Alma Mater Studiorum – Università di Bologna

DOTTORATO DI RICERCA IN

Informatica

Ciclo XXIII

Settore Concorsuale di afferenza: 01/B1

Settore Scientifico disciplinare: INF/01

Enabling a direct path from end-user specifications to executable protocols in a biology laboratory environment

Presentata da: **Alessandro Maccagnan**

Coordinatore Dottorato **Relatore**

Prof. Maurizio Gabbrielli **Prof. Tullio Vardanega**

Se hai la febbre dentro falla crescere,

metti in crisi te stesso

e le persone che ti sono accanto

T.V.

Abstract

Biomedical analyses are becoming increasingly complex, with respect to both the type of the data to be produced and the procedures to be executed. This trend is expected to continue in the future. The development of information and protocol management systems that can sustain this challenge is therefore becoming an essential enabling factor for all actors in the field. The use of custom-built solutions that require the biology domain expert to acquire or procure software engineering expertise in the development of the laboratory infrastructure is not fully satisfactory because it incurs undesirable mutual knowledge dependencies between the two camps. We propose instead an infrastructure concept that enables the domain experts to express laboratory protocols using proper domain knowledge, free from the incidence and mediation of the software implementation artefacts. In the system that we propose this is made possible by basing the modelling language on an authoritative domain specific ontology and then using modern model-driven architecture technology to transform the user models in software artefacts ready for execution in a multi-agent based execution platform specialized for biomedical laboratories.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

1.1 The scenario of Biomedical laboratories

Biomedicine and life sciences in general have been revolutionized by the introduction of high-throughput technologies. A new era, characterized by the so called "-omics" disciplines, has begun more than a decade ago. Nowadays, the "new" generations of high-throughput technologies succeed one another at increasing pace, in particular in the field of biomolecular analysis. DNA sequencers of the current second generation are able to produce frighteningly large amounts of data, in the order of the terabyte, in a single experiment. The coming next generation is expected to raise the bar by 1 or 2 orders of magnitude [94]. In figure 1.1 we can see the exponential trend observed in the volume of data produced by the sequencing methodologies.

High-throughput technology has contributed to the large-scale studies on the characterization of populations of biological entities [71]. A variety of "-omics" disciplines, such as genomics [122], transcriptomics [23], proteomics [52] and metabolomics [57, 110], have begun to emerge, with their own sets of instruments, techniques, reagents and software. The characterization of the

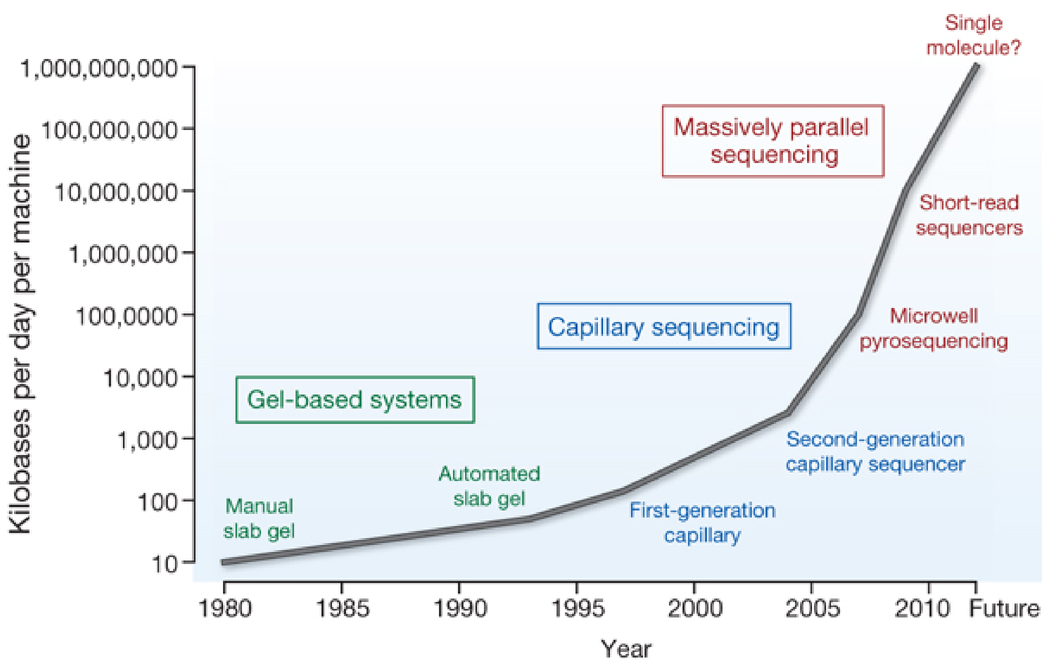


Figure 1.1: *Advances in the rate of DNA sequencing over the past 30 years and into the future. Source: Stratton et al. [118]*

"-ome" produces huge amount of data that would be impossible to process without specialised support from Information Technology.

Such a scientific and technological revolution is having a large impact on our society, life and health. Innovative tools for environmental, food and animal analysis based on the said "-omics" technologies are increasingly put into play. Medical analysis, diagnosis and therapy equally benefit from high-throughput discoveries and techniques. The advent of personalized medicine is already in sight and regenerative medicine is no longer a far dream. In figure 1.2 we can see the drop in of the cost of sequencing per genome. The diagram clearly shows the exponential decreases of the prices, especially in the last five years.

The work of life scientists is also rapidly changing. At present a researcher deals not only with laboratory equipment and *in vitro* experiments

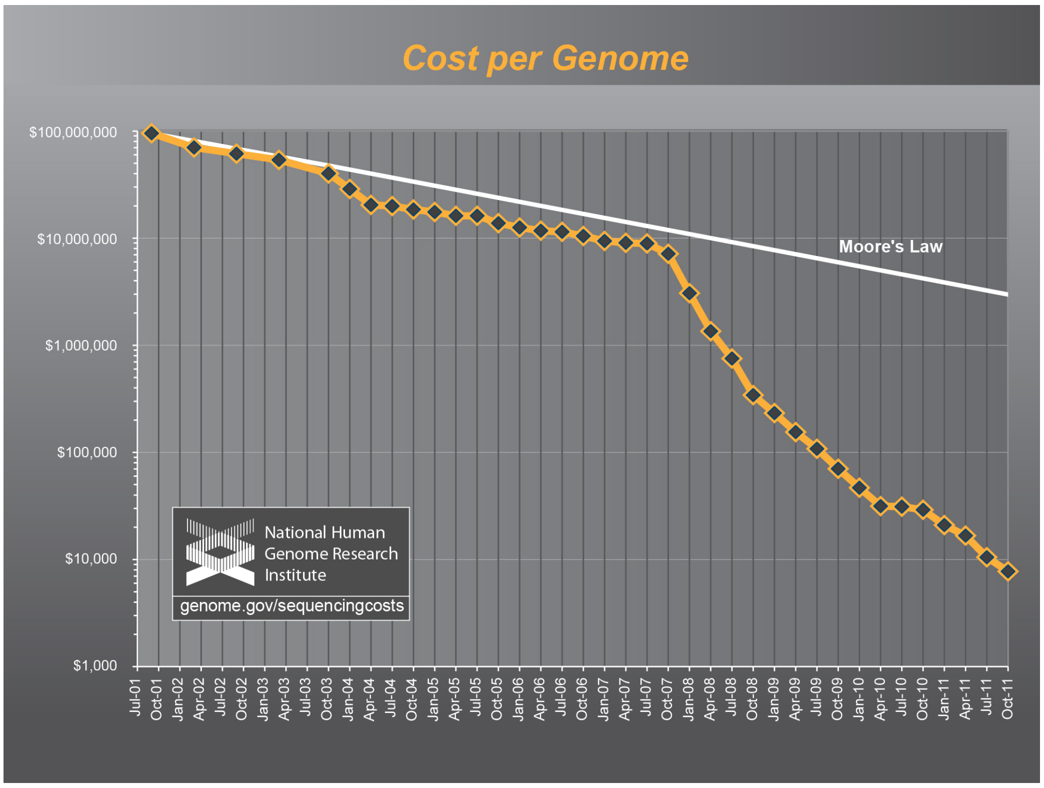


Figure 1.2: *Cost per Human Genome. Source:* *http://www.genome.gov/sequencingcosts/*

but also with software and web resources, i.e. *in silico* experiments. Scientific protocols include a very broad spectrum of activities (whether manual or automated) to be executed at the work bench and/or on computer systems. Computers play a central role in data production, collection, storage, hypothesis formation and experimentation [87]. Several sectors of science are becoming largely automated [17] and this aspect has been highlighted by the emergence of "e-Science" [42].

However, to reap the benefits of computer systems and consequently of automation, it is essential that scientists change the way in which scientific knowledge is described, reported and stored. In fact, two of the problems in contemporary life science research are the interpretation and the reproducibility of published experimental results. Hence, there is urgent need for a formal representation of scientific knowledge, including procedures (e.g., laboratory protocols, bioinformatic workflows).

Laboratory protocols and experimental methodologies are indeed an integral part of research in life sciences. The way in which protocols are described is decisive in permitting the reproducibility and the successful replication of experiments. Normally, the detailed notes about the kind of experimental procedures and their order, the type of materials and the variety of methods used by a researcher are available only inside their research group or department. The information is then disseminated through the research community by scientific publications and as a consequence it becomes available for the use of scientists who may be new to that topic. Every individual study rests on ad-hoc laboratory protocols, which are usually included in a "Materials and Methods" section defined only in natural languages.

This way of describing laboratory processes has many limitations that fatally impair their repeatability, distribution and more importantly automation. This can lead to ambiguous statements and to vastly arbitrary interpretations. Textual representation is the best choice for readability but it does not promote the re-use of parts of the protocol description and does not give a global, structured vision of the whole process also without highlighting the possibly numerous resources necessary for the execution of the experiment.

A researcher can spend weeks or months to learn, set up, and apply new experimental techniques or protocols. Thus, a significant amount of time in the laboratory is spent learning techniques and procedures published by other research groups. This is a never ending process for experimental life scientists since methodologies and their respective protocols are evolving at a dramatically fast pace.

At the same time, laboratory automation is becoming increasingly crucial in many fields of experimental research. Many wet-lab activities are becoming dependent on laboratory robots [74]. Bioinformatics encompasses automation in all the aspects related to biological data, including data collection, management and analysis. Two levels of formalization are required: one for the entities and operations deployed in protocols, another for the protocols themselves that can combine both manually executed and automated procedures.

For the first level of formalization we use the power of the Ontology [59]. Ontology is one of the strategies for the structured and formalized representation of a given domain knowledge in a formal way, thus aiding in removing ambiguity and redundancy, detecting errors and facilitating automated reasoning. Ontologies describe the entities of the specific domain but do not specify how these entities should be used and combined.

The second level, laboratory procedures, could be structured using the workflow metaphor [101]. A workflow is a representation of a sequence of operations, declared as the work of a person, a group of persons, or machines. Workflows enable the description and the orchestration of complex processes in a visual form, capturing human-to-machine interactions within those processes. Several disciplines adopt workflow systems for the automation of data processing through a series of processing stages.

