they can learn only a limited class of models: for example, they can't model stateful systems since

1 assuming the training data are linearly independent

feedback is not provided for. Likewise, the batch training modality limits adaptiveness since the whole the training set has to be used every time the parameters require some tuning. Given the localized nature of the individual basis functions, however, it is usually possible to assign the *responsibility* of an output to a limited number of kernels, so a very limited form of explanation can be provided to the user.

FUZZY SYSTEMS have a dual nature of qualitative rule-based systems and quantitative function approximators. They are built using rules, so they are usually authored manually rather than learned, but this also makes them explainable. Unlike crisp RBSs, the constraints in the head and the body are vague: this is actually a great benefit since it increases the flexibility and the gracefulness to a level that can't be achieved using a comparable number of standard rules (see for example [176]). Fuzzy systems, in fact, are very popular because they combine ease of use with robustness. However, they have to be designed by hand, a process more complex than simply writing the rules: in fact, the domains have to be partitioned choosing an adequate number of fuzzy sets and the shape of their membership functions. These parameters determine the approximating model univocally and have to be modified manually every time the model requires some change.

BAYESIAN NETWORKS are strongly focused on probabilistic relations between entities, but are nevertheless one of the most balanced tools available. There exist algorithms to learn the structure and the parameters of a network from data [210], which make BNs attractive from the point of view of learning and discovery. The interchangeable role of inputs and outputs, together with the correctness of the probabilistic inference ensured by the propagation algorithm, makes them flexible and graceful as well. Both the structure and the conditional probabilities can be exploited to give qualitative and quantitative explanations for a conclusion, showing a which entities influence it and how much (a review of the explanation techniques can be found in [177]). The main drawback of a BN is its scarce adaptivity, since once a structure has been fixed, it is hardly updated.

CLUSTERING AND CLASSIFICATION ALGORITHMS are actually a vast family of algorithms, with different properties. A clustering algorithm is usually designed to identify, isolate and model classes of homogeneous individuals within larger sets of raw data; a classification algorithm, instead, tries to assign an element to a set according to the value of some features. Hence, they are learning-oriented algorithms with strong discovery capabilities. Other properties are more difficult to evaluate in general, since they depend strongly on the specific algorithm. For example, gracefulness and flexibility are very low in the standard k-means algorithm [190], but are much higher when a probabilistic algo-

rithm such as the EM is adopted [57]. Likewise, the degree of adaptivity depends on the possibility to perform an incremental learning [159]. Being completely data-driven, however, these techniques are usually unable to explain why the data have been partitioned or assigned in a certain way, except for providing the value of one or more quantitative indicators that have been optimized by the algorithm itself.

	Learn	Discover	Adapt	Flexible	Explain	Graceful
CBR	⋆⋆	⋆	⋆⋆⋆	⋆⋆	⋆⋆⋆	⋆⋆
RBS	⋆⋆	⋆⋆	⋆	⋆⋆⋆	⋆⋆⋆⋆	⋆
NN	⋆⋆⋆⋆	⋆⋆	⋆⋆⋆	⋆⋆⋆	⋆	⋆⋆⋆
RBF	⋆⋆⋆	⋆⋆⋆	⋆⋆	⋆⋆⋆	⋆⋆	⋆⋆⋆
FS	⋆	⋆	⋆⋆	⋆⋆⋆	⋆⋆⋆	⋆⋆⋆⋆
BN	⋆⋆⋆	⋆⋆⋆	⋆	⋆⋆⋆	⋆⋆⋆	⋆⋆⋆⋆
CL	⋆⋆⋆	⋆⋆⋆⋆			⋆	

Table 3: Comparison of individual AI techniques

Being a generalization, some of the evaluations could be argued on, even if they agree substantially with the ones given in [211], but it is undeniable that no system alone possesses all the desirable properties. This fact justifies the existence of hybrid systems, where different techniques are integrated to exploit the benefits of all the components, while trying to cancel or mitigate the drawbacks.

4.2 HYBRID SYSTEMS

To be considered "hybrid", a system needs only to be built using two or more different AI techniques. So, there are many possible combinations, depending no only on the number and type, but also on the interconnection strategies. The goal of this section, then, is to introduce some classification criteria useful to describe the characteristics of an HS. After that, a brief overview of the most common architectures will be given.

4.2.1 *Properties of Hybrid Systems*

Classification by technique

The simplest classification divides HS in *homogeneous* and *heterogeneous*: the former are built using modules of the same kind (e.g. two or more NN), while the latter use different techniques.

Classification by coupling

Another criteria ([200]) is based on the degree of coupling between the different parts. *Loosely coupled* systems are formed by separate modules, connected sequentially or in parallel, and communication, if any is required, is unidirectional. In *tightly coupled* systems, the individual modules share data and bidirectional interactions are possible, although there is only one control flow. In *fully integrated* systems, in-

stead, the modules share data and knowledge, at the point that the, from an external point of view, it becomes difficult to define a neat boundary between them.

Classification by design

Goonatilake and Khebbal ([139]) divide the systems in *function-replacing*, which implement one sub-part or module of a system with a different, more efficient technique, and *intercommunicating*, where the problem is divided in sub-tasks, each one solved using the most convenient approach. *Polymorphic* systems, instead, are architectures capable of emulating different computational models within the same framework.

Classification by structure

In [168], Khosla and Dillon use another schema , not unrelated to the others. *Combination* systems are built from separate modules; *transformation* systems adapt and convert information from one representation to another, so that it can be processed using different techniques; in *fusion* systems the sub-parts of a module are designed using more than one approach. Finally, *associative* systems exploit all of fusion, transformation and combination.

Overall classification

A more detailed classification is given in [273]. This schema, summarized in charts like the example one in Figure 16, will be applied to describe the other hybrid systems discussed in this dissertation. The analysed categories are:

- **Fusion Grade** : the degree of interconnection between the modules

  LOW : the modules do not interact.

  MODERATE : the modules work in parallel on the same inputs and/or produce the same outputs.

  HIGH : the modules share data and mutually influence the processing operations.

  VERY HIGH : the modules share data (including knowledge) and structure.

- **Fusion Structure** : the interconnection topology

  INDEPENDENT (&) : the modules are not connected.

  PARALLEL (/) : the modules share inputs.

  CASCADE (−) : the output of a module is connected to the input of another.

  FEEDBACK (\) : a module is used to close a loop with the output of another's.

  DESIGN (=) : the parameters of a module are tuned using another.

  AUGMENTATION (+) : a module works in *parallel* with a hybrid *cascade* of two modules.

  ASSISTED (//) : a module processes an input and the output of a *feedback* loop fed with the same input.

- **Fusion Time** : the moment in the system life-cycle the fusion is performed

  OFF-LINE : fusion is performed at development time.

ON-LINE : fusion is performed at run-time.

BOTH : fusion is performed at different moments.

- **Fusion Level** : the layer in the architecture where fusion takes place at

  PRE-PROCESSING : preliminary operations, such as filtering and validation.

  TRANSFORMATION : extraction of meaningful information that can be processed (e.g. feature extraction, mappings, etc.).

  MODEL : proper elaboration.

  POST-PROCESSING : operations to be performed on the output, usually for presentation purposes.

- **Fusion Incentive** : the motivation behind the application of the fusion

  INDEPENDENT : different techniques are required for different sub-tasks.

  CONCURRENT : a *pool* of modules solve the same sub-task in parallel.

  COMPLEMENTARY : different components are use together to improve performance.

  COOPERATIVE : different components are used for problems that would not be able to solve individually.

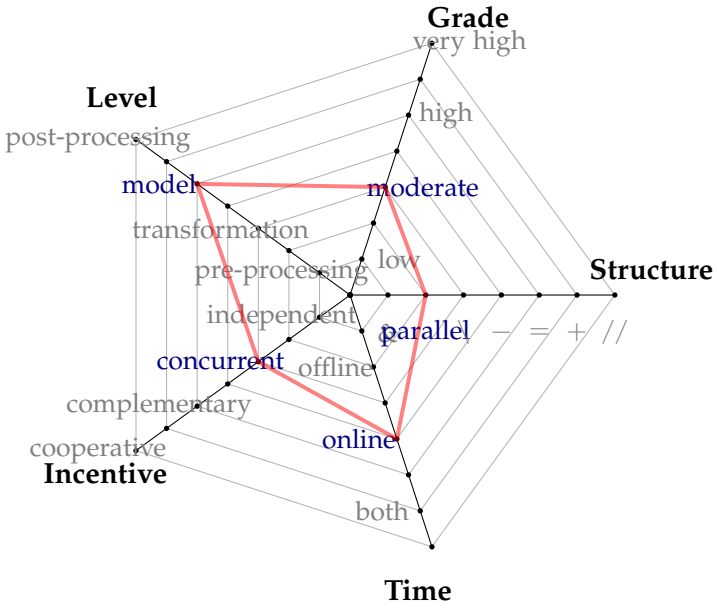

Figure 16: Classification criteria for hybrid systems

4.2.2 *Common Hybrid Architectures*

The literature is full of examples of hybrid systems, to the point that several books have been published on the topic. If one just considers loosely coupled systems, the examples would be too many to analyse, if not to cite. Hence, the discussion will focus only on relevant classes of strongly coupled systems, some of which have even been given standard names in literature, and especially on the ones that will appear in the following chapters. These systems are summarized in Table 4. The list is far from being exhaustive: in fact, it considers only systems built using two different techniques, where one has a predominant role. The primary system defines the functionality of the system, while the secondary enhances the functionalities, either by improving some core function or even adopting a polymorphic behaviour. Systems built using three or more techniques and systems build for specific applications, instead, will not treated. Notice also that this work ignores genetic algorithms, which are instead a common building block of HS. Genetic algorithms, however, are used mostly off-line, at design time for optimization/learning purposes and thus are scarcely relevant for the architectures this dissertation is centred on.

		Secondary						
		BN	**CBR**	**RBS**	**FS**	**NN**	**RBF**	**CL**
Primary	**BN**	⋆			f-BN			
	CBR		⋆	r-CBR	f-CBR			
	RBS	BLP	c-RBS	⋆	MVL	NLN		AR
	FS			FE	⋆	NFS	NFS	
	NN	(BNN)			f-NN	⋆		
	RBF						⋆	SOM
	CL			CART	f-CL	NN	SOM	⋆

Table 4: Common hybrid architectures

Hybrid CBR

Given the symbolic nature of the knowledge they encode, RBS and CBR have been used together in several systems. Some authors have used rules to enhance the case-based reasoners, if not to encode cases directly; others have used the opposite approach, using cases to improve the performance of rule-based inference (e.g. in [178] cases are used to deal with exceptions). A comprehensive overview, together with a specific classification scheme, can be found in [247]. Other CBR systems [161] have embedded sub-symbolic systems such as NNs and other clustering algorithms in their match and retrieval modules. As already discussed, the role of similarity in case retrieval makes the border between the fuzzy and the crisp versions of CBR quite vague: fuzzy CBR, then, can be further integrated with fuzzy rules [156] .

Hybrid RBS

The integration of Rule-based systems with sub-symbolic systems is less common, despite the fact that one of the most widely used rule engine architectures, the RETE algorithm [134], is based on a dataflow

network. Actually, RBSs are used in independent, cascaded or parallel hybrid architectures, but tighter forms of coupling are less common. An interesting example can be found in [103], where a "neural" model is used to implement a symbolic engine, while RBS using associative rules can benefit from the application of clustering algorithms [180]. The capability of an engine to be used as a polymorphic hybrid system, instead, depends on its supporting non boolean logics. Fuzzy logic is quite common, but is usually implemented in dedicated, "fuzzy engines" (FE). More general purpose engines, instead, also support vagueness in the form of many-valued logic [144], possibilistic logic [115] or probabilistic logic [146]. (An engine capable of supporting *all* such types of logic will be discussed in Chapter 7). Recent developments in Bayesian logic programming [167] (BLP) have contributed to reduce the gap between RBSs and BNs.

Hybrid FS

Fuzzy systems , instead, are hardly considered rule-based systems, possibly because the rules have a fixed, basic structure: the premises are conjunctions of possibly negated propositional atoms which entail one or more simple conclusions, while chaining is seldom required. The critical point in their design, instead, is the choice of the number, type and shape of fuzzy sets. In many hybrid systems the membership functions are tuned and evaluated automatically using some other tools such as a genetic algorithm or, like in neuro-fuzzy systems (NFS), by a FF-NN or a RBF.

Hybrid NN

Fuzzy Neural Networks (**FNN!**) , instead, are proper neural networks whose outputs are interpreted as fuzzy values: in fact, the networks approximate membership functions. The component that is often replaced in NNs, instead, is the training module: again, genetic algorithms are popular, but the Bayesian neural networks described in Chapter 3 and in [189] (BNN) use a Bayesian approach for training. Notice that they should not be confused with a hybrid between a Bayesian network and a neural network.

Hybrid BN

Bayesian Networks , like other tools with learning capabilities, can be trained using genetic algorithms. BNs with support for fuzzy concepts, are an interesting tool supporting both uncertainty and vagueness, but have appeared only in recent works [225, 133]. Nevertheless, the complexity of specifying the conditional probability tables makes the BN promising candidates for *fusion* hybrid systems. The linear-Gaussian models cited in Section 3.2.3 and discussed in [57] are similar, if not equivalent, to neural models embedded in the main network as local modules to perform the numerical computations necessary to approximate the evaluation of the conditional probability. Likewise, the noisy-OR and its generalization to graded variables shown in [101] could be adapted easily to fuzzy truth degrees.

Hybrid CL

A brief analysis of hybrid clustering/classification algorithms, instead, is more complicated since CL includes many techniques with different properties. Several algorithms associate an element to a cluster or to a class with a partial degree of membership that can be interpreted as probability or similarity, depending on the algorithm as-

sumptions and semantics. Notice, however, that other general-purpose tools can be used for classification and/or clustering purposes: CART trees such as the ones learned by the C4.5 algorithm ([249],[119]) are rule-based classifiers, while FF-NN are widely used for classification, exploiting their ability to approximate a characteristic function. Another type of neural network, the Self-Organizing Map [172], is widely used for clustering problems, and can be used in combination with RBF: the SOM is useful to initialize the position of the basis centres, while the RBF quantifies the degree of similarity between an input data and each prototype neuron (including the "winner") in the SOM, which can also be considered the degree of membership in a fuzzy cluster associated to the prototype itself.

4.3 CONCLUSIONS

Hybrid Systems are powerful tools that can achieve performances superior to those of their components, to the point that in the next Chapter it will be shown that a difficult problem such as the automatic management of a complex system can hardly be solved using a single technique. The main drawback is that a hybrid system requires a careful design since the possible combinations and the associated degrees of freedom increase accordingly.

However, it seems that not all combinations are equally present in literature. The order of the rows and columns in Table 4 has been chosen to separate symbolic and connectionist methods, with fuzzy systems in the middle. The emerging block structure shows that systems of the same kind are more likely to be fused together, probably because it is easier to interface systems that represent information using similar principles. This is less true when different modules are cascaded or used in parallel, since transformation adapters can be built most of the times, but stronger forms of integration are difficult to obtain. Undoubtedly, combining the different ways to encode knowledge is a major problem. The boundary between the two families, and the further benefits that could be gained using a functional - or even polymorphic - coupling of symbolic and connectionist systems will thus be discussed in the last part of the work.

Part II

MONITORING AND CONTROL FROM AN AI PERSPECTIVE

5

AUTOMATED MANAGEMENT OF COMPLEX SYSTEMS : STATE OF THE ART

Contents

This Chapter will discuss the motivations the automatic management of a "complex" system: an entity, not necessarily artificial, composed by several parts interacting in a way that is not trivial to understand. This definition is so general that it can encompass almost any real-world system, but this analysis will be focused on a very specific type of complex systems, namely the activated sludge waste-water treatment plants (WWTPs), where polluted water is processed by microorganisms in particular chemical and physical conditions, in order to remove the undesired substances. Despite the specificity of their tasks, such plants are a relevant example of complexity, since they can be considered from the biological (due to the presence of bacteria, or *biomass*), mechanical, hydraulic and electronic point of view at the same time. Nevertheless, many of the considerations made for WWTPs will be presented in a general way, so that they could be applied to a wide class of similar contexts.

In particular, the structure of the Chapter is as follows:

- A brief analysis of the concept of *management*, which, despite its apparent obvious meaning, includes a complex variety of themes itself.
- A practical motivation of the benefits of the automatic management of WWTPs, which somehow justifies their choice as a case study.
- An overview of the applicable technologies: in particular, mathematical model-based approaches will be compared to the AI techniques introduced in the previous chapters, outlining the benefits and the drawbacks of each solution, supported by some experiences found in literature.
- A study of the existing architectures which can, or have been, used to integrate different management modules

The discussion will, as usual, not be limited to a passive survey, but to a critical analysis of the existing architectures, showing their limits but also highlighting the opportunities which can be exploited to improve them, as this work has done.

5.1 AUTOMATIC MANAGEMENT

Dynamical Systems

In Systems Theory, a *complex* system is an entity composed of interconnected parts that, as a whole, exhibit one or more properties not obvious from the properties of the individual parts. A *dynamical* systems, in turn, changes it time: at any given moment, it has an internal *state* which determines its *behaviour*. This state is the result of an evolution (or *transition*) of its past internal states, possibly influenced by *inputs* and/or *disturbances* coming from the external world. Sometimes this state may be *observed*, at least partially, and possibly *controlled*, i.e. driven towards a desired value by means of some *action*.

"Hard" Control

Controllability is obviously a desirable property, since it allows a system, artificial or natural, to be not only *predictable*, but also decidable, at worst after an initial *transitory* time during which the output of the system shifts from its "natural" (*free*) values to the constrained ones. Typical control problems can be divided in *tracking*, where a variable is forced to follow a given trajectory, and *positioning*, where a variable is forced to maintain a given value, even in presence of disturbances. The definitions are natural when the variable is the position of an object, but they generalize easily. In fact, a whole discipline, *control theory*, has been developed from the second half of the XIX century and applied in almost every field of engineering.

This theory, however, is *perfect* in the sense of Chapter 2, in fact it requires a precise and certain knowledge of several pieces of information regarding the system to be controlled. First of all, the laws governing its internal behaviour are to be known; second, at any moment its internal state must be determined to compute the difference between its actual and desired value; last, the necessary control action has to

be computed and applied precisely. In practice, this is rarely accomplished for several reasons:

- The available *models* of a system are usually approximations of the real laws, which often are too complex to identify precisely (or mathematically intractable)
- Observability and controllability are not guaranteed for all systems
- Disturbances, noise and errors are not always predictable and tractable: moreover, not all systems are *robust* and capable of absorbing their negative effects
- Some systems can become *unstable* under some conditions

AI "Soft" Control

For such reasons, automatic control is not the exclusive domain of control theory, but is also a relevant field for researchers in AI. Various techniques such as the ones introduced in Chapter 3 have been used, both as an alternative to model-based controllers and together in hybrid control systems. Artificial intelligence, however, has tools suitable for even more general classes of problems.

Vision

In fact, control can be considered a specific part in the broader context of (automatic) system management. The term *management* will be used to describe a set of correlated actions tasks which, altogether, have the purpose of optimizing the operating conditions of a complex system. The actions, which are sometimes improperly referred to using the term *control* itself, are based on our own extension of the approach found in [220] and sketched in Figure 17.

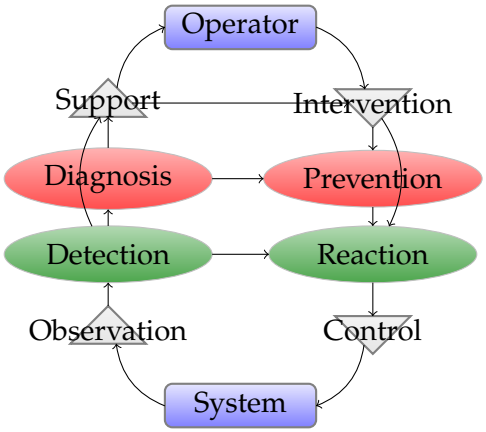

Figure 17: AI-driven management vision

An (intelligent) management system can operate at different levels of abstraction: the lower the level, the simpler but also the quicker the elaborations, usually carried out at a sub-symbolic level; moving towards higher abstractions, instead, the information processed becomes

more structured, up to the point where there can be an explicit interaction with a human user. For each level, there are two functionalities: an analytical one and an applicative one. The former are:

Generalized Observation

OBSERVATION Like in proper control theory, data have to be collected from a system in order to know its status. When quantitative values are acquired, observations are more appropriately defined *measurements*, which can be performed using *sensors* or *probes*. Repeated measurements at regular times can also be called *samples*. Independently of the used method, the data should be validated and stored, using a standard format, in an accessible location. The complexity and accuracy of this pre-processing is usually a compromise between ensuring the correctness of the observation and making the data available to the elaboration system quickly (consider, for example, hard real-time systems).

DETECTION Modules working at this level have to process the observed, raw data to extract important information. When performed in real time, notifications can be generated in the form of *events*, structures signifying that a possibly important change has happened at a certain time. Alarms, for example, are critical events generated as a consequence of the detection of an anomaly in the observed data.

DIAGNOSIS This block has the delicate task of analysing the actual internal status, possibly taking into account some or all the events detected by the lower level. Its purpose is to determine the causes of any change in the operating conditions, especially whenever they could affect the performance of the system. The isolation and identification of all failures obviously falls in this category.

SUPPORT This interface translates and presents the processed information in a human-readable way. It may also be needed to request actions which can't be applied automatically, such as the replacement of a defective component. To do so, appropriate symbolic or graphical languages have to be used, together with a possibly remote communication infrastructure, such as a network terminal.

Generalized Control

Every analytic block has an effective counterpart. While analysis is mainly a bottom-up process, where each level provides more refined information to the one above - up to a possible human supervisor - commands proceed top-down: a high-level order, in fact, is usually translated into simpler instructions. Notice, that every decision block is connected to the action block on the same level: this allows to bypass the higher levels and implement *reactive* behaviours when needed. However, no command can usually be imparted from a lower level to a higher one.

INTERVENTION This interface allows the supervisor to give additional inputs or to impart commands to the management system, at all

levels. It can be used to interact with the system, but also to override its behaviour.

PREVENTION This block is responsible for implementing all medium- and long-term policies. Its main purpose, whence the name, is to prevent failures and, more generally, undesired operating conditions. Typical actions could include the scheduling of maintenance procedures or the updating of some working parameters to reflect an environmental change.

REACTION At this level, all reactive, short-term policies are implemented. The actions are usually simple, dealing with contingent necessities which, however, may have to be solved quickly. An example is choosing the appropriate reaction in presence of an alarm, such as the decision to open a security valve, but also computing the regulation necessary to compensate a set-point error.

CONTROL In this block, the commands (e.g. the commutation of a switch) are converted into input signals for the managed system. These commands may also be actions performed on the system itself by physical (electronic, mechanical, hydraulic, ...) components, generally called *actuators*.

Proper real-time control (RTC) in the sense of (perfect) control theory takes place in the lower levels, but is not the only possible implementation of reactive policies. This architecture, while rather abstract, has been used as a guide-line in the development of the management infrastructure that will presented in the last part of this thesis, which is an example of several alternative control policies. Before discussing the relationships between AI-based techniques and this schema, however, the next section will introduce the main context which this work has been applied to.

5.2 AUTOMATIC MANAGEMENT OF WWTP: MOTIVATIONS

Water is a precious and scarce resource, but easily contaminated by other undesired substances, universally called *pollutants*, ranging from nutrients (carbon and nitrogen compounds) to chemical substances, including heavy metals and toxics, to pathogens such as bacteria and other micro-organisms. The type and concentration of pollutants determines the quality of the water, usually measured using parameters such as the concentration of the cited, undesired substances, but also chemical and physical parameters such as pH, redox potential, electrical conductibility, turbidity. Depending on the desired final use, the values of these parameters are analysed directly or combined into synthetic *water quality indexes* (e.g. see [142]): unfortunately, the admissible ranges of the parameters are subject to the directives[1] of the different countries and the validity of the indexes is often questioned [175],

Water Quality Parameters

1 http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31991L0271:EN:NOT

so there are no absolute, universally accepted criteria to define water quality. Not surprisingly, recent attempts to define quality indexes are trying to take into account the inherent imperfection, for example expressing them in fuzzy terms [288].

5.2.1 *Waste-Water Treatment Plants*

In the specific case of waste water, i.e. water discharged into sewers after domestic, industrial or agricultural use, there exists the general problem of reducing the amount of nutrients present in the water, to avoid the phenomenon of eutrophication, which would alter the fluvial and/or marine ecosystems where the water is discharged. The treatment is performed in dedicated plants, which exploit one of a few existing techniques. In the context of this thesis, only the *activated sludge* plants will be recalled, and even then only the basilar concepts necessary to understand the need and the potential for automation will be discussed: an interested reader can consult any book on the topic, such as [153].

Biological Treatment Processes

The treatment process aims at reducing the concentrations of ammonium ($[NH_4^+]$) and organic matter (carbon C compounds), which compose the main fraction of pollutants, especially in the case of urban waste-water. In activated sludge plants, the water is mixed with sludge in one or more reaction tanks, where the pollutants are consumed by different families of bacteria living in the sludge, provided that adequate environmental conditions are maintained. In particular, nitrogen removal involves two main reactions:

NITRIFICATION The reaction converts ammonium NH_4^+ into nitrate NO_3^-, passing through the intermediate form of nitrite: $NH_4^+ \rightarrow NO_2^- \rightarrow NO_3^-$. The overall reaction is[2]:

$$\begin{aligned} \mathbf{NH_4^+} + \mathbf{1.86O_2} + 1.98HCO_3^- \rightarrow \\ 0.02C_5H_7NO_2 + \mathbf{0.98NO_3^-} + 1.88H_2CO_3 + 1.04H_2O \end{aligned} \tag{5.1}$$

The balance shows that almost all available ammonium is converted to nitrate, while a small fraction - about 2% - is consumed by the bacteria which perform the reaction for their growth. Notice that the reaction requires oxygen to be available. In fact, the environmental conditions necessary for it to take place include:

- *Presence* of oxygen: $[O_2] \approx 3mg/l$ [301]. This is possibly the most critical condition to be maintained: a low oxygen concentration inhibits the process, but keeping the concentration high is rather expensive, since adequate mechanisms

2 assuming $C_5H_7NO_2$ as substrate, other types of organic matter lead to slightly different stoichiometric coefficients

are required to blow air in the tank. A typical blower may consume up to 700 KWh/day[3]
- pH in the range $7 \div 9$
- Temperature around $15 - 25°C$
- Absence of substances toxic for the bacterial populations

DENITRIFICATION The second stage of reaction involves the transformation of nitrates and organic matter into gaseous nitrogen, which is released in the atmosphere, according to the reaction:

$$\begin{aligned} \mathbf{0.07C_{18}H_{19}O_9N} + \mathbf{NO_3^-} + H^+ \rightarrow \\ \mathbf{0.5N_2} + 1.21CO_2 + 0.07HCO_3^- + 0.07NH_4^+ + H_2O \end{aligned} \tag{5.2}$$

Like nitrification, denitrification requires appropriate conditions, which are compatible with the ones required for nitrification, except for one:
- pH in the range $7 \div 9$
- Temperature around $15 - 25°C$
- Absence of substances toxic for the bacterial populations
- *Absence* of oxygen: $[O_2] \approx 0mg/l$.

Clearly, the two reactions can not take place in the same tank at the same time, since the aerobic environment required by the former is incompatible with the anoxic one needed for the latter to take place. To deal with this problem, different plant configurations can be used.

Common Plant Configurations

Unfortunately, the "logical" sequence nitrification-denitrification is not feasible in real plants: the organic matter required for denitrification enters the tank along with the water, but, should aerobic conditions be applied first, it would be oxidized along with ammonium, leaving none for the denitrification. Moreover, since water and sludge have to be mixed for reactions to happen, they have to be separated later in a dedicated *settling* tank: the cleared water is then discharged, while the sludge is reintroduced in the main tank by (sludge) recirculation. Real plants also apply other treatments, both before (e.g. filtering and gritting) and after the reaction (e.g. disinfection). This complexity leads to different possible plant configurations, including single BNR reactors, two-tank P-DEN reactors, membrane bio-reactors (MBR) and others. The plant class relevant for this work, Sequencing Batch Reactors, will be described in Chapter 9.

5.2.2 *Plant automation*

WWTPs are natural candidates for automation. According to [221], an adquate instrumentation could be used to implement the strategies

3 data coming from a real plant near Bologna, Italy

and policies necessary to handle most processes and problems in a plant, sometimes even increasing the capacity of biological nutrient removal by 10-30%.

Motivations

The advanced knowledge acquired on the relationships between the operational parameters in a treatment system and the biochemical reactions (and thus its performance), gives new possibilities to control them, in order to improve the quality of the effluent while keeping the operational costs as low as possible. Another relevant factor is energy efficiency: waste-water treatment industry is a competitive sector and

Energy Savings

energy costs typically consume from 15 to 30% of a treatment plant's operation and maintenance budget. An energy - management program may act on demand-site opportunities, including process modifications, aeration control and power-demand shift.

In order to exploit these relationships, the investments in *Instrumentation Control Automation* (ICA) may reach 20-50% of the total within the next 10-20 years. To this date, instead, ICA technologies are not widely applied, and even then the probability to find a plant equipped with advanced instruments seems to be proportional to its size. The reasons behind this fact vary between different countries: most of them are related to poor legislation, lack of acceptance within the treatment industries, lack of collaboration between stakeholders and organisations, economy and unreliable measuring devices; but probably the most fundamental barrier for a widespread acceptance of new control strategies is that existing WWTPs are not designed for real-time control [221]. In fact, the extent of control (number and type of controlled variables, complexity of the strategies, ...) that is cost efficient is dependent on size of plant. Considering Danish WWTPs, for example,

Operating Costs

the typical operating costs of small plants (< 10000 person-equivalent) are in the range of $0.3 \div 0.7 €/m^3$ of treated water. For plants over 250.000 person-equivalent, the corresponding costs are $0.1 \div 0.3\ €/m^3$ of treated water. The difference in operation costs is dependent on the relative higher load variations at small plants than at larger plants, but also on the fact the instrumentation installed on smaller WWTPs is less effective, if present at all.

Likewise, it is probably an exaggeration to state that in the rest of the European Union most WWTPs (> 10000 person-equivalent) are equipped with SCADA (Supervisory Control And Data Acquisition) systems and, even when present, they are used mainly for mere data acquisition and only rarely for the control of the operating conditions [203]. In presence of failures, success depends on the plant staff's ability to quickly identify the problem, diagnose it and start appropriate recovery actions. An intelligent management system, instead, could behave like a "virtual expert operator", monitoring the processes continuously 24/7 whereas it is practically infeasible for human operators,

Full-time Management

and it could try to optimize the yield and detect faults at an early stage, possibly even correcting them. Moreover, the collected data, properly validated and classified, could be used to build a knowledge base de-

scribing the various operating conditions of a given plant: this knowledge could be used to further improve its overall performance.

Nevertheless, there is a great potential for the applicability of automation technologies to treatment plants, and much research has been done in the last 30 years, sometimes with important results. The next section, then, will be dedicated to an analysis of the existing solutions, outlining their benefits and the limitations.

5.3 BASIC CONTROL TECHNOLOGIES

The automatic management of a complex system is a critical task, which requires an adequate infrastructure. The necessary, but hardly sufficient, condition is the presence of some kind of interface which allows to *observe* the state of the process and *act* on the plant. The trivial structure is shown in Figure 18:

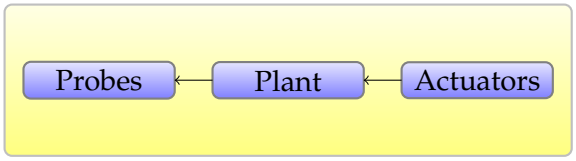

Figure 18: Basic Plant I/O

The variables sampled in a plant may be divided in two classes: the *direct* ones, which measure the state of the process and the biomass, and the *indirect* ones, chemical and physical environmental parameters which influence - and are influenced by - the process itself.

Direct Indicators

The former include the actual concentrations of the substances involved in the chemical reactions, either on an individual basis as oxygen ($[O_2]$), ammonium ($[NH_4^+]$), nitrate ($[NO_3^-]$) and nitrite ($[NO_2^-]$), or collectively as total carbon (COD or TOC), nitrogen (TKN) or biomass compounds (SSV or SST). The most common indirect signals, instead, include pH, redox potential (**orp!**), temperature, conducibility and torbidity, in addition to the various flow rates measured in the pipes. The exact definition of each measure and the characteristics of the necessary hardware are not relevant for this work and will be omitted: additional information can be found, for example, in [220] and [256].

Indirect Indicators

Most data can be acquired through manual analysis or using an adequate on-line sensor: while the first method is relatively inexpensive (but seriously time-consuming), the second can provide much more information, at the rate of several samples per minute, but for higher costs. Laboratory analysis can be sufficient for mere monitoring purposes, but real-time control is possible only through the use of in-line sensors. This poses a serious managerial problem: direct variables can give a complete knowledge of the process status, but the necessary probes usually have a cost which is one or two orders of magnitude greater than the ones for indirect signals ($\approx 10^4$ vs $\approx 10^2$ €). While the cost could be sustainable in large plants, it is clearly an issue for small

Hard and Soft Sensors

scale plants: this advocates the use of *soft sensors* (e.g. [76]), software algorithms which use the indirect indicators, collected using cheaper instruments, to estimate the value of the relevant variables. Not surprisingly, this is a source of imperfection which has to be taken into consideration. An example of imperfect soft sensor will be discussed in Chapters 9 and 10.

Control Variables

Different control strategies can be used to force one or more of these status variables (also summarized in Table 5) to assume the desired values. Some of them are easily *controllable* in the classical sense, by *manipulation* of some other variable through an appropriate *actuator*. For example, the pH and the oxygen concentrations can be kept equal to a desired set point by adding (removing) alkalinity or increasing (decreasing) the air flow rate. Other variables, instead, can still be controlled to some extent, but may require the adoption of long-term policies whose outcome is often affected by a degree of uncertainty. For example, the concentration of *volatile solids* in the tank, an indicator of the amount of active biomass, can be influenced using load and retention policies which favour the growth of the bacteria.

In general, the *manipulable* variables in a plant are less than the observable ones, and some of them are rather expensive. Most controllers, then, act on the recirculation rates between the different tanks: this allows to expose the water to different conditions (i.e. let different reactions take place) for a chosen time period. In order to force the necessary conditions not to inhibit the reactions, however, it may be necessary to add substances such as oxygen, alkalinity, carbon or even heat. The non-trivial operating costs stress the importance of choosing an optimal and useful control strategy [132], [279].

Plant Variables	
Observable	**Manipulable**
Direct (Individual):	Air flow
$[NO_2^-]$	Internal recirculation rate
$[NO_3^-]$	Sludge recirculation rate
$[NH_4^+]$	Carbon dosage
$[O_2]$	Alkalinity dosage
Direct (Collective):	
COD / TOC / TKN	
SST / SSV	
Indirect:	
pH	
Redox potential	
Temperature	
Conductibility	
Turbidity	

Table 5: Typical control variables in a WWTP

5.3.1 *Model-Based Controllers*

Benefits of Using Models

When dealing with treatment plants or, more generally, with complex dynamical systems, an automatic management system may be a worthwhile investment, with returns in terms of increased efficiency, higher reliability and lesser operative costs. Nevertheless, like any other engineering product, a management system has to be designed, implemented, tuned and tested: in such cases, the best practice is to build a model of the system to be managed. A (mathematical) model is an abstraction of a real system capable of *simulating* its behaviour in a wide variety of contexts. The model, unlike the real system, can be analyzed and studied in all its parts: in fact, the utility of models has been proven for several purposes [220], at various stages, including *design* [254], *control* [160], *diagnosis* and *prediction* [81].

ACTIVATED SLUDGE MODELS For the specific case of WWTPs, there exist a set of three standardized models ASMx [154], plus a number of variants. These models are highly non-linear and involve a number of parameters: for example, the ASM n1 uses 9 differential equations of 13 state variables; moreover, it requires a non-trivial calibration procedure [239].

Obviously, water treatment plants are *not* linear, time-invariant systems, so simple control schemas tend to fail due to the difficulty to implement them, even if some applications do exist [287]. The simplest solution, then, is to apply a linearisation procedure to simplify the model [274] and treat it as if it were a LTI system.

While model-based control is not an alternative, but rather a complementary approach to AI-based control, it will not be discussed further.

PID Controllers

LTI systems are often controlled using the proportional-integral-derivative (PID) control schema [266], a particular implementation of the *feedback* control schema shown in Figure 19.

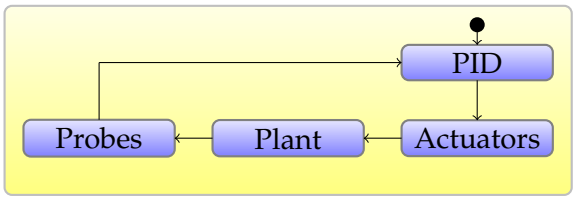

Figure 19: Feedback control with PIDs

This simple schema assumes that the controlled variable can be continuously observed, so it is possible to compute the *error* $e(t)$ by difference of the actual output $y(t)$ and the desired output $s(t)$, given as

input to the controller. The PID manipulates the actuators through a signal given by:

$$u(t) = K_p \cdot e(t) + K_i \cdot \int_0^t e(\tau)d\tau + K_d \cdot \frac{de(t)}{dt}$$

The gain parameters K_p, K_i and K_d have to be tuned accurately, to ensure an acceptable response time while keeping the controlled system stable at the same time. PIDs work best when applied to LTI which model is well-known, but unfortunately WWTPs as a whole are neither linear nor time-invariant. While it is still possible to partially overcome the non-linearity of the system [291], more often different, simpler controllers are applied to specific sub-tasks, such as keeping the oxygen concentration to a desired set-point [124] [304], or optimizing the chemical dosage in post-treatment [100]. Moreover, the *time-variance* of the plant poses the problem of updating both the set-points and the gain parameters as the environmental conditions change [209].

5.3.2 *Artificial Intelligence-based Controllers*

The complexity of systems like WWTPs manifests in the difficulty of designing a good controller, not to mention the calibration of an model of the original system itself. Moreover, its intrinsic time-variance would hamper the validity of the measurements required to tune its parameters, making the model inaccurate in a relatively short time. If one takes into account the disturbances, which may be quite relevant and unpredictable (i.e. a toxic discharge, or a strong rainfall), the plausible scenarios become much less *precise* and *certain* than the ideal situation modelled my the mathematical equations. Even when using models, it is necessary to take this uncertainty into account [132], but not surprisingly many researchers have tried alternative approaches, in particular applying *Soft Computing* techniques such as the ones introduced in Chapter 3, hoping to exploit their robust approximation capabilities. SC algorithms may be used in place of mathematical models, but sometimes they are also used in combination with controllers such as the PID, adopting the schema shown in Figure 19 or even a more complex one [164]. The upper level adjusts the se-points and the parameters of the lower level, while this one commands the actuators to minimize the error. This is actually an example of *cascaded*, *online*, *cooperative* hybrid AI system.

Adaptive Control

NEURAL NETWORKS Neural Networks are appealing tools to deal with complex, ill-known non-linear systems. Their "black-box" nature allows to approximate a non-linear function without having any explicit knowledge on it, but a large set of samples for the training, validation and test. In fact, there exist several works combining different sets of variables: a complete survey can be found in [42]. From an analysis of the recent literature, it emerges that the multilayer feed-forward network is the most common architecture, and that NNs have

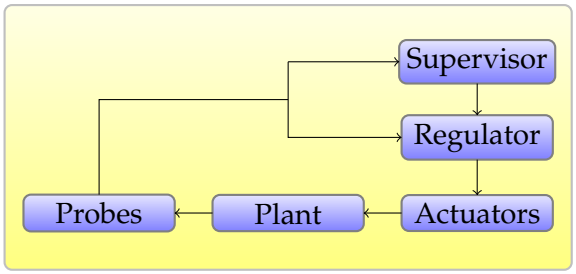

Figure 20: Feedback control with Adaptive Controller and PID

been used for general modelling ([295],[207],[290]), but also for prediction ([35],[148],[76]) and to implement control blocks even in more advanced schemas than the simple feedback loop ([240],[43]). The overall impression, however, is that despite the many, more or less successful attempts, most of them have been carried out in controlled environments such as laboratory-scale pilot plants, where the disturbances are limited and the seasonal variations are rarely taken into account. This removes much of the variability associated to a real-world complex system: the regularity in the input data causes the networks to behave like interpolators (a task they are capable of performing), whereas in a realistic context they would have to act more like extrapolators (a task they regularly fail at), unless they were constantly retrained to adapt to the changing conditions, or a data set sufficient to cover all the possible cases had already been collected to train it in the first place. This dependency on large amount of data, whose collection is an expensive and time-consuming process, is possibly the main limiting factor to the use of simple neural models in real applications of some complexity.

FUZZY CONTROLLERS The main alternative to mathematical models and neural networks is the use of fuzzy logic systems. Unlike neural networks, they require an initial knowledge of the underlying function they are to approximate, be it the model of the system or the controller's policy. In fact, in standard fuzzy control architectures the fuzzy partitions of the input and output domains are not learned from data, but generated manually by the designer, even if they can be tuned afterwards. This is usually not a drawback, since it allows to build controller acting in a way that results "familiar" to the human operators, and thus more acceptable.

The main purpose of fuzzy logic systems is to act as controllers, which have often been applied to the regulation of the oxygen concentration in the aeration tank ([128], [130],[285]), but this is not the only existing application: there are examples not only for the control of other variables (e.g. [286]), but also for approximation and prediction (e.g. [87]). A survey on the topic can be found in [38].

Like neural networks, fuzzy logic systems can be used at both levels of the architecture shown in Figure 20, implementing either the supervisor or the controller. Actually, there exist hybrid schemas where NN

and FS have been used in all the possible combinations (e.g. [224]), even with the use of PIDs at the lower level (e.g. [72], [255]).

5.4 ADVANCED MANAGEMENT ARCHITECTURES

While automated controllers have given relatively good results, especially in laboratory, it is evident that a PID or a neural network is hardly sufficient to capture all the complexity of a water treatment plant, and neither to satisfy all the needs outlined in Section 5.2. Indeed, controllers, can and should be used, but their role is that of specific modules integrated in a larger architecture, more adherent to the ideal one shown in Figure 17.

5.4.1 *Remote Management Infrastructures*

The next advanced management schema gives a relevant role to the plant operators, who act as supervisors of the behaviour of the controllers. The operations of detection, diagnosis and the appropriate consequences are not actually automated, but carried out by the human personnel. Unfortunately, it is infeasible to keep an operator full time on a hazardous plant, especially if the plant is small and located in remote areas. Nowadays, however, it is relatively cheap to transmit the information acquired on the plant to a remote location, from where the operators can monitor several plants at the same time. A bidirectional communication line also allows to send some commands back, reducing the number of interventions that must be effectively performed on site (e.g. the replacement of a defective component).

Remote Web Interfaces

Modern remoting platforms usually are based on web applications. Data are collected and stored in a database: clients needing them send a request to a centralized server which formats them using the `HTML` language and extensions thereof, so that they can be viewed using a browser interface.

Such platforms are mature, so they are not much a filed of research (although their application to novel contexts, including some environmental ones, still is: [250], [298], [135]), but rather a market for small and large companies. Obviously, commercial implementations are more sophisticated: databases are replaced by more robust data warehouses; the communications are secure and reliable; interaction with the system is regulated using some form of authentication; finally, integration with other information systems such as GIS can be provided.

While such architectures are designed for automatic control and remote management, they are still "hollow" from the point of view of automated management. Even if they can usually be extended with dedicated business logic, in practice the operative *decisions* regarding the application of medium and long term policies, as well as the handling of the anomalous operating conditions are still delegated to the human operators.

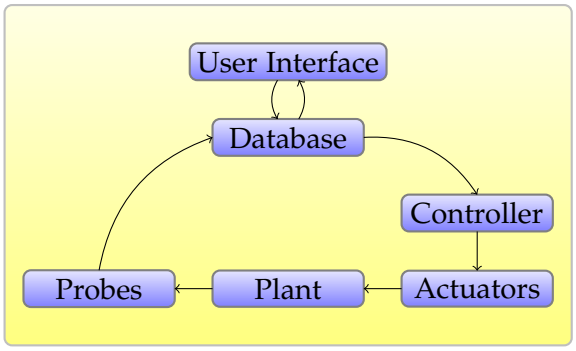

Figure 21: Remote Management Platform

5.4.2 *Decision Support Systems*

When artificial intelligence techniques are used to implement analysis, diagnosis and decision tasks, thus emulating the behaviour of a plant manager, and such modules are integrated in a (remote) management structure as shown in Figure 22, one can properly speak of (Environmental) **Decision Support Systems**, or (E)DSS.

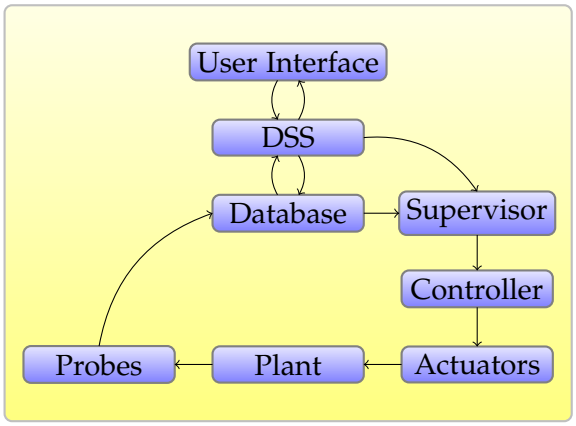

Figure 22: Decision Support System Architecture

The term DSS is quite general [246] and can be applied to any tool capable of aiding a human agent in making a decision, independently on how the tool is implemented, but here the attention is focused on the DSS which are *knowledge-based* and *hybrid*, i.e. those relying on different AI and statistical techniques. The role of DSS is usually suitable for an *Expert System*, but not all decision support systems embed an expert systems, nor all expert systems have decisional purposes. In this work, however, the terms will be used almost interchangeably: the DSS found in literature which will be discussed shortly after and the one which was implemented as a part of this thesis rely heavily on knowledge. While it is true that they can interact with human users and influence

DSS vs ES

their decisions, they also act as automated plant managers and so they have to take their own decisions - a task which requires a noticeable *expertise*. It would be wrong, however, to consider the term "Expert System" in its original acception of static, hard rule-based system. Instead, from now on, ES will be implemented using hybrid hard/soft computing techniques.

The architecture shown in Figure 22 is in fact an abstraction fitting quite a few relevant examples that can be found in literature, such as [67], all of which include a hybrid, modular core. Most authors agree that the variety of information and tasks the DSS has to solve are better dealt with using different techniques at different levels of integration, from *independent* to *cooperative* (see Section 4.2).

Component orchestration

In [162], the modules have a very fine granularity and perform simple, basic tasks. They are more properly called *components*, a term deriving from the specific implementation. The behaviour of the system is specified using *use cases*, a design choice that mixes the cases of CBR and the scenarios of SOAs. When the triggering preconditions are met, a predefined sequence of elaborations is executed, where the components invoke each other. Some of the components, moreover, employ fuzzy logic to be more robust in case of possible imperfections in the input data. While impeccable from the software engineering point of view, it remains a monolithic architecture which components are indeed modular and reusable, but hardly outside the container in which they have been developed. This, together with the use of proprietary or obsolete software tools, is the main limitation of this DSS which, nevertheless, applied promising concepts.

The architecture shown in [163] is interesting for several reasons. First, the methodology it proposes is easily generalizable to other contexts. Second, it combines many hard and soft computing techniques with different goals, from neural networks (used for prediction and approximation) to genetic algorithms for optimization to (fuzzy) rules for diagnosis and control: the underlying principle is trivial - a specific problem should always be solved using the best available tool. This, however, poses the problem of reconciling different modules which may encode the data in different ways. The proposed architecture is hierarchical: the output of a prediction module is used by a planner/scheduler to determine the necessary mid-term policies, which in turn are applied by a control/diagnosis module commanding the various controllers proper. This can be considered an *online*, *cascaded*, *cooperative* hybrid system.

Supervised Pools

The methodology and the concrete implementations proposed in [241] and [88], instead, adopt a *parallel* integration. The different DSS modules, which range from fuzzy case-based reasoning to decision trees to neural networks to embedded mathematical models, operate in parallel for diagnostic, predictive and planning purposes. Their results are combined by a dedicated *supervisor* module which, in case of conflict, uses a simple ranking criterion. This null approach to interaction is possibly the main drawback of this solution. Moreover, the implemen-

tation of [88] is based on a proprietary environment which makes it hard to integrate with other external systems. Such issues have been partially solved in [75], where a similar architecture is integrated with a domain *ontology*. An ontology is a formal representation of a set of concepts within a domain and the relationships between those concepts: in the specific case, the ontology *WAWO* has been defined on the domain of WWTPs. This ontology is used to resolve some of the apparent conflicts - the ones which could be considered "misunderstandings" - between the outputs of the modules. This is possibly the first DSS using ontologies in its development.

Use of Ontologies

A similar proposal can be found in [45], with a difference: the modules composing the DSS block are not dictated by functionality, but each of them is dedicated to the management of a single sub-part of the plant.

5.4.3 *Complex Architectures: Services, Events, Agents*

So far, the blocks of a management architecture have been discussed, but little has been said on how to *integrate* them and make them *interact*. Moreover, the analysis of the DSS block shows that the actual number of its internal modules and roles involved can be large, growing with the number of the plants and the complexity of the tasks necessary to manage them. From an engineering point of view, a real implementation of such an articulate system can't rely on ad-hoc solutions [98], but requires the use of well-defined, reliable and, most of all, *standard* principles and tools. In fact, the resulting system should be robust, but also flexible and extensible enough to be adapted to different contexts quickly, without having to rebuild it from scratch. For this reason, it is important that as many modules as possible can be reused and reconfigured quickly.

The DSS introduced so far are modular in nature, but the interaction between modules does not seem a primary topic[4]. While their internal structure is hybrid and modular - albeit often *parallel* and *concurrent*, with little interactions between the parts - their external structure is monolithic. The DSS has a definite interface used by other blocks such as the data storage and the user interface, but otherwise it is a black box. In fact, this is not the only option. To this end, the benefits and limitations of modern mainstream architectural solutions will be considered.

Service-Oriented Architectures

A Service-Oriented Architecture (SOA) is a design paradigm for large software systems, based on the vision of Service-Oriented Computing [227]:

> The visionary promise of Service-Oriented Computing is a world of cooperating services where application com-

4 at least, in the works where they are described

ponents are assembled with little effort into a network of services that can be loosely coupled to create flexible dynamic business processes and agile applications that may span organizations and computing platforms.

Services

The architecture is centred on the concept of *service*: a capability of a (software) entity which is offered as a functionality to the external world, so that it can be used by other entities needing it, but which would not be able to implement it on their own. An application, then, is composed of several self-contained modules, each of them exposing a collection of capabilities - here called *actions* - which can be exploited by other modules or external clients. For a service to be usable in practice, the granularity of its actions should not be too fine, nor too coarse, in order to obtain a good trade-off between re-usability, flexibility and performances [280]. Services may have different roles, which can be roughly divided in [125]:

ENTITY An entity service models the entities involved in an application: its actions mimic the functionalities of the modelled objects.

TASK A task service provides a *business logic* functionality, specific of the applicative domain.

UTILITY An utility service provides a generic, reusable functionality, which is not tied to any application in particular but useful in many.

Service design principles

The design of a SOA and its services is inspired by some basic principles [125], such as *re-usability*, *abstraction*, *loose coupling*, *composability*, *discoverabilty* and *negotiability*.

The adoption of such principles usually requires a larger initial investment since the development of the modules of an application is more expensive in terms of time and complexity, but a convenient return is expected when the application has to be extended, or a new one has to be developed reusing many of the available components. A service-oriented application, then, generalizes the concept of *client-server* interaction by decoupling the interactions and the roles. In fact, in a SOA three main roles are involved:

SOA Roles

SERVICE PROVIDER encapsulates a business function (or a set thereof), exposing a corresponding usage contract; it can interact with the service registry in order to publish its contract.

SERVICE REGISTRY maintains a list of known service contracts together with their location. It is actually a *well-known* service provider itself (its location is known to all other services), whose capabilities allow to search for a certain functionality (*discovery)*, returning the location of a service able to accomplish it.

SERVICE CONSUMER is the entity which requires a certain functionality. In general, it does not know where a service able to perform the requested action is located, but it can perform a *lookup* action to find a suitable service. After retrieving the contract and

location of a candidate provider, it can finally interact with it by invoking the desired functionality and obtaining a result (if any).

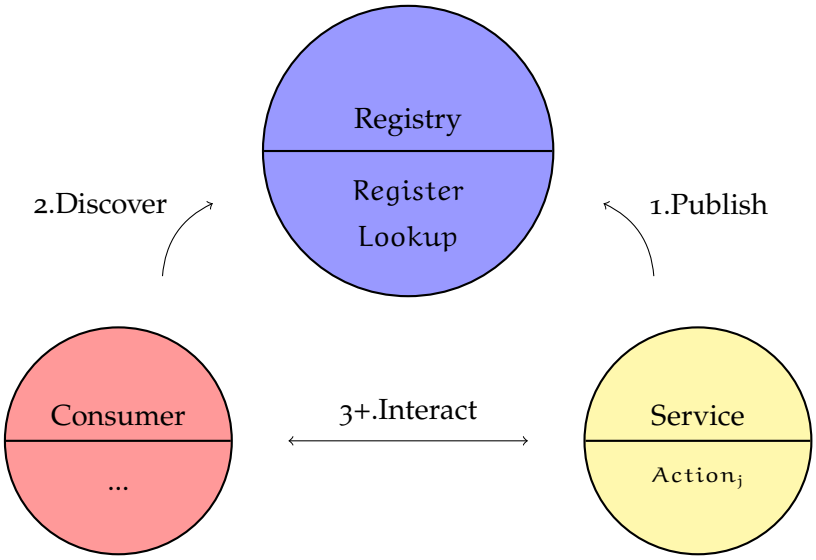

Figure 23: Service-Oriented Architecture

ORCHESTRATION AND CHOREOGRAPHY The role of service consumer could be played by a human user but also by another service provider, according to the principle of *compositionality*, giving birth to complex collaborative distributed business applications. *Loose coupling* even allows different organizations to implement and provide their own services, whose interaction can be mutually beneficial.

SOA Interaction Models

Two different complementary approaches can be followed to realize/perceive a service composition [238]:

- Orchestration perceives the collaboration by the point of view of a single entity, who defines an applicative scenario (*business process*) were interactions between internal and external services can take place, delegating the realization of some activities to such services. The execution is controlled by the generating entity: it acts as an *orchestrator*, coordinating the other services and correlating the results obtained from them. The invoked external services are unaware that they are participating in a collaboration. The orchestration itself could be exposed as a single functionality on the network, making it possible of realizing even more complex "nested" orchestrations.
- Choreography models the collaboration from a global, objective point of view, independently from the perception of the single interacting services. Unlike orchestration, its focus is not on executability, but rather on capturing the public contract which

provides the necessary rules of engagement to make all the interacting parties collaborate correctly.

While orchestration is useful when a single party is interested in aggregating the capabilities of a set of internal/external services, choreography helps when the collaboration must be achieved by taking into account the mutual requirements of interacting parties, without assuming a unique center of control.

SOA IMPLEMENTATIONS In order to effectively make heterogeneous service providers and consumers interact, the three fundamental operations as well as the published service contracts require standardized languages and protocols. Nowadays, many applications are based on SOAs: even if the paradigm poses no constraints on the implementation, many rely on WEB-services. In a WEB-service-oriented architecture, the world-wide web and its related standards are used as communication and transport infrastructure, so services interact using well-known languages such as HTTP, XML, SOAP [62], WSDL [84] and UDDI [53].

WEB SOAs

The web implementation is preferred when services are provided by different companies. When applications are developed within the context of an enterprise, the communication middleware is often implemented using an Enterprise Service Bus. In a nutshell, an ESB is a message-oriented middleware (MOM!) [262] optimized for supporting services and the communications between them, ensuring that the messages from sent one service to another are delivered in an efficient and reliable way. Concrete ESB implementations usually offer more and more sophisticated functionalities, such as adapters allowing to accept "messages" in external formats (e.g. HTTP/SOAP, SQL, FTP, ...) and embed some native utility services such as security and authentication, data transformation, service registry and more. Examples of SOAs built on top of ESBs include [219] and [96].

Enterprise SOAs

Complex Event Processing Architectures

Complex Event Processing (CEP) is an emerging approach based on the concept of event, a record *signifying* a change of state in a system at a certain time [186]. In particular, an event has a *form*, i.e. the symbolic data structures used to represent the actual activity in the real world, the *significance*. For example, the act of a temperature exceeding a safety threshold at time T could be an event, represented by a *typed* structure `AlarmEvent` holding, for example, a `DateTime` field, a `String` with the id of the temperature probe and a `Double` equal to the measured value.