Biomedical laboratories play a key role in such a scenario, as places where scientific discoveries are enabled and routine analyses are carried out. Automation and modern technologies in laboratories, besides directly enabling the “big sciences” are also indirectly catalysing new science, discoveries and innovation. In fact, they release domain experts from routine activities that would otherwise keep them occupied, preventing them from attending to more scientifically relevant work. Extremely specialized devices are able to perform routine activities in a fast, precise and reliable way. We may for example consider liquid handlers which can dispense nano-volumes of liquid, or robotic devices capable of moving samples around different locations.

1.2 Biomedical protocols

Several on-line resources are available for retrieving information about life-science protocols and experiments. A protocol could be defined as a pre-defined written procedural method in the design and implementation of experiments" [1]. Since 1997, the Science Advisory Board (SAB) [14] has been working to the goal of improving communications between biomedical scientists and suppliers of laboratory products and services. SAB also maintains an extensive database of categorized by techniques.

Protocol-Online [13] appeared in 1999 on the web as a database resource for research protocols in a variety of life science fields such as cell biology, molecular biology, developmental biology, and immunology.

In 2004, the Nature Publishing Group (NPG) launched Nature Methods [9], a monthly research journal on novel methods and significant improvements to laboratory techniques in the life sciences and related areas of chemistry. In addition, Nature Methods includes a Protocols section describing established methods written using 'bench terms'.

In 2006, JoVE [4] started to publish on-line video-protocols. The user is not required to read through a written protocol but can simply watch a

[1]`http://purl.obolibrary.org/obo/OBI_0000272`

video. Each video article includes step-by-step instructions for an experiment, a demonstration of equipment and reagents, and a brief discussion, with experts describing possible technical problems and modifications [76].

In the same year, Nature Protocols [10], became available as a cutting-edge on-line journal for biological and biomedical protocols. Protocols, written in natural language, are organized into logical categories so as to be easily accessible to researchers. They are presented in a 'recipe' style providing step-by-step descriptions of procedures that users can take to the lab bench and immediately apply to their own research.

As an example of a protocol for *in silico* experiments, Huang et al. [70] describe how to use the DAVID bioinformatic resources for the analysis of large gene lists derived from high-throughput genomic experiments, including how DAVID modules can help users to extract biological meaning from the given gene list and how individual modules should be used either independently or jointly. In that way the reader can find the procedure easier to follow to reproduce the study.

The approach used for describing a computation procedure is also adopted for laboratory protocols. For instance, the protocol suggested by Fiegler [51] is organized into several sections; first, a list of materials used in the experiment including equipment, materials and their set up is provided. The second section is a step-by-step description of the methodology used. Critical steps that must be performed in a very precise manner and all toxic or harmful chemicals are highlighted. These warnings are tagged by the heading *Critical step* and *Caution*.

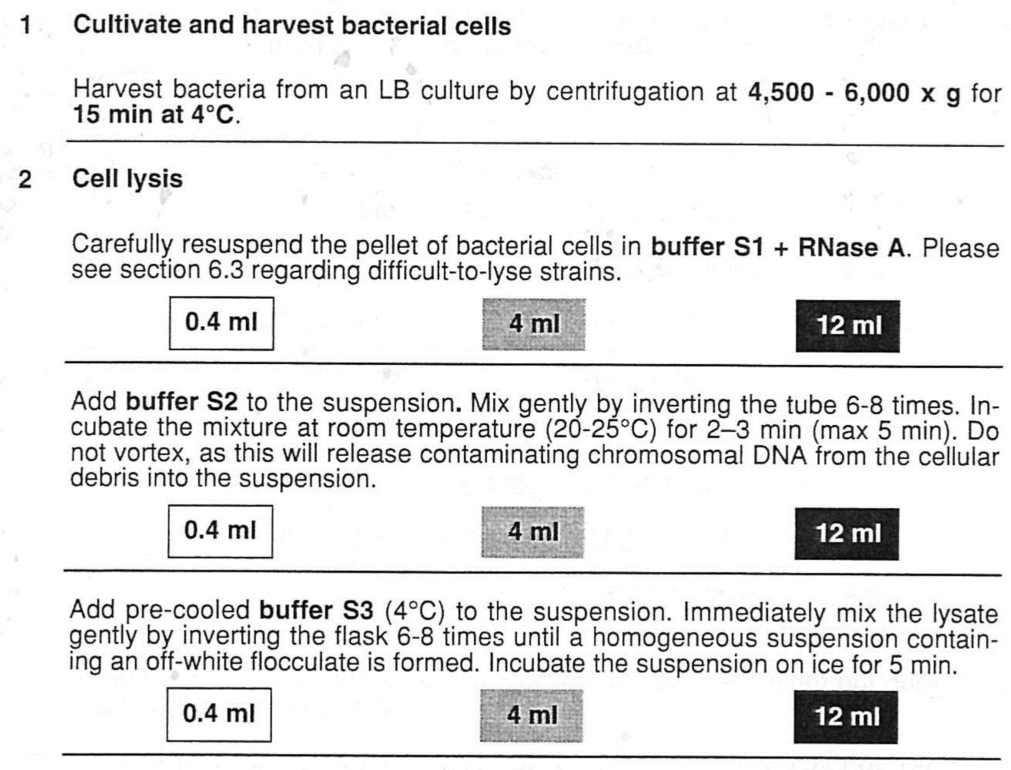
Unlike the articles in the previously cited journals, in Nature Protocols the author of a manuscript is also asked to report the timing and possible troubleshooting in order to give an idea of the duration of the procedure and on how to troubleshoot the most likely problems. Writing protocols using the same pre-defined template will help understand the procedure, as well as the critical steps and implementation of the technique reported in the published study.

In laboratory protocols there are numerous examples of ambiguous sentences. In fact, statements that can be interpreted in different ways can introduce uncertainty as to how the procedure should be performed. For example, the instruction "Remove the supernatant and dry the precipitated DNA briefly before washing with 100 μl of 70% ethanol" introduces an ambiguity of the term "briefly" [51], which may indicate different lengths of time. It could mean 30 seconds, 5 minutes, 10 minutes or a longer time. The term "gentle" in the instruction "Transfer slides into a solution of 0.1% sodium dodecyl sulphate and incubate for 5 min with gentle shaking." [51] can be arbitrarily interpreted. This problem could be overcome by providing a single value or a range of admissible values, depending on the activity performed, which can help reduce the ambiguity in the meaning of the term.

Must be also noted that the writing style of Nature Protocols is not intended to facilitate the automation of procedures. A computer machine will not be able to read it, interpret it and then replicate the original experiment.

1.2.1 Real world protocols: an example

Figure 1.3 shows a real world protocol currently used at the Centro ricerche interdipartimentale biotecnologie innovative (CRIBI) of Padua. It is written in natural language (Scientific English) as a subsequent set of actions. The protocol describes the extraction of plasmid DNA from bacteria cell colture avoiding contamination by the *E. coli* genomics DNA [28]. It has been named mini scale DNA preparation (*"miniprep"*). Each action is explained with some degree of depth depending on the particular protocol.


1 **Cultivate and harvest bacterial cells**

Harvest bacteria from an LB culture by centrifugation at **4,500 - 6,000 x g** for **15 min at 4°C**.

2 **Cell lysis**

Carefully resuspend the pellet of bacterial cells in **buffer S1 + RNase A**. Please see section 6.3 regarding difficult-to-lyse strains.

0.4 ml	4 ml	12 ml

Add **buffer S2** to the suspension. Mix gently by inverting the tube 6-8 times. Incubate the mixture at room temperature (20-25°C) for 2–3 min (max 5 min). Do not vortex, as this will release contaminating chromosomal DNA from the cellular debris into the suspension.

0.4 ml	4 ml	12 ml

Add pre-cooled **buffer S3** (4°C) to the suspension. Immediately mix the lysate gently by inverting the flask 6-8 times until a homogeneous suspension containing an off-white flocculate is formed. Incubate the suspension on ice for 5 min.

0.4 ml	4 ml	12 ml

Figure 1.3: *Example of a laboratory protocol: extraction of plasmid DNA from bacteria cell colture avoiding contamination by the E. coli genomics DNA ("miniprep") [28].*

The figure shows fragment of a protocol as reported in the reference manual of one of the several kits available for *miniprep*. Besides the commercial manual, each laboratory has its own methodology and thus its own protocols.

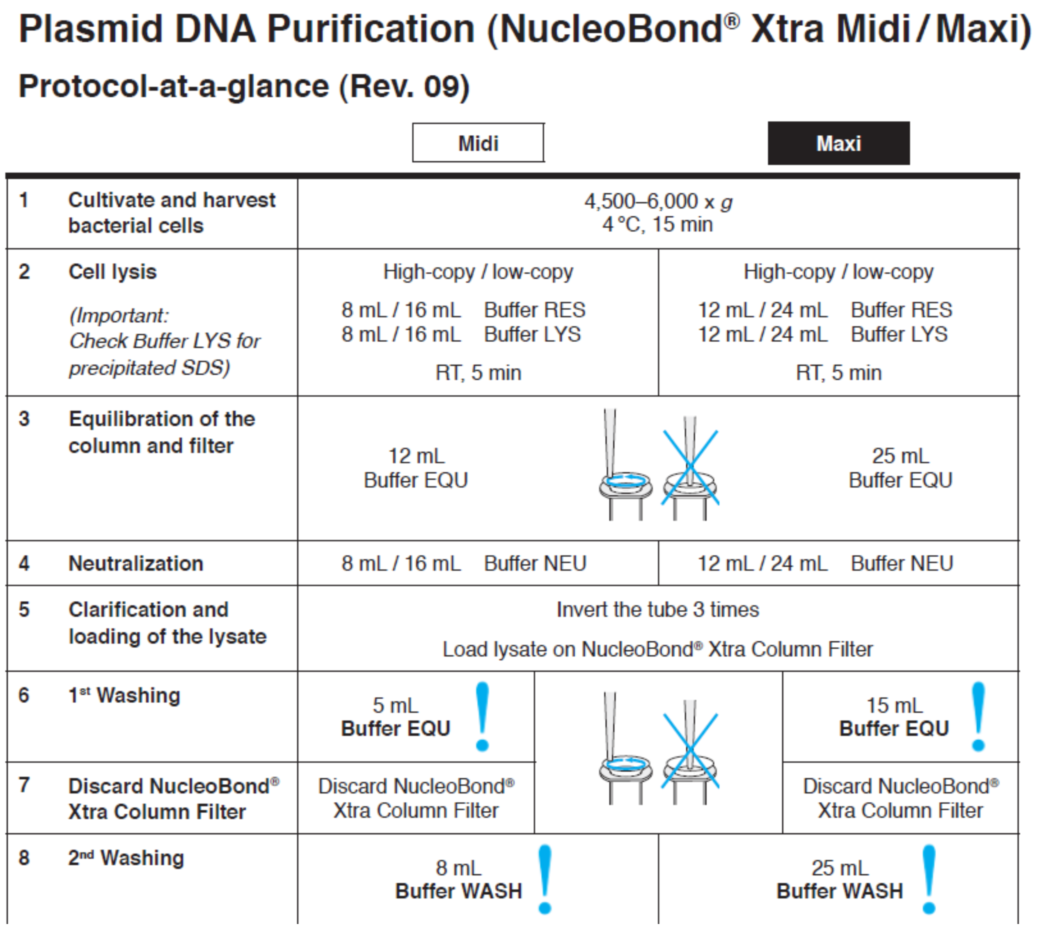
In step 1 a range is given for g-force centrifugation, but at which point of the range we could have optimality in the reaction? Further imprecise statements are given, in the number of times of inverting (step 2) or the incubation time period (also step 2). This could lead to different interpretations and consequently lead to the need for some training to determine which conditions are most suitable for a given laboratory. This could introduce noise in the process and an suboptimal use of resources.

The cited protocol is targeted to molecular biologists that work with nucleic acids, therefore the protocol is addressed for an experimenter with a specific domain knowledge. However, the circumstances does not always fit this situation. A misinterpretation could be taken over by a researcher new to the field.

Figure 1.4 shows another representation of the *miniprep* protocol. We can see the steps from 1 to 8 of the protocol. The representation helps the researcher in the execution of the protocol. The protocol is depicted as a list of actions in which every step is an action on his own. This representation is naturally near to the workflow metaphor.

1.3 Vision and interpretation

To satisfy the flexibility needs of experimental scientists and laboratory personnel, who are already faced with frequent next generations of technology, a suitable Laboratory Information Management System (LIMS) should permit a very easy and intuitive design and customization of laboratory workflows by domain experts. To this end we want to develop a general purpose LIMS that can be easily programmed by providing formal representation - the programming language - of laboratory workflows. On the other shore we have the laboratory domain expertise usually expressed in “recipe-like” protocols, expressed in natural language, with all the drawbacks discussed in [80].



Plasmid DNA Purification (NucleoBond® Xtra Midi / Maxi)

Protocol-at-a-glance (Rev. 09)

		Midi	Maxi
1	**Cultivate and harvest bacterial cells**	4,500–6,000 x *g* 4 °C, 15 min	
2	**Cell lysis** *(Important: Check Buffer LYS for precipitated SDS)*	High-copy / low-copy 8 mL / 16 mL Buffer RES 8 mL / 16 mL Buffer LYS RT, 5 min	High-copy / low-copy 12 mL / 24 mL Buffer RES 12 mL / 24 mL Buffer LYS RT, 5 min
3	**Equilibration of the column and filter**	12 mL Buffer EQU	25 mL Buffer EQU
4	**Neutralization**	8 mL / 16 mL Buffer NEU	12 mL / 24 mL Buffer NEU
5	**Clarification and loading of the lysate**	Invert the tube 3 times Load lysate on NucleoBond® Xtra Column Filter	
6	**1st Washing**	5 mL **Buffer EQU** !	15 mL **Buffer EQU** !
7	**Discard NucleoBond® Xtra Column Filter**	Discard NucleoBond® Xtra Column Filter	Discard NucleoBond® Xtra Column Filter
8	**2nd Washing**	8 mL **Buffer WASH** !	25 mL **Buffer WASH** !

Figure 1.4: *Representation of a protocol.*

1.3.1 Long term vision

The long term result aim of this work is to improve the quality of work and thus the quality of product achieved by the biologist. The envisioned picture is one in which the biologist is able to concentrate on the research goal rather than on side details. In our scenario we assume that a scientist should be able to declare which results he/she wants to obtain starting from a certain set of conditions (e.g which data he has already obtained, which samples to work on and so forth). A software able to understand this set of conditions will therefore be able to make some reasoning upon those conditions. Thereafter it will be able to produce an experiment that addresses the biologist needs. Afterwards the same software will be able to run the desired experiment employing and interacting directly with the instrumentation provided by the laboratory.

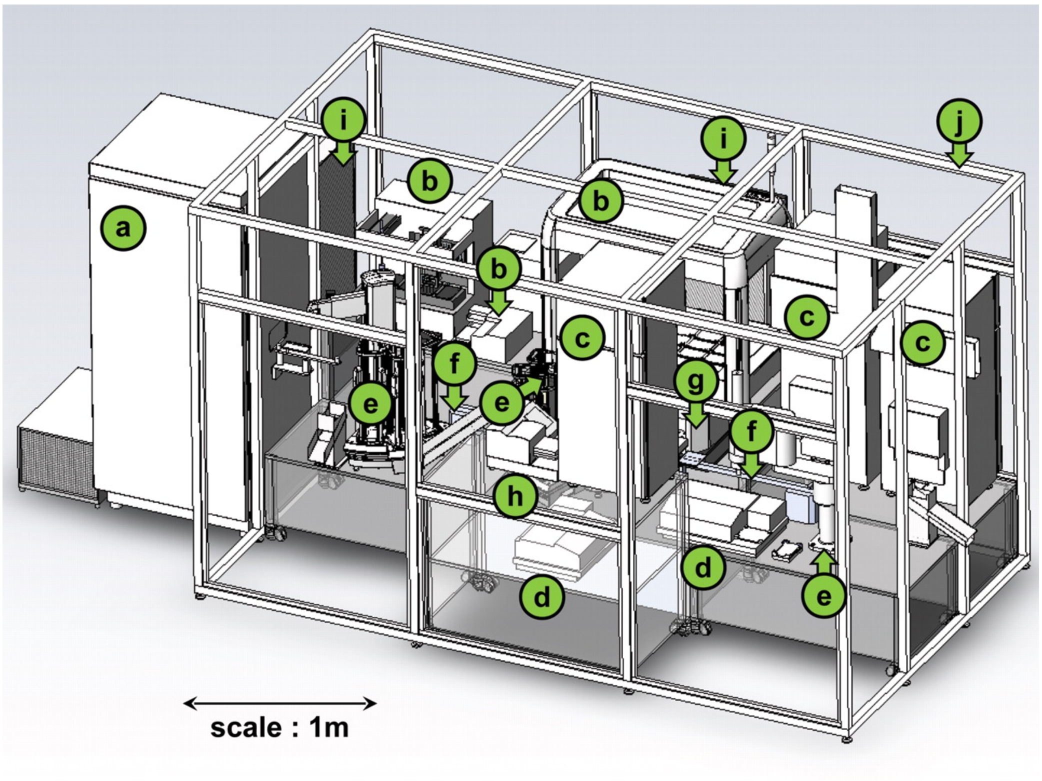


Figure 1.5: *The Robot Scientist Adam. Adam is able to independently design experiments to test hypotheses. The hardware used to build Adam is: a) an automated 20C freezer; b) three liquid handlers; c) three automated +30C incubators; d) two automated plate readers; e) three robot arms; f) two automated plate slides; g) an automated plate centrifuge; h) an automated plate washer; i) two high efficiency particulate air filters; j) a rigid transparent plastic enclosure. Source: [74]*

Elements of this vision has already been pursued by the Adam robot [74]. Figure 1.5 depicts the architecture of the Adam robot. Adam is able to automatically design experiments that test hypotheses on the domain of the yeast. The duty cycle of Adam includes the following step:

1. selection of specified yeast strains;
2. inoculation of strains into plate wells;
3. harvesting of defined quantity of cells.

. Thanks to these basic operations and to its specialised hardware and software, Adam is capable of designing and performing more than a thousand new yeast strains every day.

Our long term vision is the same as Adam. We would like to extend the capabilities of the general case of Adam. In our vision we break the problem down in some sub-problems that we aim to solve:

1. the ability to declare starting points;
2. the ability to declare goals;
3. the ability to formulate experiments as trajectories to move from a starting point to a goal.

Problems of this kind are known in computer science fall in the realm of automated planning [103]. Goals and starting points are similar objects. A way to describe them is to declare a set of conditions [115].

An experiment is a set of actions. Every action takes in input a set of conditions and transforms them into another set of conditions. An approach that tries to formulate actions in that way is described by [114]. An action could therefore be automatically selected by matching input and output conditions.

In order to automatically create experiments, a repository of available actions should be created. Actions could be described at various levels of granularity. An existing commercial kit could be regarded as coarse grained action for a particular task. Typical commercial kit includes reagents and detailed instruction for using it. The *Plasmid DNA Purification* is supplied by *MACHEREY-NAGEL*[2]. If we break the kit down in finer grained parts then, individuals actions became simpler, like for example a single step in figure 1.3.

In summary, in order to automatically create a protocol we need: a starting point; a goal; and a repository of predefined actions. The resulting protocol will be a path of actions which connect the starting point to the goal.

As seen in section 1.1, keeping track of the produced data is mandatory irrespective whether the execution of the protocol is manual or automatic. The system should be able to relate the environment with the procedure so as to ensure that the application constraints are respected. In case of malfunctioning recovery strategy must be carried out. As for example the system could decide to stop the execution in advance and to put the samples in a safe state.

In our domain of interest the environment is highly heterogeneous. The system that is in charge of the execution must be able to cope with this. A good strategy on this end is to develop a distributed system and to interact directly with the involved resources. In the case that this is not possible communication with the operator should be provided. The operator in this way will be able to provide the correct action needed by the procedure.

Our work in this thesis aim at enabling the initial steps of the long term vision. In particular we address one sub problem, to specify protocols and to enable their automatic execution.

[2]http://www.mn-net.com/

Figure 1.6 shows the principal entities in our environment. In the upper layer two kind of scientists are depicted. In a biological laboratory a *wet-lab* scientist performs experimentation producing data. The data is then analysed by bioinformatics. Both of them prepare experiments that are then physically executed using machine on the bottom layer. The work of this thesis aim to build an architecture that act as bridge of the upper and bottom layer.

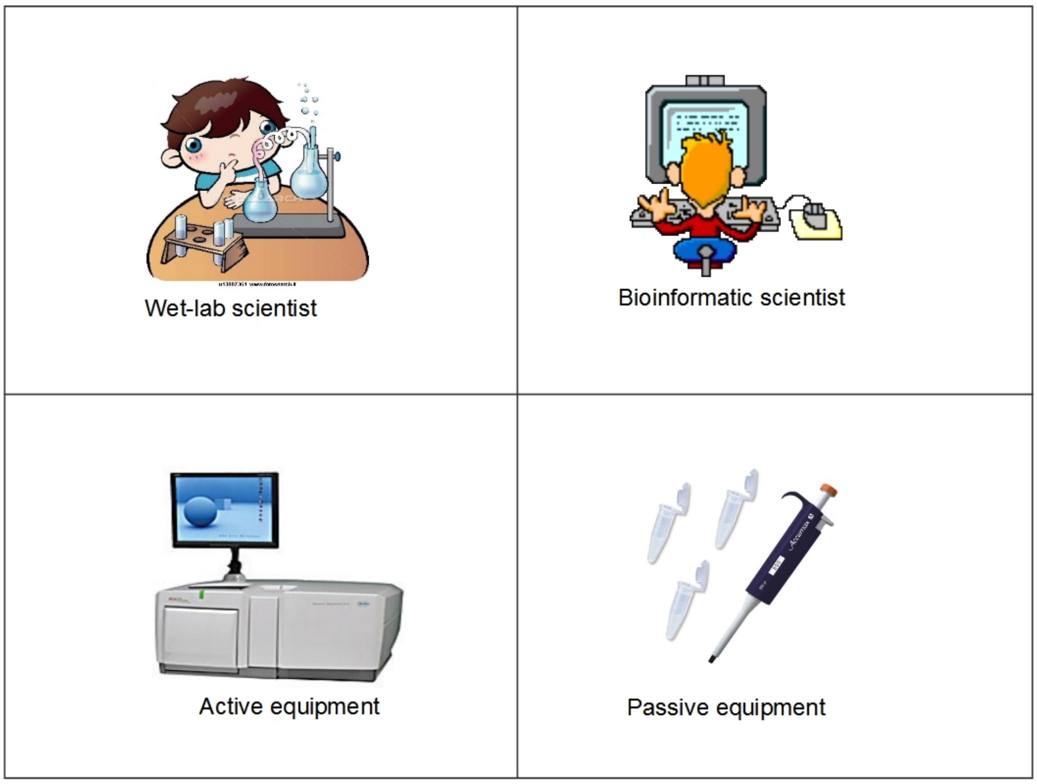


Figure 1.6: *Entities in a biological laboratory.*

1.3.2 Interpretation

Enacting the vision described above requires a system architecture in which the biologist (end-user) should be able to express his needs using a language close to his own domain knowledge. The system should be able to understand this language and to process it. The system should be able to execute the resulting procedures in the laboratory environment.