Events: Form and Significance

Most real-world systems can be viewed as event sources: some noteworthy examples are stock markets in finance, plants in chemistry and human bodies in medicine, just to cite some. The peculiarity of such systems is that they generate dozens of different events every minute - possibly even every second - but often only a few of them are actually

relevant. In the previous example, every temperature measurement, sampled with a given frequency, can be considered an event (in the specific case, an *observation*), but for the purpose of detecting alarm conditions, only the ones above the safety threshold would be of some importance. The challenge, then, is to filter, sort and analyse the events, possibly aggregating them in higher order events at different abstraction levels, so to extract and model only the information that is really relevant for an application.

Event Relations

To this end, one can exploit the different possible *(cor)relations* between the events, namely:

- **Time**: the time at which an event takes place, measured by a clock c is converted into a timestamp which allows to define an ordering relation *before* ($\leqslant_c$) on the set of events. $\leqslant_c$ is a total order, so, for any pair of events e_1 and e_2, either $e_1 \leqslant_c e_2$ or $e_2 \leqslant_c e_1$ holds. In fact, it is common to refer to a sequence of events with the term "*stream*". Notice, however, that this relation can be defined only if all the timestamps are marked using the same clock or a set of perfectly *synchronized* clocks; otherwise two events may not be comparable.
- **Cause**: even if the notion of cause-effect is less clear than the notion of time, it is possible to define a weaker notion of *computational causality*: e_1 is a cause of e_2 if and only if e_2 could not have happened without e_1, i.e. $\neg e_1 \Rightarrow \neg e_2$. This relation induces a partial order on the set of events, since independent events exists.

The two previous relations are actually typical of any event-processing system. *Complex* event processing introduces a third:

- **Aggregation**: an *aggregated* event signifies an activity consisting of the activities of a set of events $e_1, \ldots, e_n$, its *members*. Membership, too, is a partial order on the set of events. Notice that a complex event can be a member of a higher-level complex event, thus forming an event *hierarchy*

Aggregate events are *complex*, high-level events. A complex event is always (computationally) caused by its members, but it is not usually temporally comparable to them. In fact, a complex events E lasts over the time interval $T(E)$ wrapping all its members:

$$T(E) = [\min_{j:e_j \in E}\{T(e_j)\}, \max_{j:e_j \in E}\{T(e_j)\}]$$

so its timestamp is better replaced by a pair of timestamps delimiting the interval. An interval can be defined using any two quantities among start time, finish time and duration. The use of intervals expands the class of relations which can be defined between events, including concepts such as `before`, `after`, `during`: in fact, the operators form an algebra [34] on the set of events.

EVENT MATCHING The first step in event processing is the detection of relevant events. The concept of relevance is obviously relative to an applicative context, so it must be possible for a developer to define the conditions under which an event is of some importance. The canonical way is to define one or more *event patterns*, templates which match the sets of events one wants to select: a template, then, acts like a *filter*, extracting the desired events from the main stream.

Rule-Based Matching

Event matching is a particular case of pattern matching [119], where the features to be compared are the generic data structures within an event's *form*, which can be a mixture of qualitative and quantitative variables, including start and finish time. Using the approach defined in [186], objects (in the sense of object-oriented programming) are used as forms, so patterns are defined in terms of *constraints* on the event's class and fields. Such constraints can be combined using logical connectives to form complex patterns: for example, the pattern "all temperature samples over 25 °C in the last ten minutes" is translated into the conjunction of the simple constraints `class == Sample`, `type == temperature`, `value > 25` and `timestamp > (now()-10m)`, assuming the intuitive meaning for the fields. A constraint can also involve more than one event and thus be used to define patterns over sets of related events. The logic-oriented approach is not necessarily the only way to perform event matching - in fact, any pattern matching algorithm could be applied, while in [64] an automata-based solution is adopted - but is surely the most common [230]. Moreover, this class of problems is exactly the one the RETE algorithm discussed in Chapter 3 was designed to solve.

EVENT PROCESSING Once events have been selected and extracted from the stream, they can be processed by computational units which can be called *Event Processing Agents*. An EPA is associated to one or more event patterns and processes only events matching any one of them. In particular, EPA can be roughly classified into:

Event Processing Agents

- **Filters** : a filter divides a stream in two. An input event is assigned to either output stream, according to whether it matches a pattern or not.
- **Constraints** : a constraint is conceptually similar to a filter as it matches each incoming event e_j with its associated pattern. However, it does not propagate the event, but generates a higher-level event (caused by the simpler one) if e matches the pattern, or discards it otherwise.
- **Maps** : a map combines one or more relevant events, related by pattern-defined constraints, into aggregate events and thus is a building block of event hierarchies.

If patterns are defined in logical terms, an EPA is actually a miniature reactive rule engine (see Chapter 6) which executes the required actions as a consequence of the triggering of its rules.

EVENT NETWORKS The events generated by one EPA can be passed as input to another EPA, forming *Event Processing Networks* [186]. The EPAs form the nodes of an EPN, while edges correspond to communication channels between agents where events are transmitted when they are generated from an external source or by the triggering of a rule. In the first case, the sources can also be distributed, so a network can collect and integrate events generated in various locations. An EPN can then be used to build an event hierarchy, moving from lower to higher levels of abstraction as the events are processed and refined by the agents. Moreover, the use of filters and constraints can speed up the overall processing, since irrelevant events are discarded as soon as they are recognized not to be relevant; more interesting events, instead, are routed to all and only the agents which can process them. A concrete case of EPN will be discussed in Chapter 10.

Event Processing Networks

CEP ARCHITECTURES Being a relatively novel concept, not many real-world CEP applications exist, and even then sometimes they are not published for strategic reasons ([204]). Nevertheless, almost all mainstream BRMS are including complex event processing among their functionalities (see Chapter 6) and some dedicated frameworks are beginning to appear [222]. CEP-oriented architectures can be applied to any system, provided that the observation interface is configured to generate events whenever it detects a state change in the observed system: notice, however, that simple *adapters* can be used to integrate any legacy infrastructure which does not support events natively.

Given its event-oriented, reactive nature, CEP is an appealing technology for general monitoring and management applications, and all the more when complex systems such as WWTPs are the application's target. To this date, only a few very specific applications exist ([264]), and even then they are not cast in the context of CEP, even if they satisfy all the conceptual requirements. The schema in figure 17, instead, can easily be fit in the context of CEP: a plant is a system whose state changes continually as the reactions take place and the electrical and mechanical components function. Each sample collected by a probe, as well as every action performed by an actuator, can be considered a low level event: the detection layer performs the event matching and processing, possibly triggering higher level events which, in turn, can be recorded, used for diagnostic purposes and/or notified to the (remote) user. The reactive nature of event processing, moreover, is suitable to perform control and more general intervention actions. It can be argued that not all management tasks are reactive in nature, but may include planning and pro-action, so CEP would not be the killer technology for management applications. Nevertheless, it is an option which becomes even more effective when appropriately integrated in a broader context, as will be shown in the last part of this work.

CEP and WWTPs

Agent-Based Architectures

Multi-agent systems (MAS) [300] are another vast class of complex software systems founded on the concept of *agent*. An agent is [218]

> ... a component of software and/or hardware which is capable of acting exactingly in order to accomplish tasks on behalf of its user ...

This definition is rather abstract: in fact, there is no general agreement on what an agent exactly is, except that it should be an *active* and *autonomous* entity, i.e. possessing a private execution flow and an adequate interface allowing it to *interact* with the external world an other agents, both in a *reactive* and in a *proactive way* [299]. MAS are a suitable paradigm to deal with complex problems, large in terms of data and operations involved: the necessary tasks are divided among the agents, so that each one can try to solve a limited, simpler part of the whole. Moreover, the parallel elaboration increases the overall performance, while the interaction capabilities allow the agents to cooperate and share intermediate results.

Multi-Agent Systems

INTELLIGENT AGENTS There exists countless types of agents, classified according to different criteria which evaluate properties (e.g. mobility [233], proactiveness [171], rationality [5] ...) and goals (e.g. information collection rather than interface). This work, however, is focused on *intelligent agents* [299], agents with and internal *knowledge* of the environment where they act and whose actions are directed towards the fulfilment of some private *goal*.[5]

The intentional stances of an agent can be divided in two categories, *information* attitudes and *pro*-attitudes, summarized in Table 6 The former include *knowledge*, static and objective information, and *beliefs*, contingent information on the actual state of the world. The latter are more "emotional" dispositions which guide the agent's choice of actions.

Intentional stances	
Information	**Disposition**
Knowledge	Desire
Belief	Intention
	Commitment
	Obligation
	...

Table 6: An intelligent agent's intentional stances

The exact number, type and implementation of stances define the theoretical framework in which the agents are developed. For example, one of the most widely used agent models, BDI [251], is based on

5 This is more properly an *intentional* definition of intelligent agent, and not necessarily the best one.

belief, *desire* and *intention* alone. In [141], instead, also obligations are included as primitive notions. The nature of an agent is then fully determined when one decides how the stances are combined and fixes the underlying architecture, including the reasoning capabilities. Notice that this *belief* is "perfect", i.e. it is usually intended in a sense closer to that of modal logic, which is a crisp version of possibilistic logic - and not in its uncertain sense. In particular, there exist a dedicated logic, BDI logic [252].

Reactiveness vs Proactiveness

Agents can then be classified in *deliberative*, *reactive* and *hybrid*. The former use an explicit *symbolic* representation of their beliefs and use it achieve their *goals* (desires), usually through some *planning* process influenced by their *intentions*; reactive agents react in response to external stimuli, according to some policy usually encoded using sub-symbolic techniques. Hybrid agents, instead, combine both behaviours. They have a case base of pre-configured strategies which are quick to retrieve (e.g. using a CBR module, or condition-action rules[6]), even if potentially sub-optimal. Actual planning, then, takes place only when an agent has the time and resources to do so ([48]). Notice that this notion of hybrid agent is compatible with the more general notion of hybrid AI system.

Given their symbolic nature, logic programming-derived systems are often used in the development of intelligent agents ([58], [141]), but other AI tools can be applied, in particular the associative and reactive ones such as neural networks, CBR and (fuzzy) forward-chaining rule-based systems [46].

AGENT INTERACTION The "intelligence" of an agent-based application does not depend only on the nature and degree of intelligence of its component agents: for example, the well-known optimization techniques based on natural metaphors, such as colonies [105], are rightfully considered AI applications, even if the "agents" involved are rather primitive. Intelligence, instead, is an emerging behaviour which depends on the complex sequences of interactions which take place between the agents. In particular, there are two possible ways for agents to interact:

- Direct: an explicit message is passed between the agents. It is essential that the agents "speak" a common language: there exist two main language standard proposals, FIPA-ACL and KQML which define the message types and syntax. In addition to that, the agents must share the same set of concepts: to do so, ontologies are used [28] to ensure that there is no ambiguity in the terms used by two independent agents.
- Indirect: the agents interact through the environment. An agent desiring to leave a message alters the state of the environment, so that the change can be detected by other agents through their sensors [165]

6 see Chapter 7

An exchange of messages between two or more agents should be ruled by means of specific *protocols*. Much research is being carried out on the languages which can be used to define the protocols, as well as the runtime execution, verification, monitoring and analysis of interactions [205].

ENVIRONMENTAL AGENT-BASED APPLICATIONS Many environmental applications have been implemented using MAS: even some of the DSS introduced in Section 5.4.2 can be considered agent-based systems. Water treatment seems to be the main application for MAS (e.g. see [94]), followed by air pollution and meteorology [40] and forest fire prevention, but there exists even other case studies [89] (a comprehensive and recent survey can be found in [41]). In the above systems, the notion of agent is used in a wide sense and not necessarily with the meaning of intelligent agent. In [39], a methodology involving both belief-oriented reactive ("information carrier") and planning ("decision making") agents is proposed, while [94] uses generic agents communicating through a blackboard. In general, from [41] it is clear that different types of agents, from simple reactive, to planners to learners, have been used for specific environment-related tasks.

In fact, another (very general, actually) definition of agent [258] states exactly that "*An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that evnironment through effectors*". Nevertheless, this is a fitting vision for a management architecture: a single agent is unlikely to be able to handle all the complexity of systems such as WWTPs, but MAS may, provided that they can coordinate their work efficiently, either through a supervisor entity such as the ones cited in Section 5.4.2, but also interacting using a common language and especially a common ontology [74]

5.4.4 *Combining Events, Services and Agents with Imperfection*

So far, three advanced paradigms - SOA, CEP and MAS - have been presented as candidate models for the development of an automatic management system: in fact, there exist several real-world cases where each one has been applied with various, but nevertheless good, degrees of success. However, the three different approaches are not alternative, but clearly complementary. In one of his white papers, Luckham ([184] and [185]) clearly states that:

> An event driven architecture (EDA) is a service oriented architecture (SOA) in which all communication is by events and all services are reactive event processes (i.e., react to input events and produce output events).

A SOA stresses the role of modularity and the strong separation between an interface and its implementation, but poses no constraints on *what* a service should do and *how* it should be done. Nothing, then, prevents the creation of a specialized class of event-processing services,

invoked whenever the relative events actually happen. At the same time, services can generate events as a part of their execution. However, there is one important difference in the interaction modality: a service invocation usually involves a handshake protocol between the consumer and the provider, negotiating the quality of the service. The protocol is driven by the service consumer - from now on, called *client* - which is the party interested in getting the results back. When events are involved, instead, the source is never interested in knowing who will process the event, nor in getting a response back: it is instead responsibility of the event consumer - from now on, called the *handler* - to request that an event be delivered to him. This is not a limitation: the *subscription* is a specific service request, while the subsequent event notifications can be implemented using one-way, asynchronous invocations. On the other hand, should some information be returned, it is always possible to generate a response event for the original source to intercept. Thus, an event processing protocol is a special case of cascaded service invocations, even more loosely coupled than a pure SOAs would use. In this framework, then, the role of agents is clear: SOA are implementation agnostic, so an agent is a natural candidate for the implementation of a service. The agent exposes some of its capabilities through the service interface, so it can be called either directly "by name" (e.g. using an ACL) or "by role", through a service invocation protocol[7]. Depending on the application, agents can be intelligent or not; when event processing is involved, however, EPAs should be intelligent and reactive (or hybrid). Moreover, if Luckham's original vision is adopted, an EPA should also be rule-based [186].

Adopting this point of view, it is trivial to extend Luckham's vision:

> **A hybrid event-driven, service-oriented architecture is a SOA in which part of the communication regards event notifications, and all services are backed by intelligent event-processing agents.**

The idea is sketched in Figure 24, where two agents implement two services. As usual, the role of service client and service provider depends on the context. One of them is also an event source: the events it generates are processed by the second, who handles them using one of the actions he is capable of.

Fitting Vision with Design

The hybrid complex architecture fits perfectly with the ideal schema of Figure 17. Whenever it is applied to the development of an automatic management platform, the changes observed in the managed system generate low-level events which are then analysed and combined into more complex events. These events are propagated to the higher level of the architecture, according to a correspondence between level of abstraction and management role. In fact, agents operating at the detection level are mostly reactive, while agents involved in diagno-

7 This is actually a natural metaphor in everyday life: for example, you can either ask for the aid of John Doe, because you know him to be a skilled technician, or you can look in the white pages for a technician, then John Doe will answer

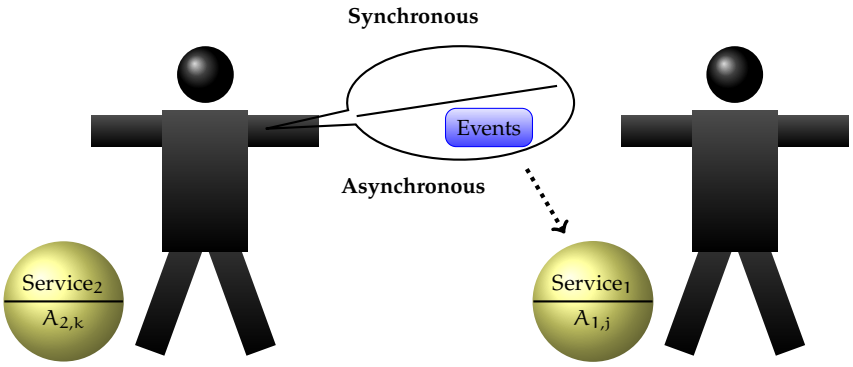

Figure 24: Integrating Agents, Services and Events

sis and intervention are hybrid, with long-term reasoning capabilities. The deliberative taks may be require different actions than simple planning, actions which can conveniently be provided by other specialized agents acting as service providers.

THE ROLE OF IMPERFECTION Despite all the reasons suggesting the adoption of a hybrid complex architecture, one should not ignore an important detail when designing a management system : like any other form of information encoding, events may be affected by imperfection. This can manifest in different ways, including:

- If some events are partially unobservable, there may be uncertainty due to the missing data.
- Events may be reported with imprecision, e.g. because the measurements are unreliable or afflicted by error.
- In presence of anomalies, the information carried by a given combination of events may be conflictual.
- The conditions used to create a complex event from simpler ones may not be certain (i.e. when detecting the insurgence of a disease from its symptoms).
- ...

Moreover, additional imperfection can be generated by the management process itself, in addition to the amount inherited from the input:

- It may be convenient (if not necessary) to express some constraints - especially the temporal ones - with some degree of vagueness (e.g., event A happens “more or less at the same time” of event B).

- Policies too complex to be applied, especially in real-time tasks, may have to be approximated to be simplified
- Any event-based prediction is intrinsically uncertain, so policies based on such predictions are not guaranteed to be successful.
- ...

As pointed out in Chapter 2, imperfection can even be considered a beneficial property of data, provided that it is possible to isolate and evaluate it. For sure, given its intrinsic presence in the data acquired from the managed system, the agents must not ignore it, or their actions risk to be founded on an idealized, unrealistic model. This poses a relevant design and implementation problem: event-processing agents are usually rule-based, but no mainstream rule-based system alone can handle imperfection in the proper way (with the exception of fuzzy rules). In order not to lose the benefits of such an implementation - namely the unified, declarative approach - the only alternative was to develop a rule engine capable of handling imperfection in a native way.

5.5 CONCLUSIONS

The automatic management of a complex system such as a WWTP can lead to several benefits: in the specific case, there are both environmental and economic advantages, since the quality of treated water can be increased while, at the same time, lowering the energetic and monetary costs. The automation of a plant, however, goes beyond the classic, *perfect* notion of automatic control, since it also includes tasks such as fault detection, diagnosis, planning and, globally, decision support and self-adaptation.

Such variety of non-trivial problems can hardly be solved by a single "killer" technology: instead, it has been shown that a combination of mathematical and statistical tools, together with various AI techniques can be largely more effective since each sub-problem can dealt with using the most appropriate technology.

The integration of the various modules, then, requires a complex architecture in order to be efficient, scalable and extensible. An elegant solution could be an architecture where knowledge-based agents offer their services to other entities, while also handling (complex) events, but the construction of the intelligent agents requires a suitable engine to process the beliefs and the knowledge of the agents themselves. Moreover, this knowledge would be *imperfect*: the reasoning processes should not ignore this, but unfortunately no engine exists capable of doing so in a general way. Hence, the next part of the work will be dedicated to the design and implementation of an engine with the required capabilities: this, in turn, will become the main building block of a hybrid "intelligent" management system.

Part III

A HYBRID RULE ENGINE

6

BUSINESS RULES MANAGEMENT SYSTEMS

Contents

In Chapter 5 it has been pointed out that rule-based systems have an important role in (complex) event processing and, more generally, in the development of reactive/hybrid intelligent agents. When the agents are used for control and management tasks, the RBS should be able to handle imperfection in a native way. If so, it would be possible to have a unified agent model to use in the development of the different modules, whereas at the current state of the art several different implementations are required, ranging from CBR to FS to NNs. On the other hand, the complexity of the system and the final goal, which remains the development of a real-world EDSS and not a theoretical prototype, suggests the use of a mainstream rule engine instead of a custom solution. The reasons are, obviously, the necessity of a mature, standard-oriented product with a sufficient level of reliability and continuously maintained. Furthermore, mainstream products often provide corollary services and functionalities that could be exploited in the development instead of having to implement everything from start. With this goal in mind, the first step was take is a deeper analysis of the state of the art of the existing rule-oriented development tools, in order to find the most suitable one for the use in a real-world project. In particular, much of the data presented in this Chapter are derived from a recent survey, whose results were partially published in [6]. Other than drawing a picture of the current situation, the survey is particularly focused on how the different available products handle imperfection.

6.1 STATE OF THE ART

Nowadays, there exist few "pure" rule engines (with the notable exception of Prolog engines). In the second half of the XX century RBS have been widely used to build Expert Systems, mainly applied to diagnosis and interpretation problems [120]. Unfortunately, ES have been applied to domains where the traditional algorithmic approaches failed mostly due to their inherent *imperfection*, but early ES like MYCIN [69] had only a very limited (and sometimes with unclear or misused interpretations [93]) notion of imperfection which mined their performance. In the years, then, rule-based expert systems have have been applied less frequently as stand-alone entities to solve artificial intelligence problems. In fact, in the 90s there has been a decline in the number of canonical, rule-based expert systems, which have been replaced by hybrid expert systems. Roughly speaking, the hybridation has followed two courses, not necessarily mutually exclusive: the HC techniques used in the implementation have been integrated - if not completely replaced - by SC methods, whose robustness and flexibility has already been discussed in Chapter 3. For example, several *Neural* [157] or *Neuro-Fuzzy* expert systems have been developed. On the other hand, ES have found other fields of application in the industry, where they have undergone an integration process with enterprise information systems such as databases and data mining tools in distributed environments [259]. In this context, to this date the most popular applications seem to be:

RBS and Expert Systems

Modern RBS Applications

- Reasoning over Semantic Web ontologies enhanced with rules [36]
- Processing business rules in enterprise contexts [257]

The first application is not relevant in this thesis, so the discussion will be focused on the latter. Rules have been used to model the behaviour of information systems because of the flexibility, uniformity and consistency of the approach, together with the possibility to expand a knowledege base ([198]). Moreover, rules are easy to understand even for non technical people - in fact, they often model a domain expert's empirical knowledge - but at the same time they are written in a formal language that allows meta-reasonings to be performed over them. When complex processes are involved, the declarative approach becomes convenient as it separates the behaviour specification from the concrete implementation. To be actually usable in an enterprise project, where several actors with different backgrounds are likely involved, rule engines have evolved into more complex *Business Rule Management Systems*, which integrate the engine proper with additional features such as guided authoring tools, versioning management and remote repositories. Moreover, while BRMS have been traditionally used to orchestrate services in enterprise SOAs, the recent advent of CEP as a complementary paradigm has urged the engine developers to inte-

grate event-processing capabilities, in order to offer tools for the development of integrated architectures such as the one shown in [229].

6.2 A COMPARISON OF MAINSTREAM BRMS

This section will be dedicated to a comparison of the main existing BRMS. Given the vast number of tools, engines, shells, frameworks and similar, it is impossible to consider them all, so the discussion will necessarily be limited to mainstream projects, i.e. general purpose tools supported by a community (to whatever extent). This excludes several student projects and many ad-hoc engines built for specific applications, none of which are suitable to become general purpose building blocks in complex applications.

Notice also that, even if the support for imperfection is a most relevant feature, pure logical reasoners and other implementations of probabilistic and possibilistic frameworks which are not rule engines in the usual sense will not be taken into account because they are too narrowly focused. Other than a minimal support base, a tool must support at least one rule language with an expressiveness at least equal to that of propositional logic and possibly have at least one other relevant feature.

To this date, the mainstream BRMS include (but likely are not limited to) *InRule* [9], *ObjectConnections Common Knowledge* [16], *Microsoft BizTalk* [4], *Fair Isaac's Blaze Advisor* [10], *ILOG JRules* [14], *OpenRules* [17], *PegaSystems PegaRules* [18], *Open Lexicon* [15], *XpertRule KnowledgeBuilder* [25] and *JBoss Drools* [8].

6.2.1 *BRMS Features*

When evaluating a BRMS, the support for imperfect reasoning is a very important factor, but inevitably other aspects have to be taken into account. Many of the features of a BRMS are stressed by vendors themselves to promote their product. They include:

- **License** : Commercial vs Free/Open Source. Academic products are always open source, but several companies release a free version (possibly with some limitations) of their products.
- **Runtime** : Execution environment. Java and .NET are the most common to this date.
- **Imperfection** : Support for non-boolean logics, such as fuzzy and probabilistic logic. Several tools use *confidence factors*, an approximate (and sometimes non completely sound) degree of belief.
- **Rule Inheritance** : This feature, directly inspired by object-oriented programming, allows the reuse and specialization of existing rules, especially in contexts where the rules share common parts.

- **Forward/Backward Chaining** : Engines operating in FC! mode are data-driven and reactive, while BC! engines are goal-oriented. The native propagation method influences the engine efficiency in different tasks.
- **Workflows** : Support for Workflows, needed to model business processes.
- **Events** : Support for Complex Event Processing, which requires an explicit notion of time and the ability to use it in the inference process.
- **Editor** : Presence of a graphical authoring tool. This determines the effective usability of the tool by people without a programming background.
- **Repository** : Presence of a centralized rule repository. This feature is mostly relevant in enterprises, where collaborative work is used.
- **Language** : The language used for the authoring of rules can be proprietary or adhere to some standard such as RuleML.
- **Standard** : Even if an engine adopts a proprietary language, it can be compliant with some rule language standard. This depends on two factors : *(i)* the expressiveness of the two languages and *(ii)* the existence of a translator, embedded in the engine or provided as an external component.

6.2.2 *Results and Considerations*

The criteria defined in the previous section allow to compare several tools available on the market. The results, summarized in Table 11, is rather clear: full-featured BRMS are mostly commercial products. They are competitors in a market which, in 2008, was worthy about 285M$ and growing, to the point that a sensitive and highly valued market survey is performed every year [193]. From a general point of view, all the products offer comparable functionalities, so one can only speculate on the reasons behind the different market shares obtained by each company. They are likely to depend on canonical factors such as the specific implementation, compatibility with legacy systems, support and the advantage gained by different times-to-market. No open source tools, on the other hand, can provide a comparable variety of features, with the only exception of JBoss Drools and, to a lesser degree, Open Rules. Such tools, usually coming from the academia, are either discontinued or developed with a specific research goal in mind, so there is little interest (and, usually, limited resources) to build a complete tool.

Interoperability: the role of RuleML and other Standards

One of the limitations of the different engines is their using proprietary languages to write logic formulas and rules in particular. The lack of a common language standard could induce a strategical lock-in effect for the vendors, but is still a major limitation to the interaction of enterprises - or even divisions within the same enterprise - using different BRMS.

To achieve a good degree of interoperation, standards on rule representation and interchange are being proposed in the last few years. The *Rule Interchange Format* (RIF) [169] is a proposed W3C standard format for rule representation and interchange, based on XML.

The *Semantics of Business Vocabulary and Business Rules* (SBVR) [20] is instead a natural language-oriented standard, focused on the logical description of business contexts. It provides a standardized vocabulary of the entities involved and allows to write rules in a way which is natural for a non-technical reader and yet compliant with the rigor of a formal language.

RuleML [2] is an initiative which develops a XML- and RDF- based markup language for rules, with Datalog-rules as the core. RuleML uses a modular approach[1] to support different rule-based logics with different types of complexity and expressiveness, in order to promote rule interoperability between industry standards. The modular structure, going from a propositional-like logic where all terms are ground to a full-fledged FOL with support for both logical negation and NAF, plus equality and function evaluation is reported in Figure 25 RuleML supports various kinds of reasoning engines (e.g., forward vs backward chaining, RETE vs Prolog, ...) and leaves knowledge engineers the choice of implementation for entities and facts (e.g., objects, plain symbols, XML trees, ...). RuleML is supported by various rules engines, such as jDREW and Mandarax. A combination of the current standard ontology language OWL and RuleML is proposed to the W3C in form of the *Semantic Web Rule Language* (SWRL) [23].

RuleML

Nevertheless, while the proposed standards SBVR and RIF are still under development, RuleML is still used primarily for academic purposes, so in practice the degree of interoperability between engines remains limited to this date, even if it is expected to increase significantly in the next years.

Imperfection in BRMS

When it comes to the support for imperfection, however, the results are diametrically opposite - and hardly satisfactory. Fuzzy logic is perhaps the easiest type of imperfect logic to implement in a rule-based system: inference in that framework is a simple generalization of the boolean case. Most importantly, the operators are truth functional, i.e. they just aggregate the degrees associated to their operands, so the complexity

1 http://ruleml.org/modularization/

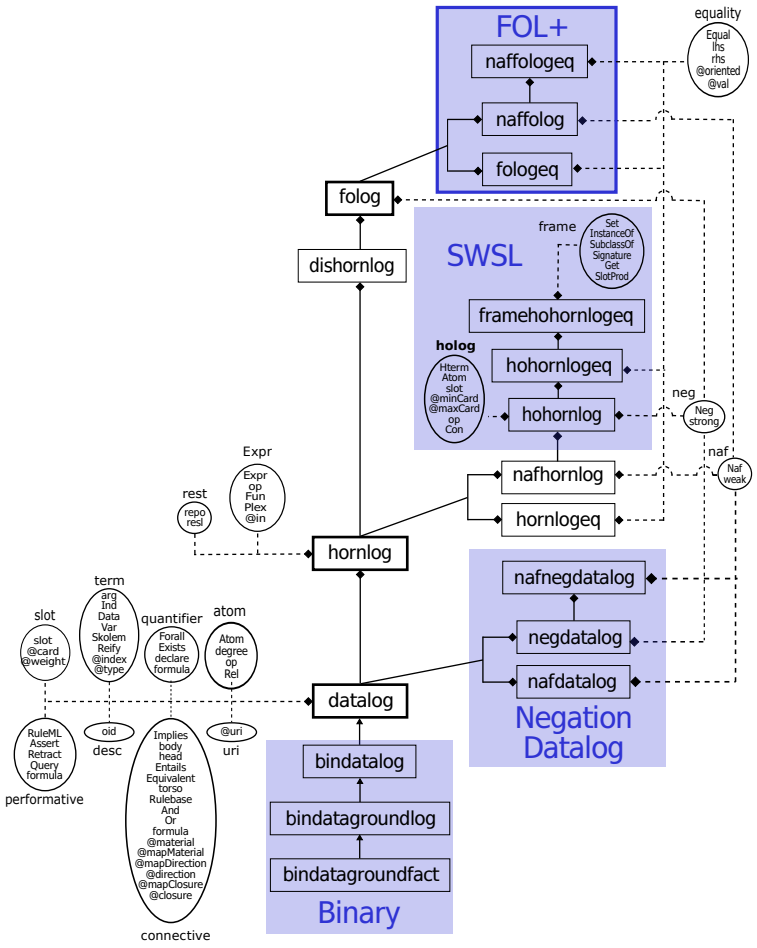

Figure 25: RuleML Modules Hierarchy (from [2])

remains limited. However, this still means that a standard rule engine can't evaluate fuzzy rules natively without an extension.

Many mainstream BRMSs, both commercial and free, usually embed a RETE-based algorithm: adding fuzzy logic (or any other type of logic) to any of these systems would require a refactoring of the internal rule engine and, possibly, the rule language, neither of which is a trivial operation.

Support for vagueness

The principale fuzzy-capable systems, instead, are open source rule shells, typically originated in an academic context, without many of the additional features of BRMSs. The most famous are possibly *FuzzyShell* [22], *FuzzyClips* [7] and *FuzzyJess* [12]; there is also a commercial data

mining tool, Scientio XMLMiner / MetaRule, which claims to have fuzzy capabilities [21], even if it was not possible to analyse it. FuzzyJess is one of the most used given its Java-oriented nature: it is actually a rewriting of FuzzyCLips, itself an extension of the CLIPS engine. FuzzyClips, moreover, has the merit of supporting two types of imperfection: fuzzy logic and confidence, in the form of certainty factors[2]. The first rule-based system to introduce uncertainty in automatic reasoning, *MYCIN* ([69]), adopted imperfect rules annotated with certainty factors to model a sort of quality score. The way of handling the factors was not theoretically very sound, so later systems used more structured approaches, even if the idea of using confidence was further developed (see for example [296]). In FuzzyClips, however, they have been introduced in a more coherent way, again truth-functional, and their evaluation proceeds in parallel with the evaluation of the fuzzy truth degrees of the formulas. Notice, however, that all these fuzzy shells support fuzzy logic in the broader sense of the term.

Support for Uncertainty

In contrast, no mainstream rule engine seems to support probabilistic logics. Whereas Bayesian Networks have become a very popular tool for handling uncertainty and many mature software packages exist which implement Bayesian networks, hardly any product already supports any of the various probabilistic logics, even if recently at least two projects have been started, namely *Balios* [166] and *BLOG* [202].

Imperfect Languages

Even if engines can hardly support imperfection, much more has been done from the language point of view. The issues related to the introduction of imperfection in rule languages have recently been discussed in [92]. In this work, an analysis of the different imperfect logic formalism is performed from the point of view of the notations required to fully express the semantics of each language. Given the different notions of imperfection, it is not surprising to find that different theoretical frameworks have been developed. Roughly speaking, probabilistic logics deal with uncertainty and thus are founded on probability theory; possibilistic and fuzzy logics are many-valued logics which handle vagueness natively; non-monotonic logics, instead, cope with inconsistency in the available knowledge.

The results given by [92] are remarkable since it shows that most types of imperfect logic - from probabilistic to possibilistic to many-valued logic - can be encoded simply by allowing predicates and connectives to be annotated using appropriate meta-data. Unsurprisingly, in presence of imperfection predicates are no longer boolean, but their evaluation returns a generalized degree, according to the possible semantics given in Chapter 2. Likewise, it is necessary to extend the concept and the modalities of combination for these degrees, which becomes more complicated, so the behaviour of the operands implementing the logic connectives has to be redefined. In [92] this is done

2 see Chapter 8

using the meta-attributes `@degree` and `@kind`, respectively, to define the desired imperfect semantics of a logic formula. These extensions proposed are compatible with RuleML: in fact, they were ideally designed to be included in an additional module, but as of RuleML version 0.91, only a partial integration with the core Datalog language has been completed (in fact, to this date, the RuleML research initiative has been inactive for some time). However, the same extensions are also compatible with a preliminary version of the W3C Rule Interchange Format (RIF) [169], while a different approach, even if limited to fuzzy logic alone, has led to the fuzzy extension of SWRL, *f-SWRL* [226].

6.3 DROOLS

Unlike many other commercial BRMS, Drools [8] is an open source project maintained by a growing community of developers who expand its functionalities and test it continuously. Excluding commercial and merely academic projects, after the dismissal of CLIPS and Jess it is possibly the only mainstream "living" engine of some importance to this date, so it deserves a more in-depth analysis. Initially born a RETE-based rule engine, it is now more properly a *Business Logic Integration Platform* which provides a unified and integrated platform for Rules, workflows and Event Processing, relying on an integrated rule Repository for cooperative work. Its modular architecture allows to use all the functionalities or just a limited part, according to the user's needs.

Drools 5.0

The characteristics described here are referred to version 5.0, but will likely be extended and improved in the following versions.

6.3.1 *Drools Expert*

Rule Engine

The rule engine is the core component of Drools, based on an object-oriented version of the RETE algorithm (see Chapter 3). The *facts* are implemented using POJOs, but the classes can be declared or extended directly in the knowledge base. As with most RETE-based engines, it

Reasoning Modes

adopts a *reactive*, forward-chaining propagation policy, even if it supports simple queries (backward chaining will likely be included in the newer releases). It can work both in *stateful* and *stateless* mode: while the former corresponds to the canonical modality, the latter discards the facts after evaluating and firing the relative rules. In stateless mode, the engine is properly reactive: the output depends on the current inputs alone, so several optimizations can be applied to increase the performance, whereas the stateful mode allows to implement more complex reasoning patterns at the cost of an increased latency.

Rule Language Expressiveness

The rules themselves are written in a proprietary language, called DRL, which will be analysed in greater detail in Chapter 7. It offers almost full support for first-order predicate logic (with a few differences outlined in the relative section), but also includes "hybrid" con-

structs such as the `accumulate`[3] which cross the border between logic-declarative and procedural semantics. The engine, in fact, is more application than pure logic oriented. First of all, the consequence of a rule can include any type of action, be it logical, procedural or functional, essentially allowing the invocation of any well-formed code block (written in a Java dialect, in particular). More interestingly, the engine also supports *custom logic evaluators*, any module with an `Object` × `Object` ↦ `boolean` interface capable of evaluating a unary or binary logic predicate for a given set of arguments. This feature, while not much publicly documented, will turn out to be essential in the integration of the rule engine with other, non symbolic techniques.

Supported Languages

Although DRL is the main language, there exists several alternative "syntactic sugar" forms:

- Domain Specific Language (DSL) : the rule language is encapsulated within templates expressed using an intermediate form between the native DRL and the natural language. These masks facilitate the comprehension of the rules by non-technicians, but still can be parsed by the compiler.
- Business Rule Language (BRL) : a web-based editor allows to author rules using a graphically enhanced interface
- Drools XML : there exists a (deprecated) XML-oriented version of the rule language. Unfortunately, this encoding is not compatible with other standard such as RuleML.
- Decision Tables (DT) : Decision tables are a compact way of defining complex decisional branches [245], easy to create even for people lacking proper programming skills. The parser, compatible with most common word processors and spreadsheets, can translate a table into a set of rules.

Drools Expert
Object-oriented RETE engine
Forward Chaining with Queries
Run-time type declaration/extension
Proprietary Language with Syntactic Sugar
First-Order Logic Support
Declarative and Procedural Side Effects
Custom Evaluators

Table 7: Drools Expert features

6.3.2 *Drools Fusion*

Event Engine

This module adds the support for CEP: RETE networks are optimized

3 http://www.jboss.org/drools/documentation.html

specifically for the many-to-many, pattern matching needed by Luckham's agents [186] and Drools' language is expressive enough to implement the required logical constraints.

Event types are defined in the knowledge base, as new classes or tagging legacy types using meta-data: each event has a timestamp - which is either set when the object is inserted in the working memory or read from a designated field of the object - and a duration, set to 0 unless otherwise specified. The timestamp and duration are used automatically by a set of temporal evaluators which are an implementation of Allen's interval operators [34].

Cloud vs Stream Mode

The engine can be configured in two modalities: the *cloud* mode, where it acts as a classical rule engine with temporal predicates, and the *stream* mode, where it uses a session clock to be explicitly aware of the passing of time. Events in stream mode are inserted sequentially; moreover, the declared duration is used to retract an event automatically when *(i)* it is expired and *(ii)* the system has determined that it can no longer contribute to the activation of any rule. This sort of automatic "garbage collection" is used to improve the response time of the engine, which may be critical when processing events, by preventing unnecessary joins in the α and β nodes. To further improve the overall performances, it is possible to define multiple *entry-points*, so that events can be channeled through different paths in the network.

Drools Fusion
Explicit Event Support
Temporal Reasoning (using Allen's interval algebra)
Session Clock
Automatic Expiration
Multiple Stream Entry-points

Table 8: Drools Fusion features

6.3.3 *Drools Flow*

Workflow Engine

Drools Flow introduces the support for *rule flows*, processes which describe the order in which a series of steps need to be executed, using a flow chart. A workflow is a procedural model of a *Business Process* (as opposed to a declarative model, a concept studied in great detail in [205]), consisting in a collection of nodes that are linked to each other using connections. Each of the nodes represents one step in the overall process while the connections specify how to transition from one node to the other. The flow is deeply integrated with rules: first, a process step can be defined in terms of rules to be evaluated and executed; then, rules can be used to evaluate the branching conditions during the process execution. Drools Flow uses a peculiar architecture: the flow engine does not hold the control and call the rule engine on request.

Flows-Agenda Interactions

Rather, objects inserted in the working memory are evaluated preventively, leading to rule activations which are placed in the agenda; the activations, then, are fired in an order controlled by the flow engine. The two components execute in parallel, using the agenda to synchronize each other. All the transitions are logged, so a limited form of process auditing and reporting is included in the bundle. The execution flow can also be influenced by asynchronous events (exploiting the Fusion module of the rule engine) and include human-performed tasks. Moreover, the engine can be extended to include custom process nodes: in fact the goal of the project is to provide an editor and an engine compatible with different business process specification standards, like WS-BPEL [24], OSWorkflow[4], jPDL [11] and BPMN [6].

Drools Flow
Process Workflows
Rule Integration
Integration of Events and Human Tasks
Domain Specific Processes
Automatic Auditing
Support for BPMN 2.0 (under development)
Transactional and Persistent

Table 9: Drools Flow features

6.3.4 *Drools Guvnor*

Rule Repository

Guvnor is Drools' business rule repository, implemented as a web application deployable on any application server and accessible through any HTTP-enabled browser. It allows to store rules and the related resources, organizing them in packages and taking care of the versioning of each item. Every package can be considered portion of a remote, shared *knowledge base*. The resources may be uploaded from different sources: in that case, the repository sees that they are properly merged and checks the coherence of the resulting package. Guvnor also supports `KnowledgeAgents`, dedicated agents which can monitor the repository, download a package and update it automatically whenever it changes.

Runtime Package Integration

6.4 CONCLUSIONS

In the last years, rule engines have lost some of their appeal as AI tools to soft computing techniques, but have found a new, important role in the management of business processes. The integration of RBS, CEP and workflows under a unified tool facilitates the development of complex

4 http://www.opensymphony.com/

Drools Guvnor
Runtime Rule-Base Update
Remote Rule and Resource Repository
Web-based administration interface
Graphical Editors for Remote Authoring

Table 10: Drools Guvnor features

enterprise applications: in fact, the development of BLIPs is mostly carried out by dedicated companies, capable of providing the expected assistance and life-cycle product management, instead of academic research groups. The noteworthy exception is given by Drools, which, due to its open source and community-driven nature, remains a competitive option even when compared to commercial products, while it can still be used for research purposes.

The advanced reasoning engine, together with the support for workflows and events, make Drools the ideal candidate platform for the development of hybrid intelligent agents, actors of a complex management system in the sense of Chapter 5. Despite all its features, Drools has two main limitations: it uses a proprietary language and does not support any kind of imperfect rules. This last drawback is not just a limitation of Drools: from a market analysis, it turns out that while there exists countless "toy" engines, most mainstream engines do not support imperfection at all, or limit themselves to fuzzy logic, whereas a user interested in dealing with uncertainty has to resort to a BN package. However, if it were possible to extend an engine to embed imperfection, the horizon of its possible applications would expand greatly. The next Chapters, then, will be focused on the analysis of the integration of imperfection in logic, with the goal of identifying the extensions necessary for a RETE network to reason with imperfect data. Afterwards, a Drools-based implementation will be discussed.

Table 11: Mainstream BRMS Features

	Vendor	Runtime	Imperfection	Inheritance	FC/BC	Flows	BRMS	CEP	GUI	Repo	Language	Compliance
Open Source/Free Systems												
Balios	Academic	Java+Sicstus	Bayesian CP	No	F+B	No	No	No	Yes	No	BLP	
DR-Device	Academic	C++	Defeasible	No	F	No	No	No	IDE	No	CLIPS/XML	OO-RuleML
Drools	Jboss	Java	Yes (Chance)	Yes	F (B?)	Yes	Yes	Yes	IDE	Yes	DRL	No
FRIL	Academic	Shell	Fuzzy + CI	No	B	No	No	No	No	No	Prolog-like	
Fuzzy + CLIPS	Academic	C	Fuzzy + CF	No	F	No	No	No	Shell	No	Custom	
Fuzzy + Jess	Academic	Java	Fuzzy + CF	No	F	No	No	No	Shell	No	Custom	
FuzzyShell	Academic	Custom C++	Fuzzy	No	F	No	No	No	Shell	No	OPS-like	
Hammurapi Rules	Hammurapi	Java	No	Yes	F (+B)	No	No	No	Yes	No	Java	
jDrew (OO)	Academic	Java	No		F+B				Yes		Prolog-like	RuleML/SWRL
Mandarax	Academic	Java	No		B				Yes		Prolog-like	RuleML
Open Lexicon	Open Lexicon	Java	No	No	F	Yes	Yes	No	Yes	Yes	Custom	
OpenRules	Openrules	Java + Xcel	No	No	F		Yes		Yes	Yes	XLS	
Prova	Academic	Java or custom	No	No	F+B					Yes	Prolog-like	Almost all
SweetRules	Cooperation	Java	Defeasible								Many	RuleML +
Take	Academic	Java	No	No	B				Yes			RuleML
ViDre	Academic	Web Service	Jess embedded							Yes		RuleML
Commercial Systems												
BizTalk	Microsoft	.NET	no (3rd part?)		F	Yes	Yes		Yes	Yes	XML	
Blaze Advisor	FICO (Fair Isaac)	Java/.NET	3-valued	Yes	F	Yes	Yes	Yes	Yes	Yes	SRL	RuleML
Common Knowledge	ObjectConnections	Java/.NET	No	Yes	F	Yes	Yes		Yes			
ILOG Jrules	ILOG	Java	No	?	F	Yes	Yes	No	Yes	Yes	BAL	
InRule	InRule Tech	.NET	No		F	Yes	Yes		Yes	Yes	MS DRL	
Oracle Rules	Oracle	Java	No	No	F	Yes	Yes	Yes	Yes	Yes	Java-like	
PegaRules	PegaSystems	Java	No	Yes	F+B	Yes	Yes		Visio	Yes		
XMLMiner + Metarule	Scientio		Fuzzy + CF	No	F	Groups		No	Yes	Yes	GUI	XML
KnowledgeBuilder	XpertRule	Win	fuzzy	Yes	F	Yes		No	Yes	Yes	Hybrid	

7

ENHANCING A RULE-BASED SYSTEM WITH IMPERFECTION

Contents

After the analysis of the existing rule languages and engines, a rather clear picture emerges. Modern BRMS have several interesting features which will prove to be useful, first and foremost the support for event processing, which makes them appealing candidates to become core building blocks in the development of "intelligent" rule-based agents. On the other hand, the imperfect nature of the information to be processed would better be treated using non-classical logic or soft-computing algorithms, which the engines embedded in mainstream BRMS do not support. The only option - other than renouncing to the benefits of their enterprise features - is then to add support for non-boolean logic to an existing rule engine, using the only one which has a sufficient level of maturity while still being open source and thus extensible (not to mention the convenience in case of application in concrete projects). So, from now on, all considerations will be applied to this specific rule engine.

Drools has a set of features comparable to that of commercial BRMS, but like the other it relies on a boolean rule engine, so it is essentially a *perfect* information processing system. In [92] it has been shown that several types of imperfect extensions of classical logic exist, and that

they can be derived from a common language by annotation with the appropriate meta-data. Ideally, it would be useful to attain the same level of customization in the rule engine, in order to be able to process a mixed rule base composed by various types of imperfect rules. It is true that Drools Expert is expressive enough to reason *over* imperfection: the consequence of a rule can analyse the degree of imperfection of the tuple which activated it and process according to the appropriate theory, but it would be much more elegant to reason *with* imperfection directly. The declaration of a pattern in a rule premise, then, would define the reactive behaviour of the engine in an ideal situation, when the available information is perfect, but the engine would try to adhere to that even when in presence of imperfect data, approximating the reasoning and degrading gracefully as the quality of the information worsens. The degree of matching between a tuple and a pattern should be taken into account automatically: the evaluators should assess the degree of imperfection associated to each constraint, then the individual degrees should be combined according to the logical connectives used in a formula, following the generalized inference pattern which will be discussed in Section 7.2. Eventually, each rule activation should be tagged by a degree - be it of truth, belief, confidence, depending on the context - which should be available in the conclusion part, to be propagated or used to condition the execution of any side effect. According to Drools philosophy, these additional features would be wrapped in a package, as shown in Table 12.

Drools Chance
Reasoning *with* different types of Imperfection
Configurable Imperfect Evaluators
Integration with Sub-Symbolic Algorithms
Full support for Logical Connectives
Granular configurability
Support for Confidence and Exceptions
Support for Induction and Abduction

Table 12: Drools Chance features

This Chapter will discuss the issues related to the development of this Drools extension. After contextualizing the execution features of engine, a generalized inference schema will be proposed in Section 7.2, to encompass different types of rules to emulate different inference processes compatible with different types of imperfect logic. Section 7.3 will show the additions to Drools' DRL language, necessary to support the generalized inference. Finally, Section 7.4 and 7.5 will discuss the relevant modifications to the underlying RETE engine, both from a theoretical and from an implementation point of view. The contents of this Chapter and the following one are an extended version of what was published in [5].

7.1 REACTION RULES

When using Drools, it is important to remember that it is a RETE-based, forward chaining production rule engine. The `if/then` rules used in (pure) logical reasoning are general constructs based on implications, which are typically used to derive *logical consequences*, i.e. the entailed formulas, from an existing knowledge base. The production rules used in production rule engines, instead, have a more behavioural connotation: a production rule states the *actions* that must be performed when the state of the world matches the given preconditions. In fact, an `if/do` formulation would better express their semantics. RETE-based engines can be considered *(weakly) reactive* production rule systems, since their rules are triggered - but not necessarily fired - by the insertion of new facts in their working memory: Drools stresses this aspect with its rule syntax, which uses a `when/then` notation. When the facts are notified in real-time and the rules fire immediately, the engine becomes properly event-oriented and thus *fully* reactive. In this case, the rules have an "event-condition-action" (ECA) semantics, which can be expressed using an `on/if/do` notation. Notice that ECA rules can be used to emulate general production rules [232] and production rules without side effects can implement forward-chaining logical reasoning.

Reactive and ECA rules

REACTION RULEML Production, reactive and ECA rules are part of the standardization iniative of the RuleML group, thanks to a dedicated sub-project called *Reaction RuleML* [26]. It extends the original RuleML hornlogeq sublanguage, adding the support for various kinds of production, action, reaction, and KR temporal/event/action logic rules as well as (complex) event/action messages [228]. It allows to tag the formulas with meta-data, which could theoretically be used for degrees, but unfortunately its evolution has proceeded in parallel with that of fuzzy RuleML, so there is no full integration between the two. Moreover, the standard language is not completely aligned with Drools rule language due to some limitations of the latter, fact which remains one of the greatest limitations to the interoperability and applicability of Drools. Nevertheless, the priority was given to the integration of Drools with fuzzy RuleML rather than with reaction RuleML, even if this second research direction has been planned as well.

7.2 GENERALIZING THE INFERENCE PROCESS

While the standardization of languages is an important result, even at the cost of some expressive power, is practical usefulness is greatly diminished by the absence of an engine capable of processing rules written in that language. In fact, the necessity for a unified imperfect reasoning framework has been advocated by Zadeh himself in a recent work [308].

The first step towards this goal is then the unification of the inference process. It is true that in probabilistic and possibilistic logics the inference process must be coherent with respect to the degree of *belief* on the actual (but ill-known) state of truth of a formula, which can only be true or false, while many valued logics deal with well-known sentences whose truth degree is gradual, but it is also true that many of these logics rely on some kind of deductive mechanism for inference. The canonical deductive inference rule is modus ponens, but the definition given in Formula 3.1 is not suitable since it does not take imperfection into account. A generalized MP expression is:

Generalized, Imperfect Modus Ponens

$$\frac{\langle P(\mathbf{x}), \varepsilon(\mathbf{x})\rangle, \forall \mathbf{X} : \langle P(\mathbf{X}) \to C(\mathbf{Y}(\mathbf{X})), \varepsilon(\to)\rangle}{\langle C(\mathbf{y}(\mathbf{x})), \varepsilon_{\Rightarrow}(\mathbf{x}, \varepsilon(\mathbf{x}), \varepsilon(\to))\rangle} \tag{7.1}$$

where the form $P \to C$ is used instead of the more generic $\varphi \to \gamma$ to stress the role of premise and conclusion. Notice that, unlike the MP of fuzzy logic, which is strictly truth functional, this form also takes the arguments into account for the computation of the degrees.

The inference proceeds in parallel along two paths: the symbolic level and the degree level. The first determines the arguments of the conclusion, while the second evaluates the associated degree of imperfection. In general, this evaluation is not truth-functional, so the output symbols may depend on the input ones, and the resulting degree may depend on both, in addition to the input degrees. An inference process using MP, then, requires the following steps (which are also a further generalization of the inference shown for possibilistic logic in [117]):

- **Evaluation** : First, atomic predicates are evaluated. Each of them returns a belief/truth/possiblity/probability/... degree, according to its semantics. The evaluation takes into account the imperfection associated to the arguments X:

$$\frac{\langle P(\ldots, A_j(x)/\varepsilon_j, \ldots), \langle P(X) \to C(Y)\rangle}{C(y)}$$

- **Aggregation** : As logical connectives aggregate atoms in complex formulas, the operator associated to each connective computes its degree, combining the degrees of its operands. In case of non truth-functional operators, this operation may require to evaluate the arguments of the operands, as well.

$$\frac{\langle \Phi(\ldots, A_j(x)/\varepsilon_j, \ldots)/\varepsilon_P, \langle P(X) \to C(Y)\rangle}{C(y)}$$

- **Projection** (proper) : The premise, taken individually, is a formula with an associated degree: the connection to the consequence is established through the (generalized) implication. The MP operator, instead, computes the actual consequence. In general, it joins the arguments of the premise with those of the implication and projects them onto the conclusion. Likewise, the conclusion degree is a combination of the premise and conclusion's

degrees, possibly conditioned by the value of the arguments (as dictated by 7.1):

$$\frac{\langle P(x)/\varepsilon_P \ , \ \rightarrow_{(X,Y)} \ \varepsilon_{\rightarrow}\rangle}{C(y)/\varepsilon_C}$$

- **Merging** : When multiple rules entail the same conclusion, the individual degrees have to be merged. In the boolean case this is not necessary, since an entailed conclusion is always certainly true. In presence of imperfection, each rule may be considered the source a piece of evidence supporting the overall conclusion. Hence, the different degrees have to be combined appropriately.

$$\frac{\frac{\langle P_1, \rightarrow_1\rangle}{C_1/\varepsilon_{C_1}}, \ldots, \frac{\langle P_n, \rightarrow_n\rangle}{C_n/\varepsilon_{C_n}}}{C(y)/\varepsilon_C}$$

This abstract schema is generic enough to accommodate several purely logical and even hybrid inference schemas. Being a generalization of MP, it clearly fits all truth-functional logics and even probabilistic logics, provided that a strong independence assumption is made, so that only conditionally independent predicates are combined. The possibility of expressing conditional probabilities $p(A|B)$ using generalized implications could be used with Bayesian-like rules: in fact, the possibility of using generic, non truth-functional operators offers several options. Despite the many examples which will be shown in Chapter 8, many others still haven't been explored yet.

The Role of $\rightarrow$

Nevertheless, the proposed inference schema stresses the role of the degree associated to the implication, which is not just limited to computing the arguments of the conclusion. In fact, while in most standard rule based systems an implication is always assumed to be constant[1] for *all* inputs, here it can have a varying degree which possibly can depend on the arguments being processed.

The Role of $\cap$

The novel component, with respect to classical logic, is effectively the final merge operation, which has many conceptual affinities with (and effectively abstracts) the Dempster-Shafer's combination rule already cited in Chapter 2. This sequence of operations defines the guideline for the reasoning steps of a configurable inference engine. Since none of the engines analysed in Chapter 6 is compliant with it, and implementing it *on top* of one of them would have led to cumbersome rules, it was preferred to implement it directly *in* Drools, enhancing its core rule engine.

7.3 LANGUAGE EXTENSIONS

Drools, like its predecessors Clips [7] and Jess [12], uses a language which is influenced by the language of the first RETE-based expert

1 In many cases, including perfect ones, it is *true*

Listing 7.1: Generic Drools' rule

```
rule "Generic"
  attribute "value"
when
//DRL syntactic features
  $pattern : Type( $var : field == "value"
            && $joinVar : intField < 0 || > 1)
  exists AnotherType( joinField != $joinVar )
then
  System.out.println("Java side effects");
  insert(new Fact());
end
```

sytem shell, OPS [68]. Drools, however, is object-oriented and fully compatible with Java, in addition to including support for a scripting oriented Java dialect, MVEL[2]. This section is dedicated to the analysis of the features of this language, showing some relevant changes made to increase its expressiveness. These extensions are general-purpose, but it will be shown that they become even more relevant when imperfection is involved.

7.3.1 *Drools DRL*

DRL glossary

The full specification of Drools' language can be found in [8], so the syntax will not be discussed here, but it is nevertheless necessary to recall the basic terminology in order to avoid ambiguities in the rest of the work. A rule such as Rule 7.1 is composed by a LHS (the premise) and a RHS (the conclusion), plus some meta-data in the form of attributes. In the RHS, any Java code can appear, but the keywords `insert`, `retract` and `update` are reserved for the manipulation of the facts in the working memory. The LHS, instead, is composed by a *conjunction* of *conditional elements* (CEs). The most common types of CEs are *quantified* CEs and *patterns*[3]. A pattern is formed by a type (the simple name of a Java class) and zero or more *constraints*, possibly combined using the conjunction (&& or ,) and disjunction (||) logical connectives. A constraint is formed by a field, possibly *bound* to a variable, and an optional *restriction*. A restriction, composed by an *evaluator* and an optional *value*, can be simple or complex: the latter, in this case, is a conjunction and/or disjunction of restrictions.