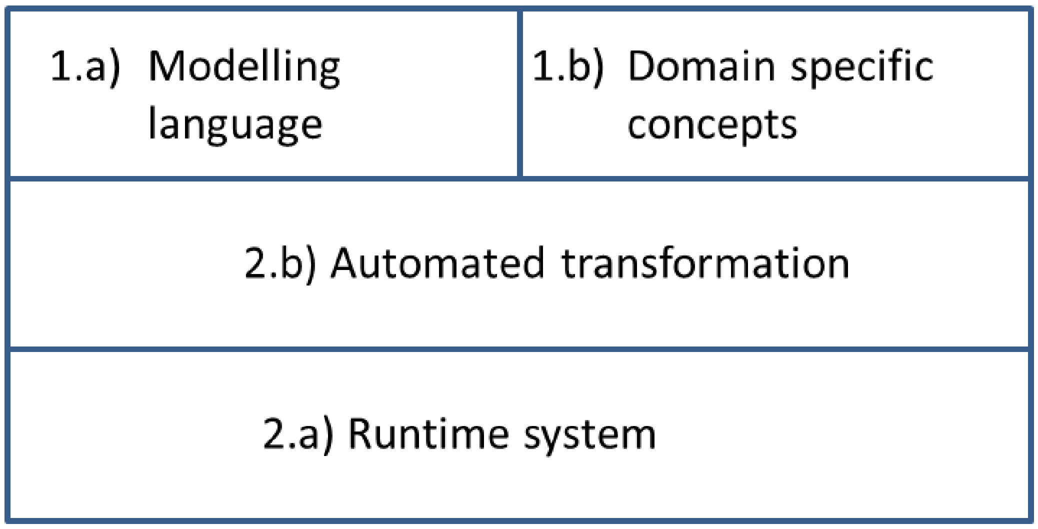


Figure 1.7: *Modelling language, domain specific concepts, runtime system, automatic transformation.*

The system we envision is organized in two layers:

1. Front-end
   (a) a modelling language that includes conditions, actions, objects;
   (b) domain specific concepts arising directly from the laboratory experience;
2. Back-end
   (a) a runtime, heterogeneous, distributed system;
   (b) a way to relate procedures with the runtime system.

The front-end in our vision is the layer of the system that interfaces directly with the end-user. We want a language able to express the flow of work of the laboratory procedures (1.a). Such a language should describe the sequence of actions that constitute protocols as well the objects involved in those actions. A graphical notation is also desirable as a way to ease the use of the language.

To be as near as possible to the domain knowledge of the end-user we want to bring in our language the concepts that directly arise from the laboratory experience (1.b). Such concepts could be effectively expressed in some consolidated and authoritative knowledge base. For this reason we have pointed our attention to the Ontology community. Ideally we want to use ontology concepts directly in our language. Objects and actions could be taken directly from a shared and acknowledged silos of terms. To this end we want to combine a modelling language able to describe workflows (1.a) and the domain-specific ontology together, thus shaping a new kind of domain-specific language.

The back-end takes charge of the implementation and execution complexity hiding them away from the front-end. Such complexity arises from the laboratory environment. Biomedical laboratories, as we have seen, are heterogeneous and distributed systems. The back-end (2.a) should take in account, and resolve, the cited requirements. The rapid evolution of agent-based systems confirms that the major advantages are significant: as for examples decentralized ownership of tasks or high degree of potential concurrency. They allow the building of highly decentralized, distributed systems, which correspond to real-world situations [27].

1.4 Summary

The central point of our vision is the development of a domain specific modelling language enriched with domain specific ontological concepts. Figure 1.7 shows that we want a way for relating the protocol expressed in the modelling language (1.a and 1.b) in the runtime system (2.a) by means of automatic transformations. Model-driven architecture (MDA) addresses the model transformation end of the problem with solid methodologies and tools. We therefore rely our vision upon MDA body of knowledge. We express the modelling language using a metamodel directly enriched with a domain-specific ontology. Then we want to use of automatic transformation (2.b) to generate artefacts suitable for execution in the runtime system.

The remainder of this dissertation is organised as follows. The relevant literature around our solution space is discussed in chapter 2. In chapter 3 we describe the architecture or our solution. In section 3.2 we present the front-end ((1.a and 1.b). In section 3.3 we describe the back-end of our solution (2.a and 2.b). Chapter 4 propose three use cases of our platform. In section 4.1 we evaluated the specification language against an end-user. In section 4.3 we used the platform in a real world environment producing a real-case biological protocol drawn from an industrial setting. In section 4.2 we focused our effort around a common pipeline used in our laboratories to analyse genomic data. Finally in chapter 5 we conclude our work recalling the main result and outlining future work.

Chapter 2

Problem analysis

In this chapter we outline the relevant literature around the solution space to which we have cast our problem. In section 2.1 we describe the experience of both industrial and academic LIMS. In section 2.2 we summarize the main concepts in the area of Ontology, with a view on the Ontologies in the domain of biology. In section 2.3 we survey the main concepts of MDA; we also described the relevant standard for business process notation and interoperability; Finally, in section 2.4 we look at the Multi-agent system (MAS) literature for the runtime system.

2.1 Laboratory Information Management Systems

Effective management of information is essential to a laboratory environment characterized by a wide variety of entities (e.g. biological samples, containers, locations, devices, experiments, protocols, laboratory personnel) and an equal large spectrum of activities. Whereas some laboratory personnel staff still annotate details of experiments into notepads, the trend is to use as more electronic resources as possible in a paperless fashion [92]. Nowadays it is fairly common to see tablet computers, PDAs and smartphones in use in laboratory routine work. For decades now many laboratories have started to use ad-hoc solutions to manage information. In the nineties the concept of Laboratory Automation System grouped robots, conveyor systems, machine vision, and computer hardware and software.

The acronyms ELN, LIS, LIMS are now commonly used in laboratory environments, to indicate software systems - often commercial - that support lab operations. An ELN, Electronic Laboratory Notebook, is a software tool designed to be a replacement of a notepad [48]. LIS, Laboratory Inventory Systems, refer to software systems assisting laboratories in keeping track of their collection of biologically relevant materials [129].

A more comprehensive solutions is a LIMS, that ideally should also track protocols and enable a smooth integration with automatic devices. The maturity of the World Wide Web has provided a good infrastructure to allow on-line access to the laboratory systems [67].

At present, automated devices combined with automated reasoning and inference permit to carry out experimentation in a completely automated fashion [74] enabling the vision of the Robot Scientist [116], able to devise new knowledge from the performed experiments and to autonomously plan and execute the next experiments to be undertaken.

In the laboratory landscape, nevertheless, a general-purpose, easy to configure LIMS is still missing, which could be effectively and efficiently customized by its end users. Laboratory experts usually do not and perhaps also should not have computer programming skills. Moreover it would be unwise and anti-economic to let them develop in-house sophisticated systems. Instead, they possess a deep knowledge of their own problem domain and a clear intuition of the protocols that an information system should implement and manage.

Actually, a long path stands between the natural language in which laboratory protocols can be expressed by domain experts and their implementation in an information system. This gap typically requires the intermediation of IT professional (e.g. LIMS designer/developers, software specialists able to configure LIMS and implement specific protocols). Usually, in-house developed solutions have rigid built-in protocols and therefore lack flexibility. New or modified protocols must be directly coded in the programming languages of the software system. Commercial systems offer more flexibility, but rely on proprietary standards and information representation. A standard formalism has not yet emerged to share laboratory protocols among different systems and in the laboratory community in general.

2.1.1 LIMS in Academic

Laboratories often adopt individually tailored protocols and in research laboratories novel strategies are typically explored. Due to the extreme level of required customization, no suitable LIMS is readily available [60]. The complexity of experiments and the amount of processed samples require more powerful tools than sophisticated electronic spreadsheets. The available commercial systems are usually too costly and complex. The “one-size-fits-all” philosophy can hardly be adopted in the laboratory world. Therefore, academic laboratories often develop their own custom LIMS. In any case important amounts of time and resources must be employed for the adaptation when a new system for managing the information of a laboratory is adopted [22]. Laboratory managers are often engaged in a difficult choice between the purchase of a costly commercial solution to be customized or the long, and perhaps eventually even more expensive, development of an in-house solution. Academic LIMS have been developed in a wide range of application fields in the life sciences and biomedicine.

Voegele et al. [126] implemented a system based on a client (platform independent web applications) - server (MySQL relational database) architecture. Their system has been tailored to a laboratory workflow aiming at high-throughput candidate gene mutation scanning and resequencing. It communicates with laboratory instruments and robots, tracking samples and laboratory information.

PIMS is a LIMS acting as a support platform in the Membrane Protein Structure Initiative [121]. It tracks essential information on the progress of cloning, expression, purification and crystallization of membrane proteins. The authors share the precious lessons learned during the challenging phases of PIMS integration and adaptation with other initiatives interested in adopting a LIMS as a data center for collaborative efforts.

RGMIMS is a modular, web-based LIMS designed in a rice functional genomics laboratory [65], that according to its authors, could be easily adapted to support general high-throughput plant research. Its web user interface enables bar-code reading and rapid data capture and tracking of biological resources.

The Emergency Response Management System is a customizable LIMS designed to support laboratory activities in chemical terrorism emergency response [104]. It adopts standardized data formats for communicating between different instrument types from different vendors.

Another example of emergency response infrastructure supported by a LIMS is that dealing with outbreaks of highly infective bovine disease in the UK [86]. That system is capable of scanning bar-codes and transferring information across computer networks.

Screensaver is a free, open source, web based LIMS to manage the information needs of a small molecule and RNAi screening facility [119]. Supporting the storage and comparison of screenings data sets, the management of information about screens, screeners, libraries and laboratory work requests, it overcomes the challenges arising when multiple independent research groups conduct numerous and interleaved screening efforts.

SLIMS is a user friendly, open source web LIMS for genotyping laboratories [125]. Studies searching for susceptibility genes for common complex diseases usually collect thousands of samples generating millions of genotypes. SLIMS aims to simplify common laboratory tasks and reducing laboratory errors permitting users to easily generate reports, shareable lists and plate design for genotyping.

iLAP, a freely available, open source, workflow driven information management system, aims at closing the gap between ELN and LIMS in the genome biology community [117]. While LIMS are supposed to manage both raw and processed data, ELN were developed to record and deal with scientific data and to enable their sharing. iLAP combines experimental protocol development, wizard-based acquisition of data and high-throughput data analysis into a single integrated system.

A fundamental issue in the design of a LIMS, and in general in software engineering, is the volatility of requirements [107]. This problem often originates from the developer holding an incomplete knowledge of the domain of interest and it is exacerbated by the extreme rate of innovation and change in technology and scientific knowledge. In their ontology-driven LIMS solution developed for web-based case reporting in medical mycology, Shaban-Nejad et al. [107] propose an original approach based on software agents for the analysis and the management of volatile and dynamic requirements.

As clearly highlighted in [107] it is difficult to combine the Information Technology and computer programming skills necessary to implement and directly customize a LIMS, with the domain knowledge and laboratory expertise needed to precisely describe laboratory workflows. Seedpod, a model driven LIMS with a web-based graphical user interface [60] goes in this direction, allowing users to create an integrated model of a LIMS without programming.

2.1.2 Lessons learned from commercial LIMS

As we have seen, a LIMS must be integrated in a concrete laboratory. In [85] two different views to build a LIMS are described. One view is to build a LIMS around a laboratory; the converse is to base the laboratory on the LIMS. A modern laboratory is a constantly changing reality in which instruments and procedures are changed every 1-2 years perhaps even less. Introducing new procedures and/or instruments could be expensive. Therefore an opportune architecture should be adopted in the design a LIMS. McDowall [85] points out two base consideration:

- The underlying processes must be streamlined and standardized as much as possible;

- The LIMS environment must be designed on the assumption that there is no single application that will automate the whole laboratory.

VelQuest Corporation [64], an industrial vendor, put forward a similar distinction. They distinguish between *thick LIMS* and a *thin LIMS*. The former is a LIMS developed from scratch around a laboratory need. The latter uses Commercial-off-the-shelf (COTS) components to build the core facilities combining them through data exchange facility. The main assumption of this vendor, for which all the corporate strategy is built around, is that more benefits are to be gained by using a *thin LIMS* approach. Traditional LIMS (*thick LIMS*) are a result of big efforts for custom coding and to follow the requirements associated with any customization in the attempt to bind an information management system to laboratory operational tasks.

GenoLogics Life Sciences Software Inc [56] is another player in the LIMS scenario that adopts a similar vision. Furthermore, they single out some other important requirements to be fulfilled by a LIMS:

- Sample traceability;
- Adaptability to changing technologies and methodologies;
- Workflow management tools.

Three simple criteria are then proposed for evaluating a LIMS:

- Does the LIMS enable labs to get up and running quickly?
- How easy is the LIMS to configure and customize by the user?
- Does the LIMS accommodate different users and workflows?

ID	**Lesson learned**
N-01	High level of customization required
N-02	Separation of concerns between IT and domain knowledge
N-03	Volatility of requirements
N-04	Integration with legacy application/Interfacing with instrumentation
N-05	Increase degree of automation
N-06	Reducing labor-intensive task
N-07	Long term support of technology
N-08	Tracking of samples and recording of data
N-09	Recording of protocols

Table 2.1: *Main needs of a modern LIMS.*

In our view we could combine the strategies proposed by the two vendors. In Table 2.1 we summarize the needs resulting from our literature review and from our own direct experience in the field. The development of a LIMS requires different IT skills and domain specific (DS) knowledge to capture the requirements fully and correctly.

Each aspects (IT, DS) is best tackled by an expert of the corresponding field. The mixing of the skills is not commonly had and should be sought with extreme attention. Having the experience of an IT expert is not required to a biological expert and vice versa. In our interpretation, separating from the start those two areas is a strategic choice which both worlds could benefit from.

Generally, an information management system should operate in an environment in which legacy applications exist. Not considering the integration with those applications could to suboptimal efficacy of the system. In particular, a LIMS has two objects: the laboratory and the organization managing it [85]. A good implementation must consider both of them. Integration

ID	Technical requirement	High-level need
R-01	A direct representation of the concept of protocol	N-09, N-08
R-02	Capability to directly express protocols by the end-user	N-01, N-03
R-03	Two layer architecture: one high-level layer able to interact with the end-user	N-02
R-04	Two layer architecture: one low-level layer of the architecture able to interact with the laboratory environment	N-02
R-05	Capability to automatic relate the high-level layer with the low-level one	N-02
R-06	Capability to add new resources during the lifetime of the system	N-03, N-01
R-07	Slim and general driver architecture able to integrate directly with legacy application and instrumentation	N-04, N-06
R-08	Runtime environment able to perform automatic actions	N-05, N-06
R-09	Long term support of technology	N-07
R-10	Distributed system	N-04, N-08

Table 2.2: *Technical requirements*

with legacy application is therefore a key asset for a LIMS. The same applies for instrumentation interfacing. The opportunity of interfacing with existing and feature, instrumentation should not missed in order to maximize the efficacy of the system. An important objective for a LIMS is to reduce the need for manual intensive tasks.

Technical requirements

From the review of the LIMS we can infer that deriving a general specification it is not an easy matter. The difficulties include volatility of requirements, incomplete knowledge, extreme rate of innovation [107]. From the literature, nevertheless, we can single out some transversally desirable characteristics.

A LIMS should not be too costly to purchase and maintain and complex to operate [22]. It should have a modular architecture, because we cannot believe in "one-size-fits-all" approach [22]. Ease of integration makes multiple, independent and interleaved efforts possible [119]. A further objective should perhaps be the simplification of common laboratory tasks so as to reduce protocol errors [125]. Experience shows that uniting software engineering skills with domain knowledge in a single profile is doomed to fail [107]. It is rather opportune to enable each to deliver the best in a "separation of concern" manner. When considering concrete protocols from the standpoint of domain experts, we want to be able to deal directly with an explicit representation of a protocol as a first-class entity. Table 2.2 shows the technical requirements extracted from the industrial needs as well from the relevant literature.

2.2 Ontology

2.2.1 Ontology in modern times

The exponential growth of experimental data, owing to rapid biotechnological advances and to high-throughput technologies, as well as the advent of the World Wide Web as a new means for data exchange, make it more complicated and difficult to find the biological meaning hidden in the heterogeneous biological data available to the scientific community. Furthermore the huge amount of information that are now produced on a daily basis, require more sophisticated management solution, while the availability of the Internet as a modern infrastructure for scientific exchange has created new demands with respect to data accessibility [95]. At the same time, in the era of genome-scale biology, the accumulation of biological data is accompanied by the widespread proliferation of biology-oriented databases [62]. The need to unambiguously classify the huge amount of data available as well as to precisely define their semantic relationship has increased the need for formal knowledge representation. In the 1980's, the ontologies entered the computer science field as a way to provide a simplified and well-defined description of a specific domain or an area of interest.

More recently, we have seen an explosion of interest in ontologies as models to represent human knowledge. Ontologies are now extensively used in applications related to areas such as knowledge management, natural language processing, e-commerce [63], web services [66], intelligent information integration, bioinformatics [1], education, life sciences [24] and medicine [31], and in widely adopted technologies such as the Semantic Web [19].

There are several reasons for this large scenario of applications. Ontologies provide a common terminology, over a domain, necessary for communication between people and organizations and also provide the basis for interoperability between systems. They can be used for making the content in information sources explicit and serve to index repositories of information [77].

The growing interest in ontologies triggered the development of Ontological Engineering, a novel field concerned with the ontology development process, the ontology life cycle, the methods and methodologies for building ontologies, and the tool suites and languages that support them [93, 41].

Despite the cited advantages, the choice of ontologies and formal representations incurs considerable costs for the retooling and upgrade of resources, and for the training of ontology developers.

2.2.2 Definitions

The word 'ontology' comes from the Greek *ontos* (being) and *logos* (word) and its conceptual origin can be traced back to early philosophers who have studied the theory of objects and their ties for centuries. In philosophy, ontology is used to name the discipline that tries to describe reality. But the term 'ontology' is still controversial because different people have different ideas on the definition of an ontology. The first formal and explicit approach to ontologies in the technical (not philosophical) sense dates back to 1900, given by Husserl. Later in the 1980's, the ontologies entered the computer science field as a way to provide a simplified and well defined view of a specific area of interest or domain. There is consensus in what an ontology is not: it is not a taxonomy (is not just a class-subclass hierarchy), a dictionary (ontology includes relationships between terms), nor a knowledge base that includes individual objects. According to Gruber, an ontology, is "the specification of conceptualizations, used to help programs and humans share knowledge" [59].

An ontology defines "a set of representational primitives with which to model a domain of knowledge or discourse" [79]. Ontologies provide a common shared vocabulary to model a domain, defining the types of objects and concepts that exist with their properties and relationships. Ontology can be classified according to the subject of conceptualization according to [84]:

1. general or common ontologies, defining concepts to represent common sense knowledge, reusable across domains;

2. top-level ontologies, defining very general concepts independent of a particular domain such as space, time, object, event, etc., and providing general notions from which all root terms in existing ontologies should

be related;

3. domain ontologies, defining concepts within a specific domain and their relationships; the concepts in this type of ontology are usually the specialization of concepts already defined in a top-level ontology;

4. task ontologies, defining concepts related to the execution of a particular task or activity and providing a vocabulary of terms used to solve problems associated with task that may or may not belong to the same domain;

5. application ontologies, containing all the definitions needed to model the knowledge required for a particular application.

2.2.3 Upper ontologies

At present the notion of Ontology has proved to be a useful tool to express domain knowledge. In the biomedical domain an important number of different ontologies has been produced. One serious problem with that is that differing ontologies may be developed and applied for the representation of the same domain. However, the mere use of ontology obviously does not warrant the elimination of heterogeneity; instead it can raise heterogeneity problems to a higher level. The development of the upper ontologies began to resolve these problems.

The function of an upper ontology is precisely ”supporting interoperability between domain ontologies in order to facilitate the share used of data both within and across disciplinary boundaries” [105]. Many upper ontologies were proposed describing a high-level of abstract concepts of reality.

BFO

In the context of bioscience a widely upper ontology used is the Basic Formal Ontology (BFO) [112] created by the Institute for Formal Ontology and Medical Information Science (IFOMIS). This ontology is based on the philosophy of realism (also called in this context BFO-realism). It takes the basics from Aristotle's concept of reality [88].

A formal ontology is a theory at the highest and most domain-neutral level and deals with the categories and relations which appear in all domains and which are in principle to reality under any perspective [58]. Object, relation, group, number, part-of are examples of categories. BFO, following the interpretation given by Husserl, describes the basic structures of reality. The two main branches are Continuant and Occurrent:

- The Continuant branch describes entities that have continuous existence and persist self-identically through time. Examples are a table, the sun, a protein.
- Occurrents otherwise are processes and hence occur in time and they unfold themselves through a period of time. Examples are the life of a person, the run of an horse.

The authors of BFO recognize the difference of being continuants and occurrents and prepared the SNAP (for continuant) and SPAN (for occurrent).

BFO has been adopted by many projects[1], some of which operating in the biomedical domain. An example application of BFO can be seen in the Ontology for Biomedical Investigations (OBI).

There are some limitations in the approach used by the authors of BFO, the question about what features should have an upper ontology is still very

[1]http://www.ifomis.org/bfo/users

open. Some opponents criticize the rigidity of the concept of existence. In [49], for example, the authors put forward the impossibility to describe the concept of hypothesis as a fundamental blockage.