DRL Features

From the point of view of a formal logical language, DRL has the following properties:

- **Constants**: All primitive (Java) types are supported.

2 http://mvel.codehaus.org/

3 Other CEs exist, but have a less logical and more functional nature

- **Variables**: Both objects matching a pattern and field values can be bound to variables. A variable can be used in a constraint, but must be ground by the time it is evaluated.
- **Functions**: Functions are allowed in restrictions: they are always interpreted and the returned value is used by the local evaluator.

  For example, in `Rectangle( height > (sqrt(area)) )` the evaluator $>$ extracts the value of a `Rectangle`'s field `height` and compares it to the square root of the rectangle's area.
- **Equality**: The equality evaluator `==` delegates to the method `equals()` of the object referenced by the field being tested. If the method is undefined, equality is evaluated by identity.
- **Atoms**: Constraints are the atomic "predicates" of DRL: the evaluators implement the unary or binary relations between the value of a field and the value(s) it is restricted to - in fact they are *test-score* functions in the sense of Zadeh [308]. However, DRL allows the use of a "syntactic sugar" which complicates things from the grammatical point of view. The complex restrictions allow to list multiple restrictions for a single field, without having to repeat multiple times. Likewise, the pattern type specification - which is equivalent to the constraint `this.class == Type.class` - groups the constraints relative to the same pattern, but at the same time creates a division between the patterns and constraints. In fact, it is possible to build complex formulas at three different levels of (syntactic) abstraction, with the role of atoms taken in turn by CEs (e.g. `Type1(...) and Type2(...)`), constraints (e.g. `Type( field1 == 1 && field2 == 2)`) or restrictions (e.g. `Type( field ( != 1 && != 2))`).
- **Connectives**: The only connectives are the logical conjunction `&&` and the logical disjunction `||`. While the two can be nested arbitrarily between constraints or restrictions, CE-level formulas are treated differently: they are manipulated until the formula is in disjunctive normal form. At this point, an individual rule is created for each alternative: thus, the LHS of a rule is always a conjunction of CEs.
- **Quantifiers**: The standard $\forall$ (`forall`) and $\exists$ (`exists`) are supported. Moreover, DRL includes the quantifier `not`, whose semantics is properly that of $\nexists$, thus implementing a kind of negation-as-failure.
- **Negation**: The logical negation `not` is supported only in restrictions, where it negates an evaluator. When used with CEs, `not` has the already cited semantics of NAF.

7.3.2 *Drools Syntax Extension*

The main limitation of Drools language is the lack of full support for logical connectives, negation above all. Moreover, the results in [92]

suggest that it should be possible to annotate the rules with meta-data to configure and customize their semantics. These reasons have motivated a direct intervention on Drools' DRL grammar, with the goal of expanding its expressiveness while ensuring *full* backward compatibility.

Drools' grammar is complex. The language is not regular [83] and, as will be shown shortly hereafter, is *self-embedded* at several levels. Moreover, it does not even belong to the class of LL(k) languages for any k: the common prefix in the structure of constraints and restrictions makes it impossible to parse the language using only a fixed lookahead. Consider, for example, two patterns with a common constraint, defined using an arbitrarily long sequence of restrictions:

```
Person( name != "A" && != "B" && ... && != "Z")
Person( name != "A" && != "B" && ... && age > 18)
```

The first ends with a restriction, but the second has a second constraint: the two cases become distinguishable only after the parser has met the evaluator ! = (resp. the field identifier `age`), i.e. after skipping an arbitrarily long number of symbols. Nevertheless, Drools uses the open-source tool ANTLR[4] to generate the parser and the lexer for its language. ANTLR is designed for LL(k) languages: in fact, with the only exception of the case cited above, DRL is at worst LL(3). The original parser, then, is LL(k)-compliant, but its behavior is overridden locally, implementing the adaptive lookahead where necessary.

The necessary language extensions have been implemented in this context: several minor modifications and additions have been performed, but the most relevant have involved attributes and connectives. The grammar shown here is actually a simplified version, deprived of many purely implementation details. The real grammar is publicly available in the Drools code repository.

- **Negation**. Logical negation is unary: its introduction has allowed to support the more general notion of unary operator, which are useful, for example, to implement the linguistic hedges used in fuzzy logic, such as `very`, `more-or-less` and similar. The syntax is :

```
<unary_op> ::= ( 'neg' | 'very' | ... ) <attribs>?
```

- **Connectives**. In addition to negation, complete support for the other canonical logical connectives has been added: implication, conjunction, disjunction and exclusive disjunction/equivalence. The connectives can be used to write complex formulas, but the role of "atom" can be taken by CEs, constraints and restrictions. Drools supports both prefix and infix connectives:

4 www.antlr.org

```
<prefix_f> ::= '(' <conn> <formula>* ')'
<infix_f>  ::=  <formula> ( <conn> <infix_f> )*
```

The former is allowed only between conditional elements, while the latter can be used with CEs, constraints and restrictions alike. Moreover, connectives can be specified using both a symbolic and a declarative form (see Table 13), which has been provided for the new connectives as well.

Explicit	Implicit
`implies`	`=>`
`or`	`||`
`xor`	`^^`
`equiv`	`^=`
`and`	`&&`

Table 13: DRL alternative connective forms

Each connective can be negated, if necessary, and customized using attributes, according to the following pattern:

```
<implication> ::= <or> <infix_impl>?
<infix_impl>  ::= <unary_op> <infix_impl>
                | IMPL_CONN <attribs>? <or>

<or>          ::= <xor> <infix_or>?
<infix_or>    ::= <unary_op> <infix_or>
                | OR_CONN <attribs>? <xor>

<xor>         ::= <and> <infix_xor>?
<infix_xor>   ::= <unary_op> <infix_xor>
                | XOR_CONN <attribs>? <and>

<and>         ::= <atom> <infix_and>?
<infix_and>   ::= <unary_op> <infix_and>
                | AND_CONN <attribs>? <atom>

<atom>        ::= <atom_cond_elem>
                (* resp. <atom_constr>
                   resp. <atom_restr> *)
                | '(' <implication> ')'
```

This abstract grammar fragment describes the infix form only, but is a pattern valid for all three levels of abstraction. From

the same productions, the priority assigned to the connectives is evident and coincides with the order they are listed in Table 13.

- **Attributes**. The role of attributes such as `@kind` and `@degree` has already been discussed in Section 6.2.2. Actually, it turns out that more attributes can become useful to control the behaviour of the engine. The adopted syntax is as similar as possible to the one already used for meta-data:

```
<attribs> ::= '@' '[' <attrib> (',' <attrib>)* ']'
<attrib>  ::= 'kind' | 'degree' | ...
```

  Attributes will be discussed in detail in a later section.

- **Custom Restrictions**. In order to customize the behaviour of evaluators as well as that of operators, it was necessary to alter the grammatical definition of restriction:

```
<atom_restr> ::= <unary_op> <atom_restr>
               | <core_restr>
<core_restr> ::= <evaluator> <attribs>? <expr>
               | '(' <implication_restr> ')'
```

  In a way similar to what has been done for connectives, the grammar allows to place one or more unary operators before the evaluator, and to decorate it with attributes afterwards. Notice that restrictions are one of the possible classes of "atoms" to use in complex, embeddable formulas, so an atomic restriction can be defined in term of a (bracketed) implicative restriction, according to the general specifications.

Notice that the resulting grammar is not LL(k), but neither was the original with which it is compatible, so this was not considered an issue.

7.3.3 *Imperfect Rule Structure*

Abstract Syntax Tree

After being lexed and parsed by the ANTLR-generated tools, the enhanced syntax tree is further processed and manipulated, again using ANTLR, to obtain the *abstract* syntax tree (AST). The LHS of each rule is converted into a tree where intermediate nodes model operators and quantifiers, while leaf nodes correspond to atomic predicates. In order to discuss the new structure of the AST, consider the following examples:

AST EXAMPLE I : PATTERN CONDITIONAL ELEMENTS Patterns are fundamental constructs in the definition of rules. With respect to the standard Drools version, the AST section has been rebuilt

in a cleaner form. Consider the rule excerpt 7.2 and its corresponding AST (Figure 26.

Listing 7.2: Syntax Example I

```
when
  Person( ... )  // a Person
then
```

The pattern CE is split in a conjunction of three formulas, fully restoring the logical, predicate-oriented nature of the AST. The root ∧ has a relevance which will become even more obvious in the next Sections: in fact it is the *pattern root*. There exists a pattern root for each pattern CE, whose children are derived directly from the rule. The type check is converted in a proper constraint, `this.class = Person.class`, and appended to the pattern root; the same happens to the pattern constraints sub-tree. The last, `holds`, is a completely novel constraint, whose semantics depends on the context. In general, its purpose is to associate a degree to an object as a whole. The insertion of an object in the WM is equivalent to acknowledging the object's existence in the model of the world the engine has. The "existence" of the object - i.e. its presence in the working memory - can however be imperfect: an agent may not be certain about it, so that a degree of belief is required. This degree should *not* be assigned to the type constraint (nor, obviously, to the field constraints): in fact, the uncertainty is not about the object being an instance of a certain class, but about the object itself in the first place. Likewise, an `enabled` constraint returning a fuzzy truth degree would mean that an object has been "partially" put in the WM (not trivially, setting this degree to false would correspond to a virtual retraction). Notice that the cases of partial class match, such as `Ellipses` trying to match `Circles`, *would* still be modelled by having the class constraint return a fuzzy truth degree.

Pattern Root Node

Holds Constraint

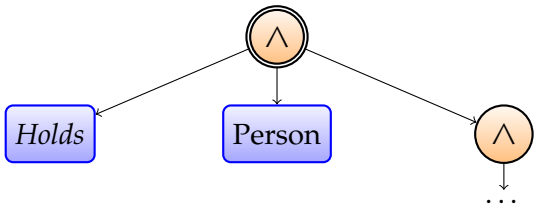

Figure 26: AST Example I

AST EXAMPLE II : INTRA-CE FORMULAS The extended grammar allows to create arbitrary formulas within patterns, using all the canonical connectives including the logical negation, as in example 7.3. Notice the difference between the constraint-level formula, where the exclusive or is the local root, and the restriction-level formula.

Listing 7.3: Syntax Example II

```
when
// A Person
  Person(
    // who is adult
    age > 18 ^^
    // or whose name is neither X nor Y
    (name == "X" neg || == "Y")
    // but not both!
  )
then
```

Extended Evaluators

Like in the case of patterns, the logical structure of the constraints is made explicit, splitting each constraint in two parts and joining them using a *conditional* conjunctive connective. The second is the constraint's restriction - or the logical composition thereof, but the former is a novel extension, at least for Drools. In presence of imperfection, the evaluators in the restrictions act as generalized test-score functions [308], returning a degree which, as usual, can be a degree of truth, belief, probability, ... according to the desired semantics. However, imperfection may affect the input of the function as well as its evaluation (see also Chapter 2): so, the first part of the constraint checks whether the extracted value is the *actual* value of an object's feature. In standard OO-RETE, the input value is extracted from an object's field, which is always certain and precise, but this is not always the case[5]. Consider, for example, a Person whose age is assumed to be 25 with degree of belief 25%: even a boolean restriction such as > 18 must necessarily reflect this uncertainty in its output.

AST EXAMPLE III : INTER-CE FORMULAS The third case study 7.4 shows that now conditional elements can be nested arbitrarily, using connectives *between* them - in the specific case, an implication.

Listing 7.4: Syntax Example III

```
when
  $p : Person( age > 18 )
  implies
  Person( this == $p, weight > 50)
then
```

The resulting AST is shown in Figure 28. Notice that the formula corresponding to the premise is not used directly, but becomes the left

5 Notice that even in "perfect" systems often fields are set to `null` because of lack of knowledge

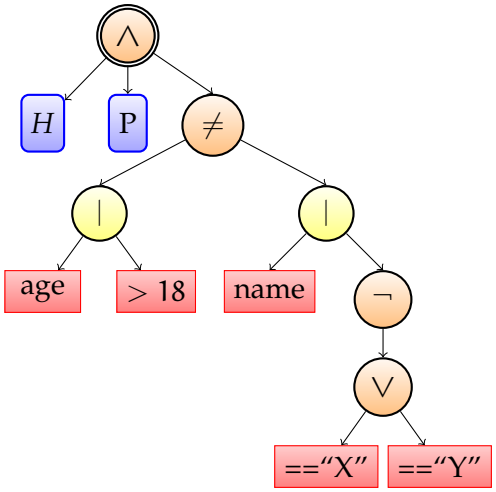

Figure 27: AST Example II

Implication and Modus Ponens

argument of a "modus ponens" connective. The other argument is an implication: in standard rule-based systems, the very act of writing a rule implicitly sets it to *certainly true* for all possible input arguments. This assumption is too strict when imperfection is involved: the importance of an explicit premise-conclusion implication, already proven in [92], will further be stressed in Chapter 8. Roughly speaking, it allows to control the way the degree associated to the premise is projected onto the conclusion. (Other than that, the considerations made for the previous examples apply to the internal constraints).

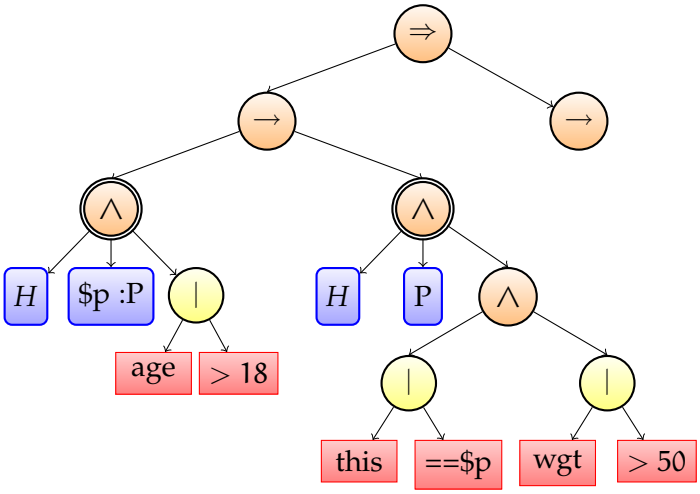

Figure 28: AST Example III

7.4 RETE ENHANCEMENTS

The extension of the language is only the first step in expanding the functionalities of Drools. The new DRL, nevertheless, has a greater level of expressiveness which makes it more similar to the standards such as RuleML, so the interventions on the engine become even more relevant in the perspective of a future full integration.

7.4.1 *Network Construction*

The original RETE algorithm ([134], Chapter 3) compiles the AST of each rule into a propagation network, sharing nodes whenever possible. Due to the different structure of the intermediate representation, however, the compilation algorithm must be modified accordingly. The AST predicate nodes are mapped onto α or β RETE nodes, depending on the predicate involving one or two objects. For reasons of compatibility and efficiency, the mapping considers the *constraint* nodes to be leaves, so any sub-trees below a conditional conjunctions is embedded directly in a single node. This is not a strict requirement and may change in a future version. More importantly, the original RETE does not consider logical connectives since, trivially, only `and` is used. As will be discussed in more detail in Chapter 8, the evaluation of a connective may require a complex operator, especially when an imperfect, non truth-functional logic is used. Thus, connective nodes are compiled in operator nodes, placed in the appropriate position within the network.

RETE Construction : AST Ordering

The compilation proceeds in two steps. First, an ordering of the AST nodes is computed: each node n is labelled with an incremental index k according to the sequence they are visited in during a post-order traversal of the tree. In this type of visit, all the children of a node are recursively visited, from left to right, before the node itself is visited. This ensures that whenever $n(k')$ depends on $n(k'')$ - either because the latter is a descendant of the former or because $n(k'')$ is a pattern root node and $n(k')$ holds a join constraint with its pattern - then $k' > k''$. For example, applying the labelling algorithm to the full parsing of rule 7.3 returns the numeration given in Figure 29.

RETE Construction: Node Deployment

Given the ordering, it is possible to deploy the augmented RETE using Algorithm 1. Each rule can be analyzed individually: given the sequence $C[i] = k | n(k)$ is the i-th pattern node, and defined $k^* = \max\{C\}$ the index of the last pattern node, the nodes with $k \leqslant k^*$ will become part of the α-network (unless they are join constraint nodes), while the others will be included in the β-network. Let also $F[i] = k$ such that $n(k)$ is the first child of $n(C[i])$ (if $n(C[i])$ has no children, take $F[i] = C[i]$). The nodes in the range $F[i] \dots C[i]$ are connected sequentially, starting from $n(F[i])$ (a class constraint node). The last, $n(C[i])$ (a pattern node), is connected to an α-memory, α_i. However, if any of the nodes holds a join constraint, it is skipped. The memory α_i is then connected to the right input of a join node Γ_i. The output

Algorithm 1: RETE network Construction

```
Require: Nodes<Id,Node> {Node map}
Require: C {Ids of Pattern Nodes}
Require: F {Ids of Class Nodes}
Require: join, alfa, beta {Generic Nodes}
  N ← length(C)
  curNode ← RETE.entryPoint
  for i = 1 to N do
    patternNode ← C[i]
    classNode ← F[i]
    for all j between F[i] and C[i] do
      if ! Nodes[j].isJoinConstraint then
        attach(curNode,Nodes[j]) {curNode is shifted}
      end if
    end for
    attach(curNode,alfa(i))
    join(i).setLeft(beta(i − 1))
    attach(curNode,join(i))
    for all j < F[i + 1] do
      if ! Nodes[j].isAttached then
        attach(curNode,Nodes[j])
      end if
    end for
    attach(curNode,beta(i))
  end for
```

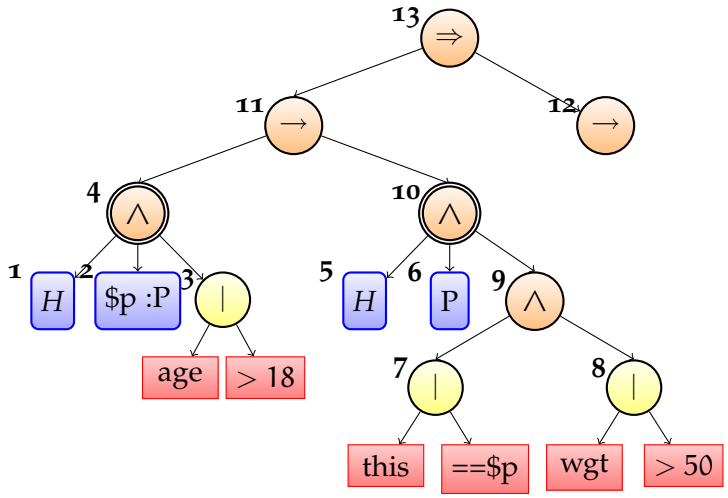

Figure 29: AST Node numbering

of Γ_i is connected to all free nodes with index less than C[i], to verify whether the joined candidate tuples are actually valid or not. Then, all free nodes with index greater than C[i], but less than F[i + 1], are connected, since they are typically operator nodes whose operands are all valid and so can be evaluated. The last node is then connected to a β-memory, β_i, where the tuple computed so far can be stored. The left input of Γ_i, instead, is connected to the output of the memory β_{i-1}, or to a dummy input in case of the first node. The nodes are created as needed or reused if already existing and correctly sequenced, as for the standard RETE algorithm. Notice, instead, that the chain of join nodes does not terminate as usual with a terminal node: a dedicated node evaluates the degree of implication between the premise and the conclusion, conditioned on the actual premise tuple (in many cases it simply returns a constant or even "*true*"); eventually, the Modus Ponens node computes the conclusion degree: only at this point the tuple is active and ready to be dispatched to the agenda, from where its firing will be scheduled. Figure 30 shows the result of the compilation.

7.4.2 *Run-time Evaluation*

The RETE network holds the *long-term* knowledge of the system in a compiled form. Once the RETE network has been built, it is used to compute the degree of match between *tuples* of objects, which are inserted in the WM at runtime, and the premises of the different rules. For each rule, this degree can be used to condition the execution of the consequences, provided that it has been modified to take into account the degree of implication between the premise (the LHS) and the conclusion (the RHS).

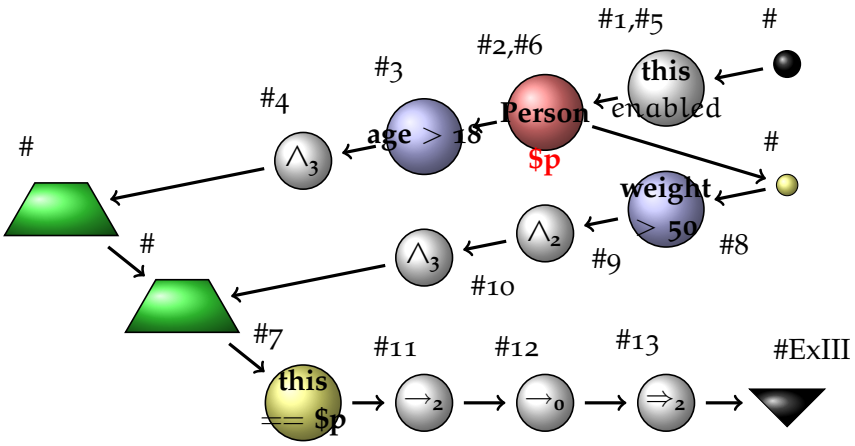

Figure 30: Extended RETE Example

This *consequence* degree, then, is the result of a recursive procedure which implements the abstract schema described in Section 7.2. The recursion is properly due to the logical operators, while the elementary steps are the evaluations of the degrees to which each constraint in the LHS of a rule is *satisfied* by an object or a tuple[6]. *Evaluation*

A constraint's satisfaction degree, in turn, is obtained by composition of the degrees resulting from the evaluation of the restrictions forming the constraint itself. Thus, the (logical) *evaluation* is the fundamental operation: it involves the equivalent of predicates and connectives, in Drools implemented by restrictions and operators. Moreover, when an evaluation is truth-functional, so that it limits to a combination of other degrees, it can be more properly called degree *aggregation*. *Aggregation* It must be remembered that, at least in Drools, more than one atomic evaluation can take place in the same RETE node. While the novel operator nodes execute only one evaluation (which is often an aggregation), α and β nodes can hold complex constraints formed by several restrictions. In fact, the difference between α and β nodes depends only on the number of objects effectively involved, but not on the complexity of the constraint they are built for.

This is one of the facts that complicates the structure of the constraint nodes: in the original algorithm, they had a simple "test-and-filter" role, where the test involved was a *perfect* symbolic comparison. A tuple, then, was propagated along the chain of nodes if and only if the constraint was satisfied. In RETE-OO, where objects are involved in place of pure symbols, a restriction may involve a more complex test than a simple comparison: in the simplest case, a comparison requires the extraction of a value from a field and the evaluation of an expression. Drools Expert adds complex restrictions, so boolean aggregations *Composite Restrictions*

6 For simplicity, objects will be considered (right) tuples composed by a single element, so the difference will be ignored from now on, unless relevant

have to be performed locally. However, the real increase in complexity is caused by the use of imperfection. Properties are no longer true *exclusive-or* false, and finding to which degree can be a non trivial operation, involving different sources of information; moreover, it is not always clear when to propagate a tuple. The simple *discard-on-false* policy is not applicable when the degrees allow to model infinitely more possible truth states - *if* a degree models a truth state at all rather than a belief. Thus, the language and semantics extension do not reflect only on the topological structure of the network, but also on the internal architecture of the nodes. Both aspects will be covered.

Imperfect Evaluation

The nodes are responsible for the *evaluation* of atomic (restrictions) of complex (logical operators) checks. The two types of evaluations share many similarities, so many of the considerations discussed here apply to both and differences will be outlined only when necessary. Denoting by σ the predicate functor, by $\mathbf{L}$ the class of degrees used in the rule, and by $\mathbf{x}$ the current set of arguments, an evaluation consists in finding the degree ε associated to the predicate-like relation $\pi = \sigma(\mathbf{x})$. σ can either be a restriction or an operator: the actual difference is given by the arguments, which are objects for restrictions, degrees for truth-functional operators and a combination thereof for generic operators. Moreover, from the point of view of evaluations, even quantifiers can be treated like connectives with a variable number of arguments, so the same considerations apply.

Evaluation Methods

The evaluation of the degree of a generic "predicate" π are can be performed using different approaches:

1. (**Factual**) ε_0^π : Prior information, available before the computation and provided as a fact.
2. (**Functional**) ε_σ^π : Direct evaluation, resulting from the *evaluator* associated to a restriction or the *operator* associated to a connective.
3. (**Logical**) ε_i^π : Logical entailment by one or more rules $r_{i \in I}$, with I indexing the set of rules providing information on the restriction as one of their consequences.

A-priori Evaluation

The enhanced DRL allows to assign a *prior* degree to an evaluation with a syntax similar to the one shown in Rule 7.6, exploiting the equivalent of the `@degree` attribute proposed for fuzzy-RuleML [92].

Listing 7.5: Restriction with prior degree

```
when
  Type( field custom @[ degree="..."] value )
then
```

This value is *static* and *universal*, so it is used every time the restriction is computed, independently of the actual tuple being checked. Even if presented with a prioristic semantics which could seem specific for probabilistic reasoning, its possible uses include, but are not limited to: fixing the result of an evaluation to a constant, setting a prior probability to be updated in a Bayesian sense or limiting the degree of possibility of a constraint.

Priors can be applied to evaluators and operators: it is remarkable that the second class also includes implications and, most importantly, the implication associated to a rule. This allows to model rules where the connection between premise and conclusion is not perfect, examples of which will be shown in Chapter 8. Since the rule implication (and the rule modus ponens, as well) are not visible, a minor language addition has been performed, exploiting the existing rule-level attributes:

Listing 7.6: Implication and MP with prior degrees

```
rule "Imperfect"
  entailment  @[ degree="..." , ... ]  // =>
  implication @[ degree="..." , ... ]  // ->
when
```

(Custom) Direct Evaluation

The second approach to the evaluation of a property requires to pass its arguments to a function, which in the specific case is embedded at the core of evaluators and operators.

As far as evaluators are concerned, Drools offers all the possible basic comparators (==,! =,>,>=,...), as well as more advanced evaluators such as `contains` and even `soundsLike` (which compares two strings according to their phonetical representation, despite its not being fuzzy), but most importantly it allows to define and plug in *custom evaluators*.

Custom Evaluators

Custom imperfect evaluators implement the test-score functions proposed by Zadeh [308], returning a degree which can be as simple as a real value or complex like a type-n fuzzy set (see also Chapter 2). The role of custom evaluator can be played by any entity implementing, directly or through an adapter, a specific Java interface[7], as indicated by the class diagram in Figure 31.

To use a custom evaluator it is sufficient to provide a *factory* class and an implementation class, in addition to registering its identifier (i.e. the string used in the rules) in the engine configuration. The support for custom evaluators is essential to build *hybrid* symbolic-connectionist

Hybrid Custom Evaluators

7 Again, only an idealized version is presented since to date the APIs are not stable and subject to change

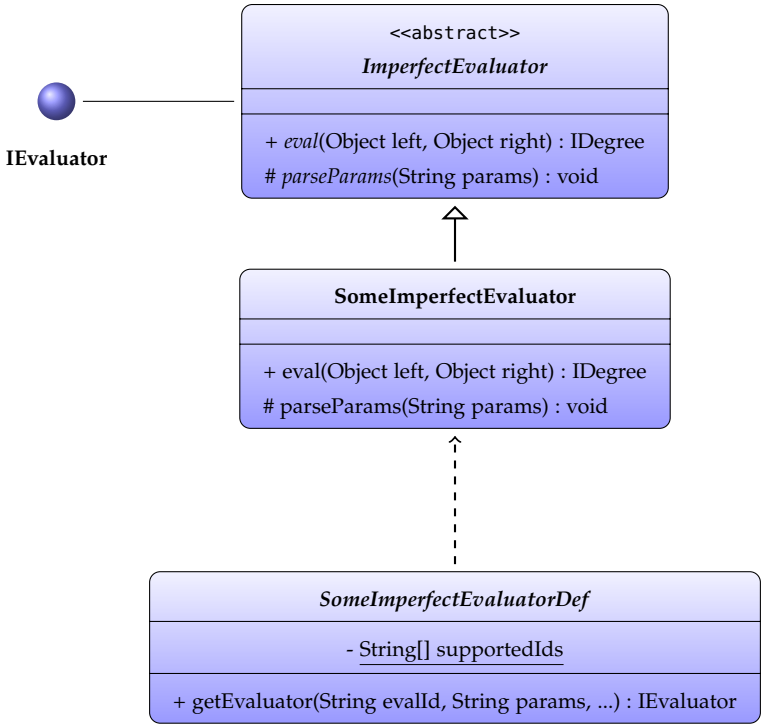

Figure 31: Imperfect Evaluator Hierarchy (excerpt)

systems: this interface is an elegant way to *embed* any module capable of evaluating a property of a given pair of objects in the LHS of a rule, whereas the only other deprecable possibility would have been the explicit invocation of an external object in the RHS of some other rule. Notice that most soft computing techniques described in Chapter 3, like Neural Networks and Bayesian Networks, are suitable candidates for the role of external evaluator. Moreover, the enhanced rule language allows a further degree of customization using attributes: the attribute `@kind`, in this context, can easily be applied to select the actual implementing class; another attribute, `@params`, has instead been defined to pass an additional initialization String to the constructor of the evaluator.

Listing 7.7: Custom Evaluator with Attributes

```
rule "CustomEvals"
when
  Type( field custom
        @[ kind="...", params="..." ]
        value )
then
```

Obviously, the same considerations can be applied to operators: in fact, the same attributes were suggested to select and customize the behavior of operators in the first place [92]. Notice that the proposed language extension allows to apply attributes to connectives at different levels. As already shown, the implicit $\rightarrow$ and $\Rightarrow$ (implication and modus ponens) operators can be exposed using the appropriate rule attributes.

Listing 7.8: Custom Operators with Attributes

```
rule "CustomOps"
  implication @[ kind="..." ]
  entailment  @[ kind="..." ]
when
  Type1( f1 == "X"
         && @[ kind="..." ]
         f2 == "Y" || @[ kind="..."] == "Z" )
  and @[ kind="..." ]
  Type2( ) @[ kind="..." ]
then
```

Moreover, notice that even *patterns* can be decorated using attributes (e.g. `Type2` in example 7.8): the attribute set is actually assigned and used to configure the implicit *pattern root* `and` connective.

Rule Chaining

Being a production system, Drools allows to change the contents of the WM at run-time. In particular, the actions executed when a rule fires may include:

- `insert` : a new fact is inserted in the WM, activating all rules with a pattern matching the object.
- `retract` : an existing WME is removed from the WM
- `update` : an existing WME is updated, re-evaluating all the rules which potentially match the object.

Logical Chaining

This allows to chain the execution of rules by *fact*, since a fact generated by the firing of one may activate another: what Drools does *not* allow, instead, is the *logical* chaining. Suppose, for example, that a rule uses the pattern `Person( age > 18)` to check whether a person is adult or not. In a perfect world, the age of all `Persons` in the WM is known, but in a realistic context it is possible that the age of one or more `Persons` is not known (e.g. the corresponding field is set to `null`). However, a person who drives a car is surely of age, so it would be useful to encode this knowledge in a rule and use it to support the evaluation of the constraint "`age > 18`", as in example 7.9.

Listing 7.9: Logical Chaining Example

```
rule "Adult"
when
  $p: Person( age > @[ id="idAdult"] 18 )
then
  // e.g. activate vote procedure
end

rule "Driver"
when
  $p: Person( )
  exists Car( driver == $p )
then
  inject($p,"idAdult")
end
```

To this end, an attribute and two new action keywords have been added. The attribute `@id` can be used on any evaluator or operator to attach an explicit identifier to the corresponding constraint/formula (if the structure is shared between different rules, it is sufficient to do it once). Notice that the name of a rule is the identifier of the associated modus ponens operator. The new consequence actions, instead, require two arguments: a tuple and an identifier. The semantics is as follows:

- `inject(tuple,id)` : adds the consequence degree of the rule it is called by to the set of degrees used to compute the overall degree of the evaluation identified by `id`.
- `reject(tuple,id)` : like inject, but applies the default logical negation to the consequence degree before the injection.

"Overriding" injections

The method `inject` is overloaded to give an additional option: different rules may be used to inject the same evaluation, but some of them may be more relevant (in some context-dependent sense) than others. This, in turn, should give more weight to their contributions in determining the overall degree for the injected evaluation. Hence, the `inject` method may be used with an additional boolean argument which distinguishes between "normal" degrees and "high-priority" ones. This additional feature has been introduced to support, even if in a limited form, rules with *exceptions*. Its use will be discussed in a specific example in Chapter 8, where it will also be discussed how to overcome the apparent limitation of having only two levels of priority. The `@overriding` attribute, instead, can be used to increase the relevance of an evaluator/operator directly, so that it will be given more relevance than injected or prior contributions.

Merging Different Sources

In conclusion, it turns out that the result of a given evaluation may be influenced by multiple sources at the same time. To reconcile the different contributions and obtain a single degree, a *merge* function $\cap : \mathbf{L}^{2+|I|} \mapsto \mathbf{L}$ is needed, such that: *Merge Strategy*

$$\varepsilon^{\pi} = \bigcap_{i \in \{0,\sigma\} \cup I} \varepsilon_i^{\pi} \tag{7.2}$$

No property other than closure is strictly required on $\cap$, even if commutativity and associativity can be useful. Notice that in a consistent boolean rule base, the definition of $\cap$ is normally redundant, and thus omitted, because all sources supporting a predicate just return `true`.

Relations with TMS

However, rule engines which rely on a *Truth Maintenance (Sub)System* (TMS) [271] use techniques that have many conceptual similarities and thus can be associated to the merge of information sources. Truth Maintenance, in a nutshell, is a feature of non-monotonic reasoning systems which ensure that the consistency of a knowledge base is maintained as its contents change in time. The basic functionality a TMS must provide is to remove (*retract*) all the logical consequences of a premise which no longer holds, possibly because it has been retracted itself. In particular, since a conclusion may be supported by different premises, the retraction must take place when there exist *no more* premises supporting a given conclusion. The simplest way to implement this, also used in Drools, is reference counting: every logically entailed fact in the WM has a reference to the rule activations which support it. This is actually a primitive form of degree merge strategy: suppose that the presence of an object in the WM is modelled by the truth of the constraint `holds`, evaluated for each object. For a logically entailed fact, the constraint is neither given a prior value, nor evaluated directly, but instead is injected by the supporting rules' activations. The standard behaviour of a basic TMS, then, can be emulated using a merge strategy which returns true if and only if there exists at least one true injecting activation, and false otherwise. Currently, the use of the `holds` constraint in an (imperfect) TMS is still under investigation.

Nevertheless, the default merge strategy used in a rule base can be overridden for any individual evaluation using the dedicated attribute `@merge`, as in:

Listing 7.10: Custom Merge Strategy

```
...
Type1( field == @[ kind="...", degree="...",
                   merge="...", missing="...",
                       override="..." ] "val" )
...
```

A merge strategy must assume that all degrees are known and have the same importance: in practice, neither may be the case, so two steps of pre-processing may be required.

Merging missing values

For each evaluation, the set of possible information sources is known a priori, but at runtime it is not guaranteed that all of them will effectively give their contribution. The prior degree may not have been specified; a functional evaluator may not be able to return a meaningful degree, for example because some of the required data are missing; eventually, not all injecting rules may have fired by the time the evaluation is performed - this last condition, in particular, is always satisfied only if strong assumptions are made on the structure of the rule base and the object insertion order. A sufficient condition is that a partial ordering can be imposed on the rules, such that it is impossible for a rule of level n to be activated unless all rules of level $n-1$ have already fired, but this requirement is hardly satisfied by a generic set of rules. In practice, this means that a degree will be determined, on average, by a number of *actual* contributions which is less than the number of *potential* contributions. By default, these "missing" values are simply ignored, but it is possible to override this behaviour using the attribute `@missing`: in practice, it allows to set a function which returns a concrete degree (whatever it means, according to the adopted semantics) in place of a missing one.

Merging values with priority

When prioritized injections and evaluators are used, instead, the degrees have to be modified. The task is demanded to a *discounting* function, which actually makes the lower-priority degrees less relevant, ideally replacing them with the neutral element of the merge function. The specific strategy can be configured using the attributed `@override`.

Filtering and Propagation

The contemporary presence of logical injections and insertions increases the reasoning capabilities of the engine, but unfortunately complicates the propagation of the tuples through the network. When an inserted object or tuple is propagated, it passes through the α, β and operator nodes: the evaluators/operators in each node perform the local, direct evaluation, but the contribution of priors and injecting rules may affect the final result, possibly even after the first passage of the object. In fact, the activation order of rules can't be predicted, especially if they depend on facts coming from external sources. Sometimes, rules could even be the only source of information for the evaluation of a constraint, so some form of synchronization between insertions and injections must be provided [145]: in such cases, in fact, the decision on whether to propagate or not should be postponed until at least one injecting rule has fired.

Propagation Policies

For this reason, the RETE nodes have been extended, adding a configurable filtering strategy which applies the conditions according to which one of the following policies is adopted:

- *Pass*: The object/tuple is forwarded.

- *Hold*: The object/tuple is held within the node (waiting for a possible injection).
- *Drop*: The object/tuple is discarded.

The filter strategy is truth-functional[8]: it checks the value of the degree associated to the object/tuple and its *support* $|\gamma|$, defined as the number of non-null partial degrees over the total number of possible degrees, which is known and equal to $|I| + 2$.

Filtering Criteria

Remarkably, the degree tested by the filtering strategy is not necessarily the degree resulting from the evaluation local to the node itself, but instead the *overall* degree so far. As objects/tuples traverse the network, every evaluation produces a degree which is wrapped in a structure called `Eval` (see Section 7.5.1 for details) and incrementally added to a stack-like structure associated to the tuple. Simple restrictions push their `Eval`s on the stack, while operators arrange the `Eval`s into a tree structure by popping a number of `Eval`s equal to their arity from the stack, aggregating them into a composite `Eval` and eventually pushing it back on the top of the stack. At any moment, the `Eval` on top of the stack is assumed to hold the current overall degree, since it is either the result of the last simple evaluation or the root of the `Eval` tree. In fact, the `Eval` tree mimics the AST of the premise of the rule as it is constructed evaluating node after node. The top of the stack, however, is not always the local `Eval` because of β constraints, which are moved into beta nodes placed *after* the join nodes in the beta network: their restrictions, in fact, can be evaluated only after a tuple has been formed, but the tree assembling process starts earlier in the alpha network. Thus, the `Eval` leaves relative to β constraints are momentarily set to $\emptyset$ ("missing" or "unknown"). Some operator nodes in the α network, then, may have to apply their filtering strategy to the root - created locally - of an incomplete `Eval` tree. The β nodes, instead, *add* their `Eval` as a leaf to the current tree, but apply their filtering strategy to its root. The evolution of the `Eval` stack for a pair of objects joined into a tuple is shown in Figure 32. Notice that considering `Eval` to be missing may be ininfluential on the overall result: for example, a conjunction with missing operands will still remain false as soon as just one of the known ones is false.

Eval Stack

Nevertheless, a node always applies the filter strategy to the `Eval` on the top of the stack, regardless of whether it was generated by the node itself or not. The default, perfect filtering strategy simply chooses `Pass` when the `Eval`'s degree is equal to true, while it opts for `Drop` otherwise. When, due to imperfection, the `Hold` policy is chosen, the tuple remains blocked at the node, in a memory called Γ-memory (for analogy with α- and β- memories). The nodes are designed using the *Observer* pattern [122], so that they can *observe* the main `Eval` and re-evaluate the filtering policy when its degree changes due to an (expected) injection.

Early injection

Notice that an injection may also happen after the tuple has been

Late injection

8 the node itself imposes a constraint on the objects!

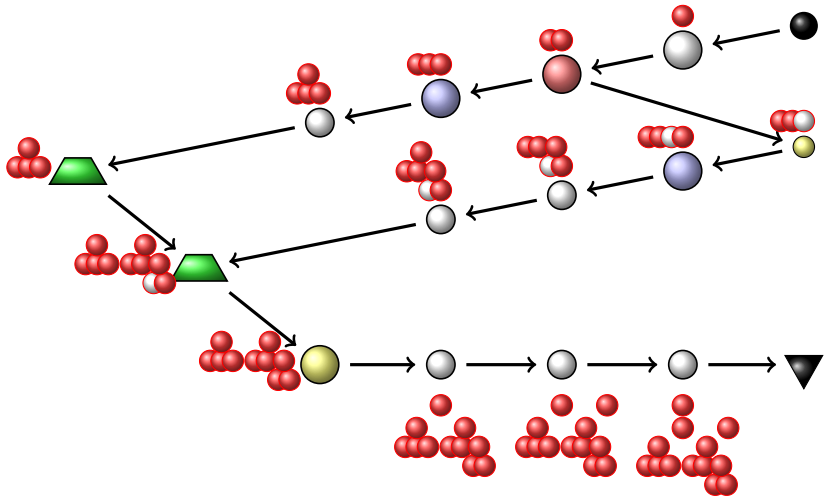

Figure 32: Eval Tree Construction

`Passed` through a constraint node, possibly even after it has already caused a rule to be activated: in this case, the the`Eval` tree is observed by the *Agenda*. The injected value is still added to the corresponding `Eval`, propagating the changes through the `Eval` tree up to the root. For clarity, the synchronization protocol between nodes and `Eval`s is outlined in algorithms 2, 3 and 4. When a tuple is inserted in a node, the relative `Eval` is retrieved (if an early injection has already built it) or built locally; this, in turn, is used to update the `Eval` tree and the filtering strategy is applied. An injection, instead, causes the target tuple to be retrieved from the Γ-memory (if held) or an `Eval` to be created and stored to be retrieved later. In the former case, the `Eval` tree is updated and, should the overall degree change, the node is notified so it can re-apply the filter policy.

Like with all other policies, the actual implementation of the filter strategy can be customized using attributes. In particular, three attributes have been defined which can influence the final decision on the propagation of a tuple or an object:

- `@filter`: Its value selects the desired filter strategy. It can also be used as a rule attribute, applying it to all nodes created by the compilation of that rule. A few examples of policies could be:
  - *Full synchronization*: `propagate` on $|\gamma| = 1$, `hold` otherwise
  - *Closed World Assumption*: `propagate` on `true`, `discard` otherwise.
  - *Open World Assumption*: `propagate` when not `false`, `discard` otherwise.
- `@boolean`: The degree returned by the tagged evaluation is effectively converted to a `boolean` before being used. The actual cast modality is not defined by the rule, but by the degree: in fact,

Algorithm 2: Node.onInsert(Tuple t)

Require: filter {S_f strategy}
  $\gamma \leftarrow \Gamma$mem.get(t)
  if $\gamma = \varnothing$ **then**
    $\gamma \leftarrow$ createEval(t,this)
    γ.set(ε_σ,this.eval(t))
  end if
  t.evalTree.add(γ)
  $[\varepsilon, |\gamma|] \leftarrow$ t.evalTree.eval() {highest ranking degree}
  if filter.decide($\varepsilon, |\gamma|$) = 'PASS' **then**
    this.remove(t)
    propagate(t)
  else if filter.decide($\varepsilon, |\gamma|$) = 'HOLD' **then**
    this.store(t)
    γ.attach(this)
  else if filter.decide($\varepsilon, |\gamma|$) = 'DROP' **then**
    γ.destroy()
    t.destroy()
  end if

Algorithm 3: Gamma.onInject(Node n, Tuple t, Rule r, Degree ε)

  $\gamma \leftarrow$ n.Γmem.get(t)
  if $\gamma = \varnothing$ **then**
    $\gamma \leftarrow$ createEval(t,n)
  end if
  γ.set(r,ε)
  γ.notify() {notifies n on cascade}

Algorithm 4: Node.onNotify(Tuple t, Degree ε, Degree $|\gamma|$)

  if this.holds(t) **then**
    if filter.decide($\varepsilon, |\gamma|$) = ...) **then**
      ... {see Procedure 2}
    end if
  end if

Listing 7.11: Generic Imperfect Rule - Example

```
rule "CustomPropagation"
when
  $x : Type( field1 == @[ filter == "..." ] "X"
             ^^
             field2 == @[ filter == "..." ] "Y" )
  Type( this == @[ cut ] $x,
        field3 == @[ boolean ] "C")
then
```

it would be more accurate to say that this attribute forces an evaluation to return either the representation of true or false according to the currently used degree model. For example, should interval values be used, the evaluation would have to return exclusively $[0,0]$ or $[1,1]$. The use of this attribute has the effect of ignoring the imperfection in the evaluation.

- `@cut`: This attribute forces the *strict* filter strategy locally: the degree is temporarily cast to boolean and, if false, the object/tuple is discarded. Unlike `@boolean`, however, the original imperfect degree is propagated.

Consider, for example, the rule of example 7.11. The exclusive or in the first pattern advocates the choice of a filter strategy which propagates objects even if they *violate* a constraint. The join in the second pattern, instead, is configured so that it allows only pairs composed by two equal objects. The imperfection in the last constraint, eventually, is removed for some reason.

7.4.3 *Summary*

In the end, it turns out that while a generalized language needs just the addition of two attributes such as `@kind` and `@degree` [92], a real implementation has a larger number of degrees of freedom which require a larger number of configuration parameters to be controlled. The goal of this brief section is to collect and summarize them for quick reference.

The configuration parameters are listed and summarized in Table 14; the different attributes are instead described in Table 15.

This allows to write rules such as the one in example 7.12, which should be compared with the one in example 7.1. The attributes allow to configure almost all operators and evaluators in the rule, including the implication and modus ponens normally considered implicit. This, together with the possibility of using custom evaluators and operators, should greatly increase the potentialities of the engine.

Table 14: Drools Chance Configuration Options

Parameter	Description
L	Partially ordered set of degrees. Its inf is **0** (certainly false) and its sup is **1** (certainly true), and may include a special value modelling complete ignorance ?. It must be possible to cast a degree into a boolean.
$\{\sigma\}$	Set of custom evaluators, implementing a test-score function which returns a value in **L**
Ω	Set of close and coherent operators. May be truth functional. Includes:
$\Rightarrow$	Modus Ponens Inference Rule, used for deduction.
$\mathbf{S}_\cap$	Degree combination rule. Combines several degrees into a single one.
$\mathbf{S}_\varnothing$	Missing degrees handling policy. Replaces missing degrees with meaningful ones or forces the combination rule to ignore them.
$\mathbf{S}_\mathbf{k}$	Discount function. Used to lower the weight of less relevant sources when merging degrees.
$\mathbf{S}_\mathbf{f}$	Propagation and filtering strategy. Controls the propagation of tuples along the network.

7.5 IMPLEMENTATION NOTES

The various enhancements have been implemented in a prototype which can be downloaded freely from the SVN repository[9]. The structure is generally consistent with the theory discussed in the previous Section, but involves lots of additional implementation details which have been omitted. A few strategic choices, however, are important enough to deserve a quick introduction. First of all, the `Eval` structures used to hold, combine and propagate the degrees will be analysed in greater detail; second, the main component responsible for the instantiation of the modules necessary to extend the network and its nodes - the `Factory` - will be illustrated; last, a complexity analysis of the modified engine will be carried out.

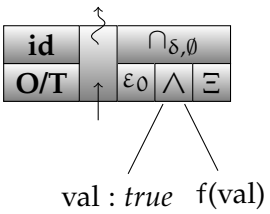

Figure 33: *Evaluator* `Eval`

9 http://anonsvn.jboss.org/repos/labs/labs/jbossrules/branches/DroolsChance/

Table 15: Drools Chance - DRL attributes

Attribute	Description
	Identification
`id`	Used to identify an evaluation univocally
	Type Selection
`kind`	Used to select an evaluator or an operator's type
`params`	Used to pass additional parameters for the creation of an evaluator or an operator
	Priors
`degree`	Used to set a prior degree for an evaluation
	Merging
`merge`	Used to choose the degree fusion strategy $\mathbf{S}_\cap$
`missing`	Used to choose the missing value strategy $\mathbf{S}_\varnothing$
`override`	Used to choose the discount strategy $\mathbf{S_k}$
`overriding`	Used to increase the relevance of the result of a direct evaluation
	Filtering
`filter`	Used to choose the propagation/filtering strategy $\mathbf{S_k}$
`cut`	Used to impose the strict boolean, "drop-on-false" filter strategy
`boolean`	Used to force the cast of an evaluation's result into a boolean

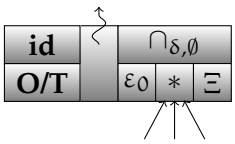

Figure 34: *Operator* `Eval`

7.5.1 *Eval Trees*

In order to facilitate the propagation as well as the combination of degrees, all the necessary information is stored in a dedicated structure, called `Eval`, for quick reference. An `Eval` can be considered the dual of a `Tuple`, but it holds degrees instead of objects.

Evals

Formally, an `Eval` like the one schematized in Figure 33 stores, manages and combines the different contributions to the evaluation of a constraint (resp. operator) σ for a tuple of objects $\mathbf{x}$. In order to build an Eval, it is necessary to know the set of rules I which may include the injection of σ among their consequences. Its fields are:

- The identifier of the evaluation
- A reference to the tuple/object $\mathbf{x}$

Listing 7.12: Generic Imperfect Rule - Example

```
rule "Imperfect"
  implication @[ ... ]
  entailment  @[ ... ]
when
  Type1( f1 == @[ ... ] "X"
         neg ^^ @[ ... ]
         f2 == "Y" || @[ ... ] == "Z" )
  neg and @[ ... ]
  $t : Type2( field custom @[ ... ] "val")
then
  // Any action here
  inject($t,"id_Eval");
```

- An array of $|I|+2$ degrees $\Xi = [\varepsilon_0, \varepsilon_\sigma, \varepsilon_{j:1..|I|}] \in (L \cup \varnothing)^{|I|+2}$
- An array of priority flags $k_{j:0..|I|+1}$
- A reference to the merge function $\cap : (L \cup \varnothing)^{|I|+2} \mapsto L$
  - A reference to the strategy $S_\varnothing$
  - A strategy S_k
- A reference to the operator $\star$ (only for operator evaluations)

The array $\Xi[1..|I|+1]$ holds the different partial contributions, or the special symbol $\varnothing$ if the corresponding piece of information has not been obtained, for example because a rule has not fired. In particular, $\Xi[0]$ is reserved for the prior degree (if specified in the DRL) and $\Xi[1]$ for the result of direct evaluation. At the moment, the POJO-oriented nature of Drools does not allow to support objects with imperfect field values, with the only exception of missing values, so a single slot in the array is sufficient (methods to overcome this limitation are currently being studied). The elements of $\Xi[i]$ are combined according to the strategy $S_\cap$, which relies on S_k and $S_\varnothing$ as explained in Section 7.4.2. The flags k_j, in particular, are used to distinguish the high-priority degrees from the normal ones. `Evals` also use both the *Observer* and *Observable* design patterns [122]. They are notified when one of the slots $\Xi[i]$ changes its value, and notify when the output of $\cap$ changes. The notified information includes the new value of ε^π and the ratio $|\gamma| = |\Xi|/(|I|+2)$, i.e. the ratio of available contributions over the total number of possible ones.

A subclass of *Composite* [122] `Evals` is defined for operator constraints (see Figure 34): they additionally store the references to a number of `Evals` equal to the operator arity. These values, conditioned by $S_\varnothing$, are aggregated using the operator evaluator $\star$.

7.5.2 *Degree Factory*

The number and type of parameters, each one configurable at the level of individual evaluations using the appropriate attributes, increases the risk of inconsistencies. While the degrees of freedom seem (and are) many, the choices are not uncorrelated: for example, once the degree set has been chosen, the choice of the operator set is automatically conditioned since the operators must be able to process and return degrees compatible with the adopted representation; moreover, the same consideration applies to the various merge and filtering strategies. It is also infeasible to let the user specify the implementing types directly in the rule file: in addition to being extremely error-prone, there would be little or no control on the components used to customize the engine. To avoid such risks, the compiler relies on an intermediary *Singleton* `Factory` based on the *Strategy* design pattern[122]. The instance of this class (outlined in Figure 35) is responsible for building the degrees (at runtime), operators and strategies (at compile time) - basically anything whose type can be specified using an attribute. The factory methods are overloaded in three versions:

Factory Interface

- `getX()` : a zero-argument version which returns the default implementation of a component.
- `getX(String type)` : a one-argument version, whose value is taken directly from the corresponding attribute in the DRL, which returns an instance of the class selected by the argument. If the type is invalid, incoherent or simply not recognized, the default implementation is returned instead.
- `getX(String type, String params)` : a two-argument version, similar to the previous one, but which passes additional arguments to the constructor/initialization method of the instantiated configuration component.

7.5.3 *Complexity Analysis*

Eventually, to complete the analysis of the proposed extension, its complexity will be discussed briefly. For this purpose, it is necessary to consider:

- The number of rules R
- The maximum length of a sequence of α-nodes A
- The maximum length of a sequence β-nodes B
- The maximum number of patterns in a rule C
- The number of objects in the working memory W

The RETE algorithm has a worst case complexity W^C that grows exponentially with the number of patterns in a rule, but the actual cost is tractable and several optimization techniques improve its performance

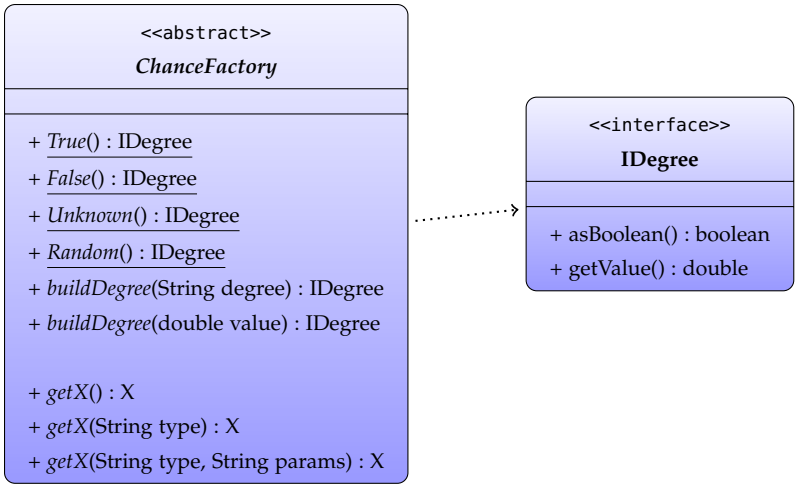

Figure 35: Centralized Factory and Degrees

[104]. Unfortunately, many optimization devised for the original RETE algorithm do not apply, since they assume that an evaluation is either true or false. So far, optimizations for the imperfect case have not been considered (but will be in future works), so it is necessary to take into account the cost of the additions: the custom evaluators in the constraint nodes, the new operator nodes and the degree updates due to injections. The complexity of an embedded evaluator can't be controlled, so it will be assumed to be constant: in any case, the benefits of invoking an external module should justify its use. The presence of operators, instead, lengthens the α and β chains by a constant-bound factor. In fact, if a is minimum arity of the operators and $L = \max\{A, B\}$, a tree with up to $\sum_{j=1}^{\log_a L} a^j = O(L)$ nodes can be built from L nodes, yielding a sequence of length proportional to $a \cdot L$. Hence, the cost of propagation is comparable to the standard case, even if C has to be increased by one due to the implication node.

Evaluator Cost

Operator Cost

$$\sum_{j=1}^{C+1} \{(W \cdot \omega(A))^{j-1} \cdot (W \cdot \omega(A)) + \omega(B)\} \tag{7.3}$$

A relevant cost increase, both spatial and temporal, can instead be ascribed to the injections. The merge at each constraint node may have to process up to $O(R)$ degrees, affecting the propagation cost $\omega(A)$ (resp. $\omega(B)$) that becomes $O(A \cdot R)$. In case of late injections, an Eval tree could be updated up to $O(R)$ times, each time with a cost $O(R \cdot \log(x))$ making the cost quadratic in the number of rules. Likewise, the storage of rule-entailed degrees requires a spatial cost $O(R \cdot L \cdot W)$. Thus, the critical point is the maximum number of injecting rules for any constraint: even if all rules could theoretically inject all others, in most practical cases the effective number of injecting rules R' will be much lesser than R.

Injection Cost

7.6 CONCLUSIONS

This Chapter has introduced an enhanced version of the RETE-OO algorithm, using Drools as a case study. The rule language has been extended to be as compatible as possible with the generalized imperfect logic language cited in Section 6.2.2. At the same time, the DRL language and the RETE-OO engine have been thoroughly modified to support a generalized inference schema suitable for that language.

In particular, the DRL extensions have involved both the expressiveness of the logic language and the set of attributes which can be used to decorate the rules with fine-grained meta-data. These additional constructs are used by the compiler to build, respectively, an extended evaluation network and configure its runtime behaviour.