2.2.4 Ontologies in biology

In biology the heterogeneity of ontologies represents an emergent issue. The use of the word ontology within biology is relatively recent. Several decades ago, the main aim of the bioinformatics was to store, retrieve and analyse the data produced by biologists; such as for example nucleotide sequences and protein structures. At that time, the limited amount of data produced by biological researchers, required simple systems for their management, organization and analysis.

However, the advent of the genome sequencing projects, high-throughput experiments, and other techniques produced a huge amount of data that needed analysis. The amount of accessible biological data started to grow exponentially. Nowadays, bioinformatics systems have to deal with large amounts of complex information, unmanageable for a scientist without sophisticated knowledge of management and information processing tools [83]. Data are now dispersed throughout several different databases and their interpretation and analysis require sophisticated tools for data management and information processing. Organized in this way, biological information is encapsulated within database schemes and is not easily available to scientist.

The available data are growing at an exponential pace but the knowledge contained in them is not growing equally fast. There are different reasons for this lack of productive knowledge and the most important one is that biological phenomena can be described in many different ways [61] and this complexity has not been tackled at semantic level. That means that usually the biologists are left with a giant base of information that they cannot access, analyse, or integrate in a pratical way [46]. The impossibility of drawing on information from the available data, adds further pressure to implement standardised and compatible nomenclature in molecular biology.

Computer scientists therefore recognized in biological data a domain in which ontologies were needed in order to solve problems of heterogeneity. A subsequent phase saw the adoption of bio-ontology by the biological community itself as a means to consistently annotate different features, from genotype (e.g nucleotide sequences, proteins) to phenotype (e.g. diseases) [29].

The fundamental problem with all that is that biomedical scientists collect facts, often recording them in natural language, and then use that knowledge to make inferences about yet uncharacterised observations. Because of this, their knowledge domain is highly heterogeneous. While it is easy to compare, for instance, nucleic acid or polypeptide sequences between bioinformatics resources, the knowledge component of these resources is very difficult to compare, both for humans and computers, because the knowledge is represented in a wide variety of lexical forms [29].

Often in biology a word refers to two different concepts: for example, the concept of 'gametogenesis' means different processes in mammals or in plants and a user, querying a database for this concept, needs to deal with these terminological and conceptual incompatibilities. This situation makes it more complex for a computer system to process biological information because it would not be able to reason over the data and capture the knowledge content. Thus, there is urgent need to find a strategy for the representation of biological knowledge in a formal way that facilitate reasoning data processing based.[39]. One way to do this is to represent the knowledge as ontologies:

the resulting 'bio-ontologies', a relatively new area of bioinformatics [24].

In the last decade, several groups have been developing controlled vocabularies and descriptors mainly for the annotation of this kind of data. For instance, the Metabolomics Standards Initiative (MSI) ontology working group is developing an ontology to facilitate the consistent annotation of metabolomics experimental data [7]. Besides the well known Gene Ontology [1], there are many other initiatives focused on standardization and ontology development that may be cited, such as MIAME [6] and PRIDE [12].

Open Biomedical Ontologies

As stated above, an ontology has to be widely disseminated and accepted among the users of the field that it aims to capture. In this respect, a strong community involvement is crucial to ensure that each specific domain is represented by a single ontology. This result is reached by the Open Biomedical Ontologies standards.

The Open Biomedical Ontologies (OBO)[2] is a collection of controlled vocabularies developed in 2001 for the ontological representation of several biological domains. The aim of this initiative, focused on object-level questions, is to represent in an exhaustive way the proteins, organisms, diseases or drug interactions that are of primary interest in biomedical research [113].

The main role of the OBO umbrella is to be an ontology resource. It is supported by the NIH Roadmap National Center for Biomedical Ontology (NCBO) through its BioPortal and it is continually kept up-to-date by ontology-based developers. There are currently over 60 live-science ontologies lodged in OBO, covering domains such as anatomy, development and phenotype, genomic and proteomic information and taxonomic information. All of

[2]http://obofoundry.org

them use a range of different attributes to describe the respective biological domain.

To be included in OBO, an ontology has to be developed following a set of principles that are used to give coherence to wider ontological efforts across the community:

- openness: ontologies must be available to all, without any constraint or license on their use and it is only asked that users acknowledge the original source. This encourages usage and community buy-in and effort;

- common representation: this is either the OBO format[3] or the Web Ontology Language (OWL)[4]. This provides common access via open tools and offers common semantics for knowledge representation;

- independence: lack of redundancy across separate ontologies encourages combinatorial re-use of ontologies and the interlinking of ontologies via relationships;

- identifiers: each term should have a semantic-free identifier, the first part of which refers to the originating ontology. This eases easy management;

- natural language definitions: terms themselves are often ambiguous, even in the context of their ontology, and definition helps ensure appropriate interpretation. Thus, the terms in each ontology must have a proper textual definition explaining clearly the exact meaning of the concept within the context of a particular ontology.

[3]http://www.geneontology.org/GO.format.shtml#oboflat
[4]http://www.w3.org/TR/owl-features/

The principles described above are necessary to ensure that the OBO ontologies remain a resource for the entire community. At the same time, the developers of a small set of OBO ontologies have initiated the OBO Foundry. The participants have established a set of additional principles over the existing and well-defined original OBO rules. These further principles require that ontologies:

1. are a result of a collaboration among the other OBO members;
2. use a set of relationships defined in the OBO Relation Ontology (RO) [111];
3. provide procedure for identifying successive versions;
4. represent a clearly specified and delineated content to ensure additivity of annotations and to bring the benefits of modular development.
5. members can propose new principles using the OBO wiki page[5].

The long-term goal is that the Foundry offers a resource, where data that are produced by biomedical researches and available to the scientific community, are collected in a consistent and algorithmically traceable way. In this way it will be possible to solve some problems associated, for instance, with the differences between technical and biological language.

Ontology for Biomedical Investigation

In the available literature we to find a good amount of ontologies mainly intended for biological data annotation. However, only a few projects have been developed for the representation and formalization of the experimental

[5]http://obofoundry.org/wiki/index.php/OBO_Foundry_Principles

protocols and the automatic operations producing such experimental data. A formal definition of scientific experimental design, laboratory entities and operations is undoubtedly important, also in the case of manually executed experiments. The development of an ontology of experiments is a fundamental step in the formalization of science, since experimentation is one of the most characteristic features of science.

In this regard, the EXPO ontology of scientific experiment has been developed to formalize generic knowledge about scientific experimental design, methodology and representation of results [114]. Ontology represents the design of an investigation, the protocols and instrumentation used, the material used, the data generated and the type of analysis performed.

EXACT [115] is an ontology of experimental actions that can be used as a formalism suitable for a structured representation of laboratory protocols.

The OBI ontology addresses the need for controlled vocabularies not only for the experimental data annotation but also for the representation of investigations in the Biological and Biomedical Sciences [40]. OBI is a controlled vocabulary with additional logical constraints expressed in OWL. OBI supports the annotation of database records, in addition can be used to improve the annotation process automatically checking the consistency. OBI contains 2,500 classes and is pubblicy available.

The authors of OBI presented some use cases in order to assess the comprehensiveness of OBI and to show how to use terms for annotation. They have discussed some real-world experimental processes to demonstrate how to model entities and relations between entities in biomedical investigations using OBI. The three use cases discussed in [32] are: i) a neuroscience investigation about the role of the primate *caudate nucleus* in the expectation of reward following action; ii) a vaccine protection investigation; iii) an auto-

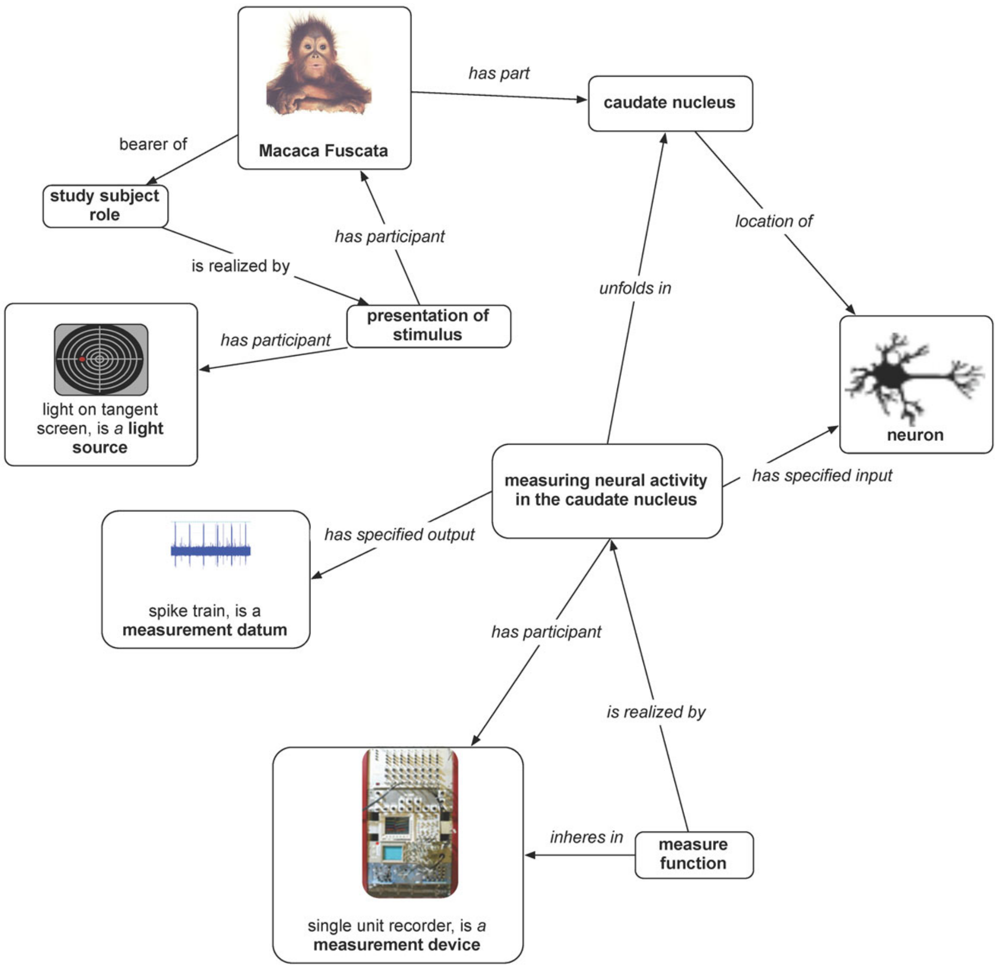


Figure 2.1: *Boxes represent instances labelled with the related class. Relationship are depicted as links labelled in italics. For example. the organism Macaca fuscata has NCBI_9542 as term ID (from the NCBI taxonomy). Source: [32].*

mated functional genomics investigation. Each of those use cases is represented by statements using terms defined in OBI. Figure 2.1, for example, depicts a model of a single trial in the aforementioned neuroscience investigation. The upper level consists of the BFO classes *material entity*, *process*, *role*, *function*, *information content entity*.

Material entity In BFO, *object*, *object part* and *object aggregate* are used to describe physical things. However, usually biomedical investigations uses entities with a large span of size (e.g. from cells to tissues to organs). In order to avoid to operate at this low level of granularity the *material entity* class has been created. The class *material entity* is an *independent continuant*. Material entities are spatially extended and their identity persist through time, for example an organism and a centrifuge. They import several subclasses from external ontologies like the *molecular entity* hierarchy from ChEBI [45] or the *organism* from the National Center for Biotechnology Information (NCBI) taxonomy [127].

Planned process The definition given for *Planned process* is: "A processual entity that realizes a plan which is the concretization of a plan specification."[6]. A *Planned process* is intentionally initiated by an agent to achieve some goals, specified as *objective specification* in OBI. An instance of a process could have inputs and outputs specified by means of *has specified input* and *has specified output*. Outputs are needed at the end of the process to attain objective. Inputs are declared in *plan specification* and are not originated during the execution.

A *plan specification* is "a directive information entity that when concretized it is realized in a process in which the bearer tries to achieve the objectives, in part by taking the actions specified. Plan specifications includes parts such as objective specification, action specifications and conditional specifications."[7].

[6]Full ID: `http://purl.obolibrary.org/obo/OBI_0000011`
[7]Full ID: `http://purl.obolibrary.org/obo/IAO_0000104`

Information entities Since the domain of OBI is important to describe information like data, results, reports. The Information Artifact Ontology (IAO) was created as a separate effort with the aim to develop a general theory of information entities. It is imported by OBI with *Information content entities* as root class. Since they are a *generically dependent* they must be borne by other entities. One of the subclasses of IAO is *directive information entity*. Both *Plan specification* and *Objective specification* are subclasses of it. Other classes include *protocol*, *study design*.

Roles and Functions As described in [20], a *Role* has two properties: an entity which the *Role* is linked with, and a process that realizes the *Role*. It is a *Realizable entity* and the realization is not typical of its bearer. The *Role* is played by an instance under some circumstances, and it is optional. A *Role* is used in OBI to define the study design of an investigation.

A *Function* is similar to the *Role*: it is a *realizable entity* and it has a bearer. The main distinction is that the manifestation is a reflection of it in-built physical structure, the given structure is designed to exercise the structure. As example, the *function* of a computer program to compute mathematical equations, conversely the *Role* of a person as a surgeon.

2.3 Model-driven engineering

2.3.1 Definitions

The IEEE Computer Society defines software engineering as “the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software” [18]. The term “software engineering” has been coined in the 1968 NATO Software Engineering Conference.

It can be divided into sub-disciplines the first of which is software requirement gathering consist as “a property which must be exhibited in order to solve some problem in the real world” [18]. Software analysis is the process of elaborating system requirements to derive software requirements. One of the outputs of this process could be conceptual models, comprising “models of entities from the problem domain configured to reflect their real-world relationships and dependencies”.

Model-driven engineering (MDE) is a software engineering method which focuses on creating such models [55]. One of the goals of MDE is to reduce the gap between models and implementations. This gap exists when a concrete realization is developed using lower abstractions that those used to express the model [54]. MDE concentrates on designing models that are closer to domain-specific concepts of some particular domain rather than to computing (or algorithmic) ones. MDE’s basic concepts are models, meta-models and transformations [50].

MDA is an instance of MDE. MDA is based on Meta-Object Facility Meta-Object Facility (MOF) by Object Management Group (OMG). A common chain of work under this methodology comprises the definition of a metamodel under a specific domain. A model conforms with a metamodel is built. Eventually, by means of specific transformation an executable representation is created. Tools supporting the above chain are vital to draw real benefits from application of the methodology. Specific technology already exist that support the whole chain (e.g. the ecosystem that revolves around Eclipse).

MDE’s basic concepts are model, metamodel and transformation [50]. A classical representation of this concept is given under the form of a 4-layers pyramid shown in figure 2.2 [47]. The real-world manifestation of a model is called M0 (bottom of the pyramid). Things that we try to represent are at the M1 level. They conform to a reference model described by a language from the M2 level, also called metamodel. The metamodel defines the concepts that should be used in defining models at M1 level. The set of concepts used to define a metamodel reside at the M3 layer and is called metametamodel. A metamodel has a metametamodel as reference model. Finally, a metametamodel is a model that is its own reference model (i.e. it conforms to itself) [72]. These layers are called the linguistic layers [34].

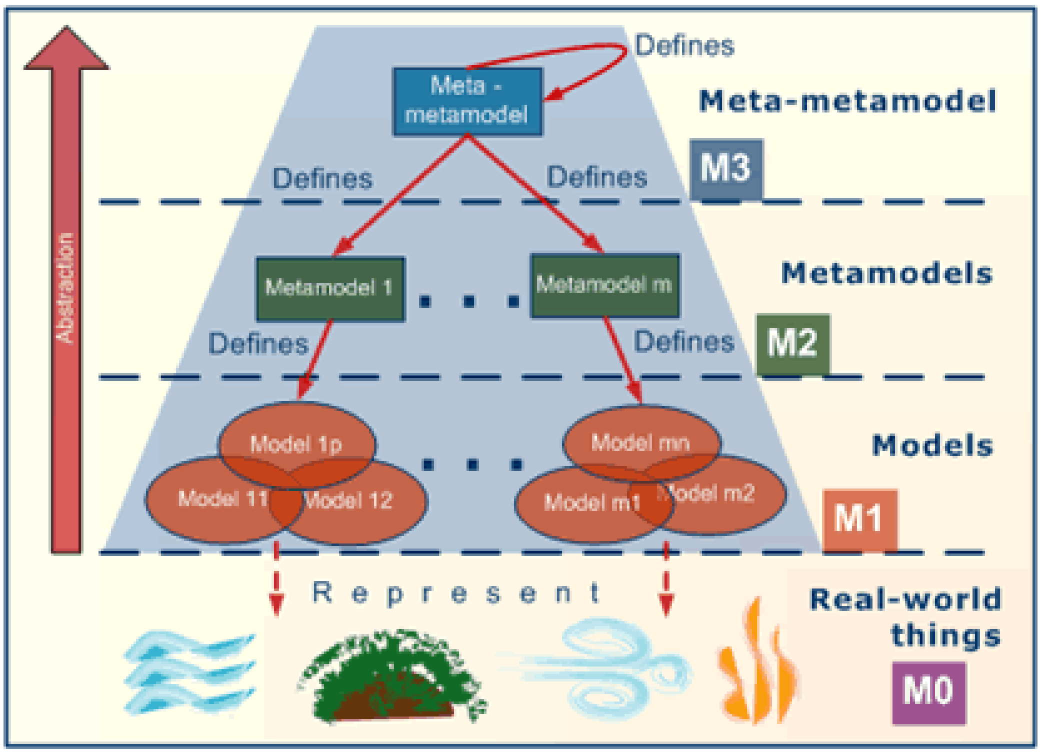


Figure 2.2: *A general modelling architecture. The M0 layer represents real world things. At M1 level there are abstraction of real things. Those are defined, as thus conform to, at the M2 level. Which is defined using a language at M3 level, that is finally conform to itself. Source: [47].*

A model-to-model transformation takes a model in input (with a reference metamodel) and produces another model in output (with a reference metamodel). It therefore enacts a bridge that transforms concepts from one modelling space into corresponding concepts in another modelling space. Conversely a model-to-code transformation produces as output the code for some programming language.

Models

We not try here to answer the question: what a model is and which features are relevant to define it.

In software engineering a model is a tool used to minimize errors in software development. It is the designer's representation of a particular aspect of a concrete reality. It is expressed by a precise language which is its metamodel. Hence the model conforms to its metamodel and this conformance enables deterministic processing of the model. By analogy a metamodel is for a model what a grammar is for a programming language [30]. The main assumption in MDE is to consider models as first class entities [34].

A more precise definition is given in [106]: “a model is a set of statements about some system under study”. The important concept of *interpretation* is given here as “a mapping of the model's elements to elements of the system under study”. In traditional scientific disciplines models are usually descriptive. However they are also used as specifications in engineering disciplines, including software design. Therefore a model could be either descriptive or prescriptive.

A distinctive aspect of models is undoubtedly their relationship with reality: “A model is an external and explicit representation of a part of reality as seen by the people who wish to use that model to understand, change, manage, and control that part of reality”[96].

Models can represent, describe, and specify things [21]. A *descriptive* model is one that "describes reality, but reality is not constructed from it". A *prescriptive* model is one that "prescribes the structure or behaviour of reality and reality is constructed according to the model; that is, the model is a specification for reality". Since in the realm of software engineering most of the models are used to construct a "reality" from them, in the remainder of this work we take a model to be prescriptive entity.

PIM and PSM MDA defines an approach that separates the specification of the intended system functionality from the implementation of it on a specific technology platform. MDA allows us to use a single model, specifying system functionality, to be realized and deployed on different platforms. System functionalities are described into a *Platform Independent Models* (PIM). How such functionalities are realized is specified in a *Platform Specific Model* (PSM). Figure 2.3 shows the typical path from PIM to PSM and code. As result a PIM makes an abstraction and in doing so it hides technical details.. A PSM instead has implementation concepts specific of a given platform. In order to relate a given PIM with a PSM of choice a transformation should be written. A model-to-model transformation may be used to project a PIM, associated with a description of the target platform, to the execution infrastructure thus creating a PSM. With the same approach, source code for the implementation could be automatically generated from a PSM. This transformation step, PSM to code, is currently more mature than PIM to PSM.

2.3.2 Models and Ontologies

In the recent years, meta-modelling and ontologies have been considered together to build a common framework. To better understand how ontology

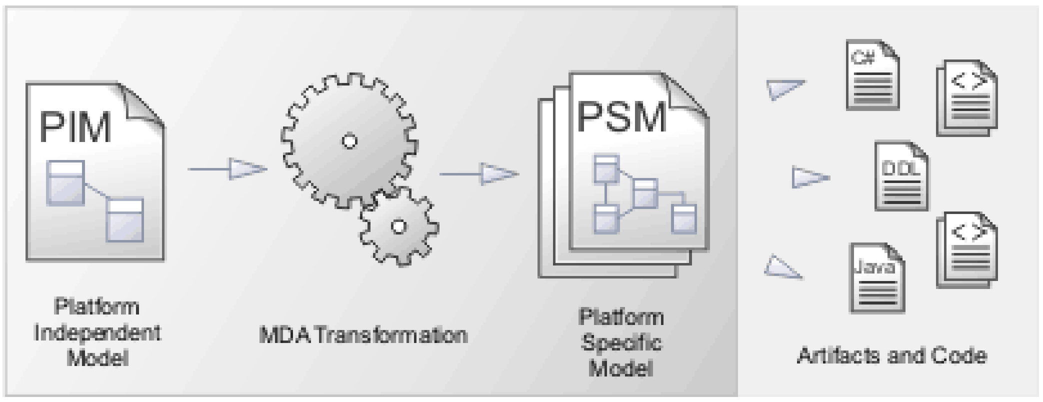


Figure 2.3: *Traditionally in MDA, a specification independent from the target platform (PIM) is first created. The PIM is converted by automatic transformation (PIM to PSM) in a PSM . If needed, the PSM could be refined (PSM to PSM). Finally executable code is generated (PSM to Code). Source:* http://www.sparxsystems.com/uml_tool_guide/mda_transformations/mdastyletransforms.htm.

and MDE could relate to each other we now discuss some qualities of models and ontologies.