The new features come with a price: the complexity of the algorithm increases, both in terms of computational requirements and usage. The language is richer and more complicated, so additional care is required to write sound and coherent rules; likewise, the attributes give many degrees of freedom in customizing the behaviour of the engine, fact which is only partially mitigated by the use of the centralized `factory` to limit the possible combinations.

However, the new features are expected to increase the potentialities of the engine in a way that makes the benefits outweigh the drawbacks, allowing to write and process imperfect rules according to the vision outlined in the previous Chapters. To test the potentialities of the engine, the next Chapter will show several case studies which have been realized for evaluation and test purposes, before moving the description of an application of the engine in a real-world case study.

8

APPLICATIONS OF IMPERFECT LOGIC

Contents

The goal of this Chapter is twofold. First, it is meant to show the potentialities of the enhanced engine, showing that there exists a rather wide class of applications that can be realized using it. Taken alone, they are by no means innovative - and neither complete, since not all possible ones have been explored - but they come from a variety of different contexts. The examples have been divided in two classes: the first ones are the "purely" logical ones, which show some imperfect extensions of a classical logic reasoner such as many-valued and non-monotonic logics; after that, several "hybrid" applications will be introduced, where the engine is integrated at various levels with tools such as neural networks.

The proposed implementations are not meant to replace the existing individual engines or tools, which are in fact often optimized for the specific types of reasoning. However, in Chapter 4 the benefits of hybrid systems were discusses extensively: it is not unlikely, then, to think of a real application (e.g. a CEP problem) where a rule-based system requires the capabilities of, say, a Bayesian Network or a fuzzy logic controller. Moreover, a complex application is likely to require *many* different SC techniques for different sub-problems. When the role of each one is limited to a very specific task, the emerging scenario is one in which dozens of different modules co-exist and need to be inter-

faced and (especially in real applications) maintained and upgraded. In this, the engine may assume a strong relevance for several reasons:

- It divides the abstract semantics of a rule from its concrete one. The rules are written as if they were perfect, but their semantics in presence of imperfection can be adapted to the actual type of imperfection, possibly dynamically. In fact, it is sufficient to change the evaluators and/or the `Factory` to modify the rule semantics as the corollary conditions change.
- It acts as a common logic "middleware", decoupling the various sub-symbolic modules by forcing them to interact at the higher, symbolic level of abstraction where they can be coordinated in a way that is also *declarative*. In fact, it will be shown both how to invoke a module from the logic level (e.g. trivially in the RHS) and how to manage its feedback (e.g. wrapping it in a custom evaluator).
- Some techniques can even be *emulated* by the engine, offering an interesting option to the developer. While a dedicated module is surely more powerful and efficient, the cost of interfacing, starting, invoking and maintaining it could be too much if compared to the real complexity of the task it is required for. Consider, for example, a Bayesian "Network" composed by 5 nodes used to process a low-priority event which may manifest itself once per day.

The second goal, instead, is more pragmatically the definition of some patterns which could be used in more concrete applications: in fact, the concrete example discussed in Chapter 10 will exploit some of them.

8.1 IMPERFECT LOGIC APPLICATIONS

8.1.1 *Boolean Logic*

Trivially, and for completeness, it must be noted that the behaviour of a standard, boolean engine can be reproduced using an appropriate `Factory`, outlined in Table 16. The degree set reduces to the binary case, so $\varepsilon \in \{0, 1\}$, and the operators are defined as in the classical case. Optionally, a weak form of *Open World Assumption* can be adopted: a tuple is `held` if the current overall degree is $\varnothing$ because the direct evaluation could not determine the truth state and no injecting rule has contributed yet. Notice that each coherent set of configuration parameters is provided by one concrete `Factory` (but the same `Factory` can provide more than one set of parameters).

Table 16: Configuration for Boolean logic

Parameter	Description
$\mathbf{L}$	$\{0, 1\}$.
σ	Any boolean property
Ω	$\wedge$ min $\vee$ max $\neg$ n.a. $\rightarrow$ ε_C
$\Rightarrow$	1 $\varepsilon_P = 1, \varepsilon_{\rightarrow} = 1$ $\varnothing$ else
$\cap$	$\vee$
$\mathbf{S}_{\varnothing}$	Ignore
$\mathbf{S}_{\mathbf{k}}$	Ignore others
$\mathbf{S}_{\mathbf{f}}$	`pass` $\varepsilon = 1$ `drop` $\varepsilon = 0$ `hold` $\varepsilon = \varnothing$

8.1.2 *MYCIN Certainty Factors*

The first example is a historical tribute. Certainty factors were introduced in MYCIN [69], one of the first expert systems ever, to deal with the statistical uncertainty typical of the medical context in which it was developed. They replaced proper joint and conditional probability distributions, difficult to elicit with absolute numerical precision from human experts.

In fact, each statement is annotated with a degree of "belief" in its truth (MB) and a degree of "disbelief" (MD) - i.e. the belief in its being false. Pairs of such values, each normalized in $[0, 1]$, give a certainty factor $CF = MB - MD \in [-1, 1]$.

It is trivial to implement MYCIN's handling of imperfection in Drools: the engine used a rule language which is a subset of DRL; moreover, CF are modelled using simple real values and are truth-functional, so the general inference framework applies directly. CFs annotate individual constraints (where they can be given a priori, by evaluation or multiple entailment) and rules alike. Formulas are formed by atomic constraints connected implicitly by the connective $\wedge$, which is implemented by the min operator; Modus Ponens then applies the product between the CF of the premise and the CF of the rule. Partial results, which can but decrease during the evaluation of a single rule, are propagated only if superior to a threshold and merged using an operator which takes

into account both the sign and the relative strength of the contributions. The mapping between MYCIN's operators and the engine's configuration options are listed in Table 27: a dedicated `Factory` instantiates degrees and connectives accordingly.

Table 17: Configuration for Certainty Factors

Parameter	Description
$\mathbf{L}$	$[-1, 1]$
σ	Predicate constraints with confidence
Ω	$\begin{cases} \wedge & \min \\ \neg & -(\cdot) \end{cases}$
$\Rightarrow$	$CF_P \cdot CF_{\rightarrow}$
$\cap$	$\begin{cases} CF_1 + CF_2 - CF_1 \cdot CF_2 & CF_1 \cdot CF_2 > 0 \\ \frac{CF_1 + CF_2}{1 - \min\{\lvert CF_1 \rvert, \lvert CF_2 \rvert\}} & \text{else} \end{cases}$
$\mathbf{S}_{\varnothing}$	Set to 0
$\mathbf{S}_{\mathbf{k}}$	Ignore others
$\mathbf{S}_{\mathbf{f}}$	$\begin{cases} \texttt{pass} & CF \geqslant \theta \\ \texttt{hold} & CF < \theta \end{cases}$

From the point of view of imperfection, CFs can be considered a subjective measure of uncertainty. However, their use as measures of uncertainty has been severely criticized ([126], [93]). In fact, CF are set arbitrarily and treated truth-functionally, which is the opposite of of what normally happens with probability. A definition more coherent with theory of probability [187], but which still relies on the assumption of *independence*, was used in EMYCIN. A CF is defined as the *variation* in belief due to the acquisition of new knowledge, i.e. the knowledge update from $p(B)$ to $p(B|A)$. Other than giving a recommendation on how to evaluate the atomic predicates, it allows a more coherent belief/certainty propagation. In this framework, a rule $\varphi \rightarrow \gamma$ - in Drools, the implication degree - is described by *two* factors, one relative to the conclusion given a true premise (CF($\gamma|\varphi$)) and one relative to the conclusion when the premise is false (CF($\gamma|\varphi$). This reflects on a different definition of the Modus Ponens operator:

Parameter	Description
$\Rightarrow$	$\begin{cases} CF(\gamma\|\varphi) \cdot CF(\varphi\|\theta) & CF(\varphi\|\theta) > 0 \\ 0 & CF(\varphi\|\theta) = 0 \\ -CF(\gamma\|\neg\varphi) \cdot CF(\varphi\|\theta) & CF(\varphi\|\theta) < 0 \end{cases}$

8.1.3 *Many-valued logics*

Unlike belief-oriented logics, the class of many-valued logics deals with partially true relations. The term includes a vast family of truth-functional logics of which fuzzy logic (in a narrow sense and, to some extent, even in a broader sense) is a specific case.

Theoretical Background

Many-valued Logics

"Fuzzy" many-valued logics [144] are a truth-functional generalization of classical logic: the set of truth values is extended from the binary true/false $\{0, 1\}$ to the whole interval $L = [0, 1]$ (even if, theoretically, any lattice would be suitable). Sentences, then, can have a *partial*, but *certain*, truth degree, since in this context $||\varphi|| \in L$. The degree of truth of an atomic predicate is given by the associated relation's membership function: in order to evaluate complex formulas, then, the canonical logical connectives `and`, `or`, `not`, ... are likewise generalized. As in the classical case, it is sufficient to define any two of them, usually the implication and the conjunction operators, so that the others can be derived accordingly. However, things are complicated by the fact that there is not a *univocal* admissible extension - actually, there are *infinitely* many ones.

Conjunctions : T-norms

The role of conjunction operator can be taken by any *T-norm* $\star$, a *commutative*, *associative*, *monotonic* binary function $L \times L \mapsto L$ with $\sup\{L\}$ as identity element. Even if any function satisfying these mild requirements is suitable, there exist only three "basic" T-norms [144]:

	Gödel	Goguen	Łukasiewicz
$\|\|\varphi \star \gamma\|\|$	$\min\{\|\|\varphi\|\|, \|\|\gamma\|\|\}$	$\|\|\varphi\|\| \cdot \|\|\gamma\|\|$	$\min\{0, \|\|\varphi\|\| + \|\|\gamma\|\| - 1\}$

Table 18: Canonical T-norms

(R-) Implications

For a chosen T-norm, the associated implication operator $\rightarrow_\star$ is defined by *residuation* [117] : $\|\varphi \rightarrow \gamma\| = \max\{\theta \in L \mid \varphi \star \theta \leqslant \gamma\}$. The definition requires, however, the T-norm to be *continuous*. In this case, the implications (or, more properly, the R-implications) are:

	Gödel	Goguen	Łukasiewicz
$\|\|\varphi \rightarrow_\star \gamma\|\|$	$\begin{cases} 1 & \|\|\varphi\|\| \leqslant \|\|\gamma\|\| \\ \|\|\gamma\|\| & \texttt{else} \end{cases}$	$\begin{cases} 1 & \|\|\varphi\|\| \leqslant \|\|\gamma\|\| \\ \frac{\|\|\gamma\|\|}{\|\|\varphi\|\|} & \texttt{else} \end{cases}$	$\min\{1, 1 - \|\|\varphi\|\| + \|\|\gamma\|\|\}$

Table 19: Canonical R-implications

Negation

Given the implication, the negation operator is defined such that $\|\neg\varphi\| \triangleq \|\varphi \rightarrow 0\|$. In Gödel logic, the evaluation of the negation returns 1 if the operand has a truth degree of 0, and 0 in all other cases. Applying the same definition to the other implications, instead, yields the *strong* negation $\|\neg\varphi\| = 1 - \|\varphi\|$. The negation operator, in turn, allows to define the *reciprocal* implication $\leftarrow$, which is very relevant from

a practical point of view. In fact, when $\neg$ is a strong negation, $||\varphi \to \gamma||$ is equal to $||\neg\gamma \to \neg\varphi||$, but knowing the value of the direct implication, in general, does not allow to evaluate the reversed formula $\gamma \to \varphi$. The evaluation of the reciprocal R-implication, then, yields:

Reciprocal (R-) implication

	Gödel	Goguen	Łukasiewicz
$\|\|\varphi \leftarrow_\star \gamma\|\|$	$\begin{cases} 1 & \|\|\varphi\|\| \leqslant \|\|\gamma\|\| \\ 1 - \|\|\varphi\|\| & \text{else} \end{cases}$	$\begin{cases} 1 & \|\|\varphi\|\| \leqslant \|\|\gamma\|\| \\ \frac{1-\|\|\varphi\|\|}{1-\|\|\gamma\|\|} & \text{else} \end{cases}$	$\min\{1, 1 - \|\|\varphi\|\| + \|\|\gamma\|\|\}$

Table 20: Canonical reciprocal R-implications

Disjunction : T-Conorms

The remaining operators are defined in terms of the previous ones. In particular, the disjunction operator $\bullet$, also called T-conorm, is obtained applying DeMorgan's law to a T-norm:

	Gödel	Goguen	Łukasiewicz
$\|\|\varphi \bullet \gamma\|\|$	$\max\{\|\|\varphi\|\|, \|\|\gamma\|\|\}$	$\|\|\varphi\|\| + \|\|\gamma\|\| - \|\|\varphi\|\| \cdot \|\|\gamma\|\|$	$\min\{1, \|\|\varphi\|\| + \|\|\gamma\|\|\}$

Table 21: Canonical T-conorms

Equivalence and Exclusive-Or

Eventually, the equivalence operator and its negation, the exclusive or [51], derive from their classical counterparts: $||\varphi \equiv \gamma|| = ||(\varphi \star \gamma) \bullet (\neg\varphi \star \neg\gamma)||$. The standard quantifiers, instead, rely on Gödel's T-norm and conorm:

$$\begin{aligned} ||\forall X : \varphi(X)|| &= \min_X\{||\varphi(X)||\} \\ ||\exists X : \varphi(X)|| &= \max_X\{||\varphi(X)||\} \end{aligned}$$

Alternative Definitions : S-Implications

However, this chain of derivations is not the only possible one. The implication $\to$, so far defined from a *material* point of view, can also be defined from an *intuitionistic* one [44], extending the classical tautology $(\varphi \to)\gamma \equiv (\neg\varphi \bullet \gamma)$. This framework requires that $\bullet$ and $\neg$ have already been defined. In particular, considering the strong negation $1 - ||\cdot||$, the choices for $\bullet$ still coincides with the definitions already given for T-conorms, but in this case are provided as primitives. The resulting definitions are not the same: in fact, they are called S-implications to distinguish them from R-implications.

	Gödel	Goguen	Łukasiewicz
$\|\|\varphi \to_\star \gamma\|\|$	$\max\{1 - \|\|\varphi\|\|, \|\|\gamma\|\|\}$	$1 - \|\|\varphi\|\| + \|\|\varphi\|\| \cdot \|\|\gamma\|\|$	$\min\{1, 1 - \|\|\varphi\|\| + \|\|\gamma\|\|\}$

Table 22: Canonical S-implications

Yet another derivation of the implication operator is found in [51]: this framework considers the exclusive or primitive, and derives the other operators from it. In particular, two more notions of implications, E-implication and X-implications are obtained.

From a practical point of view, it is clear that the use of many-valued logics requires particular attention: while connectives can be used to compute almost any combination of truth degrees, their use

in a coherent way from the logical point of view is a different matter. Łukasiewicz's logic is possibly more "coherent" than the other two [**?**], but it is also the most *strict*, since the ordering relation between the evaluation of the T-norms always holds[1]:

$$\min\{0, ||\varphi|| + ||\gamma|| - 1\} \leqslant ||\varphi|| \cdot ||\gamma|| \leqslant \min\{||\varphi||, ||\gamma||\}$$

Goguen's logic is suitable to model the operators of a probabilistic logic under the strong assumption of *statistical independece* required to make the logic truth-functional, which however is unrealistic in many cases. Gödel's logic, on the other hand, is the one commonly used in canonical fuzzy logic systems. However, there are several variants which combine two or more of the previous, like Pavelka's rational logic ([234],[235],[236]), which is an extension of Łukasiewicz's logic with an explicit notion of truth degrees. A complete picture of the different logics and their relations can be found in [144] or [117].

Kleene's 3-valued logic

Kleene's logic is a well-known formalism (e.g.[214]) which extends the admissible set of truth values with a third value, $\perp$. This value has found a dual interpretation [131]: as intermediate *truth* degree, modelling any partial degree $0 < \varepsilon < 1$ without specifying exactly which one, or as a measure of *(un)certainty*. In this second case, a sentence evaluated to $\perp$ is either true or false, but an agent does not have enough evidence to decide which is the case. The logic is truth-functional and the definition of the operators is intuitively and apparently closer to the uncertainty semantics, but the sound interpretation is actually the many-valued one [112]. In this case, it can be reconducted to the more general case by considering $\perp$ equal to *any* intermediate value between 0 and 1.

Implementation

Given the variety of definitions for the logical connectives, the role of the attribute `@kind` becomes crucial: the various families are then wrapped into operators, as summarized in Table 23, and completed with the appropriate strategies. Different sources are combined taking the maximum of the individual degrees: in fact, applying modus ponens on a single-rule basis guarantees only a lower bound to the truth degree of a conclusion ([235],[144]). To obtain the definitive value, all possible injecting rules must have fired: this is the reason why a tuple is propagated only when the associated `Eval` degree is strictly greater than a threshold (usually 0 or 0.5), but instead is *held* if the degree is below the threshold but there still exist some rules which could potentially bring it above the desired level.

Negation and Interval Degrees

Problems, in many-valued logic, arise when the theory admits the

1 non-continuous T-norms are not considered

Table 23: Configuration for many-valued logic

Parameter	Description
$\mathbf{L}$	$[0, 1]$.
σ	Fuzzy predicate constraints
$\cdot$	$\begin{cases} \neg & 1 - \cdot_1 \\ * & \wedge, \odot, \wedge_Ł \\ + & \neg(\neg\varepsilon_1 * \neg\varepsilon_2) \\ \rightarrow & \neg\varepsilon_1 + \varepsilon_2 \\ \not\equiv & \varepsilon_1 * \neg\varepsilon_2 + \neg\varepsilon_2 * \varepsilon_2 \\ \equiv & (\varepsilon_1 \rightarrow \varepsilon_2) * (\varepsilon_2 \rightarrow \varepsilon_1) \end{cases}$
$\Rightarrow$	$\varepsilon_P * \varepsilon_{\rightarrow}$
$\cap$	$\vee$
$\mathbf{S}_\varnothing$	Set to 0 (cwa) or ignore
$\mathbf{S}_\mathbf{k}$	Set non-flagged to 0 (cwa) or ignore
$\mathbf{S}_\mathbf{f}$	$\begin{cases} \texttt{hold} & \varepsilon < \theta, \lvert\gamma\rvert < \gamma_0 \\ \texttt{drop} & \varepsilon < \theta, \lvert\gamma\rvert \geqslant \gamma_0 \\ \texttt{pass} & \varepsilon \geqslant \theta \end{cases}$

entailment of the logical negation of a formula. In this case, one obtains a set of *upper bounds* to the truth degree of the formula itself: in fact, from $\varepsilon_{\neg C} \geqslant \varphi$ follows that $\varepsilon_C \leqslant 1 - \varphi$. The separate entailments of C and $\neg$C yield an *interval* $[\tau, 1 - \varphi]$, in which the actual truth degree ε_C is expected to lie. Unless intervals are *certain* (i.e. they reduce to a single number and $\tau + \varphi = 1$), their use introduces a degree of *uncertainty* on the *gradual* truth of a formula. τ and $1 - \varphi$ can be considered lower and upper previsions of a formula's truth degree: the immediate consequence of the adoption of this type of logic is the loss of truth-functionality in the strict sense. All the operators generalize naturally to be used with intervals, but the resulting intervals must always be considered bounds. For example, while $\forall\varepsilon : \varepsilon \otimes (1 - \varepsilon) = 0$, the interval-valued version gives $[\tau, 1 - \varphi] \otimes [\varphi, 1 - \tau] = [0, 1 - (\varphi + \tau)] \supseteq [0, 0]$.

Many-valued Logic with Confidence

A further extension of this model can be obtained combining the interval degrees with a specialization of certainty factors, using the latter to model *confidence*, in a way similar to what was done in fuzzyCLIPS [7]. Unlike MYCIN, however, the confidence factors lie in $[0, 1]$. The two degrees exist at different levels: the interval is used to model the imprecision in defining an exact truth value, while here the confidence degree measures the *strength* of the belief in the (imprecise) estimation.

Table 24: Configuration for interval-valued logic with confidence

Parameter	Description
L	$\{[\tau, 1-\varphi]_\chi \mid [\tau, 1-\varphi] \subseteq [0,1], \chi \in [0,1]\}$.
ff	Fuzzy predicate constraints with confidence
.	$\begin{cases} \neg & \neg_G, \neg_\pi, \neg_L \\ * & \wedge, \odot, \wedge_Ł \\ + & \neg(\neg \cdot_1 * \neg \cdot_2) \\ \rightarrow & \neg \cdot_1 + \cdot_2 \\ \diamond & \cdot_1 * \neg \cdot_2 + \neg \cdot_2 * \cdot_2 \\ \equiv & \neg\diamond, (\cdot_1 \rightarrow \cdot_2) * (\cdot_2 \rightarrow \cdot_1) \end{cases}$
$\Rightarrow$	$[\tau_1 * \tau_2, 1]_{\chi_1 \cdot \chi_2}$
$\cap$	χ-discount and interval intersection
$\mathbf{S}_\varnothing$	Set to $[0,1]_0$ or ignore
$\mathbf{S}_k$	$\chi^\star = \max\{\chi_j \mid \varepsilon_j \text{ overrides}\}$ $\chi_{j \mid \varepsilon_j \neg\text{overrides}} = \max\{0, \chi_j - \chi^\star\}$
$\mathbf{S}_f$	$\begin{cases} \texttt{hold} & \chi \cdot (\tau + \varphi) < \theta, \lvert\gamma\rvert < \gamma_0 \\ \texttt{drop} & \chi \cdot (\tau + \varphi) < \theta, \lvert\gamma\rvert \geqslant \gamma_0 \\ \texttt{pass} & \chi \cdot (\tau + \varphi) \geqslant \theta \end{cases}$

In fact, confidence is ultimately used by the merge function $\cap$. When intervals are involved, $\cap$ usually reduces to interval intersection, generalizing the concept of sup for mere lower bounds. When confidence is also present, intervals can be discounted according to their associated (lack of) confidence before they are intersected, moving from $[\tau, 1-\varphi]$ to $[\chi \cdot \tau, 1-(\chi \cdot \varphi)]$. At worst, when $\chi = 0$, the discount causes an interval to become the vacuous interval $[0,1]$. The computation of confidence factors themselves, instead, may proceed as follows. Atomic evaluators are expected to provide the confidence degree as well as the (interval) truth one; operators, then, combine the confidence of their operands by returning an aggregate confidence by means of a T-norm (usually the min). Modus Ponens, instead, always returns the *product* of the premise's and implication's confidence degrees. $\cap$, eventually, combines the confidence degrees of the different contributions using a T-conorm: in fuzzyCLIPS max is used, but the probabilistic sum is also a good candidate since it causes the overall confidence to grow as the number of merged contributions increases. The revised configuration parameters are listed in Table 24. Notice that confidence can be used as an additional factor by the filtering strategy.

Application: Gradual Rules

An important application of many-valued logic is given by fuzzy rules, where implications (or, more generally, the relation $\rho(P, C)$ between premise and conclusion) has a gradual, fuzzy value. In [117], several sub-classes of rules are defined, including:

Types of Gradual Rules

- **Gradual Rules** : *The more P, the more necessary C.*
- **Impossibility Rules** : *The more P, the less possible not C.*
- **Possibility Rules** : *The more P, the more possible C.*
- **Antigradual Rules** : *The more P, the less necessary C.*

The definitions are rather general, but with some assumptions they can be used to justify some basic inference rules. The first states that modus ponens entails a lower bound for the conclusion; the second, instead, that using a (negated) premise to entail a negated conclusion yields an upper bound for the conclusion itself (in fact, assume that $P \to C \equiv \neg C \to \neg P$: the more P, the less $\neg P$, so $\neg C$ must be low as well, even in the best case). The last two, instead, rely on *negated* implications: they allow to entail lower and upper bounds for the conclusion, but in this case the lower bound is obtained implying the negated conclusion and vice versa (intuitively, for S-implications $\neg(P \to C)$ is equivalent to $P \wedge \neg C$). (Other rules exist, such as *certainty* rules, but these involve a mapping from the level of truth degree to the level of belief degree. While theoretically supported by the engine, they have not been studied yet)

Rules like these can be implemented easily, according to the pattern 8.1.

Listing 8.1: Gradual Rules

```
rule "Gradual"
  entailment @[ kind="modus_ponens"]
             // kind="equivalence"
  /*neg*/ implication
  when
    /*neg*/ Premise(...)
  then
    inject(...);
    //reject(...)
end
```

Premise and implication can be negated as necessary; the modus ponens operator then computes the conclusion degree in the form of an interval $[\tau, 1]$ which can be applied to a target, chained formula either as a lower bound or, negated, as an upper bound $[0, 1 - \tau]$. For convenience, moreover, the `equivalence` entailment modality is provided: it applies, at the same time, the modus ponens twice: $\langle P, P \to C\rangle$ and $\langle \neg P, \neg P \to \neg C\rangle$. To do so, however, it assumes that the rule is not based on a simple implication, but on an equivalence, so that $(P \to C) \equiv (C \to P)$.

8.1.4 *Possibilistic Logic*

In its original definition, Possibilistic logic [112] is an application of possibility theory (see Chapter 2) to first-order logic, where formulas φ are annotated with a *possibility* degree ε which models the necessity of the sentence to be true, i.e. $N(\varphi) = N(\|\varphi\| = 1) \geqslant \varepsilon$. The definition explicitly involves the evaluation of the formula to stress the concept that possibility does not reason over truth degrees, but degrees of possibility that a sentence is true. However, it generalizes to a fuzzy version where necessity and possibility become the lower and upper bounds of the truth degree of a predicate.

The operators of possibility theory are *not* truth-functional, and so is the logic which maps $\wedge$ on $\cap$ and $\vee$ on $\cup$. However, it is still possible to reason with bounds for the necessity and possibility of a formula: in fact, from $N(\varphi) \geqslant \varepsilon$ one gets $\Pi(\neg\varphi) \leqslant 1 - \varepsilon$. These bounds are a specific case of (consonant) belief and plausibility functions and can be used for automatic reasoning. In [107] a mixed necessity/possibility resolution-style inference procedure is shown, but it also defines the required operators.

Possibility logic uses modus ponens as main inference rule, computing the necessity of a formula γ from the necessity of a premise ε_φ and an implication $\varepsilon_{\varphi\to\gamma}$. In particular, from $N(\varphi) \geqslant \varepsilon_\varphi$ and $N(\varphi \to \gamma) = N(\neg\varphi \vee \gamma) \geqslant \varepsilon_\to$, one entails:

$$N(\gamma) \geqslant \min\{\varepsilon_\varphi, \varepsilon_\to\}$$

Bounds for the same formula can be combined: if one has $N(\gamma) \geqslant \varepsilon_1$ and $N(\gamma) \geqslant \varepsilon_2$, the effective bound becomes $N(\gamma) \geqslant \max\{\varepsilon_1, \varepsilon_2\}$. An alternative definition of modus ponens, this time involving an implication defined in terms of its possibility, is also given in [107].

8.1.5 *Learning by Induction*

Induction is the process of inferring the degree at which a general property $P(\mathbf{X})$ holds for a set of objects $\mathbf{X}$ from the mere observation of the property on a limited subset of objects $x_1, x_2, \ldots, x_n \in \mathbf{X}$. Induction becomes particularly relevant when the formula to be learned is an implication $P \to C$ and the observations are pairs of premise/conclusion candidates, or in the data-mining problem of learning association rules $P \Rightarrow C$ [29].

The classical formulation

$$\frac{P(x_1) \ldots P(x_n)}{\forall X : P(\mathbf{X})}$$

is obviously inadequate: it is sufficient that P does not hold for *one* element x_j to make the universally quantified formula false. In practice, the $\forall$ quantifier is used in natural language expressions, but in a relaxed sense, with the actual meaning of "most" and, even then, assuming implicitly that it holds only with a certain probability/belief/-confidence. Clearly, in order to be sound, an inductive process must

Table 25: Configuration for possibilistic logic

Parameter	Description
$\mathbf{L}$	$[0,1] \times [0,1]$
σ	Gradual Necessity/Possibility
$\cdot$	$\begin{cases} \neg & \Pi = 1 - N \\ * & \wedge(N) \\ + & \vee(\Pi) \\ \rightarrow & \neg P + C \end{cases}$
$\Rightarrow$	$N_C \geqslant (N_P \wedge N_{\rightarrow})$
$\cap$	$\vee$
$\mathbf{S}_{\varnothing}$	$N = 0, \Pi = 1$
$\mathbf{S}_{\mathbf{k}}$	Set non-flagged to $N = 0, \Pi = 1$
$\mathbf{S}_{\mathbf{f}}$	$\begin{cases} \texttt{hold} & N < \theta, \lvert\gamma\rvert < \gamma_0 \\ \texttt{drop} & N < \theta, \lvert\gamma\rvert \geqslant \gamma_0 \\ \texttt{pass} & N \geqslant \theta \end{cases}$

take imperfection into account. Unfortunately, as usual there are many possible definitions, which also depend on the imperfection associated to the individual observations. From a merely syntactical point of view, DRL has been extended with one dedicated quantifier, *forany*. The potentialities of this quantifier are still being studied, so here a few basic applications will be described. The syntax is given in 8.2; informally, the equivalent predicate-style notation $\int_{C()} F() dW()$ will sometimes be adopted.

Generalized Induction

The `forany` quantifier can be customized using the `@kind` and the other common attributes. For example, in his paper [308] Zadeh generalizes the concept of (fuzzy) quantifier for fuzzy statements, introducing concepts such as "*most*" which fit well in an induction context. The attribute `@kind` can be used to choose the specific type of operation to be performed: it is an entry-point for external algorithms, but also an important extension which supports purely logical, rule-based mechanisms. The expression `weight`, instead, is meant to evaluate a degree with the semantics of weight, used by the induction operator to give different relevance to the different individuals when computing the overall degree. The actual semantics depends on the context and is delegated to the operator: the simplest induction operator is the `AverageInductionOperator` which computes a (weighted) average of the individual degrees. When the degrees are properly fuzzy degrees, it computes the *expected* fuzzy degree as suggested in Zadeh's example

Inducing Expected Fuzzy Degrees

Listing 8.2: Generalized Induction Quantifier

```
rule "Induction"
  when
    forany @[ kind="..." ] (
                    F() // any formula to be induced
      subject_to    C() // another formula
      weight        W() // one more formula
    )
  then
    inject("Target");
end

rule "Target"
  when
    ...
  then
end
```

(see rule 8.3):

Listing 8.3: *"Most Swedes are tall"*

```
rule "Fuzzy Induction"
  when
    forany @[ kind="average" ] (
              $p: Person( this seems "tall" )
      weight Person( this == @[ cut ] $p,
                     nationality == "Swedish" )
    )
  then
   ...
end
```

Inducing Objective Probabilities

The same syntax, however, can be used with boolean properties to obtain a frequentist probability, as shown in rule 8.4, where it is used to compute the probability that a 1 is rolled on some die. Notice that the same rule exploits another feature of Drools, the `from` keyword, which allows to define alternative object entry points and thus can be conveniently used to keep the flow of samples from a "training set" separate from the main WMEs.

The degrees so obtained can be used directly in the consequence, or injected in some other constraint. For example, the frequentist probability obtained in the previous example can be injected as a *prior* probability, to be combined and refined using, for example, a direct evaluator as in the second rule of example 8.4. The probability induced on a case history of previous rolls, in fact, becomes useful when the direct observation of a roll is impossible, either because the dice has not been rolled yet (there is no `Roll` in the WM) or because the information is missing (the field result is `null`). When the `Roll` is actually inserted or `updated`, the appropriate $\cap$ strategy will compute the correct value.

Listing 8.4: "*Probability Estimation*"

```
rule "Frequentist Probability"
  when
    $d : Die()
    forany @[ kind="average" ] (
             Roll( die == @[ cut ] $d,
                   result == 1 ) from Rolls
    )
  then
   inject($d,"idRoll1");
end

rule "Bad luck"
  when
    $d : Die()
    $r : Roll( die == @[ cut ] $d,
               result == @[ id="idRoll1"] 1)
  then
    ...
end
```

Induction with IDM

The possibilities increase when more complicated degrees are used in place of simple real values. In a probabilistic setting, a Bayesian update schema can be adopted, possibly in combination with the use of imprecise probabilities. For example, the previous dice example can be used with a (binomial) IDM (see [54] and Section 2.1.3), as in rule 8.5. In this case, the `@degree` attribute can be used to specify the prior probability, while the `@params` can set additional parameters such as the weight of the "hidden" observations.

Listing 8.5: "*Imprecise Dirichlet Probability Estimation*"

```
rule "IDM"
  when
    $d : Die()
    forany @[ kind="dirichlet", degree="1/6", params="0.2" ] (
             Roll( die == @[ cut ] $d,
                   result == 1 ) from Rolls
      weight ...
    )
  then
   ...
end
```

The boolean definition of $\int$ becomes more robust in a possiblistic context, by considering the necessity-possibility interval $[\forall X : P(\mathbf{X}), \exists X : P(\mathbf{X})]$. An induction operator of this kind computes the min (respectively the max) of the necessity (resp. possibility) degrees associated to the individual observations, which however must be equally relevant. This definition generalizes naturally to the fuzzy case, as well.

Learning Implications : Gradual Rules

As already noted, induction becomes a relevant tool when rules have to be learned from data. In doing so, one must try to avoid the dangerous side-effect known as *Ex Falso Quodlibet*: given an implication $P(X) \rightarrow C(Y)$, a false premise makes the operator true, regardless of the conclusion. Hence, the first time the premise becomes true, the conclusion might be assumed to be true as well, which is obviously an undesired behaviour.

In fact, it is common to partition the set of examples according to the fuzzy concept of relevance. So, for each couple <P,C>, membership in the positive, negative and irrelevant sets is given by a trio of functions that should be complementary, i.e.:

$$\forall < P, C >: \mu^+(< P, C >) + \mu^-(< P, C >) + \mu^0(< P, C >) = 1$$

In [114], Dubois and Prade study the learning of associative and gradual rules in terms of fuzzy partitions. The first case uses the conjunction, implication and negation of many-valued logic to define the partition in terms of logical formulas involving the candidate premise and conclusion: *Relevance*

$$\begin{aligned} \mu^+ &= P \star C \\ \mu^- &= \neg(P \rightarrow C) \\ \mu^0 &= \neg P \end{aligned}$$

An alternative definition, more suitable for gradual rules, is given in the same paper:

$$\begin{aligned} \mu^+ &= P \star (P \rightarrow C) \\ \mu^- &= P \star (\neg(P \rightarrow C)) \\ \mu^0 &= \neg P \end{aligned}$$

The extended syntax supports this theoretical framework as well. In [114], it is remarked that the operators $\star$, $\neg$ and $\rightarrow$ cannot be chosen independently. Only some combinations of Łukasiewicz, Goguen and Gödel operators are allowed: the `Factory` can ensure that the constraints are verified using the same specific meta-value in the atttribute `@kind` - e.g. `gradualImpl` - of all the connectives, even if the programmer is free to override it locally. Rule 8.6 shows an example for associative rules:

In particular, the memberships in μ^+ and μ^- lead to two different degrees which must be induced separately and used, for example, to define the lower and the upper bound (in a fuzzy or possibilistic sense) for the degree associated to a rule $P \rightarrow C$. The presence of μ^0, instead, led to another subtle consideration on the role of pairs for which the premise tends to be false: in fact, a pair can either be *ignored* or taken into account, but considering its contribution as an "unknown". When a single pair is involved, the false implication is considered a *lower bound* for the induced degree, so there is no real difference. When *many* pairs are involved, however, the difference may lie in the support.

Listing 8.6: "*Learning Associative Fuzzy Rules*"

```
rule "Assoc_rules"
  when
    forany @[ kind="goedel" ] ( // min in a possibilistic sense
              and @[kind](
                $p : Premise() from tsEP
                Conclusion( link == @[ cut ] $p ) from tsEP
              )
      subject_to Premise(this == @[cut] $p)
      weight     Premise(this == @[cut] $p)
    )
  then
    ...
end
```

Consider, for example, a Bayesian model such as the IDM: it is rather different to say that a formula (the general implication, in this case) is unknown because no observations have been made, rather than saying that, after observing all the training set, no additional knowledge has been gained. To this end, two different expressions can be used: the first, `weight`, is used only to compute the (relative) relevance of an element in the induction process. The second, `subject_to` is used to *discount* the degree before it is combined with the others, using the same `Factory`'s discounting strategy used by $\cap$ to merge degrees (unless modified with `@kind`). Obviously, the two can be used together.

8.1.6 *Probabilistic logics*

Various logics with support for probabilistic reasoning (purely statistical approaches as well as Bayesian reasoning) have been developed. The main problem of probabilistic logic is their general lack of truth-functionality, so several integrations of probability theory and rule-based systems have been proposed, but all with specific limitations. Given the number and variety of languages, only a few have been analysed.

Probabilistic Datalog

(Datalog$_P$) [137] is an extension of Datalog [77], a *propositional* language, which additionally allows for the probabilistic weighting of facts (but not - extensionally - of rules). Informally, the idea here is that each ground fact corresponds to an event in the sense of probability theory, and rules allow for boolean combinations of events and their probabilities, which have a probabilistic *possible-world semantics* (i.e. the degrees model a probability distribution over all possible worlds). In order to deal with the absence of truth functionality, Datalog$_P$ follows two alternative directions: basic Datalog$_P$ yields probability intervals instead of "point" probabilities in case of derived event expressions;

alternatively, $\mathtt{Datalog}_{\mathrm{PID}}$ makes the quite strong assumption of universal event independence, that is $\Pr(A \wedge B) = \Pr(A) \cdot \Pr(B)$ for any events A and B. Under this assumption boolean combinations of the constituents of probabilistic event expressions become possible, so the logic reduces to a many-valued one.

Bayesian Logic Programming

While *Bayesian networks* can emulate certain types of rules, they work only on a propositional level. However, several formal approaches exist which extend Bayesian networks with first-order capabilities (relations). *Bayesian logic programs* (BLP) [167] for example can be seen as a generalization of Bayesian networks and logic programming, implementing a possible-world semantics. The logic component of BLP consists of so-called *Bayesian clauses*. A Bayesian clause is a rule of the form $A|A_1, ..., A_n$, where each A_i is a universally quantified *Bayesian atom*. The main difference between Bayesian clauses and ordinary clauses is that the Bayesian atoms have values from a finite domain instead of boolean values. In addition to Bayesian clauses, a BLP consists of a set of conditional probability distributions over Bayesian clauses c (encoding $p(\mathtt{head}(c)|\mathtt{body}(c))$) and so-called *combining rules* in order to retrieve a *combined conditional probability distribution* from the combination of the multiple different conditional probability distributions. Given a BLP a Bayesian network can be easily computed and then queried using standard Bayesian inference. In alternative, it is possible to exploit the enhanced engine writing rules such as the ones in 8.7. Using a naive approach, the conditional probability tables are mapped on implications' degrees, while the conjunction operator in the LHS combines the premise atoms' degrees in matrix form, so that modus ponens reduces to matrix multiplication. The entailed degree, then, can be injected and combined using the chosen combination rule.

Listing 8.7: Bayesian Logic Programs

```
rule "BLP"
  entailment  @[ kind="BLP"    ]
  implication @[ degree="..." ]
  when
    $a1 : Atom1( value == @[ id="..." ] "..." )
    ...
    $aN : AtomN( value == @[ id="..." ] "..." )
  then
    AtomC cons = new AtomC(...);
    inject(cons,"idValC");
end
```

It could be also possible to write one generalize rule (i.e. without placing restrictions on the atom values), but in that case the implication operator would not be truth functional: in fact, it would be its responsibility to analyse the atoms (accessing them through the `Evals`

in order to choose the appropriate conditional probability table (which would likewise not be possible to express as a constant degree in the rule base).

Table 26: Configuration for Bayesian Logic Programs

Parameter	Description
$\mathbf{L}$	$[0,1]$.
σ	Atom value restrictions
Ω	$\wedge$ matrix composition $\vee$ n.a. $\neg$ n.a. $\rightarrow$ $p(C\|\mathbf{P})$
$\Rightarrow$	$p(C\|\mathbf{P}) \star p(\mathbf{P})$
$\cap$	$\vee$
$\mathbf{S}_{\varnothing}$	n.a.
$\mathbf{S}_{\mathbf{k}}$	n.a.
$\mathbf{S}_{\mathbf{f}}$	`pass`

HYBRID PROBABILISTIC LOGIC PROGRAMS Hybrid Probabilistic Logic Programs [91] are a hybrid combination of interval many-valued logic and probabilistic logic programs. Atoms are annotated/evaluated with a probability interval and then combined using different types of truth-functional connectives, derived from many-valued logic. The interest lies in the inference procedure, derived from probabilistic logic programs [213] where rules have the form

$$P : [p_l, p_u] \Rightarrow C : [\tau, 1 - \varphi]$$

meaning that when a fact $P' : [p'_l, p'_u]$ matches with P and its probability interval is contained in the one given for P (i.e. $[p'_l, p'_u] \subseteq [p_l, p_u]$), then it is possible to entail the conclusion C with a probability falling within the range $[\tau, 1 - \varphi]$. The only required addition is an unary meta-operator **range** which checks whether the premise degree falls within the given range, returning true ($[1,1]$) or false ($[0,0]$) as appropriate. It is also necessary to use the equivalence modus ponens operator, interpreting the rules as gradual, and specify the desired conclusion degree as the implication degree. In fact, interval based equivalence modus ponens is defined as follows, generalizing the concept of many valued modus ponens. Assuming that the premise degree is $[\tau_P, 1 - \varphi_P]$, that the implication degree is $[\tau_\rightarrow, 1 - \varphi_\rightarrow]$ and that the

equivalence holds so that $P \rightarrow C \equiv C \rightarrow P$, applying the definition of gradual modus ponens twice with any T-norm $\star$:

$$\tau_P \star \tau_r ightarrow \leqslant \tau_C \tau_P \star \varphi_r ightarrow \leqslant \varphi_C$$

When τ_P is 0, the entailed conclusion $[\tau_C, 1 - \varphi_C]$ is $[0, 1]$; otherwise, when $\tau_P = 1$, modus ponens returns $[\tau_C, 1 - \varphi_C] = [\tau_r ightarrow, 1 - \varphi_{\rightarrow}]$. The rule pattern is shown in rule 8.8.

Listing 8.8: Hybrid Probabilistic Logic Programs

```
rule "BLP"
  entailment  @[ kind="equivalence" ]
  implication @[ degree="(a,b)" ]
  when
     range @[ params="(c,d)" ] (
       // any multi-valued formula here
     )
  then
    ...
end
```

8.1.7 *Dealing with Exceptions*

The problem of exception is relevant in perfect, monotonic systems, since one has to find a trade-off between the use of general rules and the possible presence of (rare) instances of facts which violate them. Consider the classic example bird(X) $\Rightarrow$ flies(X). The rule is valid for *most* birds but not for *all*, since there can be *exceptions*, from entire subclasses of birds (e.g. penguins) to individuals (e.g. Tweety). However, even if incorrect, statements like this are convenient because (i) they are simple and (ii) still cover the majority of cases. In practice, it is not feasible to have a rule should include all the additional checks in its preconditions (e.g. $(P \wedge \neg E) \Rightarrow C$, where E denotes an exceptional condition), especially since a rule should be modified every time a new condition is found. Notice, however, that this problem is covered by the framework of imperfection since ignoring E is equivalent to treating it as if it were a missing piece of information.

Exception

Default logic [253] deals with the problem by transforming premises in *prerequisites* which must be subject to some required *justifications* before they can be effectively used to entail a conclusion.

Another widely used theoretical framework is that of *Defeasible* Logic [217]. In this framework, conflicts are solved by "defeat", i.e. proving that one of the conflicting consequences effectively overrides the other. In particular, given an implication $\varphi \rightarrow \gamma$ used for modus ponens, the knowledge of a third formula κ can be used either to attack the implication $\rightarrow$ (*undercutting* defeat), preventing γ from being deducted, or to attack the conclusion γ itself (*rebutting* defeat), or both. Defeaters may be defeated themselves: a partial order relation $\succ$ is defined over the formulas do decide their relative strength.

Defeasible Logic

Defeasible logic has been integrated with canonical logic programming (i.e. Horn clauses in a Prolog-compliant rule engine) [37]: using a defeasible theory, a goal can be solved with one of four degrees of certainty: $+\Delta$ (the goal is definitely provable), $+\delta$ (the goal is provable in a defeasible manner), $-\delta$ (the goal is provable not to be defeasibly provable) and $-\Delta$ (the goal can be proved to be definitely not provable). Another integration, this time using a forward chaining approach, is given in [192], where the complexity of the problem is also studied.

Defeaters

In Drools Chance, there exists a form of support for *undercutting* and *rebutting* defeating rules, as defined in [217] and [16], but instead a full mapping of defeasible logic, for example exploiting generalized truth degrees to model states such as $+\Delta$, $-\Delta$, $+\delta$ and $-\delta$, has not been been studied in detail yet.

Listing 8.9: "*Rules with Exceptions*"

```
rule "default"  // aka "r1"
// Bird(X) -> Flies(X)
when
  $x : Bird( )
then
  inject($x,"idFlies"); //default
end

rule "defeater" // aka "r2"
// Chicken(X) -> neg Flies(X)
// Chicken(X) -> neg (Bird(X) -> Flies(X))
when
  $x : Chicken( )
then
  reject($x,"default",true);
  reject($x,"idFlies",true);
end

rule "policy"   // aka "r3"
when
  $x : Bird( flier == @[id="idFlies"] true )
then
  ...
end
```

Example

To discuss this feature on a concrete example, consider the rule base in 8.9. The idea is that the "defeater" rule, active only in presence of exceptional conditions, alters the "default" logical sequence `Bird` $\rightarrow$ `Flies` rightarrow ..., cutting the link between `Bird` and `Flies` and replacing it with its own. To do so, it uses its own consequence degree, opportunely negated, to influence the two specific degrees: the implication of the default rule and the evaluation of the final constraint. The situation is sketched in Figure 36 and Figure 37 (where only an abstraction of the RETE nodes is drawn).

The effect of the different defeaters depends on the type of logic which is being used. As an example, the behaviour is described in

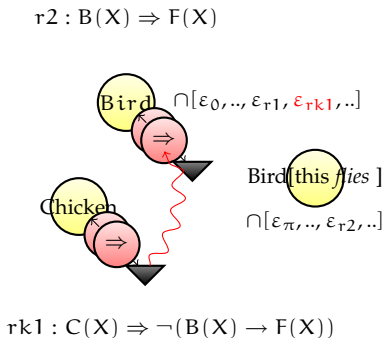

Figure 36: Undercutting Defeater

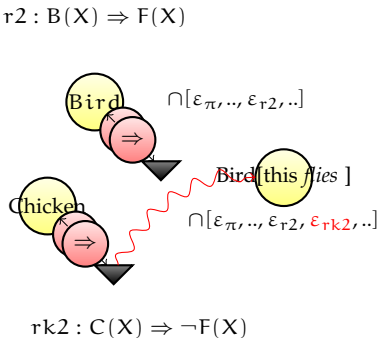

Figure 37: Rebutting Defeater

the cases of 3-valued logic and possibilistic interval logic, so the corresponding factories are assumed to be loaded. To handle exceptions, however, a further addition is required. Degrees ε (of any kind) are annotated with a higher order, real-valued degree χ to model the *confidence* in their evaluation. The implementation of the merge strategy $\cap$, then, is extended to take into account the presence of "overriding" degrees, which normally are not present in a monotonic rule base. In practice, this requires to extend the confidence-discount modality applied to interval-valued possibilistic logic to other types of many-valued logic. As described in [16], $\cap$ performs a *discount* of the different degrees before combining them: the discount, in turn, is stronger for degrees with lower confidence - at the limit, a degree with no confidence is not taken into account at all. In this context, the default override policy, as suggested by [243], lowers the *confidence* of low priority degrees using the maximum confidence of the high priority ones before they are merged. This allows exceptional degrees to override standard ones, but only if an exceptional condition is acknowledged with sufficient certainty.

Table 27: Configuration for Defeater Rules

$\mathbf{S}_k$	$\chi^\star = \max\{\chi_j \mid \varepsilon_j \text{ overrides}\}$
	$\chi_{j \mid \varepsilon_j \neg\text{overrides}} = \max\{0, \chi_j - \chi^\star\}$

EXCEPTIONS IN 3-VALUED LOGIC. The network uses 3-valued logic, where T, F and ? are used to denote true, false and unknown respectively, while $\emptyset$ is used to model a situation of conflict. Suppose also that S_f is configured to always choose the *Pass* option.

Initially, all implications are T; then, a Chicken (subtype of Bird) x is created and its field `flier` is set to `null`, so the direct evaluation of the constraint $\sigma^\star$: Bird.flier == true returns ?. After its insertion, r_1 and r_2 are T-activated, while r_3 is ?-activated. The agenda is non-

deterministic, so suppose that the rules will activate in order r_3, r_1, r_2, causing the greatest number of revisions. The first activation of r_3 returns that it is unknown whether x flies or not. When r_1 fires, the combined information for $\sigma^\star$ evaluates to T, so r_3 entails that x flies. The eventual activation of r_2 revisions the belief: the attacking F it contributes discounts the existing evidence to ?, so the aggregate degree becomes F. It also sets the implication $\text{Bird}(x) \rightarrow \text{Flies}(x)$ to F, in turn changing the contribution of r_1 to ?. In either case, the merge at the constraint node is no longer conflictual. At the end, the consequence degree of r_3 is F (x does not fly), while the degree of r_2 is T; r_1 has a true premise, but its implication is false for x, as expected given the state of its conclusion, false, and its premise, true.

EXCEPTIONS IN POSSIBILISTIC LOGIC. The same network can be used with fuzzy interval-valued possibility degrees annotated with confidence. They are in the form $[N, \Pi]_\chi$, where N and Π are, respectively a lower bound of the necessity of a constraint and an upper bound of its possibility ([113]), while χ is the confidence degree. The engine configuration parameters are the same for possibilistic logic, even if the filtering strategy is set to always *Pass* as in the previous example.

Suppose that the degree of $\rightarrow_1$ is set to $[.8, 1]_{.5}$, possibly because it has been learned by induction over a limited set of example birds (low confidence), not all of which were fliers (necessity < 1). Given a Bird x, rule r_1 returns $[.8, 1]_{.5}$. Since it does not trigger the exception, this degree is merged with the one resulting from direct evaluation at r_3. In particular, if x were known to fly, the evaluator would yield $[1, 1]_1$ so the merge would be true. If, instead, the field x.flies is null, the overall result is $[.4, 1]_{.5}$ since, due to the low confidence, N has been discounted; evaluation at r_3 returns $[0, 1]_0$, while the premise of r_2 is false, so its activation degree is $[0, 1]_0$, which alters neither r_3 conclusion nor r_1 implication. When, instead, x is a Chicken, the system behaves like in the boolean case, overriding the contributions of $\rightarrow_1$ and r_3 to $[0, 0]_1$. A schema of the final configuration (valid for both cases) is given in Figure 38.

8.2 HYBRID APPLICATIONS

This Section shows how a rule-base relates to tools which are usually considered "intelligent" in some sense, but are not purely symbolic (such as Fuzzy Logic Systems or Bayesian Networks) or even completely connectionist such as Neural Networks. Purely algorithmic modules, instead, are not considered: generally speaking, however, they can be invoked in the RHS of a rule or wrapped in custom evaluators as shown in Chapter 7.

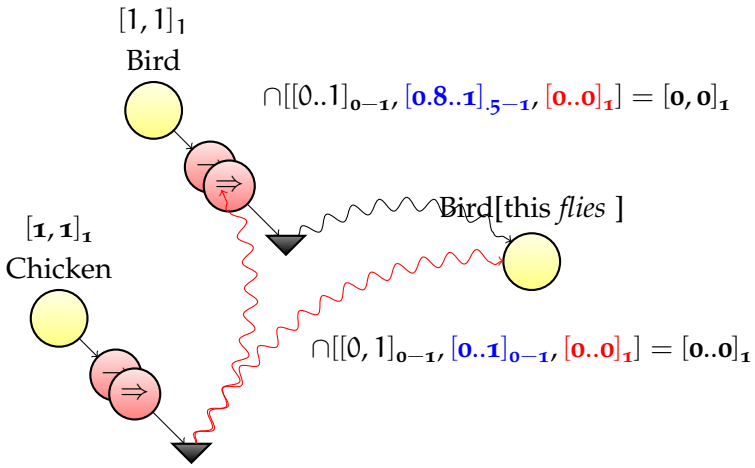

Figure 38: Defeated Rule

(Fuzzy) Feed Forward Networks

The first example shows some possible ways to integrate a rule base with a neural network. As outlined in Chapter 4, NNs have the disadvantage of being black-box models, so it is difficult to verify their behaviour and guarantee the correctness of their outputs. A common approach is to extract a rule base which emulates the network, exploiting the rules' explanatory capabilities ([283],[284]). Here, instead, the focus will be on the dual approach, i.e. how to embed a neural network into a rule base at different levels.

Cascaded Integration

The easiest way is to create a loosely integrated, *cascaded* system by having a rule invoke a neural network, as in rule 8.10. This method allows to use a pre-existing component, but the two components are essentially independent.

Listing 8.10: NN Invocation

```
rule "NeuralNet v1"
  when
    $x : Input( )
    $n : NeuralNetwork( )
  then
    $n.process($x);
end
```

Embedded Integration

In a more tightly coupled system, a neural network is used as an *external* evaluator. To do so, the output of the network must have the semantics of a degree: so, classifier networks and fuzzy networks are suitable, while predictor networks are not. The former, in fact, return a degree of belief (resp. of truth), while the latter returns a value.

Example 8.11 models a classical application of neural networks: character recognition. It is supposed that a network exists, with N input

Listing 8.11: Neural-based Evaluation

```
rule "NeuralNet v2"
  when
    CharacterShape( pattern isLetter "A" )
  then
    ...
end
```

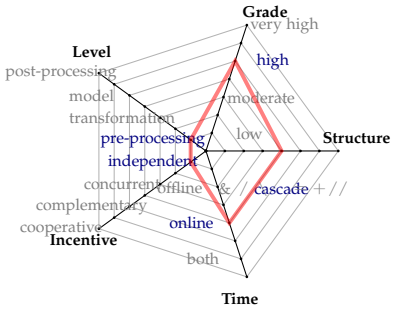

Figure 39: NN Invocation

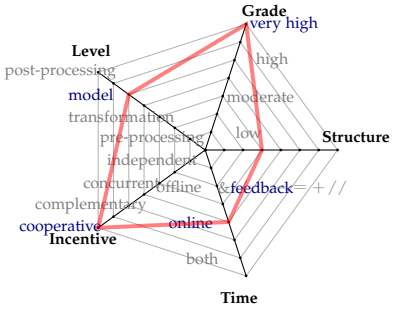

Figure 40: NN Hybridization

neurons, one for each pixel of an input image, and M output neurons, one for each symbol. The evaluator `isLetter` wraps the network and feeds it the left input (the pattern), then it uses the right input to select the output neuron. If necessary, the attributes `@kind` and `@params` can be used to pass additional configuration information. Notice also that, according to the actual structure of the network and the training procedure, the output can either be considered a degree of probability or of similarity and so should be treated accordingly.

Full Emulation

The integration of more generic types of networks is possible by having the rule engine *compute* the output of the network instead of just invoking it more or less explicitly. In fact, the enhanced engine supports the implementation of different rule-based neuronal models. In [139], a first model of fuzzy neuron is proposed which implements the max/min compositional principle exploiting the T norms and conorms of many-valued logic. A possible implementation of this model is shown in 8.12: it relies on (i) the fuzzy custom evaluator `seems` to convert a quantitative input into a fuzzy one; (ii) gradual rules to scale it; (iii) chaining to propagate the weighted inputs and (iv) the `or` connective to combine them.

Fuzzy Neuron

Generic Neuron

Other types of network can be emulated simply changing the `@kind` of the operators involved: the (bounded) linear perceptron is obtained using the product T-norm to implement modus ponens in the first level and the bounded sum T-conorm in the second level. The use of sigmoidal activation function, instead, is slightly more complicated, since the argument of the sigmoidal function $\sigma(\sum_j w_j \cdot x_j)$ is not neces-

Listing 8.12: Rule-based Fuzzy Neuron

```
rule "Link_j"  // parametric: one rule for each j
  entailment @[ kind="min"]
  implication @[ degree="..." ]  // constant degree = w_j
  ruleflow-group "Charge"
  when
    Input ( value seems "A_j" )  // evaluates x_j
    ...
  then
    inject(new Yj(),"idLinkj")
end

rule "Neuron"
  ruleflow-group "Discharge"
  when
    or @[ kind="max" ] (
      ...
      Yj ( ... ) @[ id="idLinkj" ]
      ...
    )
  then
    ...
end
```

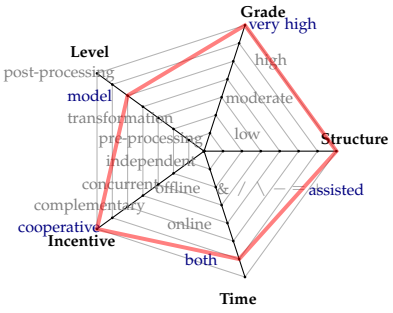

Figure 41: NN Emulation: Hybridization

sarily bounded. However, it is possible to exploit the concept of fuzzy *interactive* sigmoidal-or $\uplus$ defined in [139][2]:

$$\sigma(\sum_j w_j \cdot x_j) = \uplus_j \sigma(w_j \cdot x_j) \tag{8.1}$$

$\uplus$ can be used to implement the `or` connective; however, it is also necessary to alter the definition of modus ponens, to have it apply a sigmoidal transformation to its output after combining the premise's and the implication's degree.

Notice, however, that so far no rule-based version of a training algorithm has been implemented, even if the injection of the implications' degrees could allow to alter then dynamically.

2 actually, two kinds exist for the `tansig` and `logsig` activation functions

A "Fuzzy" Logic Controller

Fuzzy Systems are another common logic-inspired SC application, of particular interest in the development of control systems. As described in Chapter 3, standard Fuzzy Logic controllers use linguistic expression such as "*if X is A_j then Y is B_k in some degree* μ", where X and Y are objects and A_j and B_k are *linguistic* values from a fuzzy partition defined on the domain of some "hidden" quantitative variable $X.v$ and $Y.w$. A more formal description, also more suitable to be represented in a rule-based system, is the rule-equivalent:

Fuzzy Rules

$$A_j(X.v) \Rightarrow_\mu B_k(Y)$$

It states that the more the "hidden", but *perfectly known* variable v of X makes it a member of the fuzzy set A_j, the more Y can be considered a member of the fuzzy class B_k, up to a degree μ. Thus, fuzzification corresponds to evaluating the membership degree $A(X)$, inference consists in applying modus ponens to compute $B(Y)$ and defuzzification allows to obtain the quantitative value of the "hidden" variable $Y.w$, given the possibility distribution induced by B_k. In practice, however, the situation is more complicated:

- The premise may be formed by a conjunction of predicates $\Rightarrow$ an operator $\wedge$ is required to combine the different degrees.
- The conclusion degree may be limited by a value $\mu \Rightarrow \mu$ can be considered an implication degree: using a T-norm in the modus ponens operator ensures that the rule conclusion degree will not exceed μ.
- More than one rule may have the *same* fuzzy set for conclusion $\Rightarrow$ The various degrees must be combined using a strategy $\cap$, which usually coincides with the max operator $\vee$.
- Rules may entail degrees for more than one fuzzy set $B_{k_1,..,k_n}$ in the range partition $\Rightarrow$ The defuzzification operator must perform the union of these set, weighted by their associated degrees, before applying its defuzzification strategy.

Implementation Analysis

This schema translates easily in the proposed framework, as shown by the simple example in rule 8.13, where the temperature of a room is controlled by the speed of a fan. Other than the configuration of the operators and the default filter strategy, set to always `pass`, the rules deserve more comments. The rules exploit the concept of Drools Flow rule-flows for synchronization and are actually executed cyclically: the first rule (which is one of many similar ones, covering the other possible combinations of temperature and speed) determines the new value of the fan; when all of them have fired and all the contributions have been injected, the second class of rules can assign the aggregate degree to each fuzzy set in the target domain. Eventually, the overall possibility distribution can be defuzzified to update the speed of the fan. Notice that, when created, the `FanCommand` does not have its `speed`

value set, which is rather determined by the fuzzy rules. In practice, assuming that the temperature partition has N fuzzy sets and the speed partitioned is composed by M sets, $N \times M$ are required in the first group, M in the second and 1 in the third.

Listing 8.13: Fuzzy Controller

```
declare Fan
  speed : SpeedPartition
end
declare Room
  temperature : TemperaturePartition
end
declare FanCommand
  fan : Fan
  speed : SpeedPartition
end


rule "Sense 1"
  entailment @[ kind="Min"]
  implication @[ degree="..." ]  // mu here
  ruleflow-group "Sense"
  when
    $r : Room( temperature seems "hot" )
      and @[ kind="Min" ]
    $f : Fan( speed seems "slow" )
  then
    FanCommand fc = new FanCommand($f);
    inject(fc,"idFast");
    ...
end

rule "Update 1"
  ruleflow-group "Update"
  when
    $fc : FanCommand( speed seems @[ id = "idFast" ] "fast" )
  then
    $fc.update("fast",consequenceDegree);
end

rule "Act"
  ruleflow-group "Act"
  when
    $fc : FanCommand( $fan : fan )
  then
    $fan.setSpeed($fc.defuzzify());
    update($fan);
    // wait some time...
```

Fuzzy Partitions

To be implemented, this schema requires that the field of the objects are not simple values, but rather described by a fuzzy partition. To do so, a class `FuzzyPartition` has been created: it wraps a `Number` field, but it also refers a collection of fuzzy sets, each identified by a name, and a collection of degrees used to α-cut the fuzzy sets, as shown by the diagram in figure 42. The fuzzy sets, in fact, are used twice in the process: by the defuzzification strategy, which also relies on the α-cut

"Seems" Evaluator

degrees, and by the custom evaluator `seems`[3]. This binary evaluator accepts two arguments: the left one is a fuzzy partition, while the right one is a linguistic value. It extracts the wrapped numeric value and the designed fuzzy set, then has the set compute the membership of the value. This approach was used because the semantics of linguistic variables such as "hot" and "fast" depends on the actual domain, so it is unfeasible to define one custom evaluator for each fuzzy set.

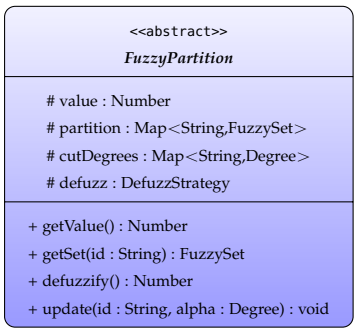

Figure 42: Fuzzy Partitions

OBSERVATIONS AND CRITICISMS ON THE APPROACH While widely applied in practice, this inference schema is questionable from the strictly logical point of view ([144], [117]). Among other things, it assumes that the original input $X.v$ is *perfectly* known. This is a rather strong assumption, especially considering that, when rules are chained, the intermediate values are fuzzy sets: the defuzzified value is only an arbitrary approximation, useful when decisions have to be taken but not during an inferential reasoning. In general, the narrow-sense interpretation of fuzzy logic in a broader sense models the latter as a mathematical logic which reasons with and over possibility distributions. In fact, when Zadeh's generalized modus ponens is applied, both the fact P' and the implication's premise P are expressed in term of fuzzy sets which must be intersected.

Given a target domain set B, the rules described so far can be summarized as:

$$\cup_j \left[A' \star (A_j \rightarrow B)\right] \tag{8.2}$$

But this is a less informative approximation (in the sense of fuzzy set inclusion) of the relation:

$$\cap_j \left[A' \star (A_j \rightarrow B)\right] \tag{8.3}$$

which, in turn, approximates Zadeh's composition principle

$$A' \star \left[\cap_j (A_j \rightarrow B)\right] \tag{8.4}$$

3 "seems" was used in place of the more common "is" not to generate confusion with other semantics

The problems become more evident if 8.4 is rewritten in expanded form:

$$B(Y.w) = \sup_{v} \min \left\{ A'(X.v), \min_{j} (A(X.v) \rightarrow B(Y.w)) \right\} \tag{8.5}$$

The approximations introduced by 8.2 are *(i)* assuming that $Y.v$ is perfectly known; *(ii)* that the rules' implication degrees are constant and that *(iii)* the rules can be separated. Unfortunately, the current state of the art does not support the full composition principle because possibility distributions are not allowed as inputs. However, Mamdani's approximation turns out to work in practice and can be used. Moreover, the effect of using different operators and merge strategies in the inference process can be experimented with using an applet provided by Dr. Wulff's research group[4], who also contributed to the development of fuzzy systems in Drools [17].

8.2.1 *Embedding a fuzzy ontological reasoner*

In addition to fuzzy controllers, fuzzy logic has several other practical applications. Recently there has been a growing interest in the combination of logic rules and ontologies. Notably, many works have focused on the theoretical aspects of such integration, sometimes leading to concrete solutions (e.g., [36]). On the other hand, there is some interest in the combination of ontologies with imperfect variants of description logic, in particular using fuzzy concepts (e.g. [313], [60]). At the moment there exist several engines integrating semantic descriptions and rule bases (notably DR-Device [173], which also offers support for defeasible logic), as well as a few ontological reasoners supporting fuzzy descriptions (notably fuzzyDL [281], which offers support for general many-valued logics), but a solution integrating all three aspects has not yet appeared as a mainstream tool to this date. Until a full *tight* integration can be achieved (some work has been planned in Drools as well), it is still possible to use a loose integration schema. The presence of custom evaluators makes the task trivial in Drools, as shown by rule 8.14.

The rule uses the custom evaluator `isA` to perform a generalized type check: this is still done after the strict `instanceof`, even if, should the approach prove to be useful and largely applied, the pattern `this isA` could be used to override the evaluator in the type node directly.

Regardless of its position in the RETE network, the evaluator `isA` uses reflection to convert the left argument into a representation compatible with the one required by the ontological reasoner (in the specific case, fuzzyDL[5] was used) and queries it according to what specified using the right argument. The result of the query is then wrapped into a `Degree`: since fuzzyDL is a many-valued reasoner using real val-

4 http://www.lab4inf.fh-muenster.de
5 http://gaia.isti.cnr.it/ straccia/software/fuzzyDL/fuzzyDL.html

Listing 8.14: Embedding a (fuzzy) DL Reasoner

```
rule "Ontology-based"
when
  Car( this isA @[ params="Cars.txt" ] "SportCar"
         && @(kind="Lukas")
       price not ~seems "low" )
then
        ...
end

/* Cars.txt - from http://gaia.isti.cnr.it/~straccia/software/
    fuzzyDL
(define-modifier very linear-modifier(0.8))
(define-fuzzy-concept High right-shoulder(0,400,180,250))
(define-concept SportCar (and Car (some speed (very High))))
*/
```

ues in $[0, 1]$, the conversion is trivial. This degree can then be combined at rule level, as if it had been obtained evaluating a normal constraint.

This approach is still rather experimental, and the implication in terms of performance have not been fully tested yet. Simple queries like the one in example 8.14 can be emulated directly rules, either using injection like in example 8.15 or, given the simplicity of the example, more directly by evaluating the constraint directly in the `Car` pattern. Nevertheless, this example may possibly become of some importance when large *legacy* ontologies are available and their translation in not a feasible option.

Listing 8.15: Sport Car : Rule-Based Version

```
rule "Fuzzy eval"
when
  $c : Car( speed very seems "high" )
then
  inject($c, "idSportCar");
end

rule "Ontology-equivalent"
when
  Car( this isA @[ id="idSportCar" ] "SportCar"
         && @(kind="Lukas")
       price not ~seems "low" )
then
        ...
end
```

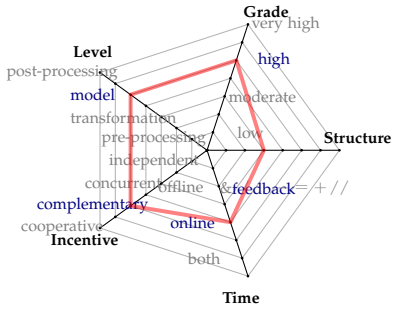

Figure 43: Rule/Ontology Hybridization

8.2.2 *A simple Bayesian network*

Reasoning with a truth-functional logic such as fuzzy logic or its variants is a computational advantage, but also a formal disadvantage when the knowledge base is affected by uncertainty. To handle it while ensuring the probabilistic coherence of the entailed information, the optimal tool balancing complexity and correctness is a Bayesian Network (see [237] and Chapter 3). While there exists ways to translate a rule base into a BN (e.g. [174]), the converse is not trivial (e.g. consider the already cited BLP [167]). The goal of this section, instead, is to show how to emulate the behavior of a small BN using some complex custom operators, adapting the message-passing propagation algorithm to the reasoning schema implemented in the enhanced RETE engine. This configuration is in no way meant to be used to implement BN, but it could allow a seamless integration between a rule base and one or more Bayesian sub-networks interfaced through custom evaluators. The performances of a rule-based version are surely worse than a dedicated implementation, but could grant access to the other features of the rule engine, such as side effects and integration with different types of logic, at least for some very specific BN nodes. In order to realize the integration in practice, several assumptions are required:

Mapping the Message Passing on Rules

- The distribution ε_i for a BN node X_i is given by the aggregation of several contributions: one, π_i, coming from the parent set Π_i as a whole, plus one, λ^i_k, from each child node C^i_k. Dummy children model priors (esp. for root nodes) or direct evaluations (esp. for leaf nodes). Each contribution is a $|X_i| \times 1$ vector and can be stored in an `Eval`: π_i occupies the third slot, while λ_k is stored in the k^{th} slot of the internal degree array. Since the contributions are independent, the merge operator $\cap$ is actually the elementwise matrix product (i.e. $C[i,j] = A[i,j] \cdot B[i,j]$), followed by normalization.
- X_i sends a message to each parent P_j and cooperates to the construction of the message for C_k with its other parents. For each

node X_i the following $n+1$ rules are written (notice that $\rightarrow_i$ is shared):

$$\vee_j P_j \rightarrow_i X_i \tag{8.6}$$

$$X_i \wedge (\vee_{j \neq j^*} P_j) \rightarrow_i P_{j^*} \tag{8.7}$$

- Rules are fired whenever the truth degree ε_i changes, until stability: S_f is fixed to `pass`.
- Computing π_i for rule 8.6: All the possible combinations $\mathbf{u_z}$ of the n parents of X_i are $Z = \prod_{j=1}^{n} |P_j^i|$; moreover, the combinations can be ordered lexicographically according to the order of the nodes and an internal ordering of each discrete domain. An aggregate prior vector, defined as $p(P) = p(\mathbf{u}_{z:1..Z}) = \prod_{j=1}^{n} \frac{\varepsilon_j}{\lambda_i}(\mathbf{u}_z[j])$, can be computed by an operator, $\vee$, since all information is stored in the Evals. A conditional table $p(X|P)$ is provided and ordered such that each column z models the probability $p(X_i|\mathbf{u_z})$: this table is the truth degree of the $\rightarrow$ node. The matrix product $\star$ can be used as Modus Ponens operator to obtain $\pi_i = p(P)^T \star p(X|P)^T$.
- Computing $\lambda_{j^*}^i$ for rules 8.7: The value to be injected is $\frac{\varepsilon_i}{\pi_i}^T \star p(X|P) \star p(P \backslash P_{j^*})$. In order to distribute the evaluation between the operators $\vee$,$\wedge$,$\rightarrow$ and $\Rightarrow$, while still being able to use the same to compute π_i, the following final definition is given:
  - $\vee$: computes $p(P \backslash P_{j^*})$ using Function 5. P_{j*} may be $\varnothing$.
  - $\wedge$: extracts $\frac{\varepsilon_i}{\pi_i}$ from its left operand.
  - $\rightarrow$: returns $p(X|P)$
  - $\Rightarrow$: computes $\varepsilon_\wedge^T \star \varepsilon_\rightarrow \star \varepsilon_\vee$. If $\varepsilon_\wedge$ is not defined, it is set equal to 1.

EXAMPLE To better show the functioning of the system, consider the canonical example shown in Figure 44. One of the required rules, instead, is shown in listing 8.16. Each rule, is used to model an unidirectional communication link between two nodes, so each arc requires two rules. The atomic variables are boolean so, for each of them, the belief degree is given by the vector $[p(\mathtt{true}), 1 - p(\mathtt{true})]$; the degree associated to the implications, instead, is a 2-dimensional conditional table which size depends on the children of a node. In the example, it is set to a constant (meaningless) value using the `@degree` attribute, but, like any other degree, can be set by implication or deduction. The `@kind` operator, instead, determines the concrete implementation of the operators, which are built by a dedicated `Factory` (its configuration is summarized in Table 28). To allow all the messages to be propagated, the filter strategy is set to "always `pass`"; moreover, the related activations are "sticky": they are not removed from the agenda until all possible injecting rules can no longer be activated themselves. This, together with the signalling capabilities of the `Evals`, allows the active rules to exchange messages through mutual injection. Eventually,

Algorithm 5: {T,rowO,colO} = buildP(j*,j,T,n,rowI,colI,p)

```
for idx = 1 to |X_j| do
  p ← p0;
  if j ≠ j* then
    p ← p · (ε_j / λ_i)[idx]
  end if
  if j < n then
    {T, rowO, colO} ← buildP(j*,j,T,n,rowI,colI,p)
  else
    T[rowI,colI] ← p, colI ← colI+1
  end if
  if j = j* then
    colI ← (colI+1) mod |X_j|
  end if
end for
rowO ← rowI, colO ← colI
```

when quiescence is achieved, the engine can proceed to eval the consequences with their entailed belief degree.

The topic still requires more investigation, but the alarm example has been implemented in three stages: simple chain, tree and polytree. In all cases, the propagation has worked in both directions (depending on the initial insertions from the main program) and the results have been compared with the ones generated by a normal BN tool[6].

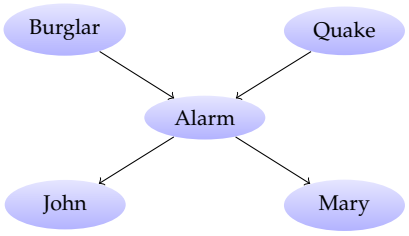

Figure 44: A Simple Bayesian Network

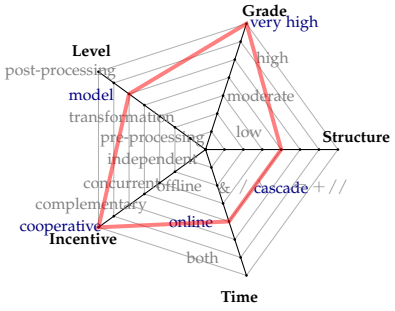

Figure 45: Hybridization Analysis

8.2.3 *The SOM training algorithm*

To conclude this section, a more articulated and complete example will be presented. The Self-Organizing Map described in Chapter 3 can be

6 JavaBayes : http://www.cs.cmu.edu/ javabayes/Home/

Listing 8.16: BN Rule

```
rule "lambda:Quake+Burglar->Alarm_1"
  implication @[ kind="bn", degree="(0.9 0.8 0.7 0.6 ; 0.1 0.2
      0.3 0.4)" ]
  ruleflow-group "Propagation"
  when
    Alarm()
    and @( kind="bn_and" )
      ((or @( kind="bn_or" args="index=1" )
        Burglar( )
      ))
  then
    Quake quake = new Quake();
      inject(quake,"idQuake");
    insertLogical(quake);
end

rule "Alarm"
  implication @[ kind="bn", degree="(1 0 ; 0 1)" ]
  ruleflow-group "Activation"
  when
    Alarm ( )
  then
    ...
end
```

Table 28: Configuration for Bayesian logic

Parameter	Description
$\mathbf{L}$	Probability distributions (joint and conditional) over finite domains, including boolean
σ	Field selectors
Ω	$\begin{cases} \wedge & \text{cause elimination} \\ \vee & \text{conditioning} \\ \rightarrow & p(\mathbf{C}\|\mathbf{P}) \end{cases}$ Not truth-functional
$\Rightarrow$	$\star$ (matrix product)
$\cap$	normalized $\odot$ (element-wise vector product)
$\mathbf{S}_\varnothing$	Set to uniform distribution
$\mathbf{S}_\mathbf{k}$	n.a.
$\mathbf{S}_\mathbf{f}$	`pass`

considered a neural clustering algorithm. It is recalled here to show how it can be used quite easily in a rule base.

KOHONEN'S ALGORITHM A detailed description of the original SOM training algorithm can be found in [149], so it will only be outlined here.

Given a training set $X = \{x_{s1}, .., x_{sD}\}_{s:1..S}$ of S samples in a D-dimensional space and a set $W = \{n_{j1}, .., n_{jD}\}_{j:1..N}$ of N neurons constrained on a d-dimensional lattice, the problem is to deploy the neurons in the space. The goal is to minimize the distance $\delta(\mathbf{n}, \mathbf{x})$ between each data point and its nearest neuron while preserving the neighbourhood relations.

1. *Initialize*: Choose the initial position of the neurons, usually at random or by using some elements of X.
2. *Sample*: Draw a sample $\mathbf{x}_s$ from X with probability $p(\mathbf{x}_s)$
3. *Match*: Find the winning neuron using a min-distance criterion:

$$\text{win}(\mathbf{x}_s) = \mathbf{n}_{j*} = \arg\min_j \delta(\mathbf{n}_j - \mathbf{x}_s)$$

4. *Update*: Move the winning neuron towards the input sample; its neighbour neurons are moved depending on their topological distance from the winner:

$$\Delta\mathbf{n}_{j*} = \eta(t) \cdot \nu(j, j*) \cdot (\mathbf{n}_{j*} - \mathbf{x}_s)$$

   The learning rate η determines how much the winning neuron should be shifted towards the input sample: it decreases with time to grant plasticity to the net in the early stages of training and stability in the late ones. The neighbourhood function ν has a single maximum for $j = j^*$ and models the links between neurons, causing the winner to drag its neighbours along as it moves.
5. *Continue*: Go to step 2 unless some condition is satisfied. Typical indicators include performance (e.g. the mean distance $\frac{1}{S}\sum_s \delta(\mathbf{n}_{j*} - \mathbf{x}_s)$), the number of training epochs t and/or the network stability (e.g. $\sum_{t-\tau}^{t} \|\Delta\mathbf{n}_{j*}(t)\|$ for some time window τ).

For simplicity, it will be assumed that the training data to be processed are 2-Dimensional, while the neurons lie on a 1-dimensional chain, so that the concept of neighbourhood is limited to a single dimension; moreover, all the coordinates are normalized so that the domain is $[0, 1] \times [0, 1]$.

Predicates

Before showing the DRL implementation, a first definition is provided using a predicate logic formalism. This allows to better explain the semantics of both the atomic and composite predicates. If not specified differently, the variable terms are actually structures storing the information on training `Samples` (`X`), `Neurons` (`N`) and time epochs (`T`).

`Samples` and `Neurons` have a field (position) which stores their coordinates; `Neurons`, moreover, have and identifier field.

Recall(N,X,T): "Neuron N recalls data X at epoch T"

Evaluation : $\exp^{-\frac{\delta(X,N)}{\sigma}}$

In a SOM, the activation of a neuron depends on the similarity between an input pattern and the prototype stored in the neuron. Thus, the evaluation exploits the monotonically decreasing relation between similarity and distance δ: the exponential form emphasizes nearby inputs, but a larger scope parameter σ can be used to have neurons recall less similar data to a higher degree.

Far(N) : "Neuron N is located far from the beginning of the linear grid"

Evaluation : $\frac{N.id}{|N|}$

The $|N|$ neurons are ordered by an index ranging from 1 to $|N|$: the value of the index expresses directly how "far" the neuron is placed along the linear chain. Notice, then, that neighbour neurons are "far" in a similar degree.

Young(T): "The Net is responsive to novel inputs"

Evaluation : $\frac{1}{1+\frac{T}{\tau}}$

The "age" of a network is measured in term of training epochs: a "young" neural network usually learns faster to fill its memory of meaningful data, while an "old" one must not forget the data already learned. In this case, the time constant τ models the speed at which the network ages.

Acknow(X,T): "The Net recognizes the input X (at time T)"

Evaluation : $\exists N : \text{Recall}(N, X, T)$

This predicate becomes more true the more the network could recognize an input sample. The overall degree depends on the degree of the neuron which was activated most.

Winner(N,T): "Neuron N has the highest activation at T"

Evaluation : $\text{Recall}(N, X, T) \equiv \text{Acknow}(X, T)$

With this formulation, the concept of "winner" neuron is gradual: the role of the winner is an ideal one, which is matched by the individual neurons depending on how much their activation is similar to the maximum one. Notice that no explicit ordering nor comparisons are performed as there can be many more-or-less winners.

FarActiv(T) : "The farthest part of the Net recognizes the input"

Evaluation : $\int_{Winner(N,T)} N : Far(N) dRecall(N, X, T)$

The degree of truth of this predicate is correlated to the location along the grid of the winner neuron. In fact, the position of the ideal winner neuron(s), modelled by `Far`, is averaged to locate the neighbourhood to move. The induction process is constrained to the winner neurons, and weighted by the degree of recall, as discussed in the induction example. The resulting position determines the neighbourhood to be shifted.

Near(N,T) : "Neuron N is topologically near to the winner position"

Evaluation : $FarActiv(T) \equiv Far(N)$

Before and during the organization phase, topological neighbour neurons may be far apart in space. In order to realign the network, an activated winner neuron activates its neighbours, albeit to a lesser degree, regardless of their position. To do so, a neuron's topological position is compared to the active neighbourhood: this compatibility is usually maximal for the winner neuron.

Fast(N): "Neuron N adapts easily to new inputs"

Evaluation : μ

This is a logical interpretation of a neuron's learning rate. Unless influenced otherwise, it is set to a prior, fixed value.

Lure(N,X,T): "Neuron N is attracted by input X at input T"

Evaluation : $\{Young(T) \wedge Fast(N) \wedge \texttt{very}Near(N, T)\}$

Reaction : $N.pos+ = \mu \cdot \varepsilon \cdot C \cdot (X.pos - N.pos)$

The more a neuron belongs to the active neighbourhood, the more it is to be attracted by the current input. This condition is further affected by three factors: first, the hedge `very` causes only those for which `Near` is true to a high degree to actually move. Second, as the number of epochs increases, the attraction and thus the learning capabilities are dampened. Last, the neuron's learning rate affects its elasticity. The overall consequence truth degree is eventually used to perform the actual neuron shift.

SOM Algorithm : DRL Version

The predicate logic version of the algorithm translates directly into DRL rules and can take advantages of the expressiveness of the language. All intermediate "virtual" facts can be declared in the knowledge base directly, so that only the real entities of the domain, samples and neurons, are effectively defined as Java POJOs. The rule base also exploit the sequencing capabilities of Drools Flow (see Chapter 6):

since the rules have a clear sequencing order, but they must be executed in parallel on the different neurons, the use of flow groups ensures that rule activations are not mixed. This is a safer approach than the use of salience or, worse, the introduction of synchronization facts. Finally, the rules make extensive use of attributes to influence the behaviour of the operators. Being many-valued rules, the same `Factory` used in Section 8.1.3 applies.

Listing 8.17: SOM Training, part I

```
global int N; // = 10

declare Young
end

declare Recall
        neuron : Neuron
        sample : Sample
end

declare Acknowledge
end

declare Winner
        neuron : Neuron
        id : int
end

declare HotSpot
        position : Double
end

declare Lure
        neuron : Neuron
end
```

Listing 8.18: SOM Training, part II

```
rule "Insert"
  ruleflow-group "Init"
  entailment @[ kind="equivalence" ]
  when
    $p : Sample( epoch young @[ params="0.015" ] )
  then
    Young young = new Young();
      inject(young,"idClsYoung");
    insertLogical(young);
end
```

Results and Extensions

The algorithm has been tested on some synthetic data sets, namely groups of 1000 random data points generated adding some white noise to the functions $y = x$ and $y = (x - 1/2)^2$. A 10-neuron network has been trained in the two cases, setting the values of the parameters σ, τ and μ to 0.05, 0.015 and 0.2 respectively (as shown in the DRL). The results, after 1000 training epochs, are shown in Figure 46 and 47.

Listing 8.19: SOM Training, part III

```
rule "Recall"
  ruleflow-group "Excite"
  entailment @[ kind="equivalence" ]
  when
    $p : Sample(  )
    $n : Neuron(  position near @[ params="0.05" ] $p.position )
  then
    Recall r = new Recall();
      r.setSample($p);
      r.setNeuron($n);
      inject(r,"idClsRecall");
    insertLogical(r);
end
```

Listing 8.20: SOM Training, part IV

```
rule "Acknowledge"
  ruleflow-group "Ack"
  entailment @[ kind="equivalence" ]
  when
    exists $rec : Recall() @[ id="idClsRecall"]
  then
    Acknowledge ack = new Acknowledge();
      inject(ack,"idClsAck");
    insertLogical(ack);
end
```

Listing 8.21: SOM Training, part V

```
rule "Winner"
  ruleflow-group "Locate"
  entailment @[ kind="equivalence" ]
  when
    ( equiv
      $rec : Recall( $n : neuron ) @[ id="idClsRecall" ]
      Acknowledge() @[ id="idClsAck" ]
    )
  then
    Winner winner = new Winner();
      winner.setNeuron($n);
      winner.setId($n.getId());
      inject(winner,"idClsWinner");
    insertLogical(winner);
end
```

Listing 8.22: SOM Training, part VI

```
rule "Position"
  ruleflow-group "Isolate"
  entailment @[ kind="equivalence" ]
  when
    forany (
      $n : Neuron( id far N )
      subject_to  Winner( neuron == @[cut] $n  ) @[ id="idClsWinner"]
      weight      Recall( neuron == @[cut] $n  )
                )
  then
    HotSpot position = new HotSpot();
      inject(position,"idHSfarN");
    insertLogical(position);
end
```

LEARNING WITH RELEVANCE The results have been obtained assuming that all inputs were equally important and relevant. Rele-

Listing 8.23: SOM Training, part VII

```
rule "Excitation"
  ruleflow-group "Update"
  entailment @[ kind="equivalence" ]
  when
    ( equiv
      HotSpot(position far
                      @[ id="idHSfarN" ]
                      N
      ) //injected
    $n : Neuron( id far N)
    )
  then
    Lure lure = new Lure();
      lure.setNeuron($n);
      inject(lure,"idClsLure");
    insertLogical(lure);
end
```

Listing 8.24: SOM Training, part VIII

```
rule "Lure"
  ruleflow-group "Update"
  entailment @[ kind="equivalence" ]
  when
    (and @[ kind="Lukas" ]
      $p : Sample()
      Young() @[ id="idClsYoung"]
      (very
         Lure( $neur : neuron fast @[ degree="0.2" ]
              ) @[ id="idClsLure"] )
      )
  then
    double alpha = drools.getConsequenceDegree().getValue();
    $neur.moveTo($p,alpha);
end
```

vance, in general, is instead a fuzzy concept: its definition depends on the actual data and, possibly, the sources they have been collected from. So, one can assume the additional predicate:

TrainSample(X,T): "The point X is a relevant training sample for the Net at time T"

Evaluation : varies

`TrainSample` can be set a priori, evaluated, or deducted from other rules. It is meant to replace the `Sample` pattern in rule `Lure` and can be used for many purposes: its crisp form selects the sample(s) to be presented to the network for sequential or batch training, since inputs for which its evaluation are do not cause the neurons to shift. Using intermediate truth values, instead, differentiates the samples by importance and conditions the outcome of the training. A simple test has been performed on the quadratic data set, considering the relevance to be function of the first coordinate: in particular, relevance was 1 for samples with coordinate $x < 0.5$; otherwise it decreased linearly to reach 0 for $x = 1$. The results are shown in Figure 48: notice that in this case only 2/3 neurons are used to cover the second half of the data, whereas in the case with equal relevance they were uniformly distributed.

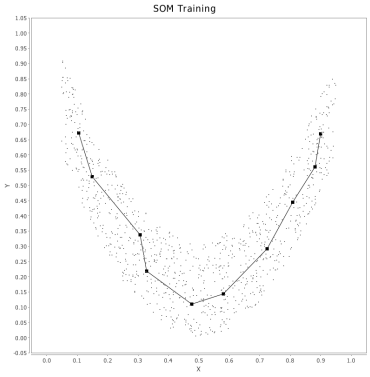

Figure 46: SOM with linear dataset

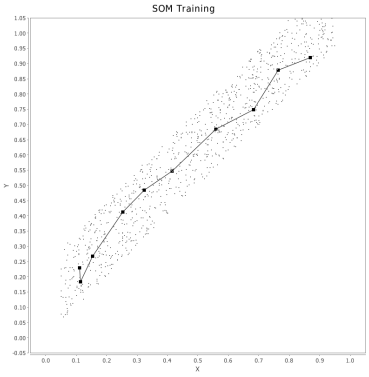

Figure 47: SOM with quadratic dataset

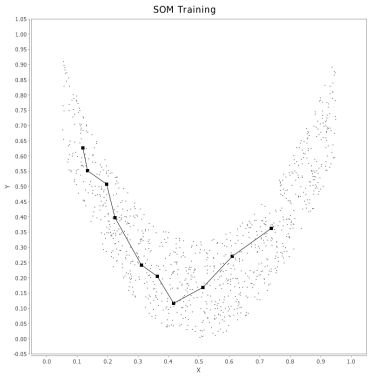

Figure 48: SOM with partial relevance

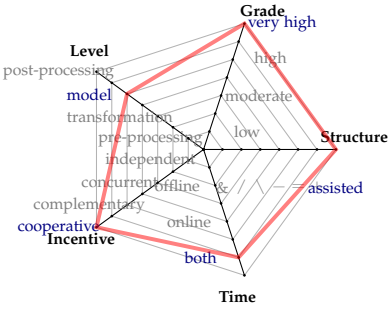

Figure 49: SOM Hybridization

SOM-BASED REACTIONS Remarkably, many of the rules such as the `Recall`, `Acknowledge` and `Winner` predicate/rules can be used not only for training, but also for normal recognition purposes. The updated neurons, in fact, remain in the WM and can be used in reaction rules like 8.25: the consequence degree, then, is function of the degree of similarity. Moreover, this degree can be controlled by the implication degree associated to the rule itself. It can be set a priori, but an induction schema such as the one shown in Section 8.1.5 can be adopted.

Listing 8.25: SOM Usage

```
rule "React"
  when
      Recall( $n : neuron, $x : sample )
  then
    //...
end
```

This is possibly the highest grade of fusion between the two systems: the rules are used both to train the network and to invoke it. From the point of view of hybrid systems classification, this can be considered an *assisted* system, since reactive rules like 8.25 rely on the result of a closed-loop feedback procedure - the training of the network - which was guided at rule level.

SOM VALIDATION Eventually, rules can be used to write validation expression, such as the ones in rule 8.26. The simple idea behind the first is that a neuron which does not effectively recognize at least one (training) sample is useless and so could be pruned or retrained. The second, instead, has a proper validation nature: it expresses, on average, how much the training set has been covered by the neurons.

8.3 CONCLUSIONS

While many of the examples presented in this Chapter are more patterns studied from a theoretical point of view than a set of concrete applications, the results are nevertheless promising. First of all, the language remains compatible with standard boolean logic, but adds support for several types of extensions with different semantics, from many-valued to possibilistic to probabilistic, provided that they are used in the context of production rules, which propagate the information entailed using modus ponens using a forward-chaining strategy. More interestingly, however, it was possible to integrate different SC techniques in a natural way. The simplest way is cascading, where a SC module is invoked to produce new data to be assembled into new facts. The second modality involves a feedback connection: the SC modules are used to evaluate logical properties, so they process data (terms) to return a degree which is reintroduced in the reasoning flow. More in-

Listing 8.26: SOM Validation

```
rule "Validate I"
  when
    $n : Neuron( )
    neg exists $r: Recall( neuron == @[ cut ] $n)
  then
    // Neuron is useless
end

rule "Validate II"
  when
    forany (
      ( $s : Sample( )
        exists Recall( sample == @[ cut ] $s)
      )
      weight TrainSample( sample == @[ cut ] $s)
    )
  then
    // Trainset is covered
end
```

terestingly, it has been shown that using imperfect reasoning it is also possible to *emulate* the behaviour of some non-symbolic tools, at least partially. In conclusion, an imperfect reasoning engine is not only a rule-based system, but a *polymorphic* component which can be conveniently used to build strongly hybrid systems, possibly composed by many (logically) interacting sub-parts. The next Chapters, then, will show a concrete application exploiting the potentialities of the novel component.

Part IV

CASE STUDY: A HYBRID ENVIRONMENTAL DECISION SUPPORT SYSTEM

9

SEQUENCING BATCH REACTORS - OPTIMIZATION

Contents

The examples provided in Chapter 8 may be interesting from a theoretical point of view, but they are abstract test cases rather than concrete applications. The last part of this dissertation, then, will be dedicated to the discussion of a more realistic case study which better justifies the use of all the techniques recalled or introduced in this work. In fact, the full features of the BRMS Drools, enhanced with the support for imperfection, will be used in the development of a complex management architecture applied to waste-water treatment plants, according to the principles outlined in Chapter 5. In particular, the research and development activities have been carried out on a pilot-scale treatment plant, choosing a plant from a class - the Sequencing Batch Reactors - which is particularly suitable for the application of control and diagnosis policies, to the point that the scientific literature - including several works of the author - is full of results, more or less relevant, on the automatic management of the process. Remarkably, in many cases the difference between a relevant result and a standalone study of little practical importance lies in the possibility of applying it in the context of a broader architecture providing, in a robust way, all the corollary services which are not strictly part of the technique itself, but are required for it to work in practice.

Before discussing the proposed architecture, however, this Chapter will be dedicated to the description of the particular applicative domain and to an overview of the various, standalone techniques which can be applied to it. The next Chapters, then, will show how it is possible to integrate them in a more general framework. It must be remembered, though, that this dissertation is not focused on water treatment itself, but on the applicability of AI techniques to it, so we excuse preventively if a competent reader will find in the description of the process rather simplified or imprecise. More detailed works, explicitly

Disclaimer

focused on SBRs and the related processes, can be found for example in [282] and [66].

9.1 BACKGROUND : SEQUENCING BATCH REACTORS

Sequencing Batch Reactors SBR are a particular class of activated sludge WWTPs, with a layout characterized by the presence of a single treatment tank where all the necessary reactions take place sequentially in time [19]. SBRs have several applications [188], but when used for urban waste-water treatment, the main reactions involved are the *nitrification* and *denitrification* of nitrogen compounds and the simultaneous removal of organic matter, as described in Chapter 5.

SBR Cycle

Due to the necessity of different environmental conditions, a SBR operates in a cyclic modality: a batch of water is loaded, treated alternating anoxic and aerobic conditions, and finally discharged. The effectiveness of the treatment process depends on many factors, including the hydraulic retention time (HRT) [297]: roughly speaking, this factor determines how long the polluted water remains in a tank and thus the amount of time it is subject to the particular environmental conditions which favour the different bio-chemical reactions. Thus, the timing of the phases must be chosen carefully, finding a trade-off between ensuring that the HRT is sufficient to complete the reactions and the contingent necessity to process a certain number of batches every day, fact which determines a limit to the maximum duration of a cycle.

The actual sequence of phases may vary according to the type and concentration of pollutants, but the "standard" cycle is composed by six (or seven) phases, ordered as in Figure 50. They are:

LOAD During this phase, a volume of water ΔV is loaded in the tank. This volume is a fraction of the total volume V of the tank - usually around $40 - 50\%$, even if it can be as low as 25% or as high as 70%. The loaded water is mixed with the activated sludge left in the tank after settling during the previous cycle. The duration T_{load} of this phase depends on the flow rate q_l of the pumps the tank is equipped with, according to the relation $T_{load} = \frac{\Delta V}{q_l}$. In any case, the load terminates when the water level in the tank reaches a maximum admissible level, or after a maximum time in case there is no water to treat for some reason.

REACT The reactions in the tank start as soon as the biomass gets in contact with the water to be treated, so this phase may effectively overlap with the load phase, depending on the load modalities. The phases can be considered distinct only if a *step-feed* load is performed: the water is brought in the tank very quickly and mixed with the sludge only after the load has been completed. If, instead, the mixing starts as soon as the load operation begins, the two phases are effectively concurrent. In any case, the reactions taking place during this phase can be controlled by dosing the concentration of dissolved oxygen in the tank:

Anoxic : When no free oxygen is present in the tank, the bacteria extract it from the oxidized forms of nitrogen (nitrates and nitrites), phosphor (phosphates) and sulphur (sulphates), *reducing* them to the basic elements. In particular, this allows to remove the nitrogen from the water, since it is released in the atmosphere in gaseous form.

Aerobic : The concentration of dissolved oxygen in the tank can be increased by blowing air in the water. When this concentration is sufficiently high, the bacteria consume ammonia - oxidizing it to nitrate - and organic matter - transforming it into carbon dioxide.

SETTLING : After the reactions have been completed, the blowers and the mixers (if any) are turned off, so that the sludge can settle by gravity on the bottom of the tank, becoming separated from the (clean) water.

DRAW : The clarified water is eventually removed from the top of the tank, extracting a volume ΔV equal to the loaded one. It is assumed that the sludge has had enough time to settle, otherwise the biomass would be discharged and lost: in fact, the inability of the sludge to settle is one of the main problems in WWTPs.

- Sludge draw: One of the ways to ensure a good sludge quality is to keep the *sludge retention time* (SRT) within certain specific ranges. In fact, the bacteria in the sludge are living creatures and, as such, have a limited life span: in order to remove the inert (dead) biomass, some sludge has to be removed periodically from the tank, albeit at a much slower rate and in much smaller amounts than the water.

IDLE : During this phase, while a new batch of water to be treated is collected, nothing happens in the tank.

As long as the treatment is concerned, the only relevant phases are the reaction ones. The concentration of pollutants in urban wastewater is usually low enough that a single sequence of anoxic/aerobic conditions is sufficient to meet the law requirements. However, to ensure that the HRT is sufficient for the reactions to be completed, the sub-phases must last for *enough* time. Unfortunately, the effective load of pollutants in the influent is not constant, but varies with factors such as the time of the day (few pollutants are discharged in the sewers at night, when people sleep), the period of the year (e.g. the population of an area may vary significantly during holidays), the presence of periods of intense rainfalls or draught and so on. Thus, the maximum duration of the phases is tuned on the average case, plus a relevant safety margin, with the ratio of the durations of the anoxic and aerobic sub-phases normally close to 1 : 2. However, it turns out that this conservative approach often leads to a waste of time (the reactions are completed well before the deadlines), energy (air blowers are expensive) and, ultimately, opportunities (with shorter cycles, more water can be treated). For this reason, much research has been carried out on

Phase Duration

the *optimization* of the duration of the SBR cycles, the topic which will be discussed in the next Sections.

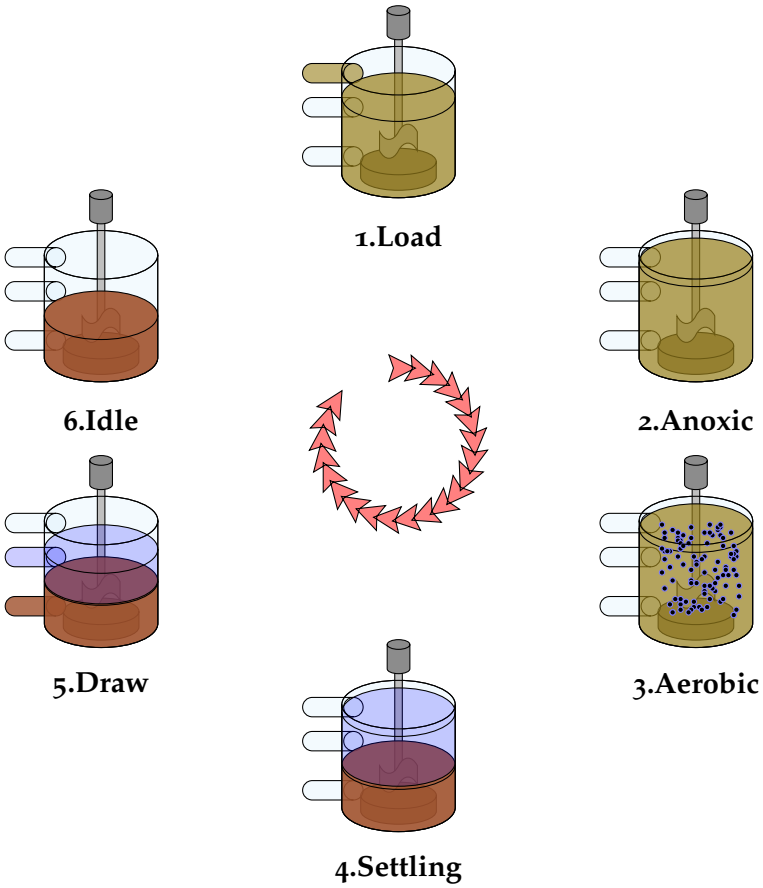

Figure 50: SBR Cycle

THE PILOT PLANT In order to experiment with the SBR technology and collect experimental data, a pilot-scale SBR plant has been built by ENEA and placed inside the municipal plant of Trebbo di Reno (BO, Italy). The plant, which has a working volume of 500L (of which, about 150L of working load) and an overall hydraulic retention time (HRT) of 20h, is fed with real sewage drawn after grit removal. Biomass is automatically wasted on a daily basis to maintain the sludge retention time (SRT) between 15 and 20 days. The plant is operated with 4 cycles a day, consisting in the canonical sequence of feed, reaction (anoxic, then aerobic), sludge wasting, settling, draw and idle phase (see Figure 50). It is equipped with a mechanical mixer, a variable-flow blower connected to a membrane diffuser, two peristaltic pumps for influent loading and effluent discharge (flow rate = 6L/min) and a pump for sludge wastage (flow rate = 1L/min). A multi-parameter converter continuously measures pH, ORP and DO signals during the whole process:

Pilot Plant Equipment

Phase	Max. Duration	Load Pump	Mixer	Blower	Sludge Pump	Draw Pump
Load	30m	√	√			
Anoxic	1h		√			
Aerobic	3h		√	√		
Sludge Draw	3m		√	√	√	
Settling	40m					
Discharge	30m					√
Idle	[1]					

Table 29: Pilot Plant Static Configuration

the samples are acquired by a National Instruments data acquisition board and stored in a MySQL database for both immediate and later access. The actuators, instead, are controlled by a PLC interfaced to the acquisition card. The operating phases, their maximum duration and the actuators active during each phase are reported in Table 29.

9.2 PROCESS OBSERVATION

As outlined in the previous Section, the critical point in the management of a SBR is the optimization of the duration of the reaction phases. In fact, the SBR *allows* to change the **HRT!** dynamically - an action which in other types of plants would require to change the *volume* of a tank - but to do so in a sensible way, one should be able to monitor the evolution of the treatment process as it takes place. In Chapter 5 it was noted that there exist online sensors which can sample the concentrations of nitrogen and carbon compounds, but their high costs make their use hardly convenient, especially since the lower costs are one of the strong points ofSBRs [223]. Instead, several authors have pointed out that there exists a strong correlation between the state of the process and the time evolution of three indirect signal, namely pH ORP and DO, which can be sampled using much cheaper instrumentations (e.g. see [33], [181], [78]). Figure 56 shows the result of a track study on the pilot plant: the measured concentrations of $[NH_4^+]$ and $[NO_3^-]$ confirm the theoretical expectations.

Signal Trend Explanation

As soon as the water is pumped in the tank, the process of denitrification starts: the reduction of nitrates to nitrogen increases the pH of the environment and lowers the oxidation potential to values around $-150mV$. After the nitrates are depleted, the biomass starts to reduce other oxidized compounds such as phosphates and sulphates, further lowering the ORP; at the same time, a decrease in pH can usually be observed. When the blower is turned on and the aerobic conditions are restored, the oxygen is consumed by the bacteria to oxidise the remaining organic matter and, most importantly, ammonia to nitrates (passing through the intermediate step of nitrite). For this reason, the

level of DO remains low until the nitrification process is completed, while the ORP is still increased by the presence of the oxidized compounds. The same reaction also causes the release of H^+ ions, which lower the pH. When nitrification stops, instead, the oxygen is free to saturate the water rapidly; at the same time, the pH increases again, possibly due to the stripping of CO_2 from the water which is no longer contrasted by the consumption of ammonia.

9.3 SBR MANAGEMENT: STATE OF THE ART

Accounting for Process Variability

Actually, the motivations adduced to explain the signal trends are not absolutely *certain*. Moreover, it must be remembered that nitrogen and carbon compounds are not the only chemical substances present in a waste-water and, especially in the case of real sewage, their presence (and their effective concentrations) can't be predicted a priori, but they could influence the process in some way. If one ignores the theoretical justifications and limits to the empirical observations, however, it turns out that changes in the signal trends may effectively be correlated to and denote the completion of the main reactions involved in the treatment process. From Figure 51, which shows a typical signal set acquired when the plant operated in normal conditions of efficiency, it is evident that the effective reaction time is much lesser than the corresponding phase time. As already pointed out, the phase time must take into account the variability of the required reaction time, which may be influenced by a number of factors including the state of the biomass, the variable load, the temperature, the presence of toxics substances inhibiting the bacteria, so the maximum duration is usually overdimensioned. An example of the variability of the signals, assumed due to variations of the process conditions, is shown in Table 30 at the end of this Chapter: the signals, acquired in subsequent cycles, show a progressive profile change during the anoxic phase. Safety, however, comes with a price, since much time and energy and time is wasted in the average case: in the example in Figure 51, more than 50% of total reaction time turns out to be useless.

In this context, the variability of the process conditions motivates the use of any technology allowing to control the process in real time, with the dynamic regulation of the phase durations being the minimum requirement to be satisfied. The indirect, non-linear correlation between the measured signals and the process evolution, in turn, makes it a relevant candidate for the application of AI techniques, especially *imperfect* ones. From an analysis of the literature, it turns out that almost all possible approaches (see Chapter 3 for their properties) have been applied with different degrees of success, even if their usage can be roughly divided in two main families:

Process Optimization Approaches

PROCESS STATE ESTIMATION These *quantitative* techniques try to predict the current level of concentration of the different substances such as nitrates and ammonia. If it is not possible to obtain quantita-

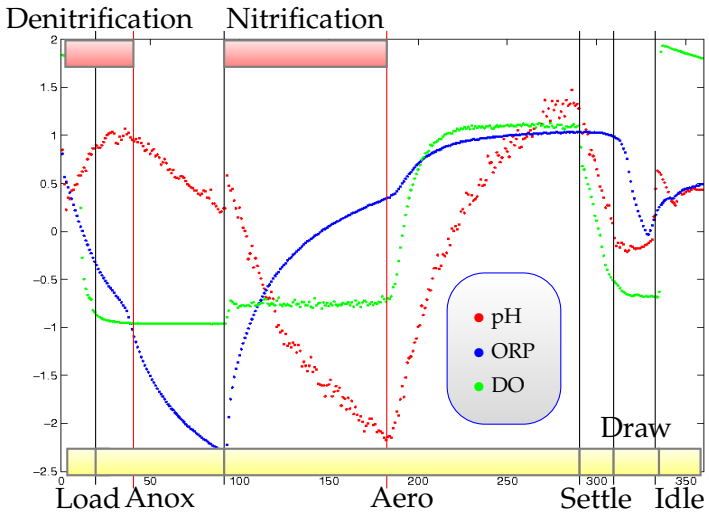

Figure 51: Evolution of pH, ORP and DO during an SBR process

tive values in a reliable way, a weaker form of estimation may be sufficient, which distinguishes between the mere *presence* or *absence* of a substance. To this end, specific instrumentation has been applied [129], but most of the times neural networks such as the FF-NN or the Elman network have been used ([183],[127],[32]). We tried this approach in [11].

In alternative, statistical approaches based on (Multi-Variate) Principal/Independent Component Analysis try to associate the observed data to a hidden, partially unobservable state which has not necessarily a direct physical meaning, but has the advantage of being learned from a historic data set in controlled conditions. At runtime, then, the same procedure is applied and the estimated "state" is analysed to decide whether the reactions can be considered complete or not. Examples of this can be found in [31], [303] and [85].

In [194], instead, the estimated state is given by the derivatives of the (denoised) indirect signals, but the matching between the current state and the desired ones is performed using fuzzy clustering. We tried a conceptually similar approach, but instead using a SOM for clustering, in [9] and evolved it in [4].

SIGNAL TREND CHANGE DETECTION Another large family of control methods is based on the detection of the *qualitative* trend changes (also called discontinuities, breakpoints or characteristic points) in the indirect signals, which are usually pH, ORP and DO. Alternatively, the trends themselves may be used [265]. Either case usually requires an analysis of the time derivatives of the signals: their numeric computation is the critical point of this approach since derivation is an unstable operation when the signal is noisy, so a some kind of filtering/denois-

ing algorithm is always applied (common choices are low-pass filters and wavelet filters). More generally, for each time t, some features of each signal σ are computed and analysed in an interval $[t - l, t + u]$, not necessarily symmetric, and then used to *classify* the content of the window. The possible *patterns* to be matched include:

- Maximum / Apex: a rising trend followed by a falling one.
- Minimum / Valley: a falling trend followed by a rising one.
- Knee : two consecutive falling (resp. rising) trends. The slope of the second is usually greater than that of the first.
- Step : quick shifts from one level to another.

In [152], the matching is performed using different types of NN; others, instead, prefer fuzzy rules to express conditions on the time derivatives, usually included among the features; we defined and matched such characteristic points expressing conditions on a sigmoidal approximation in [1] exploiting a result presented in [12]. When patterns are recognized in the signals, the recognition of the completion of the different phases can be expressed in terms of conditions on the patterns themselves. An (imperfect) rule-based approach would seem the most appropriate tool for this task, but this is not always the case. In fact, several works (e.g. [73], [66]) express the conditions in terms of flow diagrams, which seems to hint to a procedural approach rather than an "intelligent" one. The conditions, however, are almost always equivalent to logic conjunctive rules [1] such as:

`max(pH,T)` $\wedge$ `knee(orp,T)`

$\rightarrow$ `complete(denitrification,T)`

`min(pH,T)` $\wedge$ `step(do,T)` $\wedge$ `knee(orp,T)`

$\rightarrow$ `complete(nitrification,T)`

Nevertheless, this *perfect* formulation is too brittle to be usable in practice: the trend change patterns are rarely matched with precision by the signals to be analysed, and even more rarely at the same time. In fact, in [305] a disjunctive ($\vee$) form is preferred, but, more often, what is used in practice is a relaxation of the above constraints. A full treatment of imperfection is rarely applied, possibly because the ultimate decision - deciding whether to turn the air on or off - is crisp. Instead, partial matches are accepted when the similarity exceeds a threshold and a tolerance on the temporal alignment is admitted as well. In many cases, even this is not sufficient, so additional temporal constraints are added: the duration of a reaction must be comprised between a minimum and a maximum value. The upper bound is especially necessary because, most of the times, the algorithms are trained to recognize the features typical of a correct completion, but not the ones relative to a *failed* reaction. Thus, failures are undecidable: "*not-know*" situations, where the recognition algorithm fails, can't be distinguished from "*know-not*" ones, where the problem for some reason lies in the process. In either case, the upper deadline serves to resolve

Imperfection in SBR Management

the deadlock. While this approach justifiable for reasons of safety and soundness, it would be more coherent to handle the uncertainty on the correctness of the response directly at the AI level, which could also be trained to recognize and diagnose failures, instead of subordinating the "intelligent" control algorithm to a procedural one.

9.4 OFFLINE MANAGEMENT

The considerations made so far can be summarized as follows: the SBR process is cyclic, consisting in a sequence of phases, each one with a maximum duration. Some process phases, however, can be terminated before their deadline, according to some criteria which may involve the analysis of signals samples in the plant itself. The factors influencing the decisions may be as simple as "turn of the pump when the tank is full" or as complicated as the predictors described in the previous session. Nevertheless, this vision can be modelled conveniently using a Petri net [61] such as the one shown in Figure 52

Abstract SBR Model

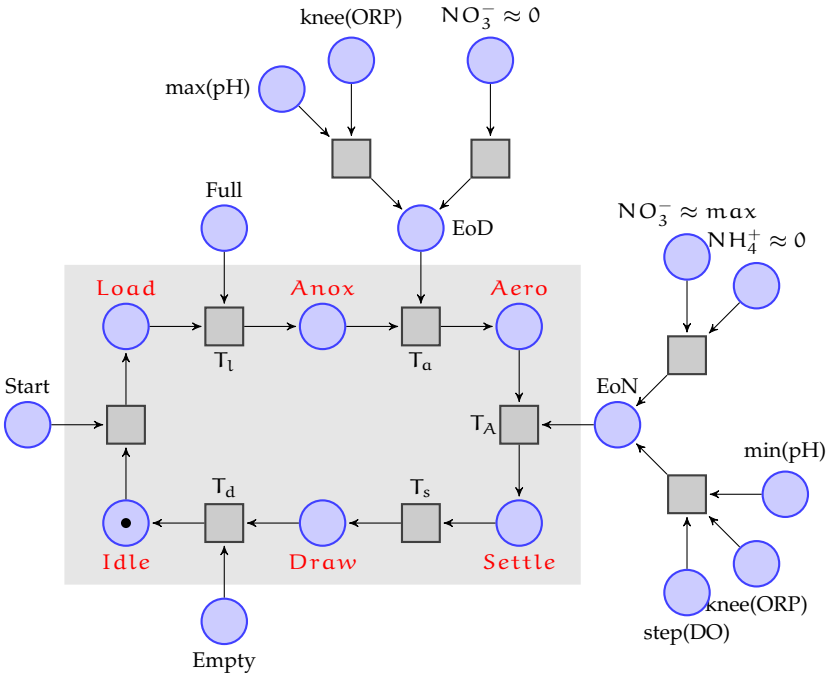

Figure 52: Petri Net model of an (optimized) SBR Cycle

This model is interesting for two reasons. First, the existence of alternative criteria should be considered an benefit for a management system: a system can implement two or more of them and have them reinforce each other. Moreover, should one not be applicable to a given input, there is still hope that another may entail something useful. In [11], three different techniques were applied, all relying on the sampling of pH, ORP and DO. Even if in that paper the analysis was per-

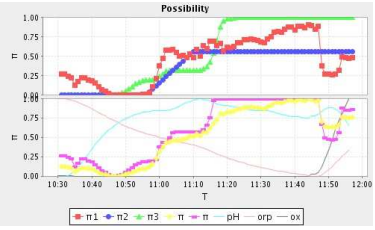

Figure 53: Combining Possibilistic Estimations

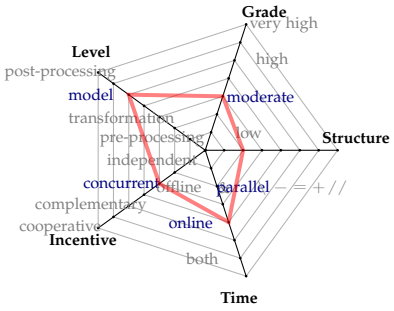

Figure 54: Hybridation Analysis

formed only on the anoxic phase, the techniques used are not dissimilar from the ones existing in literature:

- An Elman NN-based predictor, similar to the one used in [183] (of which the work is an evolution), but uses a SOM in parallel to estimate the reliability of the predictions.
- A signal pattern detection algorithm, dual to the one used in [265]. The focus is not on trends, but on trend changes; moreover, the features are extracted from the analysis of an approximation of the signals performed using sigmoidal basis functions.
- A SOM-based state tracking, comparable to the ones used in [303] and [194]. The signals and their derivatives are used directly as state indicators, but the trajectory in the state-space is analyzed using a SOM with the functions of a fuzzy clusterer.

Fusing Multiple Approaches

Interestingly, the modules tried to cope with the imperfection of the input data and the one added by the elaborations themselves. For them to comparable and combined, the response $\pi_j(t)$ of each module j was cast into the degree of *possibility* that the reaction had been completed at time t: the different degrees, then, were *merged* using a possibilistic combination function suggested in [109]. An example of its application is shown in Figure 53: in the upper part, the evolution of $\pi_j(t)$ is shown, while in the lower part the combined possibility $\pi(t)$ is compared to the input signals. The chart in Figure 54, instead, summarizes the characteristics of this approach from the point of view of the classification of hybrid systems.

SBR from a CEP Point of View

The second reason is that the Petri net suggest a (complex) event-based interpretation of the SBR management policies. A transition from one state/phase to another, in fact, is always triggered by an event: it is of little importance whether it is actually generated by a timer or by a physical source in the plant. In the case of reaction phases, in particular, the switch is expected to be generated by an "`EndOfReaction`" event, signalling the (correct) completion of the current reaction. This event is clearly complex: it is caused[2] by an implicit event signifying

2 in the CEP sense of causality

that a certain substance has reached concentration close to zero in the tank. This last event, which is unobservable, may be estimated directly or indirectly, by aggregation of simpler events detected on the probe signals, the trend changes, which in turn are detected analysing even simpler events, the samples coming from the probes. This vision has been adopted in [1], where the event analysis was performed offline, to validate the cycles after their completion rather than to control the process in real time (the conceptual schema is shown in Figure 55). Other than showing a first connection between plant management and CEP, the same paper also analysed, for the first time, the water treatment process from the point of view of business processes and business rules.

SBR from a BP Point of View

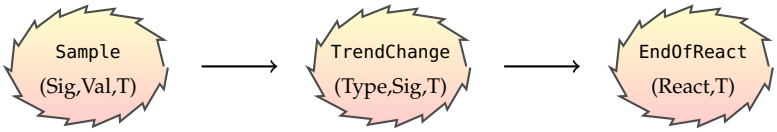

Figure 55: Basic Event Hierarchy

While interesting and complementary, the approaches followed in [11] and [1] are more suitable for an off-line analysis of the treatment process. The last step, then, consists in adapting and extending those concepts, deploying them in a suitable real-time management infrastructure.

9.5 CONCLUSIONS

This Chapter has shown the potentialities and the issues of using a SBR plant for water treatment. To achieve a satisfactory level of treatment efficiency while still being convenient from an economical point of view, a SBR has to be optimized, but this is not a trivial task. In fact, this class of plants turns out to be extremely relevant from a scientific point of view because it is the ideal candidate for the application of *imperfect* AI techniques.

The reasons are various and include:

- The management policies are based on good sense rather than an in-depth knowledge of the status of the process and the biomass
- A *partial* knowledge on the state of the process can only be obtained through complex, time consuming laboratory analysis.
- The criteria are based on predictions, or the evaluation of qualitative features of noisy indirect measures.
- The effective outcome of the optimization strategies is uncertain.

Given this scenario, it is not surprising that AI - and especially SC algorithms - is preferred. However, the variability of the conditions to be analysed makes it complicated for a single tool to be always successful

in optimizing the treatment processes: in fact, many of the proposals which can be found in literature are actually hybrid systems, either "pure" or supported by additional, more traditional control modules.

Notice also that, in order to be optimized, a process must work properly in the first place: for this reason, the problem of optimizing the performance of the plant can't be kept separate from a diagnosis of the health status of the plant, from the biomass to the electro-mechanical components.

This, together with the implicit requirement of usability by the plant operator, justifies the development not only of an intelligent control algorithm, but of a full remote EDSS integrating the different tasks, possibly aware of the event-oriented nature of the monitored system. In its development, the role of imperfection can't be ignored or just reduced to the application of a few fuzzy definitions. In fact, it is not an exaggeration to say that:

> Most existing SBR control policies are defined in ***vague*** terms, used to describe ***uncertain*** relations between a process and its observable variables.

The last Chapter, then, will show how to combine all the concepts introduced so far - namely imperfection-aware hybrid systems, rule-based CEP, SOAs and BRMSs - in the development of one such architectures applied to the real-time control and optimization of a SBR plant.

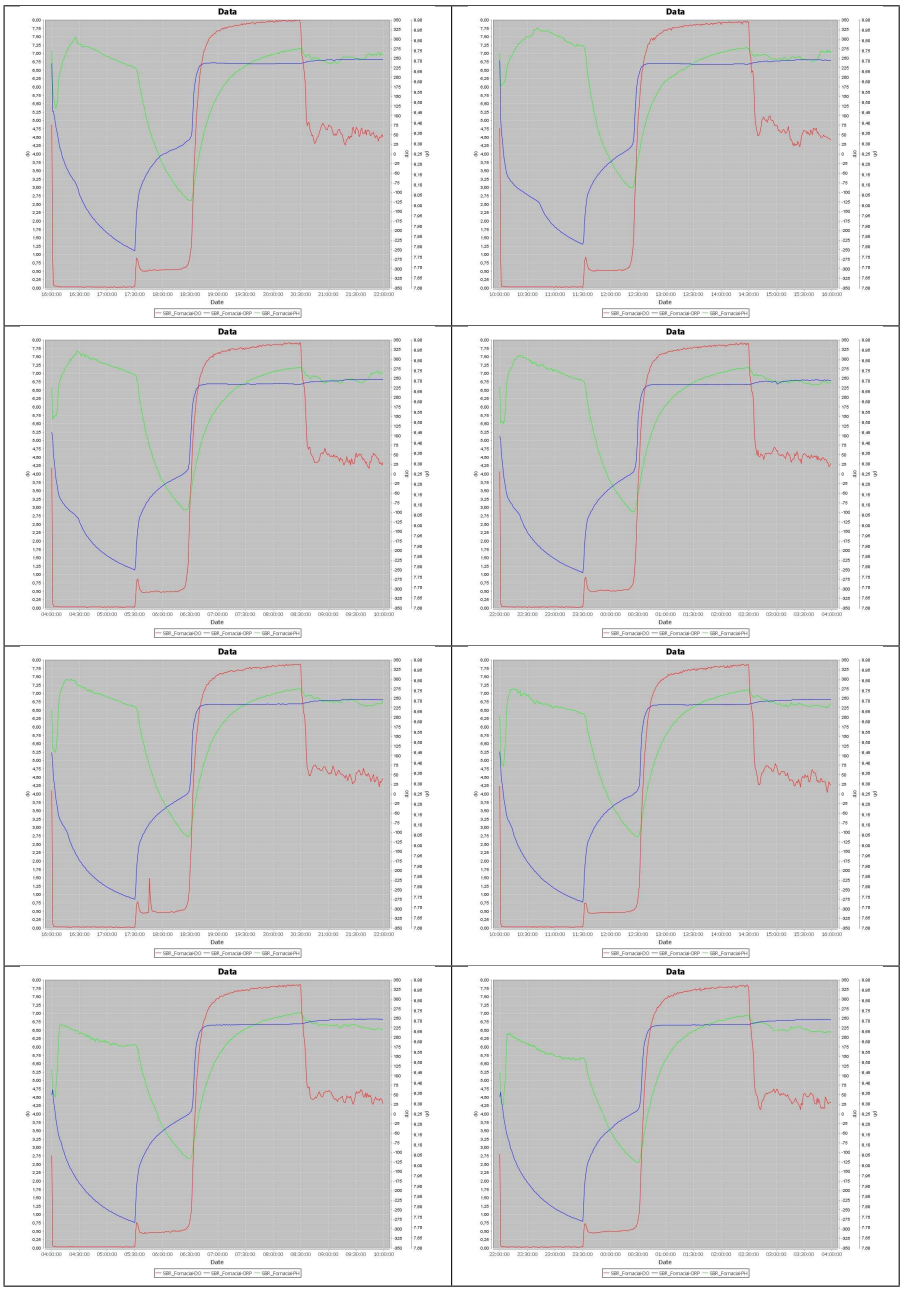

Table 30: SBR Signals

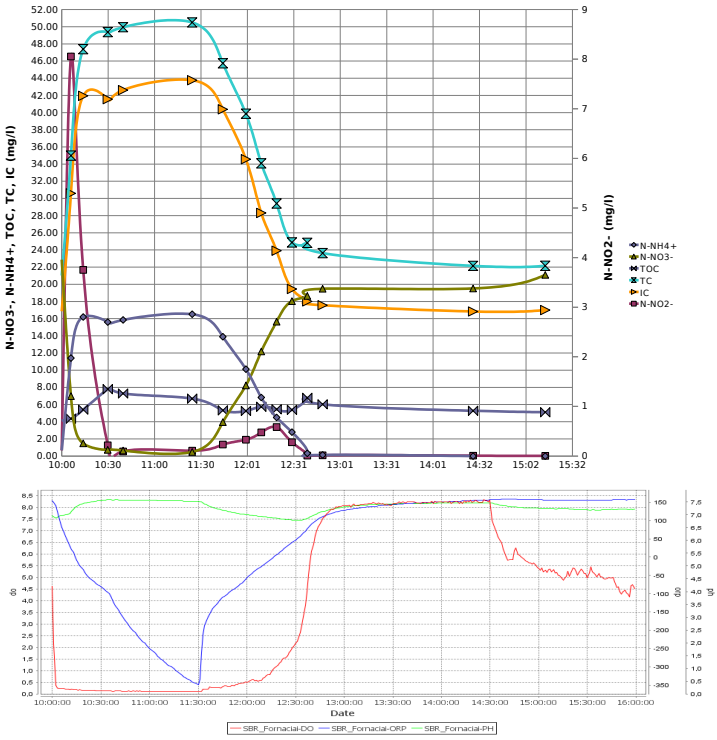

Figure 56: Track Study

10

DESIGNING A COMPLEX EDSS

Contents

This Chapter will discuss the design and implementation details of a hybrid EDSS infrastructure, applied to the real-time management wastewater treatment plants and, in particular, sequencing batch reactors. This infrastructure summarizes the results obtained trying to achieve all the objectives which motivated this whole work.

In Chapters 5 and 9, it has been shown that various reasons (environmental, energetic, economical, ...) have motivated much research on the automation of treatment plants, which is sponsored by private companies and public research centers alike. The very context in which this research has been carried out was born out of a cooperation between the university of Bologna, the municipal multi-utility HERA and the national agency for the energy and the environment, ENEA. This generalized interest, together with the complexity of the problem, has led to the development and application of a number of technologies, with varying degrees of success. However, the simultaneous and integrated application of the results, both found in literature and developed internally, requires necessarily a global support infrastructure like the one that will be presented here. A general-purpose middleware, in fact, provides all the "horizontal" services required to test and to run different control and monitoring modules and have them interact as needed, facilitating their development and (re)use.

In doing so, other advantages will be discussed at the same time:

- The use of a hybrid architecture, supporting both events and services
- The use of hybrid AI techniques, processing information affected by imperfection
- The use of open-source tools as integration platforms.

The proposed approach is general, but has been applied and tested with the problem of integrating the different optimization criteria studied for the pilot-scale sequencing batch reactor described in Chapter 9.

10.1 RELATED WORKS : COMPLEX MANAGED DOMAINS

As discussed in Chapter 5, many commercial EDSS are actually extremely valid remote management platforms, providing facilities for the operators but adding little from the point of view of automated "intelligent" control. More integrated solution of this kind have been devised in academic contexts (especially in Spain) and then applied to real plants. Such solutions, dating back about 10 years, introduce the concept of a modularized DSS, but still have a data-centric architecture (see Figure 57), where different modules work in parallel on the data, possibly under the supervision of a dedicated entity. More recent architectures, instead, adopt a more granular approach: SOAs, EDAs and combinations thereof stress, with different modalities, the role of interaction between different parts/entities/modules/agents, so that not only a single task is solved by the cooperation of more than one module, but the same module can also take part in different tasks. Moreover, in such architectures the distinction between the parts is not strictly hierarchical, but depends on the granularity and the generality of the capabilities of each module.

Here this second approach has been adopted. All the required functionalities are implemented as services (see Figure 58 for a conceptual schema), including both "standard" services, invoked explicitly by a consumer, and reactive, event-triggered handlers. In addition to what previously stated, the services are implemented by rule-based, intelligent agents. This is not a limitation, since Drools supports workflows, events and, as shown in Chapter 8, different types of imperfect reasoning and both tight and loose models of integration with various SC techniques.

Excluding private enterprise applications which are not divulged publicly, similar architectures are beginning to appear in other contexts such as the Semantic Web. The pre-eminent example is possibly the Rule Responder architecture [231]: in that case, however, the focus is on the coordination and the exchange of information between heterogeneous systems. The problem considered here, instead, is more oriented on data processing and imperfection handling, so the two systems remain complementary (and may benefit from each other in their future developments). The approach seems to be novel, instead, for the water treatment domain: complex applications exist based either

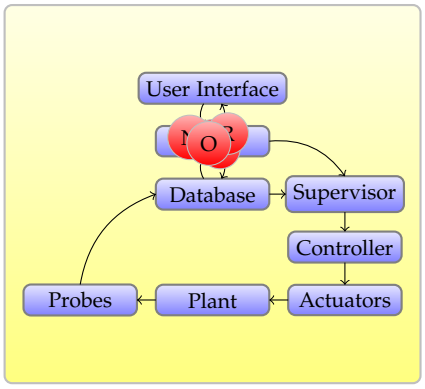

Figure 57: Data-centric Architecture

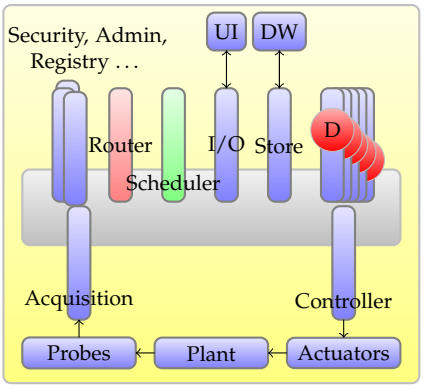

Figure 58: Service-centric Architecture

on agents, hybrid AI systems, events or services (see Chapter 5 for a survey), but none that we are aware of integrates all the aspects at the same time. Remarkably, recently there has been much interest in *dynamic signal trend* analysis for diagnostic and fault detection purposes ([80], [197]): these works, however, focus on a very specific class of events. This does not diminish their relevance - in fact, one of the modules of the proposed application uses a similar if more primitive approach - but they are often standalone solutions which, instead, could benefit greatly from being used in a wider context.

10.2 ARCHITECTURE

10.2.1 *Enterprise Service Bus*

Communication Middleware

Before discussing the structure of the agents, it is important to cite the communication infrastructure they rely on. Almost all comparable architectures use *Enterprise Service Buses* (ESB) at their core. An ESB is a software infrastructure providing support services to complex SOA architectures: it is a standard-based messaging engine acting as *message broker* between applications [79]. This allows to reduce the number of point-to-point connections and allows a reliable routing of the messages to the appropriate applications. The message model has to be standardized and shared between modules using the ESB, but adapters ("*gateways*") can be easily used to transform messages to and from legacy and custom formats, including JMS [13], SQL, HTTP/SOAP [62], e-mails and FTP.

The bus usually supports different types of communication models, including the canonical publish/subscribe and point-to-point schemas. Interactions, then, can be mono- or bidirectional, synchronous and asynchronous. A typical implementation may involve the use of reliable message *queues*, which store undeliverable message until the

intended receiver can retrieve them, and one-to-many *channels* (also called *topics*) for broad- and multi-casts.

Moreover, the service bus is inherently distributed and can be used to connect services deployed on different nodes in a way that is completely transparent to the services themselves. If needed, the bus automatically handles replication and load balancing, increasing the fault tolerance and the flexibility of the system: in fact, services can usually be hot-plugged on the bus, so the number and type of modules can change dynamically.

JBOSS ESB The proposed architecture has been implemented on top of the `JBossESB` 4.7 middleware, a Java-based, open source ESB with a set of features comparable with that of mainstream commercial products, which is also natively compatible with other JBoss products including Drools itself. (Notice that while convenient, this choice is not strategic as the ESB could possibly be replaced by another implementation).

The main features of JBossESB include the expected functionalities of an ESB, in addition to some "horizontal", generic services:

- *Reliable Message Delivery* : The ESB allows to create communication endpoints and the services can use them to listen for messages. These endpoints propagate ESB-oriented messages, implemented using a JBoss proprietary format: should a legacy format be used, specific gateway endpoints can be used to transform the messages before they are delivered. The delivery itself is reliable and a storage mechanism can be used: if a message is sent to a certain endpoint where no listener is available, it will be stored and the delivery will be delayed until a listener appears on that endpoint.
- *Composite Service* : A service is composed by a pipeline of *actions*: an incoming request message is processed by each module and then passed to the next one, or discarded if the computations must be stopped for some reason.
- *Rich Messages* : Messages have a complex structure, which includes a header, a body - which is essentially an identifier-`object` Map - and some optional attachments. Thus, any `serializable` content can be passed between a consumer and a provider.
- *Transformation Service* : Message contents can automatically be converted from a format to another by specifying the transformation patterns using either Smooks[1] or XSLT. The transformation is performed automatically by a dedicated service, in a way that is transparent to the final destination.
- *Web Service Integration* : Services on the bus can be exposed through a web-service interface simply by providing the associated validation schemas.

1 http://www.smooks.org

- *Registry Service* : A message can always be sent to a specific named endpoint, but, given the loosely coupled nature of a SOA, a client is not likely to know the exact identity of the module implementing the required service. In general it is sufficient to know a role identifier, in the specific case a name-category pair. The ESB, in fact, provides a registry lookup mechanism that, given a service descriptor, will look for an implementing module and automatically forward the message with the request. Registration is also done automatically when a service is deployed. The implementation supports registries compliant with standards such as UDDI[2], accessed through API such as JAX-R[3] or RMI.
- *Content-Based Routing* : A service consumer needs not necessarily know the name/category identifier of the appropriate service provider: a specific service can be used to analyse the content of a message at run-time and deliver it to the most appropriate service. This concept has been further extended, completely redefining the native CBR routing service. The details will be discussed in Section 10.2.3
- *Location Transparency* : As long as they point to the same registry cluster, different bus instances on different nodes can invoke services regardless of their effective location, using the API .
- *Load Balancing* : Services are identified by a name and a category, independently of the underlying implementation: thus, it is possible for the same service to be implemented multiple times (or for a single implementation to be replicated, e.g. for load balancing or fault tolerance purposes). When a request has to be handled and multiple candidates exist, the bus automatically selects one of them according to some configurable policy. Predefined options include round-robin, random and lighter-load-first policies.
- *Security Service:* Critical applications can exploit the security and authentication service to restrict deployment and access to the resources on the bus.

JBossESB has many advantages, but also some limitations which influenced the final implementation. The most relevant are:

- *Overhead* : While instrumental in decoupling the different services, the delivery of a standardized request through the ESB necessarily adds some overhead if compared to a direct, point-to-point communication in a native binary language. This could be an important issue when critical events have to be processed in hard real-time, which is seldom the case when treatment plants are involved - with the possible exception of some alarms.

2 http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm
3 http://java.sun.com/webservices/jaxr/index.jsp

- *Non-structured interactions* : While it allows great flexibility in encoding and delivering messages, the current implementation is totally lacking when it comes to the message structure. Unless *Web* services are involved, there is no way predefined way for a service provider to publish its interface so that it can be accessed by the potential consumers. Thus, the consumer must already know the request and response format, usually obtained through an "offline" channel. Moreover, the binary format requires that both parts share the same class model, which can be a serious limitation for distributed systems, where services developed by different parties are involved.

10.2.2 *Rule-Based agents*

The integration of the different components on the ESB is trivial:

- **Service**: Services are nothing more a name/category pair, from now on denoted by N/C, connected to one or more communication channels. The service is defined by a chain of actions, i.e. methods of some implementing classes. Interactions take place using `Messages`, whose format must have been agreed upon by both sender and receiver: to implement an action, a class must define a method with the signature `process(Message) : Message`.
- **Events**: Events are wrapped and delivered using `Messages`. The format is essential: the `Message` body contains a slot, indexed by the key "`event`", which contains the instance (*form*) of the event. Unlike generic interactions, event notifications are always *asynchronous* and *one-way*, so the event source never expects (nor waits for) a response. If an event requires a response, it must be delivered by generating a new event. Notice that if an event is *imperfect*, i.e. it has an associated degree, this is propagated as well.
- **Rule-Based Agents**: Agents are implemented using Drools, so their behaviour is defined using rules. Agents may be purely reactive or hybrid[4]: the former are implemented using a *stateless* Drools session, while the latter use a *stateful* one. The former are more efficient, but lack the memory of the latter. The session, if present, is created when the agent is initialized, but can be disposed or renewed at any moment. Drools' implementation also allows to change the rule base dynamically: a rule agent polls one or more rule resources (either in the local file system or stored in a Guvnor repository) at constant intervals and updates its internal RETE network whenever one of them changes.

  Every service is implemented using a single action, which acts as an adapter between the communication channel and the agent.

4 in the sense of mixed reactive/proactive behaviour

The method `process`, in fact, limits to extract the payload object(s) from the incoming message body and `inserts` them in the agent's working memory. If an event has an associated imperfect degree, it is automatically injected on the event's `holds` constraint before the event is inserted. While this is sufficient for event processing (if needed, new events are generated by the rules themselves), in case of proper interactions the adapter also collects the results from the WM (e.g. using a Drools query or a global object, both features of the engine) and wraps them in a new Message to be returned to the sender.

Rule-based processing has the further advantage of being less strict than pure Java methods and does not require explicit type checks and/or casts: in fact, the RETE algorithm automatically tries to match a new fact with the existing rules. To overcome - at least partially - the problem of class sharing, all agents load a model `jar` from a common, known repository, implemented using Drools Guvnor. The structure is summarized in Figure 59.

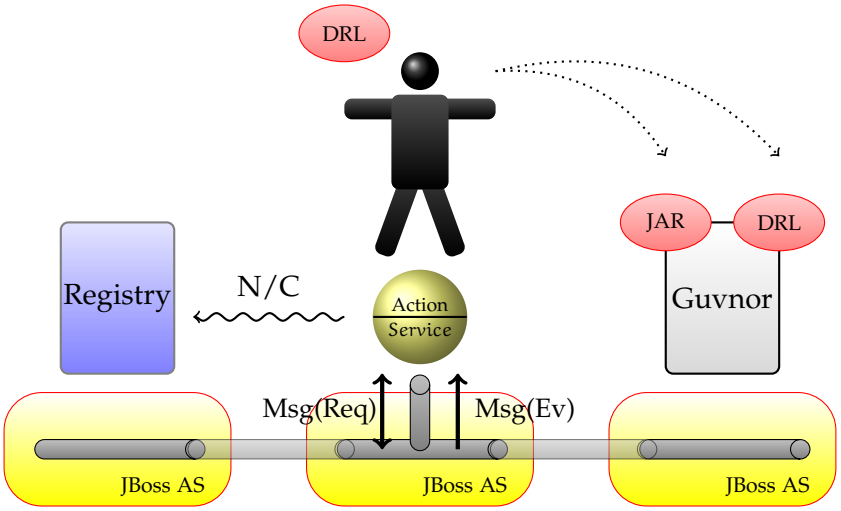

Figure 59: Agent Architecture

10.2.3 *(Dynamic) Content-Based Routing*

In normal service invocations, the consumer has the responsibility of finding a suitable provider: it usually knows its N/C identifier and invokes it using the `ServiceInvoke` API, which performs a lookup on the registry to obtain the service's endpoint, where the message can be delivered. In some cases, however, the consumer does not know the N/C or, like in the case of event sources, is not even interested in knowing who will handle it. While JBossESB supports the publish/subscribe interaction pattern using *topics*, it also allows more structured alternatives.

Dynamic CBR

Dynamic Content-Based Routing is an enterprise integration pattern [155] which extends the concept of content-based routing. Unlike publish/subscribe, where the destination(s) of a message depend uniquely on the channel, when CBR is used the destinations are chosen according to some conditions expressed on the actual contents of the message. Reactive rules such as rule 10.1 are a natural implementation for this pattern, which is used for the routing of events.

Listing 10.1: Basic Routing Rule

```
rule "Routing Example"
  //perfect rule
  when
    $e : Event( ... ) //conditions here
  then
    insert(new Destination($e,"Name","Cat"));
end
```

In fact, the router is a specialized agent identified by the N/C pair "`Router:Routing`". The use of Guvnor, together with the rule-base update capabilities of the agents, actually allows to implement the *dynamic* version of the router. Standard CBR uses a fixed rule base, while in D-CBR the rules are provided at run-time by one or more external sources, usually the destinations themselves, according to the schema in Figure 60.

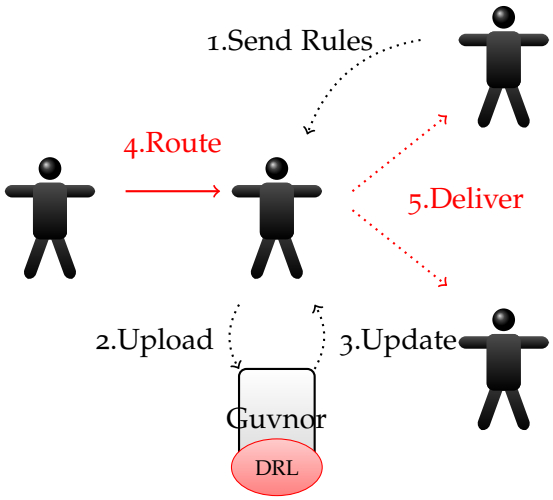

Figure 60: Dynamic Content-Based Router

The routing rules are sent to a routing agent, which loads them on the repository. Guvnor sees that the rules are joined in a single package, which is then downloaded and used by all routing agents to update their internal RETE networks. This solution allows to hide the

use of Guvnor from the potential destinations - in fact, only the router is aware of the presence of the routing rule package in the repository; at the same time, it allows to share the package between multiple instances of routing agents replicating the routing service.

The use of routing rules generalizes the concept of Event-Processing Network advocated by Luckham [186]: an EPN, in fact, can be emulated by writing a CBR rule for every edge, having each rule test the source of an event to decide which destination to route it to. The solution has the advantage of greater flexibility and adapts to the effective number and type of event handlers active at any moment, but it is assumed that the destinations upload "safe" routing rules, since no control is actually performed on them. Notice that, if needed, the routing rule-base can still be provided by a centralized orchestrator. In particular, an agent sends its routing rules to the router when it is created; the same agent notifies the router of its destruction, so that the router may decide to remove its rules. In the current implementation, the router checks whether the agent's service was replicated: if no more instances of the same service exist, it removes the rules from the repository (and, subsequently, its RETE network).

D-CBR vs EPN

Moreover, the behavior of the routing agent can further be enhanced with additional capabilities:

CONTEXT-BASED ROUTING A standard routing agent uses Drools in stateless mode: the entailed destinations, then, depend on the content of the event alone. In more complicated situations, the routing decision may depend on other events or some additional *context* information, but for the agent to retain them, it must be configured to use a stateful session. This can actually be considered a form of *context-aware* routing ([90]): sources and handlers send context facts to the routing agent, which adds them to its working memory. The routing rules, then, can express conditions involving them. The drawback of a stateful session is its increased memory occupation: to limit it, the session is configured properly using Drools Fusion STREAM mode. In this mode, facts are retained until retracted explicitly, but events are retracted automatically when they can no longer match any rule. The automatic retraction of routed events, in fact, would not allow to use recent events as part of the context; Drools fusion, instead, permits it while at the same time avoiding the accumulation of stale events.

A practical example of context-based routing is the dynamic reconfiguration of the routing paths. Suppose that a handler A_1 consumes the events produces by another EPA A_2, but this one is not available for some reasons - either because its presence is *optional* or due to a failure. The router, then, could try to deliver to A_1 the events it would normally have sent to A_2[5], assuming that A_1 can still perform at least part of its tasks using the lower-level events. The conditions may be

5 in practice, the router does so on explicit A_1's request

expressed on the event flow itself, as in rule 10.2, or on some other context facts provided by the sources and/or the destinations.

Listing 10.2: Context-Based Routing

```
rule "Default"
  //perfect rule
  when
    $e : EventA()
  then
    insert(new Destination($e,"A1","Cat"));
end

rule "Best Effort"
  //perfect rule
  when
    $e : EventB()
    not EventA() over:time(15m)
  then
    insert(new Destination($e,"A1","Cat"));
end
```

MULTIPLE DELIVERY For any message, the rules may entail zero, one or more destinations, in the form of N/C pairs: for each N/C pair, then, a corresponding service is invoked. When the N/C pair is replicated, the `ServiceInvoker` chooses one instance of the service according to the default policy, but there are cases when this is not acceptable. Consider, for example, the case of an agent with the task of activating a siren in presence of an alarm: its replication would clearly not be for load balancing purposes. In this case, it would be desirable that the event was routed to all instances capable of handling it. Since JBoss-ESB native CBR does not allow multiple deliveries, the feature has been added: when a rule entails a Destination, it can add a flag (false by default) which specifies to all service instance of the same *category*, regardless of their name.

SOURCE REDIRECTION The main drawback of using a dynamic CBR is the additional overhead. While this is acceptable for complex events, it becomes less sustainable for simpler events, which usually are generated with a higher frequency only to be handled by the same service. To avoid this problem, a router may decide to override the behaviour of an event-generating agent, having it deliver its events directly to the designated handlers. The protocol is as follows:

1. The source sends its events to the router to be delivered
2. For simple events only, if the router recognizes that in the last δT seconds all the events of the same type C, coming from the

same source S, have been delivered to the same destination D, it sends a *one-way* asynchronous message to S, asking it to deliver its events of class C directly to the final destination D.

3. The source may decide to ignore the suggestion, but if it does not, it sends all events of type C to D instead of the router. The redirection lasts for ΔT seconds, after which S starts again to sends the events to the router.

The redirection is temporary, since the final destination could be replaced at any time, or others may appear: while the router is aware of this because the new handlers send their rules, the source is not. The router, however, may redirect the source again, so for a fraction of time equal to $\frac{\Delta T - \delta T}{\Delta T}$ the router is effectively bypassed. A rule implementing this feature is shown in 10.3.

Listing 10.3: Routing : Source Redirection

```
rule "modifyDestination"
//perfect rule
  when
    // on routing certain types of event:
    $e : Event($srcCat : srcCat, $srcName : srcName,
                $type : class memberOf ... )
    $d : Destination(event == $e, $cat : category, $ser : name)
    // if all previous similar events
    // have been sent to the same destination:
    forall (
      $evt : Event(this before @[ params="1ms,10m" ] $e, class
          == $type
                  srcCat == $srcCat, srcSer == $srcSer )
      Destination(event == $evt, category == $cat, name == $ser)
          )
        // and the event is sufficiently frequent:
    $total : Number( doubleValue > 10 )
       from accumulate( Event(this before[1ms, 10m] $e,
                         class == $type,
                         srcCat == $srcCat,
                         srcSer == $srcSer )
                       init( double total = 0; ),
                       action( total += 1; ),
                       result( total ) )
  then
       // have the source route them directly to $d
  end
```

10.3 CASE STUDY

The proposed architecture facilitates the development and the deployment of agents with complex event-processing capabilities, supporting their interactions. The EPN-equivalent of the team of agents developed for the management of the SBR is shown in figure 61 (in particular, the proper EPAs are drawn in blue, while the other blocks are corollary services and interfaces used by the agents). Remarkably, the network is

neither static nor has a direct "concrete" representation, but is instead the result of the various routing rules provided by the agents themselves. This section, then, will discuss each agent: all of them are rule-based, but most have a hybrid structure which includes one or more SC techniques at different levels of integration, discusses according to the criteria defined in Chapter 4. To better manage the imperfection, the agents use non-boolean rules with slightly different semantics, discussed on a case-by-case basis. The simulations have been performed using a `Factory` which models degrees using a simple real value for all cases, so it was necessary to make some additional assumptions to mix the different imperfection types.

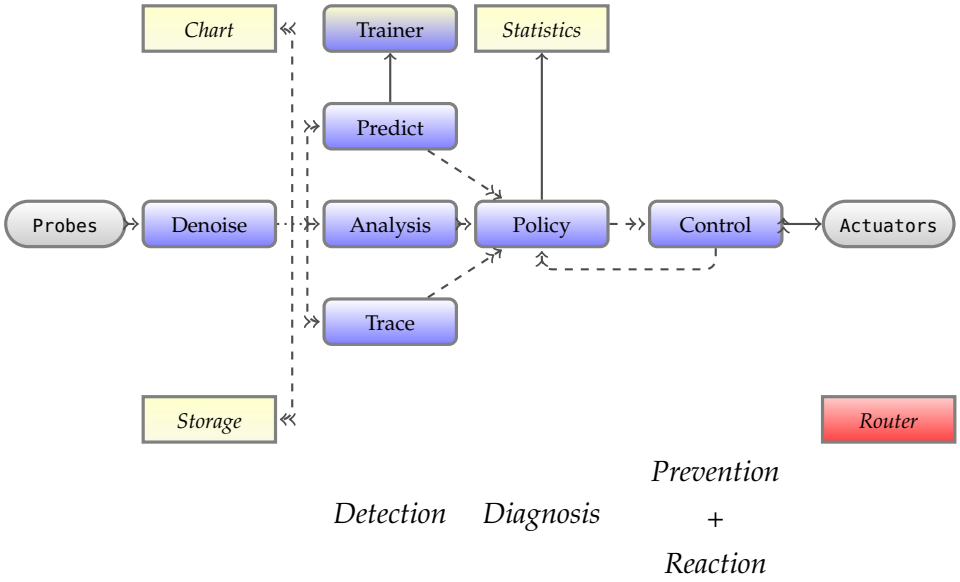

Figure 61: EPN-equivalent interactions between SBR agents

10.3.1 *Event Model*

In [75] and other related works, it has been shown that a complex management system may benefit greatly from the use of an *ontology*. Unfortunately, to our knowledege a standard and universally accepted ontology covering the SBR domain does not exist to this date. The development of one was beyond the goals of this work, so a simpler POJO hierarchy has been adopted instead and will be briefly discussed in this Section. Nevertheless, the use of ontologies, especially the ones supporting imperfection natively, will be considered in future versions.

A canonical abstract `Event` class is at the top of the hierarchy. It has fields for its `start`, `end`, `duration` and `expiration`, in order to be compliant with Drools Fusion. Moreover, two additional fields are used for the N/C identifier of the `source` who generated it. This class is further extended by `PlantEvent`, adding a reference to a `Plant`, a

SBR Glossary

model class described by its `id`, `class` (SBR, MBR, ...), `location` and `name`.

Sample

The fundamental event is the `Sample`, the act of acquiring a small amount of water from the tank to perform one or more `Analysis` on it. An `Analysis` is the measure of the `value` of a `Variable` (pH, nitrate concentration, temperature, ...), performed using a `method` which is affected by an `error`. The measure can also be associated to a `derivative`, estimating the trend of the evolution of that variable in time. Mere analysis are `raw` and do not always come with all this additional information, which may require a pre-processing treatment to be evaluated.

Analysis

Variable

Track

A sequence of `Samples` with the explicit goal of monitoring the process during its whole execution is called `Track`. For quick reference, a `Track` has a reference to the `Set` of all the variables analyzed in the `Samples`, which need not be the same for every `Sample`. Before a track can be used, it must be `validated` to ensure that all the data are complete, correct and coherent.

The relations between these classes are shown in Figure 62; the model, however, is not limited to them.

TrendChange

The event `TrendChange` models the signal-level events: the `type` can be any one among *Apex*, *Knee* and *Step* [1]. Its features, other than the signal `source`, are the `verse` (upwards or downwards), the time extension `deltaT`, the range extension `deltaY` and the slopes `leftDer` and `rightDer`.

Cycle- and Phase-related Events

In order to model the cyclic process, instead, the simple plant events `InitCycle`, `InitPhase` and `ReactionComplete` have been defined. Notice that `InitPhase` declares the `phase` about to begin, while `ReactionComplete` refers the `reaction` which just terminated. The two are connected by the fact that certain reactions can take place only during certain phases due to various bio-chemical and physical reasons. Moreover, the event `Switch` is used to denote a phase transition. Ideally, `ReactionComplete` should cause `Switch` which, in turn, causes `InitPhase`.

10.3.2 *General Purpose Services*

To be usable in practice, the system requires some general-purpose functionalities. They are of little relevance from the AI point of view, so the description will be summary, outlining a few architectural features.

- **Gateway** : This module interfaces with the legacy data acquisition system, a combination of a National Instruments analog/digital sampling and conversion board and a Labview management application. The board samples the probes (pH, ORP and DO) with a 1kHz frequency and averages them over 1000 samples to return a vector of 3 values every second; the software further averages the data over 60 seconds and stores them in a temporary archive on the local file system. The gateway agent monitors the data source and generates an event of type `Sample` (raw) every minute.

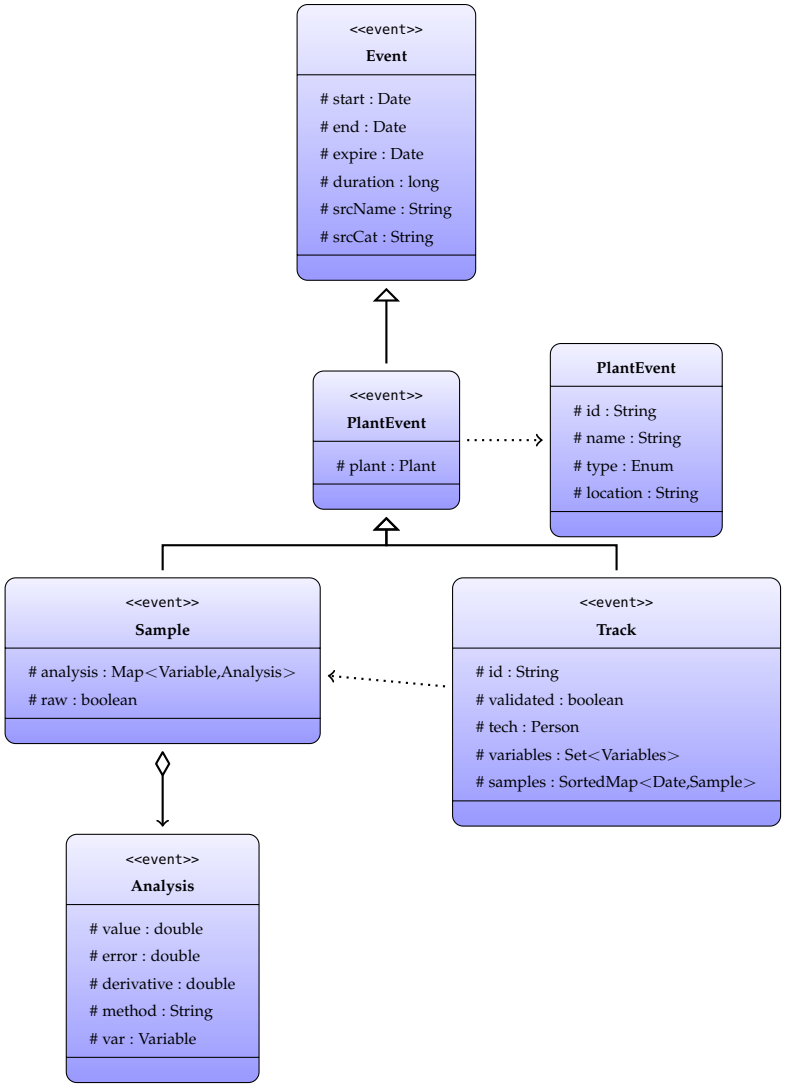

Figure 62: Event Model - Excerpt (UML)

- **Data Storage** : This agent is an interface for a standard relational database (even if future releases may rely on a more structured data warehouse). Remarkably, in the proposed architecture the DB is moved away from the main data flow. In data-centric architectures, DBs are often used with the dual role of storage facilities *and* messaging middleware. This reduces the number of components, but the data consumers have to wait for the information to be stored before they can access it, with the additional overhead of two DB accesses. Moreover, not all DBs support notification mechanisms, so the consumers must *pull* the data from it. The use of the ESB, instead, allows to bypass the DB, at the same time using a "push" model which notifies the handlers

only when new data are effectively present. The DB, however, remains exposed through the storage agent and can be accessed like a normal service (for simplicity, the request Message holds the query to be performed).

- **Charting** : This basic GUI service interacts with the storage agent and allows to plot charts of different time series. Remarkably, it can handle `Sample` events, updating the charts in real-time if new data relative to the visualized signals are acquired. This service also exploits the multiple delivery capability of the event router, allowing multiple displays to be active at the same time.
- **Actuation** : The agent responsible for actually sending the commands to the actuators is not active: the legacy control system does not have an external command interface and due to unfortunate practical reasons it has not yet been possible to upgrade it.

10.3.3 *Data/Event Processing Agents*

This section will be dedicated to the proper "intelligent" EPA, outlining their architecture. In discussing their behaviour, only a subset of the rules will be presented, and even then they will be somewhat simplified, ignoring the details of implementation nature to stress, instead, the underlying logical processes.

Pre-Process

onSample

This pre-processing agent reacts to a `raw Sample`, regenerating the events with `raw` set to false, after denoising the data and computing the accessory information such as the (estimated) time derivative and measurement noise. Notice, however, that in some deployments the pre-processing agent can be *optional*: in that case, a router with the context-aware modality enabled can bypass it. Since `Samples` are collections of several measurements, they are internally separated, processed individually and then recombined. Each `Variable` value is assigned to its context, namely the plant it has been collected on, and stored in a limited buffer associated to that context. The agent, in fact, delegates the actual computation to an internal pre-processing strategy, which is typically a numeric algorithm that requires a number of past (raw) samples to be effective. A temporal window is then created the first time the agent has to process a new variable (and retracted when empty). An outline of the behaviour is listed in rule-base 10.4.

Regularization-based Denoising

While a simple moving average could be enough, the agent uses the regularization algorithm described in [82]. Given a time series $x(t) = x(0)..x(T)$, the algorithm actually smooths the time derivative x' of the signal, computing the clean version x'_c before rebuilding the denoised series x_c by integration. To do so, it minimizes the regular-

Listing 10.4: Pre-processing Agent Policy

```
rule "Init Window"
// perfect rule
  when
    $raw : Sample( $plant : plant, raw == true )
    Analysis( $var : var ) from $raw.analysis
    not Window( src == $plant, var == $var )
  then
    insert(new Window($plant,$var));
end

rule "PreProcess"
// perfect rule
  when
    $raw : Sample( $id : id, $plant : source, raw==true )
    Analysis( $var : var, $newVal : value ) from $raw.analysis
    $w : Window( src == $plant, var == $var )
    $s : PreprocessStrategy()
  then
    $w.update($newVal);     // process individual variables
        Analysis clean = new Analysis( $id, $var, ... ,
                                       $s.process($w));
    insertLogical(clean); // main entry point
end

rule "Recombine"
// perfect rule
  when
    $raw : Sample( $id : id, raw == true)
    $data : Collection from accumulate (
              Analysis( source == $id )
              ...
            )              // recombine the results
  then
    // raw = false
    Sample sample = new Sample($id, ..., $data, false);
    dispatch(sample);       // generate the event
end
```

ization functional 10.1[6], which finds a trade-off between the approximation error and the smoothness of the approximation, shifting the relevance from the former to the latter according to the value of the parameter α.

$$\alpha \int_0^T |x_c''| + \frac{1}{2} \int_0^T |Ax_c' - x| \tag{10.1}$$

For better results, but at the expense of some delay, the clean value can be taken to be $x_c(T-\tau)$ instead of $x_c(T)$, since the algorithm is less stable near the borders of the window. Whatever the value of the parameter τ, the strategy also returns the estimated derivative $x_c'(T-\tau)$; the error, instead, is defined as the difference between the original and the denoised values.

6 A is an integration matrix

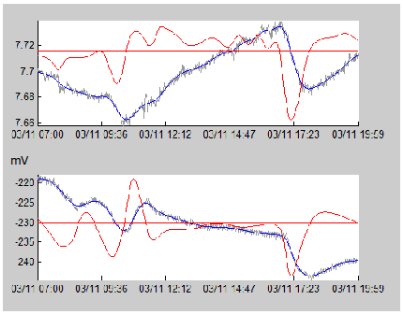

Figure 63: Original (grey), filtered (blue) and derivative (red) signals

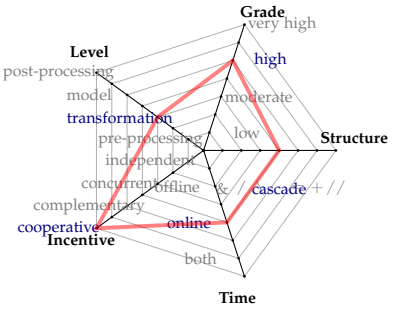

Figure 64: Preprocessing Agent Hybridization

Trend change Detection

The correlation between changes in the observed signals and the completion of the bio-chemical reactions is universally acknowledged (see Chapter 9): however, this poses the problem of identifying the trend changes, fact which becomes more complicated in an online application. To do so, a dedicated agent exists which analyses the different signals and generates events of class `TrendChange` when needed. The agent ignores the differences between the signals and treats them equally from a purely numeric point of view. Using a set of rules similar to the ones used for the pre-processing agent (see 10.3.3), it handles `Sample` events by assigning the value of each `Variable` to a corresponding window. Given the slow dynamics of the processes, the window's size is set to $20 \div 30$ to have it capture the necessary changes. Every time a window is updated, the new content is processed by an algorithm which extracts a (linear) model which, in turn, is used by a pattern matching module with the goal of identifying the relevant trend change patterns. This abstract behaviour can be implemented using a number of techniques ([197]): the approach adopted here consists in the combination of a piece-wise linear approximation algorithm ([1]) for feature extraction, together with a (fuzzy) rule-based classification.

onSample

Signal Feature Extraction

Given a window, the linear approximation module returns a sequence of contiguous segments $\langle [x_j, y_j], [x_{j+1}, y_{j+1}] \rangle_{j:1..n}$. It was assumed that any one of the trend changes of interest (Apex, Knee and Step) can be defined by three consecutive segments, so up to $n-2$ candidate piece-wise tracts have to be tested every time. The module does not return the segments directly, but actually `FeatureSets`, structures holding the features such as amplitudes and slopes which define the shape of the approximation, as shown in Figure 65.

Fuzzy Rule Classification

The use of fuzzy rules to classify the features allows to associate each feature set to a pattern type with a degree that is greater as its similarity with an ideal prototype increases. The consequence degree of classification rules such as the ones in example 10.5 can be consid-

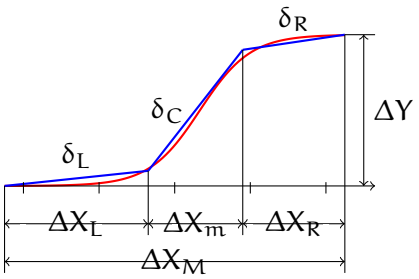

Figure 65: Approximation Features

ered a similarity degree, or a measure of the necessity that a pattern should be assigned to a class. It must not be forgotten, however, that the rule is applied to an approximated model and not to the original data, which is affected by (measurement) error itself in the first place. The feature set, then, includes an `error`, computed by taking the sum of the maximum (average) approximation and maximum (average) input errors. The rules should take this into account: when confidence factors are used, the confidence of a classification should be inversely proportional to this error; more generally, a high error should drive the consequence degree towards "unknown", whatever the representation according to the adopted degree set. A possible way to do it is to use a custom evaluator in the rule's implication, conditioning the connection between premise and conclusion on the value of a specific field. Rule 10.5 shows the tentative evaluation of a positive Apex (i.e. a maximum), conditioned on the approximation error. The rule is actually a simplified version, since the number of features and the variability of the patterns complicate the classification, so that additional restrictions are required in practice. In the current implementation, the fuzzy partitions for the features' domain and the classification rules have been written by hand, but will likely be replaced by a hybrid cluster/rule algorithm.

Imperfect Classification

The rules do not generate the events immediately: since the sliding window is analysed for every new sample, a relevant pattern may remain in the window for some time, but the event should be generated only once. Nevertheless, when the pattern enters the window, a partial match might still generate an event which could and would be better upgraded when the whole pattern is observed shortly afterwards. So, a temporary `TCCandidate` is generated instead from the feature set: this has the form of an event - including start and duration which, for this type of events, is not null, but estimated from the feature Δ_{XMax} - but is not propagated unless it has been validated. A possible solution is to consider only events which start and finish sufficiently far from the borders of the window, but this introduces a delay. The adopted solution, shown by rule **??**, instead takes advantage of imperfection.

Before a `TrendChange` is generated, the rule checks whether there already exist an equivalent event in the immediately preceeding in-

Listing 10.5: Trend Change Detector Rule

```
rule "Maximum"
  // not supported yet, emulated using injection:
  implication @[ kind="predicate", args="#0.error seems small" ]
when
    $f : FeatureSet(
           // change should be evident
           // filters out local "noise"
           deltaXMax neg seems "short"
                   && deltaLeft neg seems "short"
                   && deltaRight neg seems "short"
                   && deltaY seems "high"

                   // required traits
                   && leftDer seems "positive"
                   && ( cenDer seems flat || deltaXMin seems "
                        short" )
                   && rightDer seems "negative"
    )
then
  TCCandidate ec = new TCCandidate(MAX, $f , ... );
  insert(ec);
end
```

Listing 10.6: Trend Change Detector: Filter

```
rule "TrendChange"
  entailment @[ filter="Threshold(0.5)" ]
when
    $tc : TCCandidate( $type : type,
            $verse : verse, $src : source )
      and @[ kind = "Lukas" ]
    neg exists TrendChange( type == $type, verse == $verse,
            source == $src,
            this before @[ params="trapez,5m,30m" ] $tc)
then
  TrendChange ev = new TrendChange(tc);
    inject(ev,"holds");
  insert(ev);
  dispatch(ev, consequenceDegree);
  //dispatch event with associated degree
end
```

stants. The use of Łukasiewicz's and causes the rule to fire only if the new event is "truer" (i.e. the pattern is more similar) than was acknowledged before: in fact, $a \otimes \neg b > 0 \Leftrightarrow a > b$. The `before` evaluator is actually fuzzified, using a trapezoidal membership function: it discounts past events, allowing for successions of similar events which are sufficiently separated in time. When this constraint is satisfied, the event is effectively generated from the `TCCandidate`. Notice also that the match degree must be superior to a certain threshold to consider the event relevant.

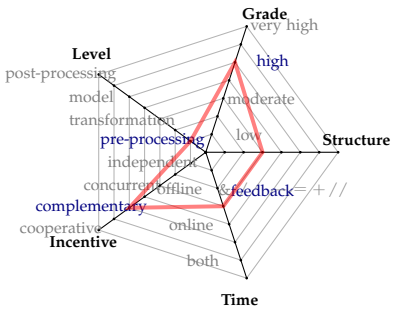

Figure 66: Trend Change Detector Hybridization

Concentration Estimation

One of the methods of determining the evolution of the process is to estimate the concentration of the unobservable pollutants in the tank, using the current values of the measured variables. In this section, it will be assumed that the former are the nitrogen compounds $[NO_3^-](t)$ and $[NH_4^+](t)$ and the latter are the usual $pH(t)$, $ORP(t)$ and $DO(t)$, but the same system can be used with other variables as well.

onSample

The predictor[7] agent reacts to pre-processed events of kind `Sample` which contain the *input* variables, but not the *target* ones. It estimates their value and (optionally) their derivatives and error, setting the `method` to a value which describes the actual technique used for the estimation. In particular, the agent embeds a prediction module for each target variable: for every prediction of each variable, it generates an appropriate `Analysis` event.

Prediction Strategy

The prediction strategy can be implemented using different tools, but neural networks are ideal candidates. Experiments have been performed using both FF-NN and Elman networks, exploiting the open source tool JOONE[8] for the actual implementation. The invocation pattern, shown by rule 10.7, follows the template outlined in Chapter 8.

Imperfect Estimation and Validation

The rule, which requires Drools Chance, is rather complex: first, it finds a suitable predictor for the sample in the WM - only one match is supposed to exist. The predictor declares its input variables, which are extracted from the sample and returned in a collection. Actually, some of the required values may be missing: in that case, the returned collection will have a size lesser than the expected one, but the imperfect evaluator `similar` will determine to what degree the actual size is acceptable. Moreover, a further condition on the input is imposed. As discussed in Chapter 3, predictors and neural networks in particular are bad extrapolators, so they are likely to give unreliable results when processing inputs quite different from the ones they have been trained on. Hence, the predictor's training set of N elements is retained and compared to the input. Since training sets can be large while only a

7 the predicted values are not the future ones, but the current, unobservable ones

8 http://www.jooneworld.com/

summary comparison is required, the set is compressed using a clustering algorithm (the test were performed using a SOM), so that it is sufficient to test the membership of the input in each of the n cluster, performing only $n \ll N$ comparisons. The invocation pattern is the one described for SOMs in Chapter 8. Eventually, the consequence truth degree is used in the estimation of the error: the lower the degree, the less reliable is the estimation. This definition may be used either as an alternative or in conjunction with the predictor's native error estimation technique, if any exists.

Listing 10.7: Predictor Agent Rule

```
rule "Predict"
  // many-valued rule
  when
    $s : Sample(  )      // routing ensures raw = false, when
        possible
    $p : PredictorStrategy(
      // conditions on the plant source, if any
      $in : inputVarSet,
      $tgt : targetVariable;
      $m : method,
    )
    $input : Collection( size similar $in.size) from accumulate
        (
         Analysis( var memberof @[ boolean ] $in,
                   ...
                   $val : value  ) from $s.analysis
    ) // extract input variables
    TrainSet( pred == @[ boolean ] $p, $proto : prototypes)
    exists $clus : Cluster( this covers $input) from $proto
  then
    Analysis a = new Analysis();
      a.setVariable($tgt);
      a.setMethod($m);
      a.setValue($p.predict($input));
      // a.setDerivative($p.predictDer($input));
      double error = 0;
        //error += $p.predictError($input);
        error += $tgt.maxValue() * (1-consequenceDegree.value())
            ;
      a.setError(error);
    dispatch(a);
end
```

A pair of prediction examples for nitrates and ammonia concentrations are shown in Figure 67 and Figure 68, respectively. The predictions have been performed using simple FF-NN predictors with 8 and 7 neurons in the hidden layer - sizes chosen after applying a canonical train/validate/test training procedure with various values for the parameters such as hidden layer size, learning rate and momentum.

TRAINER Remarkably, the training of the predictors is not performed manually, but delegated to a specific agent, who prepares the modules on behalf of the predictor agent. The latter, in fact, requests a pre-

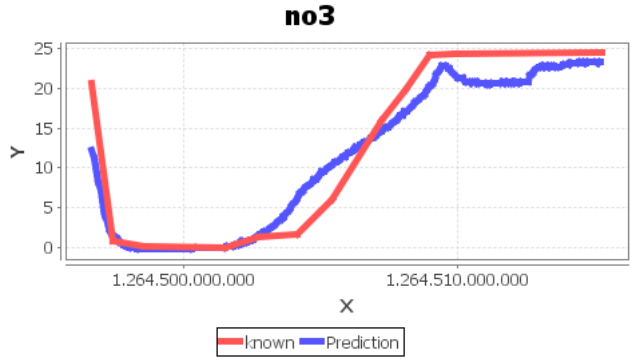

Figure 67: $[N - NO_3^-]$: predicted vs real values

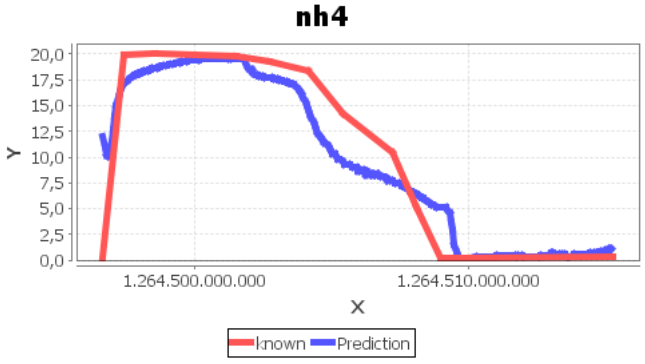

Figure 68: $[N - NH_4^+]$: predicted vs real values

dictor module to the trainer whenever (i) it does not have a suitable predictor strategy for a variable, (ii) the current module's performance degrade too much or (iii) after a fixed amount of time which takes into account the seasonal behavior of the process, so that a previously learned signal-process correlation is no longer assumed to be reliable.

Trainer Types

The role of trainer agent is an abstract one: the concrete subclasses act as factories for different types of predictor, so there is a FF-NN trainer, an Elman network trainer and more can be added. All of them, however, require a training set to train the predictors on. The training set is built incrementally, as a consequence of the completion of track studies. The trainer agent, in fact, reacts to `Track` events, updating its internal knowledge. The operation is not trivial, since initially a `Track` contains only sparse information for a number of `Variables` which are likely to become the targets of a predictor. The corresponding input variables, instead, are normally measured by probes, but their values are not added to the `Track`. Hence, before a `Track` can be used to populate training sets, several operations have to be performed: the offline (target) data must be populated fully, the online (input) data must be retrieved from the storage and the two sets have to be merged - only then the data can be validated and used in practice. Given the com-

onTrack

Track Processing

plexity of the problem, it has been formalized using a BPMN notation, considering the validation a particular case of business process. This allows to implement the different stages using rules and exploiting the synchronization features of Drools Flow. The resulting flow chart is shown in Figure 69.

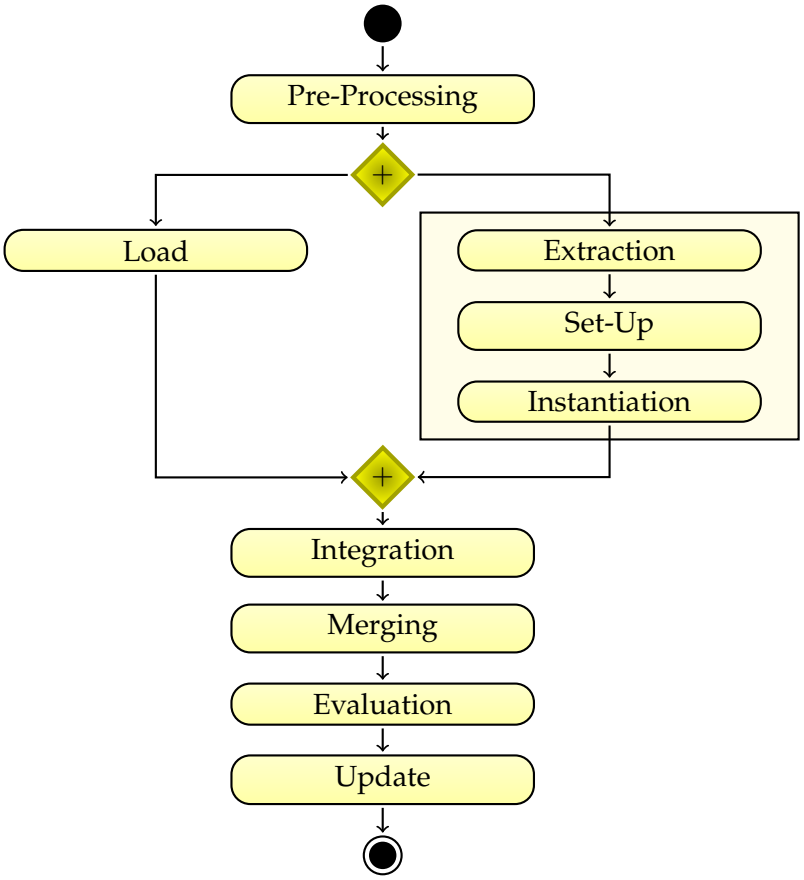

Figure 69: OnTrack Flow

Formally, the `Track` contains J `Samples`, performed at times $\tau_{j:1..J}$ which form the sparse time set T_Δ. Each sample holds m `Analysis`, which are to be matched to n online signals acquired at a much higher rate. The actions performed by the rules in every group can be summarized as follows.

- **Pre-Process**: Global information is extracted from the `Track`; the temporary data structures necessary for the elaboration are prepared.
- **Load** : The online analyses (e.g. pH, ORP, DO) relative to the period [`Track.start`,`Track.end`] are retrieved from the storage. To do so, the trainer invokes the Storage Agent, who returns a data set composed of K $(n+1)$-element tuples such as $\langle t_k, pH(t_k), ORP(t_k), DO(t_k)\rangle$.

For convenience, each tuple is split in $(n+1)$ pairs: $\langle t_k, t_k \rangle$, $\langle t_k, pH(t_k) \rangle$, ... The first pair defines the *dense* time set T_δ.

- **Extraction** : In parallel, all the `Analysis` relative to the same sparse `Variable` are extracted from the `Track` and combined so that each variable can be processed separately. In order to estimate the values for times other than T_Δ, the sparse time series must be *approximated*, possibly using one of the techniques described in Chapter 3.
- **Set-Up** : The Set-Up stage selects the concrete Approximation strategy and configures the necessary parameters. So far, two are supported: RBF functions and the Bayesian GPM method [57].
- **Instantiation** : Eventually, each sparse time serie $\langle \tau_j, \gamma(\tau_j) \rangle$ is used to train an approximator f_γ. Figure 70 and Figure 71 show the result obtained using the two techniques on the same data set.

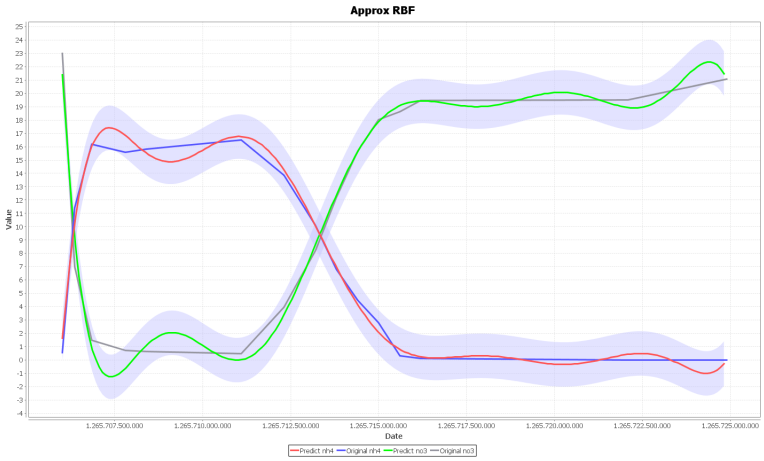

Figure 70: Example: Approximation using RBF model

GPM has the advantage of returning not only an estimate $\gamma = f(t)$, but a full Gaussian probability distribution on the variable's range $p(\gamma|t)$ which can be converted in a confidence interval and thus in an error for the virtual `Analysis`. An RBF approximator, instead, returns a definite value, but, assuming that the variance of each element in the training set (i.e. the `error` of the `Track`'s `Analysis`) is known, it can be shown that this error can be converted into an uncertainty first on the combination weights of the RBF [244] and then on the final output [123]. This method, however, turned out to suffer from some numerical stability problems.

- **Integration** : Once the approximators are ready, the sparse time series can be "filled": for each $t \in T_\Delta$ and for each approximator f_γ, the pair $\langle t, f_\gamma(t) \rangle$ is built.

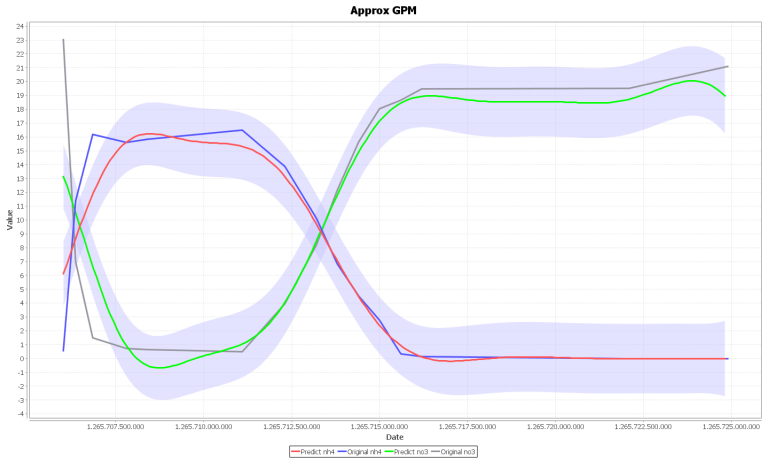

Figure 71: Example: Approximation using GPM model

- **Merging** : All the pairs $\langle t, \cdot \rangle$ relative to the same t are joined in a single tuple $\langle t, \varphi_1(t), \ldots, \varphi_{n+m}(t) \rangle$, where φ denotes any input or target variable.
- **Evaluation** To have a predictor learn meaningful relations, the candidate training data must be validated accurately. The criteria are expressed using the many-valued rule 10.8, whose consequence degree is attached to the tuple to measure its reliability. Different aspects are taken into account at the same time: the experience of the operator who made the analysis and the method used for it (a novice tends to make more errors than an expert one); the compatibility between the *tracked* plant and the one the final predictor will work on (if plants of a similar class work in similar operating conditions, it is reasonable to transfer the information acquired on one to the other, albeit with a lower degree of confidence); the uncertainty on the measures, denoted by the error, real or estimated.

Logic-Based Validation

- **Update** The `Track` `validated` status is set to true; moreover, the weighted tuples are inserted into a collection, called `TrainingBag`, from where they can be extracted to build training sets. A training set of X elements is formed extracting X elements from the bag, each with a probability proportional to its weight. This allows less reliable input-target pairs to be presented to a predictor with lower frequency during its training phase. In practice, the `TrainingBag` component allows to implement *bootstrap* training procedures [63] which are useful especially when the training data are scarce. Since tracks are expensive procedures, both in terms of money and time, this scenario is not unlikely. To this date, only 10 Tracks were available, 2 of which were kept for the validation and test of the neural predictors.

Listing 10.8: Trainer Agent - Data Validation Policy

```
rule "Validate Data"
when
  $t : Track( this valid,
              $tid : id, technician expert,
              $pid : source.id,
              $p_loc : source.location,
              $p_type : source.type
      )
    //track execution criteria
    $trainerPlant : Plant(
              id == $pid
       || location near $p_loc
       || type compatible $p_type
      )
    //source compatibility criteria
    $s : Sample( source == @[cut] $tid,
              $sid : id, $time : TStart,
              $anMap : analysis
      )
    //measurement criteria
    forall (
         $a : Analysis( ...
              error small && method reliable
        ) from $anMap
      )
then
  // weight = consequenceDegree;
  ...
end
```

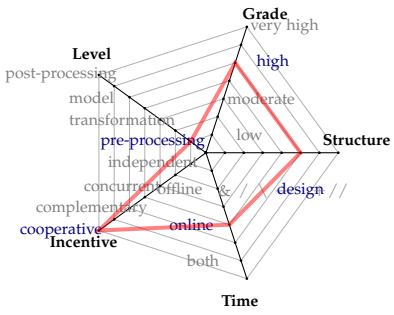

Figure 72: Trainer Agent Hybridation (w.r.t. to Predictor)

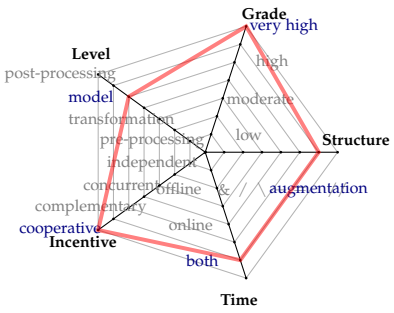

Figure 73: Predictor Agent Hybridation

Time Series Tracking

Another possible, imperfect approach to the monitoring of the process can be based on the comparison between the time series measured on-line by the sensors and the ones stored in a historical database. Conceptually, this approach has some similarities with CBR, since it

tries to associate the current values of the signals to the values they had in some predefined and well-known situations.

To create the case history, a set of *valid* cycles is extracted and prepared. The concept of a-posteriori validity defined in [1] applies, but here it can be easily mapped onto rules using the `ReactionComplete` events, as shown in 10.9.

Listing 10.9: Cycle validation

```
rule "Predict"
  // probabilistic rule
  entailment @[ boolean ]  // only certainly valid cycles
      allowed
when
  ...
  $eod : ReactionComplete( $p : plant, $c : cycleId, reaction ==
       DENITRIFICATION)
  $eon : ReactionComplete( plant == $p, cycleId == $c, reaction
      == NITRIFICATION,
                    this after @[ params="..." ] $eod ) // Tmin
                        ..Tmax
  ...
then
  // $c is valid
end
```

Knowing the timestamps of the events `ReactionComplete` for the relevant reactions, nitrification and denitrification, allows to define a fuzzy partition of the time domain of each reaction phase (anoxic and aerobic), using the fuzzy sets "`pre`", "`inter`" and "`post`" to denote the fuzzy periods preceding and following the completion of the reaction, separated by the fuzzy instant during which the reaction effectively completes (e.g. see Figure 74). This fuzzification is useful for many reasons: reactions do not actually finish abruptly; the effect on the signals may not be immediate and there may be some error in the measurements.

Sub-Phase definition

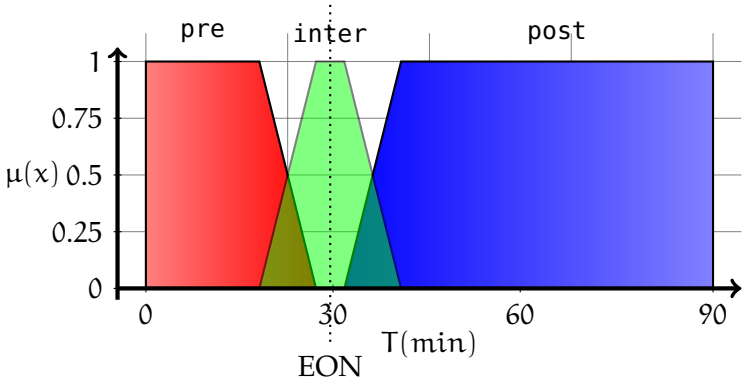

Figure 74: Anoxic Sub-Phase Fuzzy Partition

The goal is to estimate the membership of a signal vector

$$\mathbf{x} = [pH(t), ORP(t), DO(t), pH'(t), ORP'(t), DO'(t)]$$

obtained from a `Sample`, in order to generate up to three `AckStage` events, respectively holding the memberships μ_{pre}, μ_{inter} and μ_{post}. However, the membership degrees are not static : they are not defined directly on the time domain, but for each cycle they are conditioned by the value of the signals, i.e. $\mu(t) = \mu(t|\mathbf{x}(t))$. In practice, the memberships are *known* a posteriori for the signals of the valid cycles, but not for the new ones. To obtain the memberships at run-time, an imperfect reasoning based on the principle of similarity is adopted: "*The more similar an input* $\mathbf{x}$ *is to a prototype* $\mathbf{c}$ *for which the memership* $\mu(\mathbf{c})$ *is known, the more similar* $\mu(t|\mathbf{x}(t))$ *will be to* $\mu(\mathbf{c})$".

Training Set Reduction

This reasoning is implemented in different steps. First, the training set is encoded in a more compact form using a SOM on a bidimensional grid to take into account variations along the temporal and measurement dimensions (see Figure 75). In particular, two distinct neural networks are used for the anoxic and aerobic phases, each deployed on a fixed 5x20 grid.

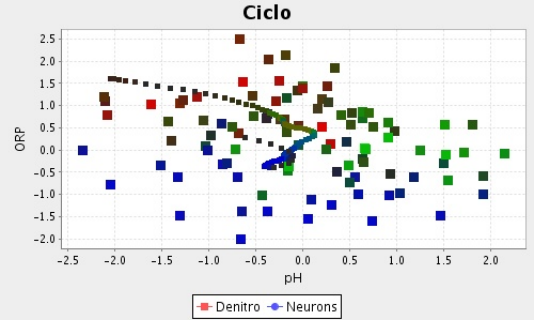

Figure 75: Anoxic Phase - SOM Layout (PCA projection) : colors correspond to `pre`, `inter`, `post` sub-phases

The Neurons are used to evaluate the similarity between the current input and the prototype ones in a more efficient way than using the redundant training set directly. The agent can choose the appropriate network simply by intercepting the `InitPhase` event.

On-line Classification

More formally, the classification problem can be stated using the approximate inference schema in possibilistic logic:

$$\frac{\texttt{Recall}(N, X(t)), \texttt{Recall}(N, X) \rightarrow SubPhase(N)}{SubPhase(X(t))}$$

The predicate-equivalent `Recall` models the similarity between the input and the prototype (see also Section 8.2.3); the implication $\rightarrow$, instead, models the correlation between a prototype and the current subphase (respectively `pre`, `inter` and `post`). It is expressed using a gradual rule (see Section 8.1.3), in order to obtain a conservative lower bound for the sub-phase membership of the current input.

The language and the engine support the expression and the evaluation of this schema. First, the implication is learned by induction, using a rule pattern similar to the one shown in 8.1.5. The induction is constrained by the activation of the neuron, in order to use only the training data which are relevant for the implication (Rule 10.10 is a simplified example of the training rule set). The actual semantics of the output degree depends on the form of the degrees: when simple degrees are used, it can be considered the *expected* degree of truth of the gradual implication since it is an average of the individual degrees evaluated for each training pair. The problem is that, when using induction, gradual degrees of truth mix with frequencies, which leads to a belief distribution on the set of truth values. So, a gradual membership - i.e. a prototype has mixed traits which could be typical of a *transition* between two sub-phases - can't be distinguished from an uncertain one - i.e. a prototype has ambiguous traits which can appear in different sub-phases. Early experiments with more expressive truth degrees were performed and shown in [9] and [4], but have not yet been upgraded to the online version.

Imperfect Interpretation

Listing 10.10: Tracking Agent Training

```
rule "TrainPre"
when
  ...
  forany (
     // Recall degree is given by the implicit "holds"
     $r : ( implies
            Recall($n : neuron, $x : input, $t : start) from ...
            Input( this == $x , this seems "pre" ) from ...
          )
     subject_to  Recall( this == @[ cut ] $r )
     weight      Recall( this == @[ cut ] $r )
  )
  ...
then
  ...
  inject($n, "idPre");
end
```

Nevertheless, the inductive rules allow to obtain an implication degree for each possible neuron-subphase pair: this, in turn, is used to estimate the a lower bound for the membership of the current input, as shown by the rules in 10.11. According to the SOM principle, the winning neuron is used for the association. The strict Lukasiewcz's and is used to obtain a safer lower bound: notice that using the conjunction in the premise with a true implication is equivalent to applying a gradual implication, since $a \otimes b \Rightarrow_1$ yields the same degree as $a \otimes \Rightarrow_b$. Moreover, if intervals are adopted, the partition condition $\mu_{pre} + \mu_{inter} + \mu_{post}$ can be translated in three rules, each one stating that the membership in any two sets excludes the membership in the third. In case, the necessity of intervals derives from the negation in the entailed conclusion.

Listing 10.11: Tracking Agent Training

```
rule "Associate_Pre"
ruleflow-group "Step1"
when
  $x : Input( ... )
  $win : Winner( input == $x )
  ( Recall($n : neuron == $win)
      and @[ kind="Lukas" ]
    Neuron(this == @[ boolean ] $n,
           this seems @[ id="idPre"] "pre" )
  )
then
  inject($x, "idInputPre");
end
/*
rule "Interact"
ruleflow-group "Step2"
when
  Input( this seems @[ id="idInputPre" ] "pre"
         || @[kind="Lukas"]
         this seems @[ id="idInputInter" ] "inter"
       )
then
  reject($x, "idInputPre");
end
*/
rule "Generate"
ruleflow-group "Step3"
when
  $x : Input (this seems @[ id="idInputPre" ] "pre")
then
  dispatch(new AckStage($x,PRE),consequenceDegree);
end
```

Eventually, the estimated degrees are used to generate the `AckStage` events, up to one for each subphase, which hold partially. In the ideal case, the neuron which recalls perfectly an input activates during and only during a given sub-phase, so only one certainly true `AckStage` event is generated. When, instead, an input is perfectly recognized by a neuron which activates with imperfect degrees in all phases, three `AckStage` event will occur, each one with a partial degree. The interpretation of this degree is subtle when simple degrees are used. A possible interpretation is to consider them belief (necessity) degrees, but this is equivalent to assuming that the output fuzzy partition is actually crisp - i.e. no partial memberships are allowed, but uncertain ones are. This is currently not a limitation, since what is really relevant is the transition from the `pre` status to the `post` one and the distinction between the two is boolean.

Policy Enactment

The policy agent's knowledge concerns the proper information on the management of aSBR: in fact, the other agents described so far perform

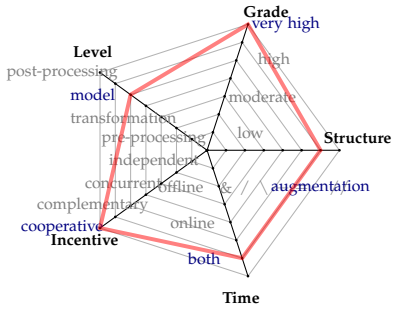

Figure 76: Tracking Agent Hybridation

generic tasks which could be easily applied to other types of plants, if not to other data processing contexts altogether. The policy agent, among other things, has the task of determining the state of the process, generating the event `ReactionComplete` when necessary. Moreover, it decides when to `Switch` from one phase to another, according to its belief and knowledge on the process. The reasoning is properly performed in two sequential steps: the switch is generated as consequence of the completion of the reactions, which in turn is generated by aggregation and analysis of the simpler events.

REACTION COMPLETION The agent intercepts all the events generated by the analysis agents, namely: *Evidence*

- Processed `Sample(No3,Nh4)`, holding the estimated values of the relevant nitrate and ammonia concentrations. The degree of imperfection is given qualitatively by the `error` associated to the values, which is easily converted in the variance of a gaussian distribution centered on the value.
- `TrendChange(Type`, each annotated by a lower bound of a fuzzy similarity degree, which is higher as an event matches an ideal prototype.
- `AckStage(Subphase)`, annotated with a degree describing the belief in the necessity of the truth of stating that the current phase is actually the given one.

Given the current phase, any of these events can be used to entail conclusions on the state of the reactions in the tank. These conclusions are always *uncertain*: the `Samples` would be reliable if they were measured directly using a (precise) sensor, but the ones here are obtained by prediction; the other criteria, instead, assume a correlation between indirect signals and processes which is abducted from experimental data and not guaranteed to hold in any case, especially in case of failures in the probes or in the process. The correct approach would require a Bayesian Network, expressing the probability that a reaction is complete, conditioned by all the available information. This is not feasible in practice, mainly because the evidence-generating agents may not *Uncertain Decision Model*

be present, and even then, they generate information asynchronously. To approximate the probabilistic model and decouple it from the information sources, a TBM model is used instead.

The property of completion is boolean, so the underlying frame of discernment has two elements: $\Omega = \{complete, \neg complete\}$. It will be assumed that the focal elements are $X = \{complete\}$ and Ω, with all the mass assigned initially to Ω. The existing criteria can only bring evidence to X, so at each update the current distribution $m_0 = \{m(X), m(\Omega) = 1 - m(X)\}$ must be merged with the contribution $m_j = \{m_j(X), m_j(\Omega) = 1 - m_j(X)\}$. Applying Dempster-Shafer's combination rule, it turns out that the new evidence is given by $m = m_0 \cap m_j$ and, in particular:

$$m(X) = m_0(X) + m_j(X) - m_0(X) \cdot m_j(X)$$

Since this is actually the probabilistic disjunction of two belief degrees, the rule-based pattern shown in 10.12 can be used:

Listing 10.12: Belief Update

```
rule "Belief Update"
when
  $b : Belief( $iter : step, $topic : topic ) // "holds" <-> m_o
    or @[ kind="Probabilistic" ]
  $ev : Evidence( refers = $topic )           // "holds" <-> m_j
then
  retract($b);
  Belief b1 = new Belief($iter+1, $topic);
    inject(b1,"holds");
  insert(b1);
end
```

The various events, then, can be converted into Evidence, each contributing with a mass w to the overall belief. The rules are sketched in 10.13 and 10.14: notice that they are domain-specific for the two reactions in the two phases.

Imperfect criteria

The implication in the rules are gradual and return a degree of belief, applying the empirical principle "*The more compatible the conditions with the ideal, expected ones, the more it can be believed that the reaction is complete*". In particular, when the premises have maximum degree the evidential contribution is w_1, w_2 and w_3 for each criteria, respectively. w_1 and w_3 are low (~ 0.1), but the events are repeatable; w_2, instead, is higher (~ 0.7). When the events do not match the premises, instead, the contribution is lower, possibly 0. Notice that since no rule provides evidence *against* the completion of a reaction, the belief is monotonic and non-decreasing. Moreover, the use of implications to vehicle belief is compatible with the definition of rules to handle exceptions, as shown in Section 8.1.7.

I - low nitrates

Other than that, the rules are an imperfect reformulation of the common criteria adopted in literature, as discussed in Chapter 9. The first looks for a (reliable) estimate stating that the nitrate concentration is

Listing 10.13: Anoxic Phase Rules

```
rule "Denitrification Evidence 1"
  implication @[ kind="mixed", degree="w1" ]
when
  CurrentPhase( type == ANOX, $den : reaction )
  $s : Sample( $analysis : analysis, ... )
  Analysis( variable == @[ cut] NO3,
            range seems "zero" ) from $analysis
then
  Evidence e = new Evidence($den);
    inject(e,"holds");
  insert(e);
end

rule "Denitrification Evidence 2"
  implication @[ kind="mixed", degree="w2" ]
when
  CurrentPhase( type == ANOX, $den : reaction )
  ( and @[ kind="Min" ]
  $max : TrendChange( source == "pH",
          type == APEX, verse == UP )
  $k   : TrendChange( source == "ORP",
          type == KNEE, verse == DOWN, this overlaps $max )
  neg exists TrendChange( source == "DO",
          type != STEP, verse == DOWN)
  )
then
  Evidence e = new Evidence($den);
    inject(e,"holds");
  insert(e);
end

rule "Denitrification Evidence 3"
  implication @[ kind="mixed", degree="w3" ]
when
  CurrentPhase( type == ANOX, $den : reaction )
  exists AckStage( phase == ANOX, subphase == POST  )
  exists AckStage( phase == ANOX, subphase == INTER )
  exists AckStage( phase == ANOX, subphase == PRE   )
then
  Evidence e = new Evidence($den);
    inject(e,"holds");
  insert(e);
end
```

close to zero, using a fuzzy approach to improve robustness. Since the estimate is given in a form `value±error` (returned by the convenience getter `range`), the fuzzy evaluator actually returns a necessity/possibility interval, and only the lower bound is propagated. The second rule requires the contemporary presence of a pH maximum and an ORP knee. Imperfection is taken into account twice: by the degree at which the events *Hold* (in turn derived from their similarity with a prototypical pattern) and by the degree of overlap of the two events. The third, eventually, requires that the correct sequence of sub-phases has been acknowledged. Even if only the rules for denitrification have been dis-

II - Simultaneous Trend Change

III - Sub-Phase Sequence

cussed, the rules for the aerobic phase are specular and reported for completeness in the rule set 10.14.

Listing 10.14: Aerobic Phase Rules

```
rule "Nitrification Evidence 1"
  implication @[ kind="mixed", degree="w4" ]
when
  CurrentPhase( type == AEROBIC, $nitro : reaction )
  $s : Sample( ... NH4 seems o ... )  //actually, the variable
       is extracted from the analysis
then
  Evidence e = new Evidence($nitro);
    inject(e,"holds");
  insert(e);
end

rule "Nitrification Evidence 2"
  implication @[ kind="mixed", degree="w5" ]
when
  CurrentPhase( type == AEROBIC, $nitro : reaction )
  $min  : TrendChange( source == "pH",
            type == APEX, verse == DOWN )
  $step : TrendChange( source == "DO",
            type == STEP, verse == UP, this overlaps $min )
  $knee : TrendChange( source == "ORP",
            type == KNEE, verse == UP, this overlaps $min )
then
  Evidence e = new Evidence($nitro);
    inject(e,"holds");
  insert(e);
end

rule "Nitrification Evidence 3"
  implication @[ kind="mixed", degree="w6" ]
when
  CurrentPhase( type == AEROBIC, $nitro : reaction    )
  exist AckStage( phase == AEROBIC, subphase == POST  )
  exist AckStage( phase == AEROBIC, subphase == INTER )
  exist AckStage( phase == AEROBIC, subphase == PRE   )
then
  Evidence e = new Evidence($nitro);
    inject(e,"holds");
  insert(e);
end
```

PHASE SWITCH When the `Belief` in the completion of a reaction exceeds a threshold, an event `ReactionComplete` is fired, with degree equal to that of the current `Belief`. The basic policy would be to `Switch` to the next phase as soon as the reaction is believed to be complete, but doing so without taking into account the uncertainty would not be safe. Instead, rule 10.15 is used:

Safe Switch

The rule integrates, putting them at the same logical level, the requirement that the reaction has been completed with the requirement that a phase must last for a minimum amount of time. `before`, in this

Listing 10.15: Optimal Switch Rule

```
rule "SwitchAnox"  // resp. SwitchAerobic
when
  $c : CurrentPhase( type == ANOX )
  ( and @[ kind == mixed ]
     $rc : ReactionComplete( reaction == DENITRIFICATION )
     InitPhase( phase == $c, this before @[ params="tMin"] $rc )
  )
then
  Switch sw = new Switch(...);
  schedule(Switch, tMax, 1-consequenceDegree);
end


rule "Safety Check"
when
  Switch( newPhase == AEROBIC, $now : start )
  $cp : CurrentPhase( $cid : cycleId, phase == ANOXIC, $dur : (
       start - $now))
  CurrentPhase( cycleId == ($cid - 1), phase == AEROBIC,
        $prev : duration neg compatible @[ filter="..." ] $dur )
then
  Alarm a = new Alarm(...  , consequenceDegree, WARNING );
  dispatch(a);
end
```

case, has a trapezoidal membership degree which grows linearly from 0 to the passed argument tMin and stays bounded to 1 afterwards. Thus, the consequence degree of the first rule decreases as (i) the reaction is not fully believed to be complete and (ii) there is not enough temporal distance between the start of the phase and the moment it could theoretically be ended. Thus, the `Switch` event is not generated immediately, but after a time $\Delta T = (tMax - now) \cdot (1 - \varepsilon)$.

Consistency Checks

Notice that other operations can be performed at this point: for example, it was found that there is a weak correlation between the duration of a nitrification reaction and the duration of the following denitrification, as shown in Figure 77. On leaving the anoxic phase, a rule retrieves the duration of the aerobic phase of the previous cycle (assuming its duration has been optimized, so that it coincides with the effective duration of the nitrification reaction) and checks the compatibility between the two durations. The relation has been modelled using a GPM, so the response is actually a degree of probability. If this value is too low, the rule triggers and generates an `Alarm`. Obviously, similar rules can be written for other correlations that exist between phases of different cycles.

Plant Controller

onSwitch

The controller agent takes care of the actual phase changes in the plant: it reacts to a `Switch` request event, executing the actions necessary to move from one state to the following, then notifies the completion of

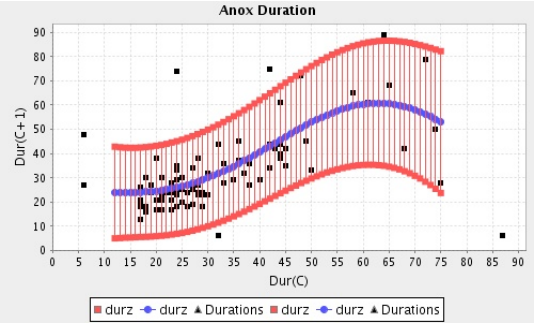

Figure 77: Correlation between the Duration of an Aerobic and following Anoxic Phases

the transition generating the event `InitPhase`. Its semantics is exactly the one defined by the Petri Net in Section 9.4: Drools, in fact, allows to translate the theoretical model directly into a concrete implementation. Drools Flow has a (limited, but growing) support for BPMN models, which can be mapped onto Petri nets ([102]). While it has not been verified if the mapping is biunivocal, so that also the converse holds, it was feasible to translate the necessary parts.

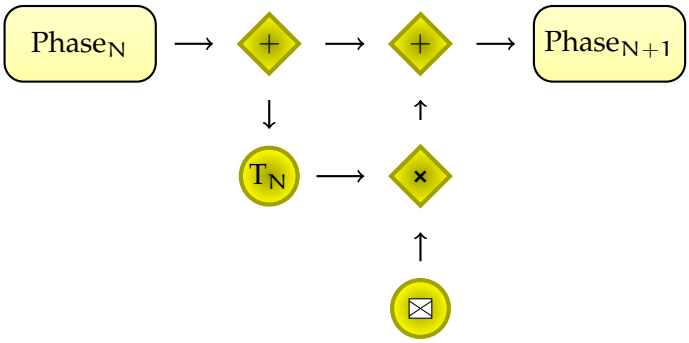

Figure 78: SBR Process Flow : Switch

Formally, the SBR cycle can be defined applying the pattern shown in Figure 78, where the passage from one phase to the next depends on any one between the triggering of a watchdog timer and the occuring of an external event. The event, usually generated by the policy agent, is a combination of the different on-line analysis techniques; the timer, instead, is triggered when the previous swith is completed.

The watchdog, in turn, is activated after the previous transition has been completed.

Drools Flow ensures that rules such as 10.16 are executed only when ruleflow group the rule belongs to is active, so the necessary actions on the actuators (blower, mixer, pumps, ...) can be applied as a *consequence* of a switch. The concrete actions for each phase are derived, trivially, from Table 29.

Listing 10.16: Controller Agent Policy

```
rule "Apply Switch"
// perfect rule
  ruleflow-group "F"
  when
    CurrentPhase( id == "F-1" )
  then
    // control actuators
    ...
    CurrentPhase newPhase = new CurrentPhase("F");
    dispatch(new InitPhase(newPhase));
end
```

10.4 CONCLUSIONS

10.4.1 *Summary : Default Event Flow*

Figure 79 summarizes the events which are involved in a normal, managed SBR cycle. The level of abstraction increases and the frequency lowers when moving from raw `Samples` to `Switch` commands: in particular, raw observations performed on the plant (the `Samples`) are used to detect relevant signal state/trend changes, which are mapped onto process state changes, which in turn cause phase changes in the plant. The events are correlated by computational *causality* and/or *aggregation*, but the criteria used to generate a complex event from simpler ones may involve several reasoning steps, conveniently performed by the agents, possibly exploiting a short-term memory (i.e. stateful reasoning sessions) and the integration with sub-symbolic modules. Remarkably, *logical* causality would characterize the *reversed* sequence of events: phase changes in the plant alter the process state, which in turn manifest as trend changes in the sampled signals. The use of imperfection in the reasoning allows to take care of the inherent uncertainty associated to this form of implicit abduction, in addition to the one generated by the measurement and process noise.

10.4.2 *Considerations*

The migration of the decisional applications from an off-line implementation to an on-line infrastructure has been possible thanks to the adoption of an appropriate infrastructure. Thus, it was possible to reproduce the results obtained from an a-posteriori analysis of the SBR cycles with minor conceptual modifications (e.g. see [11],[4],[1]). Unfortunately, the only serious limitation is that, given the current impossibilty of commanding the actuators directly, the only tests have been simulations on historical data, which reproduced the results obtained and published in the cited works. So, the effective improvements have involved the major structural (and in some cases formal) modifications:

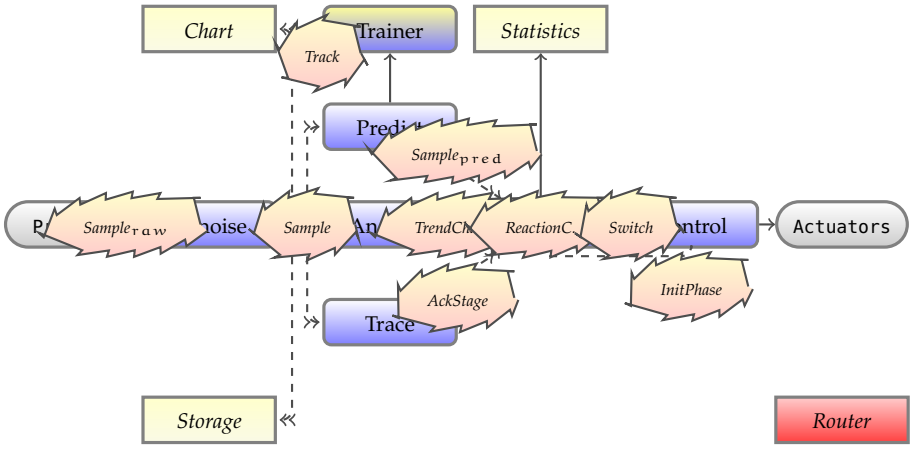

Figure 79: Conceptual Event Flow

with respect to other existing solutions, the proposed one has several advantages - albeit also some drawbacks. The former include:

- *Event-Orientation* : the application is possibly the first SBR management system completely based on CEP.
- *Flexibility and Reliability*: the number and type of agents can be varied dynamically at runtime. Additional modules can extend the functionalities of the system, improving the performance of existing ones or adding completely new ones. Likewise, the content/context-based routing allows to reconfigure the event flow dinamically, so that performances can degrade gracefully in case some services are not available.
- *Interactivity*: The different interaction options allow the development of systems where the different modules interact and cooperate in more complex ways than simply processing information in a parallel or cascaded way.
- *Abstraction*: Other than modelling some problems in a natural way, the underlying CEP approach allows to reason at different levels of abstraction. Complex problems can be decomposed in simpler ones, resolved individually and then recombined at the higher level.
- *Data Robustness* : The native use of imperfection in the reasoning allows to manage a source - the plant - which naturally produces noisy, partial, uncertain and/or vague data.
- *Integration*: Drools has turned out to be convenient tool for the integration of AI tools: while it remains a symbolic, declarative tool (with all the advantages of expandibility and explainability), it also allows to *use*, *embed* or even *emulate* other sub-symbolic

modules or techniques. For a quick reference, Figure 80 summarizes the technologies which have been used in the development of the different agents.

Its main limitation is essentially the lack of a standard model and a standard language to describe it. The use of a POJO class hierarchy can be a convenient internal representation, but may fail when the information have to be shared between agents. This, together with the lack of support for a communication standard, limits the interoperability of the system with third-party services.

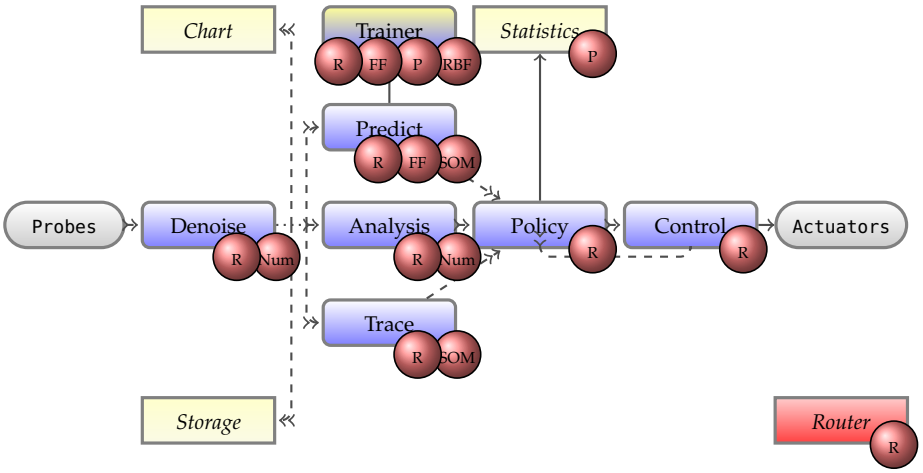

Figure 80: Use of (hybrid) AI techniques by the EPAs

Part V

CONCLUSIONS AND FUTURE WORKS

11

CONCLUSIONS AND FUTURE WORKS

Contents

11.1 CONCLUSIONS

The initial motivation of the long-term project discussed in this dissertation was to determine to what extent it could be possible to integrate symbolic and non symbolic AI techniques in the development of a complex automated management infrastructure for a complex system, and which benefits would come from it.

This has led the work in two main directions, although strongly correlated: the study of advanced software architectures and the research on the inference mechanisms of production rule systems. Interesting results were obtained in both fields.

11.1.1 *Results in the Development of (Production) Rule-Based Systems*

From a theoretical point of view, it was argued that the main benefit of SC techniques is their ability to deal with various types of imperfect information. This property is normally not shared by HC systems, whereas instead it would be useful in many practical cases, since imperfection is possibly an inherent property of information coming from the real world. However, it turned out that even a "hard" computing tool such as a rule-based system can be enhanced to support more general inference schemas, and can adapt to different types of imperfect reasoning simply using meta-data in the knowledge bases to configure it.

This result is relevant when paired to a recent result [92] which showed how several logic languages can be unified, but a similar generalization had not yet been performed on rule engines. While it introduces a possibly relevant computational overhead, which makes it less suitable for the large scale processing of a single type of rules, the enhancements make the engine important for applications where a minor number of rule activations, but with different semantics have to be processed.

In fact, the rule engine enhanced with imperfection allowed to be integrated with most types of SC techniques in more advanced ways than simply applying them in cascade or in parallel. Imperfect reasoning reduces the distance between the two worlds, to the point that rules have been proved to *emulate* the behaviour of some sub-symbolic algorithms. In more general cases, *strong* hybrid systems can be developed using one or more of the following integration patterns, ordered from the looser to the tighter:

- The imperfect rule engine invokes a SC module, passing the information it generated.
- A SC can provide imperfect information which is evaluated by the imperfect rule engine.
- The rule engine delegates the evaluation of an imperfect property to a SC module, which returns a generalized degree.
- The imperfect rule engine acts as a *polymorphic* hybrid system itself, emulating all or part of the behaviour of a SC module.

Remarkably, the integration is *native*: while rule-based systems usually allow the execution of generic code as a consequence of the firing of a rule, so that any algorithm can be implemented there, the proposed engine supports integration in the *premise* part of the rules.

Moreover, the enhancement was not performed on an ad-hoc engine, but rather as a development branch of a mainstream, open source BRMS - Drools - which features and potentialities can be compared to the most advanced rule management systems present on the market, further improving it.

11.1.2 *Results in the Development of (Environmental) Decision Support Systems*

The BRMS, enhanced with imperfection, has found a relevant role in the development of a hybrid Decision Support System - which actually shares some features of an Expert System - applied to the on-line monitoring and control of a SBR treatment plant. The proposed architecture has several interesting points, as well.

First, it combines agents, services and events in a unified framework. This is not particularly innovative, but becomes so if one considers that such integrated platforms are hardly applied to water treatment systems and even then, the existing ones rarely are completely

imperfection-aware (if any). In particular, the proposed application is possibly the first CEP-oriented solution to the problem of managing a SBR.

The architecture has all the advantages of a SOA, but, being event-driven, is even more loosely coupled and thus more flexible. From an implementation point of view, it relies on agents implemented using the full capabilities of the BRMS Drools - namely events, workflows, and repository in addition to the rule engine - and exploits the custom enhancements to support imperfection. In fact, it has been proved that all these components, originally designed for an enterprise context, can be successfully applied to the water treatment domain as well. Remarkably, in [1] we were among the first to propose the applicability of concepts taken from the field of business process management to the specific domain: this dissertation possibly completed the picture by showing the utility of events and shared rule bases.

The event-aware engine alone, integrated with imperfection, allowed to translate into rules - and thus to apply in real time - several management criteria which were initially developed for off-line control. Moreover, it facilitated their *contemporary* application, in a framework where different functionalities can be added or removed at run-time.

In conclusion, then, it was proved not only that symbolic and sub-symbolic techniques can be integrated tightly to improve the mutual performances, but also that an integrated intelligent system facilitates the development of advanced management infrastructures which outperform many of the existing ones, both in terms of flexibility and robustness. Such infrastructures are suitably applied to complex systems, such as waste-water treatment plants, where the only available knowledge and information are both characterized by a high degree of imperfection.

11.2 FUTURE WORKS

Despite the results obtained, this dissertation leaves many open challenges and issues for future research. They can be divided in two groups: improving the imperfect reasoning engine and expanding the applicative fields.

Improving the Engine

- *Upgrading the Implementation*: So far, `Drools Chance` has been implemented as a prototype in a branch of Drools 5.0. The next step will be an integration of the branch in the main engine, passing from a development version to a release one. Since *Chance* effectively alters the structure of the core nodes, it can't be added on top of the existing architecture, but must redefine it. In order not to force users to adopt it when not necessary, it has been planned to customize the behaviour of the compiler, which will

generate either a standard RETE or an imperfect one, according to the desired configuration.

- *Extending the support to other logics* : Only a few among all the possible `Factory` configurations have been implemented. Ideally, one `Factory` for each compatible logic should be prepared and released as an additional component, with an adequate documentation on all the possible values of the construction parameters.
- *Standardizing the Language* DRL is a popular and expressive language in the Drools community, but is not compliant with any standard. Instead, Reaction RuleML is an appealing standard language, with an expressiveness adequate to cover that of an production rule system. So, a DRL $\leftrightarrow$ RuleML translator would enhance the compatibility of Drools with other BRMS. Before doing so, however, it will be necessary to compare the two languages in detail to find the optimal mapping. Reaction RuleML, moreover, does not support imperfection, so any necessary extension of the same will be considered.
- *Adding support for Semantic Reasoning*: Recently, there has been much interest in the integration of rules and ontologies, as well as the integration of ontologies with imperfection. `Drools Chance` would then allow to reason with imperfect ontologies, but up to what level of expressiveness has not been studied yet.
- *Defining a Benchmark Suite*: It is difficult to evaluate the performance of the RETE algorithm in an objective way, especially because they depend on how much a rule base and the current set of WME can exploit the node sharing and the other optimization techniques. Although there exist some benchmarks, imperfection complicates things because the structure of an imperfect rule base tends to be quite different (on average, the rules are fewer, but with a more complex structure). Thus, it would be more appropriate to devise one or more ad-hoc, standard benchmarks, so to evaluate the computational cost variations due to the engine's implementation and not due to the rules themselves.

Finding additional Applications

- *Adding Ontologies to the architecture*: As discussed in Chapter 10, the main limitation of the proposed architecture is the lack of a standardized language for the agents to communicate with. Even if a language was chosen or devised, however, the concepts expressed in this language would still have to be shared. To do so, an ontology for the water treatment domain in general, and SBRs in particular, would be the best option.

- *Expanding the Knowledge Base*: The implemented agents cover but a small part of the existing management policies for SBR. In fact, no faulty condition is treated, and the diagnostic capabilities are very limited. The proposed methodology, however, allows to build the system incrementally, even as new knowledge is acquired.
- *Applying the architecture to different contexts*: SBRs are not the only possible type of plant the architecture has been devised for. The next immediate target will be a full-scale plant situated near Calderara di Reno (Bologna, Italy), owned and managed by HERA, which has already been equipped with probes and a remote data acquisition system. The possible applications, however, would not be limited to treatment plants, but possibly to all contexts where a knowledge-based decision support system would be appropriate.

BIBLIOGRAPHY

[1] *A simple algorithm for efficient piecewise linear approximation of space curves*, volume 2, 1997. (Cited on page 227.)

[2] Ruleml - rule markup language. URL http://www.ruleml.org. (Cited on pages xviii, 107, and 108.)

[3] W3c uncertainty reasoning for the web incubator group, http://www.w3.org/2005/incubator/urw3/xgr-urw3. (Cited on page 12.)

[4] MS BizTalk. URL www.microsoft.com/biztalk/en/us/default.aspx. (Cited on page 105.)

[5] *Reasoning about rational agents*. MIT Press, Cambridge, MA, USA, 2001. ISBN 0-262-23213-8. (Cited on page 94.)

[6] Business process model and notation, 2009. URL www.omg.org/spec/BPMN/2.0. (Cited on page 113.)

[7] Fuzzy CLIPS. URL ai.iit.nrc.ca/IRpublic/fuzzy/fuzzyClips/fuzzyCLIPSIndex.html. (Cited on pages 108, 121, and 158.)

[8] JBoss Drools - Business Logic Integration Platform. (Cited on pages 105, 110, and 122.)

[9] Scientio InRule. URL www.inrule.com. (Cited on page 105.)

[10] FICO Blaze Advisor. URL www.fico.com/en/Products/DMTools/Pages/FICO-Blaze-Advisor-System.aspx. (Cited on page 105.)

[11] Jboss jbpm. URL www.jboss.com/products/jbpm. (Cited on page 113.)

[12] Jess. URL www.nrc-cnrc.gc.ca/eng/ibp/iit.html. (Cited on pages 108 and 121.)

[13] Java messaging system. URL http://java.sun.com/products/jms/. (Cited on page 213.)

[14] JRules. URL www.ilog.com/products/jrules. (Cited on page 105.)

[15] OpenLexicon. URL www.openlexicon.org. (Cited on page 105.)

[16] ObjectConnections CommonKnowledge. URL www.objectconnections.com. (Cited on page 105.)

[17] OpenRules. (Cited on page 105.)

[18] PegaRules. URL www.pega.com/Products/RulesTechnology.asp. (Cited on page 105.)

[19] Technical report. (Cited on page 198.)

[20] Sbvr - semantics of business vocabulary and business rules, 2008. URL http://www.omg.org/spec/SBVR/1.0/PDF. (Cited on page 107.)

[21] Scientio XMLMiner. URL www.scientio.com. (Cited on page 109.)

[22] FuzzyShell. URL cobweb.ecn.purdue.edu/RVL/Projects/Fuzzy/. (Cited on page 108.)

[23] Swrl: A semantic web rule language combining owl and ruleml. URL http://www.w3.org/Submission/SWRL/. (Cited on page 107.)

[24] Web services business process execution language version 2.0, 2007. URL docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html. (Cited on page 113.)

[25] XpertRule. URL www.xpertrule.com. (Cited on page 105.)

[26] Paschke A., Kozlenkov A., Boley H., Kifer M., Tabet S., Dean M., and Barrett K. Reaction ruleml, 2006. (Cited on page 119.)

[27] Agnar Aamodt and Enric Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches, 1994. (Cited on page 60.)

[28] Chris Van Aart, Giovanni Caire, Ruurd Pels, and Federico Bergenti. Creating and using ontologies in agent communication, 1995. (Cited on page 95.)

[29] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Mining association rules between sets of items in large databases. pages 207–216, 1993. (Cited on page 161.)

[30] Gennady Agre. Diagnostic bayesian networks, 1996. (Cited on page 50.)

[31] D. Aguado, A. Ferrer, J. Ferrer, and A. Seco. Multivariate spc of a sequencing batch reactor for wastewater treatment. *Chemometrics and Intelligent Laboratory Systems*, 85(1):82 – 93, 2007. ISSN 0169-7439. doi: DOI:10.1016/j.chemolab.2006.05.003. (Cited on page 203.)

[32] D. Aguado, J. Ribes, T. Montoya, J. Ferrer, and A. Seco. A methodology for sequencing batch reactor identification with artificial neural networks: A case study. *Computers & Chemical Engineering*, 33(2):465 – 472, 2009. ISSN 0098-1354. (Cited on page 203.)

[33] B.S. AkIn and A. Ugurlu. Monitoring and control of biological nutrient removal in a sequencing batch reactor. *Process Biochemistry*, 40(8):2873 – 2878, 2005. ISSN 1359-5113. (Cited on page 201.)

[34] James F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, 1983. ISSN 0001-0782. (Cited on pages 91 and 112.)

[35] M N Almasri and J J Kaluarachchi. Modular neural networks to predict the nitrate distribution in ground water using the on-ground nitrogen loading and recharge data. *Environmental Modelling & Software*, 20(7):851e871, 2005. (Cited on page 83.)

[36] Grigoris Antoniou, Carlos V. Damásio, Benjamin Grosof, Ian Horrocks, Michael Kifer, Jan Maluszynski, and Peter. Combining Rules and Ontologies. A survey., 2005. (Cited on pages 104 and 179.)

[37] Grigoris Antoniou, David Billington, Guido Governatori, and Michael J. Maher. Embedding defeasible logic into logic programming. *TPLP*, 6(6):703–735, 2006. (Cited on page 170.)

[38] Aleksander Astel. Chemometrics based on fuzzy logic principles in environmental studies. *Talanta*, 72(1):1–12, 2007. ISSN 1873-3573. (Cited on page 83.)

[39] Ioannis N Athanasiadis and Pericles A Mitkas. A Methodology for Developing Environmental Information Systems with Software Agents. *Technology*, pages 119–137, 2009. (Cited on page 96.)

[40] Ioannis N. Athanasiadis, Marios Milis, Pericles a. Mitkas, and Silas C. Michaelides. A multi-agent system for meteorological radar data management and decision support. *Environmental Modelling & Software*, 24(11):1264–1273, 2009. ISSN 13648152. (Cited on page 96.)

[41] Montse Aulinas. Agents as a Decision Support Tool in Environmental Processes: The State of the Art. *Water*, pages 5–35, 2009. (Cited on page 96.)

[42] M Azlan Hussain. Review of the applications of neural networks in chemical process control - simulation and online implementation. *Artificial Intelligence in Engineering*, 13(1):55–68, January 1999. ISSN 09541810. (Cited on page 82.)

[43] A Azwar, M Hussain, and K B Ramachandran. The study of neural network-based controller for controlling dissolved oxygen concentration in a sequencing batch reactor. *Bioprocess and Biosystems Engineering*, 28(4):251–265, 2005. ISSN 1615-7591. (Cited on page 83.)

[44] MichałBaczyński and Balasubramaniam Jayaram. (s, n)- and r-implications: A state-of-the-art survey. *Fuzzy Sets Syst.*, 159(14): 1836–1859, 2008. ISSN 0165-0114. (Cited on page 156.)

[45] J Baeza, D Gabriel, and J Lafuente. An expert supervisory system for a pilot WWTP. *Environmental Modelling and Software*, 14(5): 383–390, March 1999. ISSN 13648152. (Cited on page 87.)

[46] Juan Baeza, David Gabriel, Javier Bejar, and Javier Lafuente. A Distributed Control System Based on Agent Architecture for Wastewater Treatment. *Computer-Aided Civil and Infrastructure Engineering*, 17(2):93–103, March 2002. ISSN 1093-9687. (Cited on page 95.)

[47] Ying Bai, Hanqi Zhuang, and Dali Wang. *Advanced Fuzzy Logic Technologies in Industrial Applications (Advances in Industrial Control)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 1846284686. (Cited on page 52.)

[48] Debdeep Banerjee and Jeffrey Tweedale. Reactive (re) planning agents in a dynamic environment. In *Intelligent Information Processing*, pages 33–42, 2006. (Cited on page 95.)

[49] Regina Barzilay, Daryl Mccullough, Owen Rambow, Jonathan Decristofaro, Tanya Korelsky, Benoit Lavoie, and Cogentex Inc. A new approach to expert system explanations. In *In 9thInternational Workshop on Natural Language Generation*, pages 78–87, 1998. (Cited on page 59.)

[50] CÃ©dric Baudrit, InÃ©s Couso, and Didier Dubois. Probabilities of events induced by fuzzy random variables. In Eduard Montseny and Pilar Sobrevilla, editors, *EUSFLAT Conf.*, pages 541–546. Universidad Polytecnica de Catalunya, 2005. ISBN 84-7653-872-3. (Cited on page 26.)

[51] BenjamÃn C. Bedregal, Renata H.S. Reiser, and GraÃ§aliz P. Dimuro. Xor-implications and e-implications: Classes of fuzzy implications based on fuzzy xor. *Electronic Notes in Theoretical Computer Science*, 247:5 – 18, 2009. ISSN 1571-0661. Proceedings of the Third Workshop on Logical and Semantic Frameworks with Applications (LSFA 2008). (Cited on page 156.)

[52] Richard E. Bellman. *Adaptive control processes - A guided tour*. Princeton University Press, Princeton, New Jersey, U.S.A., 1961. (Cited on page 58.)

[53] Tom Belwood, Luc Clément, David Ehnebuske, Andrew Hately, Maryann Hondo, Yin Leng Husband, Karsten Januszewski, Sam Lee, Barbara McKee, Joel Munter, and Claus von Riegen. UDDI Version 3.0. http://uddi.org/pubs/uddi_v3.htm, 2000. (Cited on page 90.)

[54] Jean-Marc Bernard. An introduction to the imprecise dirichlet model for multinomial data. *International Journal of Approximate Reasoning*, 39(2-3):123 – 150, 2005. ISSN 0888-613X. doi: 10.1016/j.ijar.2004.10.002. (Cited on pages 18 and 164.)

[55] Philippe Besnard and Anthony Hunter. A logic-based theory of deductive arguments, 2001. (Cited on page 59.)

[56] J.C. Bezdek. The thirsty traveler visits gamont: a rejoinder to ldquo;comments on fuzzy sets-what are they and why? rdquo;. *Fuzzy Systems, IEEE Transactions on*, 2(1):43 –45, Feb 1994. ISSN 1063-6706. doi: 10.1109/91.273125. (Cited on pages 24 and 27.)

[57] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 1 edition, October 2007. ISBN 0387310738. (Cited on pages 16, 17, 42, 44, 47, 49, 50, 62, 66, and 234.)

[58] Jeff Blee, David Billington, and Abdul Sattar. Reasoning with levels of modalities in bdi logic. pages 410–415, 2009. (Cited on page 95.)

[59] James L. Blue and Patrick J. Grother. Training feed-forward neural networks using conjugate gradients. In *In SPIE*, pages 179–190, 1992. (Cited on page 41.)

[60] Fernando Bobillo and Umberto Straccia. Fuzzy description logics with general t-norms and datatypes. *Fuzzy Sets and Systems*, 160(23):3382 – 3402, 2009. ISSN 0165-0114. Theme: Computer Science. (Cited on page 179.)

[61] G Bortone, S Longhi, L Luccarini, E Porrà, and P Ratini. On-line Control of a SBR Reactor for the Biological Wastewater Treatment. *Time*. (Cited on page 205.)

[62] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Nielsen, S. Thatte, and D. Winer. Simple Object Access Protocol (SOAP) 1.1. http://www.w3.org/TR/soap, 2000. (Cited on pages 90 and 213.)

[63] Leo Breiman and Leo Breiman. Bagging predictors. In *Machine Learning*, pages 123–140, 1996. (Cited on page 235.)

[64] Lars Brenna, Johannes Gehrke, Mingsheng Hong, and Dag Johansen. Distributed event stream processing with nondeterministic finite automata. In *DEBS '09: Proceedings of the*

Third ACM International Conference on Distributed Event-Based Systems, pages 1–12, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-665-6. (Cited on page 92.)

[65] R. Bringhurst. *The Elements of Typographic Style*. Version 2.5. Hartley & Marks, Publishers, 2002. (Cited on page 287.)

[66] SebastiÃ Puig Broch. Operation and control of SBR processes for enhanced biological nutrient removal from wastewater, February 2008. (Cited on pages 198 and 204.)

[67] Re Bronstein, Joydip Das, Marsha Duro, Rich Friedrich, Gary Kleyner, Martin Mueller, Sharad Singhal, Ira Cohen, G. Kleyner, M. Mueller, S. Singhal, and I. Cohen. Self-aware services: Using bayesian networks for detecting anomalies in internet-based services. In *Northwestern University and Stanford University. Gary (Igor*, pages 623–638. Publishing, 2001. (Cited on page 86.)

[68] Lee Brownston, Robert Farrell, Elaine Kant, and Nancy Martin. *Programming expert systems in OPS5: an introduction to rule-based programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1985. ISBN 0-201-10647-7. (Cited on page 122.)

[69] Bruce G. Buchanan and Edward H. Shortliffe. *Rule-based Expert Systems : the MYCIN experiments of the Stanford Heuristic Programming Project*. Addison-Wesley, Reading, Mass. [u.a.], 1984. ISBN 0-201-10172-6. (Cited on pages 104, 109, and 153.)

[70] Bruce G. Buchanan and Edward H. Shortliffe. *Rule-based Expert Systems : the MYCIN experiments of the Stanford Heuristic Programming Project*. Addison-Wesley, Reading, MA, 1984. ISBN 0-201-10172-6. (Cited on page 59.)

[71] Martin Dietrich Buhmann. *Radial basis functions*. Cambridge University Press, 2003. ISBN 0521633389, 9780521633383. (Cited on page 43.)

[72] Sergiu Caraman and Marian Barbu. The identification and robust control of a biological wastewater treatment process. *2008 IEEE International Conference on Automation, Quality and Testing, Robotics*, pages 37–42, May 2008. (Cited on page 84.)

[73] M. Casellas, C. Dagot, and M. Baudu. Set up and assessment of a control strategy in a sbr in order to enhance nitrogen and phosphorus removal. *Process Biochemistry*, 41(9):1994 – 2001, 2006. ISSN 1359-5113. (Cited on page 204.)

[74] Luigi Ceccaroni. What if a wastewater treatment plant were a town of agents. *Review Literature And Arts Of The Americas*. (Cited on page 96.)

[75] Luigi Ceccaroni, Supervisors Ulises CortÃ©s, and Miquel SÃ nchez-marrÃ¨. Ontowedss - an ontology-based environmental decision support system for the management of wastewater treatment plants, 2001. (Cited on pages 87 and 222.)

[76] D. Cecil and M. Kozlowska. Software sensors are a real alternative to true sensors. *Environmental Modelling and Software*, 25 (5):622 – 625, 2010. ISSN 1364-8152. Thematic Issue on MODELLING AND AUTOMATION OF WATER AND WASTEWATER TREATMENT PROCESSES. (Cited on pages 80 and 83.)

[77] S. Ceri, G. Gottlob, and L. Tanca. What you always wanted to know about datalog (and never dared to ask). *IEEE Transactions on Knowledge and Data Engineering*, (1(1)):146–66, 1989. (Cited on page 166.)

[78] C. H. CHANG and O. J. HAO. Sequencing batch reactor system for nutrient removal : ORP and pH profiles. *Journal of chemical technology and biotechnology*, 67(1):27–38, 1996. ISSN 0268-2575. (Cited on page 201.)

[79] David Chappell. *Enterprise Service Bus*. O'Reilly Media, Inc., 2004. ISBN 0596006756. (Cited on page 213.)

[80] S Charbonnier, C Garciabeltan, C Cadet, and S Gentil. Trends extraction and analysis for complex system monitoring and decision support. *Engineering Applications of Artificial Intelligence*, 18 (1):21–36, 2005. ISSN 09521976. doi: 10.1016/j.engappai.2004.08.023. (Cited on page 213.)

[81] a. Charef, a. Ghauch, and M. Martin-Bouyer. An adaptive and predictive control strategy for an activated sludge process. *Bioprocess Engineering*, 23(5):529–534, November 2000. ISSN 16157591. doi: 10.1007/s004499900191. (Cited on page 81.)

[82] Rick Chartrand. Numerical differentiation of noisy , nonsmooth data. *Energy*, (1):1–9, 2005. (Cited on page 225.)

[83] N. Chomsky. Three models for the description of language. *Information Theory, IRE Transactions on*, 2(3):113–124, January 2003. doi: 10.1109/TIT.1956.1056813. (Cited on page 124.)

[84] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl, 2001. (Cited on page 90.)

[85] F. Ciappelloni, D. Mazouni, J. Harmand, and L. Lardon. On-line supervision and control of an aerobic SBR process. *Water science and technology : a journal of the International Association on Water Pollution Research*, 53(1):169–77, 2006. ISSN 0273-1223. (Cited on page 203.)

[86] Giulianella Coletti and Romano Scozzafava. Conditional probability, fuzzy sets, and possibility: a unifying view. *Fuzzy Sets and Systems*, 144(1):227–249, 2004. (Cited on pages 26 and 27.)

[87] J. Comas, I. Rodríguez-Roda, K.V. Gernaey, C. Rosen, U. Jeppsson, and M. Poch. Risk assessment modelling of microbiology-related solids separation problems in activated sludge systems. *Environmental Modelling & Software*, 23(10-11):1250–1261, 2008. ISSN 13648152. (Cited on page 83.)

[88] Joaquim Comas. Development, implementation and evaluation of an activated sludge supervisory system for the granollers wwtp. *AI Commun.*, 14(1):65–66, 2001. ISSN 0921-7126. (Cited on pages 86 and 87.)

[89] Juan Manuel Corchado and Aitor Mata. Osm: A multi-agent system for modeling and monitoring the evolution of oil slicks in open oceans. *Expert Systems*, pages 91–117, 2009. (Cited on page 96.)

[90] Gianpaolo Cugola, Alessandro Margara, and Matteo Migliavacca. Context-aware publish-subscribe: Model, implementation, and evaluation. *2009 IEEE Symposium on Computers and Communications*, pages 875–881, July 2009. (Cited on page 219.)

[91] Carlos Viegas Damasio and Luis Moniz Pereira. Hybrid probabilistic logic programs as residuated logic programs, 2001. (Cited on page 168.)

[92] Carlos Viegas Damásio, Jeff Z. Pan, Giorgos Stoilos, and Umberto Straccia. Representing uncertainty in ruleml. *Fundam. Inf.*, 82(3):265–288, 2008. ISSN 0169-2968. (Cited on pages 109, 117, 123, 129, 134, 137, 144, and 254.)

[93] Q. Dan and J. Dudeck. Some problems related with probabilistic interpretations for certainty factors. pages 538 –545, jun 1992. (Cited on pages 104 and 154.)

[94] D Davis. Agent-based decision-support framework for water supply infrastructure rehabilitation and development. *Computers, Environment and Urban Systems*, 24(3):173–190, May 2000. ISSN 01989715. doi: 10.1016/S0198-9715(99)00056-3. (Cited on page 96.)

[95] Ramon Lopez de Mantaras and Enric Plaza. Case-based reasoning: An overview, 1996. (Cited on page 39.)

[96] Michael Decker and Rebecca Bulander. A Platform for Mobile Service Provisioning Based on SOA-Integration. pages 72–84, 2008. (Cited on page 90.)

[97] Thierry Denoeux. Modeling vague beliefs using fuzzy-valued belief structures, 1998. (Cited on page 26.)

[98] R Denzer. Generic integration of environmental decision support systems-state-of-the-art. *Environmental Modelling & Software*, 20 (10):1217–1223, 2005. (Cited on page 87.)

[99] Ralf Der, Gerd Balzuweit, and Michael Herrmann. Building nonlinear data models with self-organizing maps. (Cited on page 45.)

[100] B Dieu. Applications of automatic control systems for chlorination and dechlorination processes in wastewater treatment plants. *ISA Transactions*, 34(1):21–28, March 1995. ISSN 00190578. doi: 10.1016/0019-0578(94)00041-J. (Cited on page 82.)

[101] Francisco Javier Diez and F. J. D'iez. Parameter adjustment in bayes networks. the generalized noisy or-gate. In *In Proceedings of the 9th Conference on Uncertainty in Artificial Intelligence*, pages 99–105. Morgan Kaufmann, 1993. (Cited on page 66.)

[102] Remco M Dijkman, Marlon Dumas, and Chun Ouyang. Formal Semantics and Analysis of BPMN Process Models using Petri Nets. *Language*, pages 1–30. (Cited on page 246.)

[103] Liya Ding, Hoon Heng Teh, Peizhuang Wang, and Ho Chung Lui. A prolog-like inference system based on neural logic—an attempt towards fuzzy neural logic programming. *Fuzzy Sets Syst.*, 82(2):235–251, 1996. ISSN 0165-0114. (Cited on page 66.)

[104] Robert B. Doorenbos. Production matching for large learning systems. Technical Report CS-95-113, Carnegie Mellon University, School of Computer Science. (Cited on pages 37 and 149.)

[105] Marco Dorigo, Mauro Birattari, Thomas StÃ¼tzle, UniversitÃ© Libre, De Bruxelles, and Av F. D. Roosevelt. Ant colony optimization - artificial ants as a computational intelligence technique. *IEEE Comput. Intell. Mag*, 1:28–39, 2006. (Cited on page 95.)

[106] D. Dubois and H. Prade. *Fuzzy Sets and Systems: Theory and Applications*. Academic Press, 1980. (Cited on page 53.)

[107] D. Dubois, J. Lang, and H. Prade. Automated reasoning using possibilistic logic: Semantics, belief revision, and variable certainty weights. *IEEE Trans. on Knowl. and Data Eng.*, 6(1):64–71, 1994. ISSN 1041-4347. (Cited on page 161.)

[108] Didier Dubois and Henri Prade. The three semantics of fuzzy sets. *Fuzzy Sets Syst.*, 90(2):141–150, 1997. ISSN 0165-0114. doi: http://dx.doi.org/10.1016/S0165-0114(97)00080-8. (Cited on page 22.)

[109] Didier Dubois and Henri Prade. Possibility theory and data fusion in poorly informed environments. *Control Engineering Practice*, 2(5):811–823, 1994. (Cited on page 206.)

[110] Didier Dubois and Henri Prade. Fuzzy sets and probability : Misunderstandings, bridges and gaps. In *In Proceedings of the Second IEEE Conference on Fuzzy Systems*, pages 1059–1068. IEEE, 1993. (Cited on pages 24 and 26.)

[111] Didier Dubois and Henri Prade. Possibility theory, probability theory and multiple-valued logics: A clarification. *Annals of Mathematics and Artificial Intelligence*, 32(1-4):35–66, 2001. ISSN 1012-2443. doi: http://dx.doi.org/10.1023/A:1016740830286. (Cited on pages 22 and 24.)

[112] Didier Dubois and Henri Prade. Possibilistic logic. . doi: 10.1.1.54.7792. (Cited on pages 157 and 161.)

[113] Didier Dubois and Henri Prade. Possibilistic logic. . doi: 10.1.1.54.7792. (Cited on page 172.)

[114] Didier Dubois, Eyke Hullermeier, and Henri Prade. A systematic approach to the assessment of fuzzy association rules. (Cited on page 165.)

[115] Didier Dubois, JÃ©rÃ´me Lang, and Henri Prade. Possibilistic logic, 1992. (Cited on page 66.)

[116] Didier Dubois, Laurent Foulloy, Gilles Mauris, and Henri Prade. Probability-Possibility transformations, triangular fuzzy sets, and probabilistic inequalities. *Reliable Computing*, 10(4):273–297, 2004. doi: 10.1023/B:REOM.0000032115.22510.b5. (Cited on page 26.)

[117] Didier Dubois, Francesc Esteva, Lluis Godo, and Henri Prade. Fuzzy-set based logics - an history-oriented presentation of their main developments. In D. M. Gabbay and J. Woods, editors, *Handbook of the History of Logic, Volume 8: The Many Valued and Non-monotonic Turn in Logic*, pages 325–449. Elsevier, 2007. (Cited on pages 24, 37, 53, 120, 155, 157, 160, and 178.)

[118] Didier J. Dubois and H. Prade. *Fuzzy Sets and Systems : Theory and Applications (Mathematics in Science and Engineering)*. Academic Press, October 1980. ISBN 0122227506. (Cited on page 26.)

[119] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2 edition, November 2000. ISBN 0471056693. (Cited on pages 58, 67, and 92.)

[120] John Durkin. Expert systems: A view of the field. *IEEE Intelligent Systems*, 11:56–63, 1996. ISSN 0885-9000. doi: http://doi.ieeecomputersociety.org/10.1109/64.491282. (Cited on page 104.)

[121] Richard Dybowski and Stephen J. Roberts. Confidence intervals and prediction intervals for feed-forward neural networks. In

Clinical Applications of Artificial Neural Networks, pages 298–326. University Press, 2001. (Cited on page 47.)

[122] E.Gamma, R.Helm, R.Johnson, and J.Vlissides. *Design Patterns Elements of Reusable Object-Oriented Software*. Addison-Wesley, Massachusetts, 2000. ISBN 0-201-63361-2. (Cited on pages 141, 147, and 148.)

[123] Ralf Eickhoff and Ulrich Rückert. Robustness of radial basis functions. *Neurocomput.*, 70(16-18):2758–2767, 2007. ISSN 0925-2312. (Cited on page 234.)

[124] Mats Ekman, Berndt Bjorlenius, and Mikael Andersson. Control of the aeration volume in an activated sludge process using supervisory control strategies. *Water research*, 40(8):1668–76, 2006. ISSN 0043-1354. doi: 10.1016/j.watres.2006.02.019. (Cited on page 82.)

[125] Thomas Erl. *SOA Principles of Service Design (The Prentice Hall Service-Oriented Computing Series from Thomas Erl)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2007. ISBN 0132344823. (Cited on page 88.)

[126] Certainty Factors. Probabilistic Reasoning and Certainty Factors. 186(1975):263–271, 1976. (Cited on page 154.)

[127] Liping Fan and Yang Xu. A pca-combined neural network software sensor for sbr processes. In *ISNN '07: Proceedings of the 4th international symposium on Neural Networks*, pages 1042–1047, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-72392-9. (Cited on page 203.)

[128] J Ferrer, M Rodrigo, a Seco, and J Penyaroja. Energy saving in the aeration process by fuzzy logic control, 1998. ISSN 02731223. (Cited on page 83.)

[129] N Fiocchi, E Ficara, R Canziani, L Luccarini, F Ciappelloni, P Ratini, M Pirani, and S Mariani. SBR on-line monitoring by set-point titration. *Water Science and Technology: A Journal of the International Association on Water Pollution Research*, 53(4-5):541–549, 2006. ISSN 0273-1223. PMID: 16722107. (Cited on page 203.)

[130] M. Fiter, D. Guell, J. Comas, J. Colprim, M. Poch, and I. Rodriguez-Roda. Energy Saving in a Wastewater Treatment Process: an Application of Fuzzy Logic Control. *Environmental Technology*, 26(11):1263–1270, 2005. ISSN 0959-3330. (Cited on page 83.)

[131] Melvin Fitting. Kleene's three valued logics and their children. *Fundam. Inf.*, 20(1-3):113–131, 1994. ISSN 0169-2968. (Cited on page 157.)

[132] Xavier Flores-Alsina, Ignasi RodrÃguez-Roda, GÃ¼rkan Sin, and Krist V. Gernaey. Multi-criteria evaluation of wastewater treatment plant control strategies under uncertainty. *Water Research*, 42(17):4485 – 4497, 2008. ISSN 0043-1354. doi: 10.1016/j.watres.2008.05.029. (Cited on pages 80 and 82.)

[133] Christopher Fogelberg, Vasile Palade, and Phil Assheton. Belief propagation in fuzzy bayesian networks. 2009. (Cited on page 66.)

[134] Charles Forgy. Rete: A fast algorithm for the many patterns/many objects match problem. *Artif. Intell*, 19(1):17–37, 1982. (Cited on pages 37, 65, and 130.)

[135] M Frehner and M Brandli. Virtual database: Spatial analysis in a Web-based data management system for distributed ecological data. *Environmental Modelling & Software*, 21(11):1544–1554, 2006. ISSN 13648152. doi: 10.1016/j.envsoft.2006.05.012. (Cited on page 84.)

[136] Nir Friedman, Dan Geiger, Moises Goldszmidt, G. Provan, P. Langley, and P. Smyth. Bayesian network classifiers. In *Machine Learning*, pages 131–163, 1997. (Cited on page 49.)

[137] Norbert Fuhr. Probabilistic datalog - a logic for powerful retrieval methods. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1995. (Cited on page 166.)

[138] Lou Goble. *The Blackwell Guide to Philosophical Logic*. Wiley-Blackwell, August 2001. ISBN 0631206930. (Cited on page 35.)

[139] Suran Goonatilake and Sukhdev Khebbal, editors. *Intelligent Hybrid Systems*. John Wiley & Sons, Inc., New York, NY, USA, 1994. ISBN 0471942421. (Cited on pages 63, 174, and 175.)

[140] J. Goppert and W. Rosenstiel. Regularized SOM-training: a solution to the topology- approximation dilemma? In *ICNN 96. The 1996 IEEE International Conference on Neural Networks*, volume 1, pages 38–43. New York, NY, USA, 1996. (Cited on page 45.)

[141] Guido Governatori, Vineet Padmanabhan, and Antonino Rotolo. A.: Rule-based agents in temporalised defeasible logic. In *Ninth Pacific Rim International Conference on Artificial Intelligence. Number 4099 in LNAI*, pages 31–40. Springer, 2006. (Cited on page 95.)

[142] A. K. GUPTA, S. K. GUPTA, and Rashmi S. PATIL. A comparison of water quality indices for coastal water. 38(11):2711–2725, 2003. (Cited on page 75.)

[143] Ian Hacking. *Logic of statistical inference*. CUP Archive, 1965. ISBN 0521290597, 9780521290593. (Cited on page 16.)

[144] Petr Hájek. *Metamathematics of Fuzzy Logic (Trends in Logic)*. Springer, 1 edition, November 2001. ISBN 1402003706. (Cited on pages 34, 66, 155, 157, and 178.)

[145] Lawrence Hall. Rule chaining in fuzzy expert systems. (Cited on page 140.)

[146] Joseph Y. Halpern. An analysis of first-order logics of probability. *Artificial Intelligence*, 46:311–350, 1990. (Cited on page 66.)

[147] Joseph Y. Halpern. *Reasoning about Uncertainty*. The MIT Press, October 2003. ISBN 0262083205. (Cited on page 12.)

[148] M M Hamed, M G Khalafallah, and E A Hassanien. Prediction of wastewater treatment plant performance using artificial neural networks. *Environmental Modelling & Software*, 19(10):919–928, 2004. (Cited on page 83.)

[149] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Macmillan, New York, 1994. (Cited on pages 40, 42, 44, and 185.)

[150] David Heckerman. A tutorial on learning with bayesian networks. Technical report, Learning in Graphical Models, 1996. (Cited on page 48.)

[151] David Heckerman, Abe Mamdani, Michael P. Wellman, F. Nadi, A. Agogino, and D. Hodges Use. Real-world applications of bayesian networks, 1995. (Cited on page 49.)

[152] D Hegg, T Cohen, Q Song, and N Kasabov. Intelligent Control of Sequencing Batch Reactors (SBRs) for Biological Nitrogen Removal 1 . Description of the task. pages 1–6. (Cited on page 204.)

[153] M. Henze, P. Harremoes, J. Jansen, and E. Arvin. *Wastewater treatment*. Springer, third edition, 2002. (Cited on page 76.)

[154] Mogens Henze, Willi Gujer, and Takashi Mino. *Activated Sludge Models ASM1, ASM2, ASM2D and ASM3*. IWA Publishing, 1 edition, February 2007. ISBN 1900222248. (Cited on page 81.)

[155] Gregor Hohpe and Bobby Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003. ISBN 0321200683. (Cited on page 218.)

[156] Eyke HÃ¼llermeier, Didier Dubois, Henri Prade, De Toulouse, and Universit'e Paul Sabatier. Fuzzy rules in case-based reasoning. In *in Conf. AFIA99 Raisonnement Ã Partir de Cas*, pages 45–54, 1999. (Cited on page 65.)

[157] Sima J. Neural expert systems. *Neural Networks*, 8:261–271(11), 1995. (Cited on page 104.)

[158] Peter Jackson. *Introduction to Expert Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998. ISBN 0201876868. (Cited on pages 35 and 37.)

[159] A K Jain, M N Murty, and P. J. Flynn. Data clustering: A review, 1999. (Cited on pages 47 and 62.)

[160] I. Jubany, J. a. Baeza, J. Carrera, and J. Lafuente. Model-based Design of a Control Strategy for Optimal Start-up of a High-Strength Nitrification System. *Environmental Technology*, 28(2): 185–194, 2007. ISSN 0959-3330. doi: 10.1080/09593332808618780. (Cited on page 81.)

[161] Kim K.-S. and Han I. The cluster-indexing method for case-based reasoning using self-organizing maps and learning vector quantization for bond rating cases. *Expert Systems with Applications*, 21:147–156(10), October 2001. (Cited on page 65.)

[162] S Kaloudis, C Costopoulou, N Lorentzos, a Sideridis, and M Karteris. Design of forest management planning DSS for wildfire risk reduction. *Ecological Informatics*, 3, 2008. ISSN 15749541. doi: 10.1016/j.ecoinf.2007.07.008. (Cited on page 86.)

[163] Akimoto Kamiya, Seppo J. Ovaska, Rajkumar Roy, and Shigenobu Kobayashi. Fusion of soft computing and hard computing for large-scale plants: a general model. *Appl. Soft Comput.*, 5(3):265–279, 2005. (Cited on page 86.)

[164] M. R. KATEBI, M. A. JOHNSON, J. WILKIE, and G. MCCLUSKEY. Control and management of wastewater treatment plants. *IEE conference publication*, pages 433–438, 1998. (Cited on page 82.)

[165] David Keil and Dina Goldin. Indirect interaction in environments for multi-agent systems. pages 68–87. 2006. (Cited on page 95.)

[166] Kristian Kersting and Uwe Dick. Balios: the engine for bayesian logic programs. In *PKDD '04: Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 549–551, New York, NY, USA, 2004. Springer-Verlag New York, Inc. ISBN 3-540-23108-0. (Cited on page 109.)

[167] Kristian Kersting and Luc De Raedt. Bayesian logic programs. In *Proceedings of the Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming*, pages 138–155, 2000. (Cited on pages 66, 167, and 181.)

[168] R. Khosla and T.S. Dillon. Welding symbolic ai systems with neural networks and their applications. In *Neural Networks Proceedings, 1998. IEEE World Congress on Computational Intelligence. The 1998 IEEE International Joint Conference on*, volume 1, pages

29–34 vol.1, May 1998. doi: 10.1109/IJCNN.1998.682231. (Cited on page 63.)

[169] Michael Kifer. Rule interchange format: The framework. In *RR '08: Proceedings of the 2nd International Conference on Web Reasoning and Rule Systems*, pages 1–11, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-88736-2. (Cited on pages 107 and 110.)

[170] George J. Klir. Generalized information theory. *Fuzzy Sets Syst.*, 40(1):127–142, 1991. ISSN 0165-0114. doi: http://dx.doi.org/10.1016/0165-0114(91)90049-V. (Cited on page 12.)

[171] Kevin Knight. Are many reactive agents better than a few deliberative ones? In *IJCAI'93: Proceedings of the 13th international joint conference on Artifical intelligence*, pages 432–437, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc. (Cited on page 94.)

[172] T. Kohonen. The self-organizing map. 78(9):1464–1480, 1990. (Cited on pages 45 and 67.)

[173] Efstratios Kontopoulos, Nick Bassiliades, and Grigoris Antoniou. Deploying defeasible logic rule bases for the semantic web. (Cited on page 179.)

[174] M. Korver and P.J.F. Lucas. Converting a rule-based expert system into a belief network. *Medical Informatics*, 18:219–241, 1993. (Cited on page 181.)

[175] Dinesh Kumar and Babu J. Alappat. NSF-Water quality index: Does it represent the experts' opinion? *Practice Periodical of Hazardous, Toxic, and Radioactive Waste Management*, 13(1):75–79, 2009. (Cited on page 75.)

[176] Isao Kuwajima, Yusuke Nojima, and Hisao Ishibuchi. Effects of constructing fuzzy discretization from crisp discretization for rule-based classifiers. *Artificial Life and Robotics*, 13(1):294–297, December 2008. doi: 10.1007/s10015-008-0515-7. (Cited on page 61.)

[177] Carmen Lacave and Francisco J. Diez. A review of explanation methods for bayesian networks. *Knowledge Engineering Review*, 17:2002, 2000. (Cited on page 61.)

[178] Mal Rey Lee. An exception handling of rule-based reasoning using case-based reasoning. *J. Intell. Robotics Syst.*, 35(3):327–338, 2002. ISSN 0921-0296. doi: http://dx.doi.org/10.1023/A:1021161418286. (Cited on page 65.)

[179] Werner Van Leekwijck and Etienne E. Kerre. Defuzzification: criteria and classification. *Fuzzy Sets and Systems*, 108(2):159 – 178, 1999. ISSN 0165-0114. (Cited on page 53.)

[180] Brian Lent, Arun Swami, and Jennifer Widom. Clustering association rules, 1997. (Cited on page 66.)

[181] Baikun Li and Shannon Irvin. The comparison of alkalinity and orp as indicators for nitrification and denitrification in a sequencing batch reactor (sbr). *Biochemical Engineering Journal*, 34(3):248 – 255, 2007. ISSN 1369-703X. (Cited on page 201.)

[182] D. V. Lindley. Scoring rules and the inevitability of probability. In *System design for human interaction*, pages 182–208. IEEE Press, 1987. ISBN 0-87942-218-1. (Cited on page 26.)

[183] L. LUCCARINI, E. PORRA, A. SPAGNI, P. RATINI, S. GRILLI, S. LONGHI, and G. BORTONE. Soft sensors for control of nitrogen and phosphorus removal from wastewaters by neural networks. *Water science and technology*, pages 101–107, 2002. ISSN 0273-1223. (Cited on pages 203 and 206.)

[184] David Luckham. Soa, eda, bpm and cep are all complementary - part i, 2008. URL http://complexevents.com/wp-content/uploads/2007/07/Soa_EDA_Part1.pdf. (Cited on page 96.)

[185] David Luckham. Soa, eda, bpm and cep are all complementary - part ii, 2008. URL http://complexevents.com/wp-content/uploads/2007/07/Soa_EDA_Part2.pdf. (Cited on page 96.)

[186] David C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001. ISBN 0201727897. (Cited on pages 90, 92, 93, 97, 112, and 219.)

[187] Xudong Luo and Chengqi Zhang. Proof of the correctness of emycin sequential propagation under conditional independence assumptions. *IEEE Trans. on Knowl. and Data Eng.*, 11(2):355–359, 1999. ISSN 1041-4347. (Cited on page 154.)

[188] S. Mace and J. Mata-Alvarez. Utilization of SBR technology for wastewater treatment: An overview. *Industrial & Engineering Chemistry Research*, 41(23):5539–5553, November 2002. ISSN 0888-5885. (Cited on page 198.)

[189] David J. C. Mackay. A practical bayesian framework for backpropagation networks. *Neural Comput.*, 4(3):448–472, May 1992. ISSN 0899-7667. (Cited on pages 42 and 66.)

[190] J. B. Macqueen. Some methods of classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967. (Cited on page 61.)

[191] P. C. Mahalanobis. On the generalised distance in statistics. In *Proceedings National Institute of Science, India*, volume 2, pages 49–55, April 1936. (Cited on page 45.)

[192] M.J. Maher. Propositional defeasible logic has linear complexity. In *of Logic Programming*, pages 691–711, 2001. (Cited on page 170.)

[193] MarketResearch. Excerpt : Worldwide business rules management systems 2009-2013 forecast update and 2008 vendor shares, 2009. (Cited on page 106.)

[194] S Marsili-Libelli. Control of SBR switching by fuzzy pattern recognition. *Water Research*, 40(5):1095–1107, March 2006. ISSN 0043-1354. PMID: 16494923. (Cited on pages 203 and 206.)

[195] Thomas Martinetz and Klaus Schulten. A Neural-Gas network learns topologies. In T. Kohonen, K. Makisara, O. Simula, and J. Kangas, editors, *Proc. International Conference on Artificial Neural Networks (Espoo, Finland)*, pages 397–402, Amsterdam Netherlands, 1991. North-Holland. (Cited on page 45.)

[196] O. Matan, R. K. Kiang, C. E. Stenard, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel, and Y. Le Cur. Handwritten character recognition using neural network architectures. In *In Proceedings of the 4th United States Postal Service Advanced Technology Conference*, 1990. (Cited on page 46.)

[197] M Maurya, R Rengaswamy, and V Venkatasubramanian. Fault diagnosis using dynamic trend analysis: A review and recent developments. *Engineering Applications of Artificial Intelligence*, 20 (2):133–146, 2007. ISSN 09521976. doi: 10.1016/j.engappai.2006.06.020. (Cited on pages 213 and 227.)

[198] Peter McBrien, Anne Helga Seltveit, and Benkt Wangler. Rule based specification of information systems. In *In Proceedings of the International Connference on Information Systems and Management of Data (CISMOD'94*, pages 212–228, 1994. (Cited on page 104.)

[199] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. pages 15–27, 1988. (Cited on page 39.)

[200] L. Medsker. Design and development of hybrid neural network and expert systems. In *Neural Networks, 1994. IEEE World Congress on Computational Intelligence., 1994 IEEE International Conference on*, volume 3, pages 1470–1474 vol.3, Jun-2 Jul 1994. (Cited on page 62.)

[201] Jerry M. Mendel. Advances in type-2 fuzzy sets and systems. *Information Sciences*, 177(1):84 – 110, 2007. ISSN 0020-0255. doi: 10.1016/j.ins.2006.05.003. (Cited on page 25.)

[202] Brian Milch, Bhaskara Marthi, Stuart Russell, David Sontag, Daniel L. Ong, and Andrey Kolobov. Blog: Probabilistic models with unknown objects. In *In IJCAI*, pages 1352–1359, 2005. (Cited on page 109.)

[203] B Mishra. Distributed digital processing and closed loop computer control of wastewater treatment. *Automatica*, 16(1):73–82, January 1980. ISSN 00051098. doi: 10.1016/0005-1098(80)90088-6. (Cited on page 78.)

[204] Adam Mollenkopf and Edson Tirelli. Applying drools fusion complex event processing (cep) for real-time intelligence, September 2009. (Cited on page 93.)

[205] Marco Montali. Specification and verification of declarative open in- teraction models: a logic-based framework, 2009. (Cited on pages 96 and 112.)

[206] John E. Moody, S. J. Hanson, and John E. Moody. Number of parameters: An analysis of generalization and regularization in nonlinear learning systems. (Cited on page 60.)

[207] Hakan Moral, Aysegul Aksoy, and Celal F. Gokcay. Modeling of the activated sludge process by using artificial neural networks with automated architecture screening. *Computers & Chemical Engineering*, 32(10):2471–2478, 2008. ISSN 00981354. doi: 10.1016/j.compchemeng.2008.01.008. (Cited on page 83.)

[208] B. Moulin, H. Irandoust, M. Belanger, and G. Desbordes. Explanation and argumentation capabilities: Towards the creation of more persuasive agents. *Artificial Intelligence Review*, pages 169–222, May 2002. ISSN 0269-2821. (Cited on page 59.)

[209] Ioan Nascu, Grigore Vlad, Silviu Folea, and Tudor Buzdugan. Development and application of a PID auto-tuning method to a wastewater treatment process. *2008 IEEE International Conference on Automation, Quality and Testing, Robotics*, pages 229–234, May 2008. doi: 10.1109/AQTR.2008.4588827. (Cited on page 82.)

[210] Richard E. Neapolitan. *Learning Bayesian Networks*. Prentice Hall, April 2003. ISBN 0130125342. (Cited on page 61.)

[211] Mircea Negoita, Daniel Neagu, and Vasile Palade. *Computational Intelligence: Engineering of Hybrid Systems*. Springer, 1 edition, April 2005. ISBN 3540232192. (Cited on pages 57, 58, and 62.)

[212] Arnold Neumaier. Clouds, fuzzy sets and probability intervals. *Reliable Computing, Kluwer Academic Publishers*, 10:249–272, 2004. (Cited on page 26.)

[213] Raymond Ng and V.S. Subrahmanian. Probabilistic logic programming, 1992. (Cited on page 168.)

[214] Flemming Nielson, Hanne Riis Nielson, and Mooly Sagiv. Kleene's logic with equality. *Information Processing Letters*, 80(3): 131 – 137, 2001. ISSN 0020-0190. (Cited on page 157.)

[215] Daniel Nikovski. Constructing bayesian networks for medical diagnosis from incomplete and partially correct statistics. *IEEE Transactions on Knowledge and Data Engineering*, 12(4):509–516, 2000. ISSN 1041-4347. doi: http://doi.ieeecomputersociety.org/10.1109/69.868904. (Cited on page 50.)

[216] VilÃ©m NovÃ¡k. Abstract: Mathematical fuzzy logic in narrow and broader sense - a unified concept. (Cited on page 52.)

[217] Donald Nute. Defeasible logic. In *INAP*, pages 87–114, 2001. (Cited on pages 169 and 170.)

[218] Hyacinth S. Nwana and Martlesham Heath. Software agents: An overview, 1996. (Cited on page 94.)

[219] R Olejnik, T Fortis, and B Toursel. Webservices oriented data mining in knowledge architecture. *Future Generation Computer Systems*, 25(4):436–443, 2009. ISSN 0167739X. doi: 10.1016/j.future.2008.09.011. (Cited on page 90.)

[220] Gustaf Olsson and Bob Newell. *Wastewater treatment systems*. IWA Publishing, 1999. ISBN 1900222159, 9781900222150. (Cited on pages 73, 79, and 81.)

[221] Gustaf Olsson, Marinus K. Nielsen, and Zhiguo Yuan. *Instrumentation, control and automation in wastewater systems*. IWA Publishing, 2005. ISBN 1900222833, 9781900222839. (Cited on pages 77 and 78.)

[222] T J Owens. Survey of event processing, 2007. (Cited on page 93.)

[223] Clark B. Pace and Randy Harlow. Sbr vs. continuous flow: A cost comparison of waste treatment technologies. volume 278, pages 99–99. ASCE, 2000. (Cited on page 201.)

[224] T.Y. Pai, T.J. Wan, S.T. Hsu, T.C. Chang, Y.P. Tsai, C.Y. Lin, H.C. Su, and L.F. Yu. Using fuzzy inference system to improve neural network for predicting hospital wastewater treatment plant effluent. *Computers & Chemical Engineering*, 33(7):1272–1278, 2009. ISSN 00981354. doi: 10.1016/j.compchemeng.2009.02.004. (Cited on page 84.)

[225] Heping Pan. Fuzzy bayesian networks - a general formalism for representation, inference and learning with hybrid bayesian networks. (Cited on page 66.)

[226] J Z Pan, G. Stoilos, G. Stamou, V. Tzouvaras, and I. Horrocks. f-SWRL: a fuzzy extension of SWRL, 2006. (Cited on page 110.)

[227] Mike P. Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. Service-oriented computing: a research roadmap. *Int. J. Cooperative Inf. Syst.*, 17(2):223–255, 2008. (Cited on page 87.)

[228] Adrian Paschke. A homogenous reaction rule language for complex event processing. In *In Proc. 2nd International Workshop on Event Drive Architecture and Event Processing Systems (EDA-PS*, 2007. (Cited on page 119.)

[229] Adrian Paschke and Alexander Kozlenkov. A rule-based middleware for business process execution. In Martin Bichler, Thomas Hess, Helmut Krcmar, Ulrike Lechner, Florian Matthes, Arnold Picot, Benjamin Speitkamp, and Petra Wolf, editors, *Multikonferenz Wirtschaftsinformatik*. GITO-Verlag, Berlin, 2008. ISBN 978-3-940019-34-9. (Cited on page 105.)

[230] Adrian Paschke and Alexander Kozlenkov. Rule-based event processing and reaction rules. In *RuleML '09: Proceedings of the 2009 International Symposium on Rule Interchange and Applications*, pages 53–66, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-04984-2. (Cited on page 92.)

[231] Adrian Paschke, Harold Boley, Alexander Kozlenkov, and Benjamin Craig. Rule responder: Ruleml-based agents for distributed collaboration on the pragmatic web. In *ICPW '07: Proceedings of the 2nd international conference on Pragmatic web*, pages 17–28, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-859-6. (Cited on page 212.)

[232] Norman W Paton. Supporting production rules using eca rules in an object-oriented context. *Information and Software Technology*, 37(12):691 – 699, 1995. ISSN 0950-5849. (Cited on page 119.)

[233] HervÃ© Paulino. An overview of mobile agent systems, 2002. (Cited on page 94.)

[234] J. Pavelka. On fuzzy logic I: Many-valued rules of inference. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 25:45–72, 1979. (Cited on page 157.)

[235] J. Pavelka. On fuzzy logic II: Enriched residuated lattices and semantics of propositional calculi. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 25:119–134, 1979. (Cited on page 157.)

[236] J. Pavelka. On fuzzy logic III: Semantical completeness of some many-valued propositional calculi. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 25:447–464, 1979. (Cited on page 157.)

[237] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988. ISBN 0-934613-73-7. (Cited on pages 48, 49, 50, and 181.)

[238] Chris Peltz. Web services orchestration and choreography. *Computer*, 36(10):46–52, 2003. ISSN 0018-9162. (Cited on page 89.)

[239] Britta Petersen, Krist Gernaey, Mogens Henze, and Peter A. Vanrolleghem. A comprehensive model calibration procedure for asm1. (Cited on page 81.)

[240] Emil Petre and Cosmin Ionete. Nonlinear and Neural Networks Based Adaptive Control for a Wastewater Treatment Bioprocess. *Simulation*, pages 273–280, 2008. (Cited on page 83.)

[241] M Poch, J Comas, I Rodriguezroda, M Sanchezmarre, and U Cortes. Designing and building real environmental decision support systems. *Environmental Modelling & Software*, 19(9):857–873, 2004. ISSN 13648152. doi: 10.1016/j.envsoft.2003.03.007. (Cited on page 86.)

[242] John L. Pollock. Defeasible reasoning with variable degrees of justification. *Artificial Intelligence*, 133:233–282, 2001. (Cited on page 21.)

[243] John L. Pollock. Defeasible reasoning with variable degrees of justification. *Artificial Intelligence*, 133(1–2):233–282, 2001. (Cited on page 171.)

[244] B. T. Polyak and S. A. Nazin. Interval solutions for interval algebraic equations. *Math. Comput. Simul.*, 66(2-3):207–217, 2004. ISSN 0378-4754. (Cited on page 234.)

[245] Udo W. Pooch. Translation of decision tables. *ACM Comput. Surv.*, 6(2):125–151, 1974. ISSN 0360-0300. (Cited on page 111.)

[246] D J Power. A brief history of decision support systems. (Cited on page 85.)

[247] Jim Prentzas and Ioannis Hatzilygeroudis. Categorizing approaches combining rule-based and case-based reasoning. *Expert Systems*, 24(2):97–122, 2007. (Cited on page 65.)

[248] Cecilia M. Procopiuc. Applications of clustering problems, 1997. (Cited on page 47.)

[249] J. Ross Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993. ISBN 1-55860-238-0. (Cited on page 67.)

[250] N Quinn. Environmental decision support system development for seasonal wetland salt management in a river basin subjected to water quality regulation. *Agricultural Water Management*, 96(2): 247–254, 2009. ISSN 03783774. doi: 10.1016/j.agwat.2008.08.003. (Cited on page 84.)

[251] Anand S. Rao and Michael P. Georgeff. Bdi agents: from theory to practice, 1995. (Cited on page 94.)

[252] ANAND S. RAO and MICHAEL P. GEORGEFF. Decision procedures for BDI logics. *J Logic Computation*, 8(3):293–343, June 1998. (Cited on page 95.)

[253] R. Reiter. A logic for default reasoning. *AI*, 13, April 1980. (Cited on page 169.)

[254] a Rivas, I Irizar, and E Ayesa. Model-based optimisation of Wastewater Treatment Plants design. *Environmental Modelling & Software*, 23(4):435–450, 2008. ISSN 13648152. doi: 10.1016/j.envsoft.2007.06.009. (Cited on page 81.)

[255] M a Rodrigo, a Seco, J Ferrer, J M Penya-roja, and J L Valverde. Nonlinear control of an activated sludge aeration process: use of fuzzy techniques for tuning PID controllers. *ISA transactions*, 38 (3):231–41, January 1999. ISSN 0019-0578. (Cited on page 84.)

[256] J.F. Roesler and R.H. Wise. Variables to be measured in wastewater treatment plant monitoring and control. *Journal of the Water Pollution Control Federation*, 46(7):1769–1775, 1974. cited By (since 1996) 0. (Cited on page 79.)

[257] Ronald G. Ross. The brs rule classification scheme. (Cited on page 104.)

[258] S. J. Russell and Norvig. *Artificial Intelligence: A Modern Approach (Second Edition)*. Prentice Hall, 2003. (Cited on pages 34 and 96.)

[259] Galina Rybina and Victor Rybin. Static and dynamic integrated expert systems: State of the art, problems and trends. (Cited on page 104.)

[260] M. Krishnaveni S. Santhosh Baboo, P. Subashini. Combining self-organizing maps and radial basis function networks for tamil handwritten character recognition. *ICGST International Journal on Graphics, Vision and Image Processing, GVIP*, 09:1–7, 2009. (Cited on page 44.)

[S.[1] nd C.(1998)]kundu1998 Kundu S.[1] and Jianhua C. Fuzzy logic or lukasiewicz logic: A clarification. *Fuzzy Sets and Systems*, 95: 369–379, May 1998.

[261] J.J. Saade. A unifying approach to defuzzification and comparison of the outputs of fuzzy controllers. *Fuzzy Systems, IEEE Transactions on*, 4(3):227–237, Aug 1996. ISSN 1063-6706. doi: 10.1109/91.531767. (Cited on page 53.)

[262] Kai Sachs, Samuel Kounev, Stefan Appel, and Alejandro Buchmann. Benchmarking of message-oriented middleware. In *DEBS '09: Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, pages 1–2, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-665-6. (Cited on page 90.)

[263] Regivan Hugo N Santiago and Christian Maeder. Linguistic Variables of Type-N. A Mathematical Model. (x), 2009. (Cited on page 26.)

[264] R Sarrate, J Aguilar, and F Nejjari. Event-based process monitoring. *Engineering Applications of Artificial Intelligence*, 20(8):1152–1162, 2007. ISSN 09521976. doi: 10.1016/j.engappai.2007.02.008. (Cited on page 93.)

[265] R. Sarrate, J. Aguilar, and F. Nejjari. Event-based process monitoring. *Engineering Applications of Artificial Intelligence*, 20(8):1152 – 1162, 2007. ISSN 0952-1976. (Cited on pages 203 and 206.)

[266] M Schutze, a Campisano, H Colas, W Schilling, and P Vanrolleghem. Real time control of urban wastewater systems?where do we stand today? *Journal of Hydrology*, 299(3-4):335–348, 2004. ISSN 00221694. doi: 10.1016/j.jhydrol.2004.08.010. (Cited on page 81.)

[267] Kari Sentz and Scott Ferson. Combination of evidence in dempster-shafer theory. Technical report, 2002. (Cited on page 19.)

[268] G. Shafer. *A mathematical theory of evidence*. Princeton university press, 1976. (Cited on page 19.)

[269] Glenn Shafer. 1 introduction what is probability? 1. (Cited on page 16.)

[270] R. Shao, E.B. Martin, J. Zhang, and A.J. Morris. Confidence bounds for neural network representations. *Computers and Chemical Engineering*, 21(Supplement 1):S1173 – S1178, 1997. ISSN 0098-1354. (Cited on page 47.)

[271] Stuart C. Shapiro. Belief revision and truth maintenance systems: an overview and a proposal. Technical report, 1998. (Cited on page 139.)

[272] B. Sick. Fusion of soft and hard computing techniques in indirect, online tool wear monitoring. *IEEE Transactions on Systems, Man, and Cybernetics - C*, 32(2), 2002. (Cited on page 57.)

[273] B. Sick and S.J. Ovaska. Fusion of soft and hard computing techniques: a multi-dimensional categorization scheme. In *Soft Computing in Industrial Applications, 2005. SMCia/05. Proceedings of the 2005 IEEE Mid-Summer Workshop on*, pages 57–62, June 2005. (Cited on page 63.)

[274] Ilse Y. Smets, Jeroen V. Haegebaert, Ronald Carrette, and Jan F. Van Impe. Linearization of the activated sludge model asm1 for fast and reliable predictions. *Water Research*, 37(8):1831 – 1851, 2003. ISSN 0043-1354. doi: 10.1016/S0043-1354(02)00580-8. (Cited on page 81.)

[275] Ph Smets, Y. Hsia, A. Saffiotti, R. Kennes, H. Xu, and E. Umkehren. The transferable belief model. pages 91–96. 1991. doi: 10.1007/3-540-54659-6_72. (Cited on page 19.)

[276] Philippe Smets. Analyzing the combination of conflicting belief functions. *Information Fusion*, 8(4):387 – 412, 2007. ISSN 1566-2535. doi: 10.1016/j.inffus.2006.04.003. (Cited on page 19.)

[277] Philippe Smets. *Imperfect Information: Imprecision and Uncertainty*, pages 254, 225. 1996. (Cited on pages 11 and 12.)

[278] Barry Smyth and Mark T. Keane. Remembering to forget: A competence-preserving case deletion policy for case-based reasoning systems. pages 377–382. Morgan Kaufmann, 1995. (Cited on page 60.)

[279] a Stare, D Vrecko, N Hvala, and S Strmcnik. Comparison of control strategies for nitrogen removal in an activated sludge process in terms of operating costs: a simulation study. *Water research*, 41(9):2004–14, 2007. ISSN 0043-1354. doi: 10.1016/j.watres.2007.01.029. (Cited on page 80.)

[280] Claudia Steghuis. Service granularity in soa-projects : a trade-off analysis, June 2006. (Cited on page 88.)

[281] Giorgos Stoilos, Giorgos Stamou, and Stefanos Kollias. fuzzyDL: An expressive fuzzy description logic reasoner. *2008 IEEE International Conference on Fuzzy Systems IEEE World Congress on Computational Intelligence*, (1):923–930, 2008. (Cited on page 179.)

[282] LluÃs Corominas Tabares. Control and optimization of an SBR for nitrogen removal: from model calibration to plant operation. (Cited on page 198.)

[283] Ismail A. Taha and Joydeep Ghosh. Symbolic interpretation of artificial neural networks. *IEEE Trans. on Knowl. and Data Eng.*, 11(3):448–463, 1999. ISSN 1041-4347. (Cited on page 173.)

[284] A. B. Tickle, R. Andrews, M. Golea, and J. Diederich. The truth will come to light: directions and challenges in extracting the knowledge embedded within trained artificial neural

networks. *Neural Networks, IEEE Transactions on*, 9(6):1057–1068, 1998. (Cited on page 173.)

[285] a Traore, S Grieu, S Puig, L Corominas, F Thiery, M Polit, and J Colprim. Fuzzy control of dissolved oxygen in a sequencing batch reactor pilot plant. *Chemical Engineering Journal*, 111(1):13–19, 2005. ISSN 13858947. doi: 10.1016/j.cej.2005.05.004. (Cited on page 83.)

[286] a Traore, S Grieu, F Thiery, M Polit, and J Colprim. Control of sludge height in a secondary settler using fuzzy algorithms. *Computers & Chemical Engineering*, 30(8):1235–1242, 2006. ISSN 00981354. doi: 10.1016/j.compchemeng.2006.02.020. (Cited on page 83.)

[287] Raynitchka Tzoneva. Method for Real-Time Optimal Control of the Activated Sludge Process. (Cited on page 81.)

[288] Raman Bai. V, Reinier Bouwmeester, and Mohan. S. Fuzzy logic water quality index and importance of water quality parameters, 2009. (Cited on page 76.)

[289] Jordi Vallverdu. The false dilemma: Bayesian vs. frequentist, 2008. (Cited on page 16.)

[290] a. Venu Vinod, K. Arun Kumar, and G. Venkat Reddy. Simulation of biodegradation process in a fluidized bed bioreactor using genetic algorithm trained feedforward neural network. *Biochemical Engineering Journal*, 46(1):12–20, 2009. ISSN 1369703X. doi: 10.1016/j.bej.2009.04.006. (Cited on page 83.)

[291] Norhaliza a. Wahab, Reza Katebi, and Jonas Balderud. Multivariable PID control design for activated sludge process with nitrification and denitrification. *Biochemical Engineering Journal*, 45 (3):239–248, 2009. ISSN 1369703X. doi: 10.1016/j.bej.2009.04.016. (Cited on page 82.)

[292] P. Walley. Inferences from multinomial data: learning about a bag of marbles. *Journal of the Royal Statistical Society, Series B*, 58: 3–57, 1996. (Cited on page 18.)

[293] Peter Walley. Coherent upper and lower previsions, 1998. (Cited on page 17.)

[294] Peter Walley. *Statistical Reasoning with Imprecise Probabilities*. Chapman & Hall/CRC, 1st edition, December 1990. ISBN 0412286602. (Cited on page 17.)

[295] Aijie Wang, Chunshuang Liu, Hongjun Han, Nanqi Ren, and Duu-Jong Lee. Modeling denitrifying sulfide removal process using artificial neural networks. *Journal of hazardous materials*, 168(2-3):1274–9, 2009. ISSN 1873-3336. doi: 10.1016/j.jhazmat.2009.03.006. (Cited on page 83.)

[296] Pei Wang. Confidence as higher-order uncertainty, 2001. (Cited on pages 21 and 109.)

[297] Yayi Wang, Yongzhen Peng, and Tom Stephenson. Effect of influent nutrient ratios and hydraulic retention time (hrt) on simultaneous phosphorus and nitrogen removal in a two-sludge sequencing batch reactor process. *Bioresource Technology*, 100(14): 3506 – 3512, 2009. ISSN 0960-8524. (Cited on page 198.)

[298] I Wong, R Bloom, D Mcnicol, P Fong, R Russell, and X Chen. Species at risk: Data and knowledge management within the WILDSPACE Decision Support System. *Environmental Modelling & Software*, 22(4):423–430, 2007. ISSN 13648152. doi: 10.1016/j.envsoft.2005.12.012. (Cited on page 84.)

[299] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10:115–152, 1995. (Cited on page 94.)

[300] Michael Woolridge and Michael J. Wooldridge. *Introduction to Multiagent Systems*. John Wiley & Sons, Inc., New York, NY, USA, 2001. ISBN 047149691X. (Cited on page 94.)

[301] L. Yang and J.E. Alleman. Investigation of batchwise nitrite build-up by an enriched nitrification culture. *Water Science and Technology*, 26(5-6):997–1005, 1992. cited By (since 1996) 59. (Cited on page 76.)

[302] H. J. Yin. ViSOM - A novel method for multivariate data projection and structure visualization. *IEEE Transactions on Neural Networks*, 13(1):237–243, January 2002. (Cited on page 45.)

[303] Chang Kyoo Yoo, Dae Sung Lee, and Peter a Vanrolleghem. Application of multiway ICA for on-line process monitoring of a sequencing batch reactor. *Water research*, 38(7):1715–32, 2004. ISSN 0043-1354. doi: 10.1016/j.watres.2004.01.006. (Cited on pages 203 and 206.)

[304] Changkyoo Yoo and Min Han Kim. Industrial experience of process identification and set-point decision algorithm in a full-scale treatment plant. *Journal of environmental management*, 90(8): 2823–30, 2009. ISSN 1095-8630. doi: 10.1016/j.jenvman.2009.04.004. (Cited on page 82.)

[305] R Yu, S Liaw, C Chang, H Lu, and W Cheng. Monitoring and control using on-line orp on the continuous-flow activated sludge batch reactor system, 1997. ISSN 02731223. (Cited on page 204.)

[306] L. A. Zadeh. Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, 100(Supplement 1):9 – 34, 1999. ISSN 0165-0114. doi: 10.1016/S0165-0114(99)80004-9. (Cited on pages 24 and 25.)

[307] L. A. Zadeh. Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, 100(Supplement 1):9 – 34, 1999. ISSN 0165-0114. doi: 10.1016/S0165-0114(99)80004-9. (Cited on page 53.)

[308] Lotfi Zadeh. *Generalized Theory of Uncertainty (GTU) - Principal Concepts and Ideas*, pages 3–4. 2006. (Cited on pages 119, 123, 128, 135, and 162.)

[309] Lotfi A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965. (Cited on page 21.)

[310] Lotfi A. Zadeh. The concept of a linguistic variable and its application to approximate reasoning - i. *Inf. Sci.*, 8(3):199–249, 1975. (Cited on pages 25, 52, and 54.)

[311] Lotfi A. Zadeh. Fuzzy logic, neural networks, and soft computing. *Commun. ACM*, 37(3):77–84, 1994. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/175247.175255. (Cited on page 39.)

[312] Guoqiang P. Zhang. Neural networks for classification: a survey. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 30(4):451–462, 2000. doi: 10.1109/5326.897072. (Cited on page 46.)

[313] Jidi Zhao, Harold Boley, and Weichang Du. Knowledge representation and consistency checking in a norm-parameterized fuzzy description logic. In De-Shuang Huang, Kang-Hyun Jo, Hong-Hee Lee, Hee-Jun Kang, and Vitoantonio Bevilacqua, editors, *ICIC (2)*, volume 5755 of *Lecture Notes in Computer Science*, pages 111–123. Springer, 2009. ISBN 978-3-642-04019-1. (Cited on page 179.)

COLOPHON

This thesis was typeset with LaTeX 2_ε using Hermann Zapf's *Palatino* and *Euler* type faces (Type 1 PostScript fonts *URW Palladio L* and *FPL* were used). The listings are typeset in *Bera Mono*, originally developed by Bitstream, Inc. as "Bitstream Vera". (Type 1 PostScript fonts were made available by Malte Rosenau and Ulrich Dirr.)

The typographic style was inspired by Bringhurst's genius as presented in *The Elements of Typographic Style* [65]. It is available for LaTeX via CTAN as "`classicthesis`".

Final Version as of March 15, 2010 at 3:48.