The *open-world assumption* is a characterizing property of ontologies. It states that anything not explicitly expressed by a knowledge base is unknown. If a particular statement is not made we cannot infer a false value about that it, but we only know that the information is not known [69]. Conversely, the *closed-world assumption* expresses that any statement that is not known to be true is false (this is usually the case in databases).

Another important issue is to discern whether models describe or control reality. Hence whether they are descriptive or prescriptive [106]. A model is descriptive if it describes reality thus makes statements about some system. The reality is not however constructed from it. A model instead is prescriptive if it prescribes structure or behaviour of reality. In this case a model is a specification of reality. In general models could be equally descriptive or prescriptive. However, since they specifies a set of statements from which a system could be developed, in software development they usually have a prescriptive nature [21]. Since the open-world assumption does not allow a complete and final description, an ontology has always a descriptive nature.

Ontology-aware meta-pyramid

As seen in subsection 2.2.3, an upper ontology is built for better interoperability. A domain ontology (see subsection 2.2.2) specializes concepts taken from an upper ontology. Using the previous consideration, Assmann et al. [21] propose the ontology-aware meta-pyramid. Figure 2.4 shows the corresponding model: domain ontologies live at level M1 and correspond to models. An upper ontology, providing a language for ontologies, should live at level M2. Assmann et al. [21] argue that one meta-metamodel language could be used to specify both ontology and meta-models. This is depicted in Figure 2.4 in which at level M3 there is one artifact. Both the ontology dimension and the model-driven dimension instantiate from this meta-metamodel.

Linguistic and ontological dimensions in the MDE layers

Bézivin [34] use a different approach to relate ontology and MDE. The four MDE levels are called in this contest linguistic layers. They point out that concepts from the same linguistic layer can be at different ontological layers. Figure 2.5 depicts the four meta-layers considering this important remark. The linguistic instantiation runs on the vertical dimension; conversely the ontological instantiation runs on the horizontal dimension.

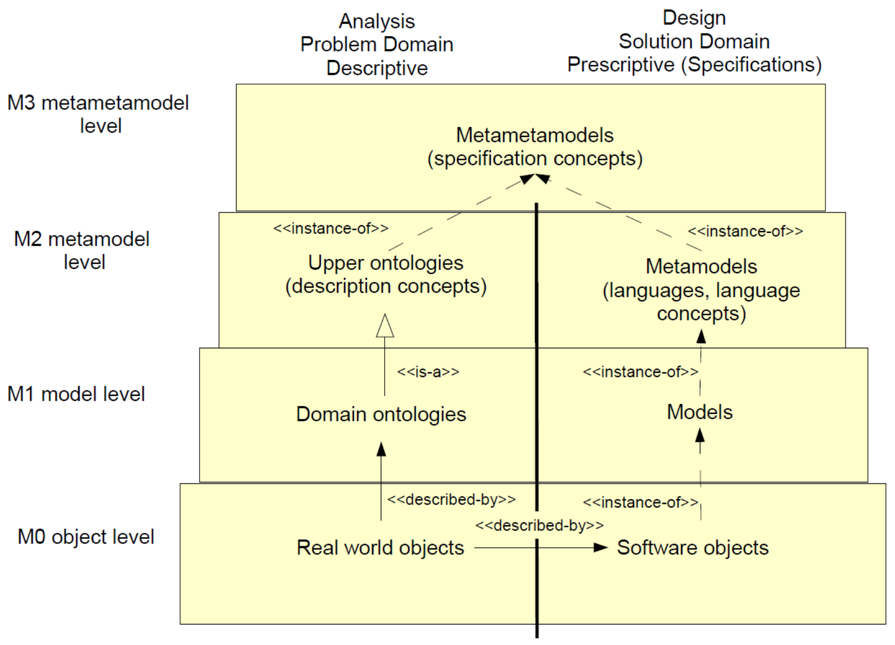


Figure 2.4: *The ontology-aware meta-pyramid. Domain ontologies live at level M1. Upper ontologies live at level M2. Ontology metalanguages live at level M3. Source: [21].*

Mapping from model and ontological technical spaces

Another formulation is given by [73]. The authors propose a mapping from the model engineering to the ontology technical space. Their work is focused on a mapping between Ecore and the Ontology Definition Metamodel. Some caveats following from the difference between those technical spaces are discussed (e.g. differences from *EClass* to *OWLClass*) in the cited work.

Mapping between EMF objects OWL/RDF

Hillairet Guillaume and Yves [68] proposed another mapping between EMF objects with OWL/RDF Resources. Also this mapping presents some difficulties: class membership is an example. In object-oriented languages the

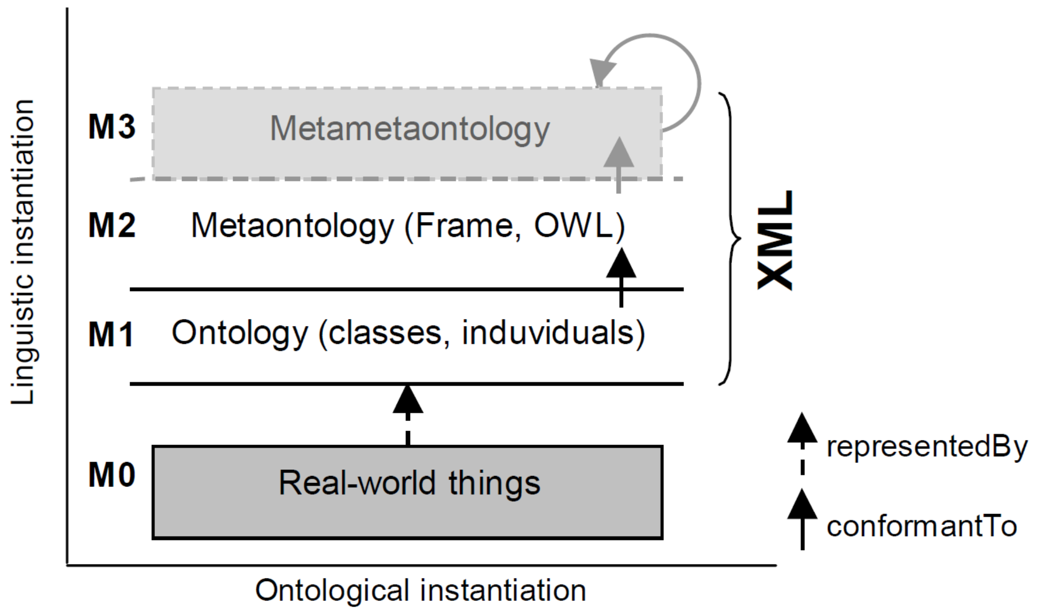


Figure 2.5: *The four meta-layers in terms of ontological engineering and its orthogonal instance-of relations: linguistic and ontological. Source: [34].*

membership of objects is fixed. In OWL instead an individuals can belong to multiple classes. The authors propose a set of Eclipse plugins that are able to make a round-trip transformation between OWL and Ecore. The project (named EMF4SW[8]) is not yet mature enough to cope with large and complex ontologies. However it is in very active development and it is able to handle with relatively small ontologies.

2.3.3 Workflows

Definitions

In the workflow context, a process can be considered as the set of activities performed by different entities, and their execution ordering through different constructors, which make it possible to control the flow of execution

[8]http://code.google.com/p/emftriple/

(e.g. sequence, choice, parallelism and join synchronization). An elementary activity is an atomic piece of work [101].

A workflow is a representation of a sequence of operations, declared as the work of a person, a group of persons, or machines. Workflows make the description and the orchestration of complex processes possible in a visual form, capturing human-to-machine interactions within those processes. Several disciplines adopt workflows systems for the automation of data processing through a series of processing stages.

A workflow is therefore the structured definition of a process used for the automatic management of particular activities. The formalization of a process (workflow schema) involves the definition of activities, the specification of their order of execution (i.e. the routing or control flow) and of the responsible actors. Other features should be taken into account too, e.g. the data flow [101] or the various ways in which resources are represented and utilized in workflows [102]. Many formalisms and notations exists, we concentrate our description on Business Process Modelling Notation (BPMN) and XML Process Definition Language (XPDL).

In the last few years the interest for workflow development has seen a considerable growth also in the scientific community [44]. Scientific workflows can be considered as the executable description of scientific processes [109]. Similar in nature to business workflows, they have the distinct characteristic of operating on large amounts of heterogeneous data. In particular, they are generally data-flow oriented instead of being control flow and event-based. They also are very versatile in composing flows of execution. In bioinformatics, in particular, workflows are extremely valuable for programming the steps of *in silico* experiments in a visually intuitive manner. However, workflows are still not commonly adopted in the formalization of protocols for biological laboratory experiments.

There are several available tools for workflow design and enactment [100], for instance JPEd [5], an open-source visual editor for general-purpose workflows. Taverna [89], developed by the myGrid project, is the workflow platform most commonly used for the systematic analysis of vast amounts of data, but it does not allow the description of laboratory experimental procedures. Taverna workflows can be shared among the scientific community thanks to Web 2.0 initiatives like myExperiment [8]. This social web site enables scientists to publish their workflows and in addition to execute, reuse and share workflows of other groups. In this way myExperiment contributes in reducing time-to-experiment, in sharing knowledge and expertise and in avoiding reinvention [43].

BPMN

BPMN [3] is a graphical notation based on intuitive flowcharts for the definition of business processes. Originated from the Business Process Management Initiative, in 2005 it was merged into OMG [11] and in 2009 1.2 became a standard. A major revision process for BPMN 2.0 is in progress. BPMN aims to support both technical and business users. The notation provided is based on simple graphical elements, the main goal of which is to provide a standard notation understandable by all the experts involved in the business process.

The set of graphic elements is relatively small and comprises just four main categories: *flow objects*; *connecting objects*; *swimlanes*; *artifacts*. In Figure 2.6 we can see the main elements for each of those categories.

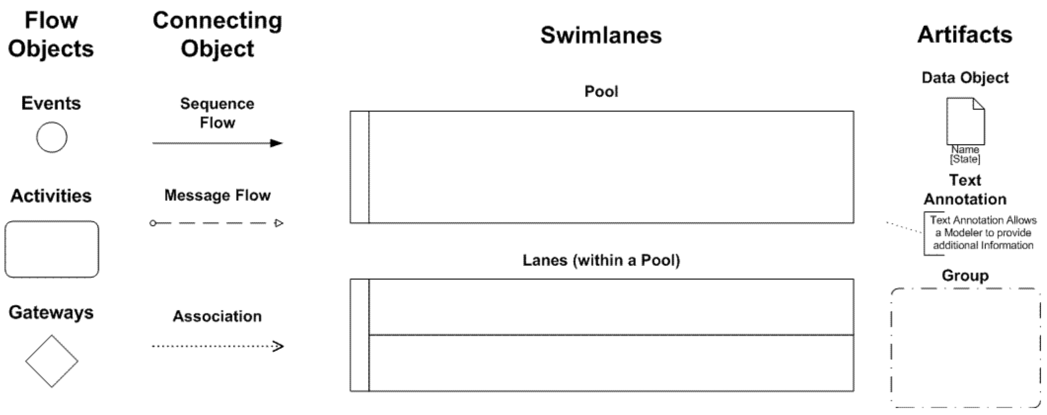


Figure 2.6: *Core set of BPMN elements. Source:* `http://www.bpmn.org/Samples/Elements/Core_BPMN_Elements.htm`

Flow object The *Flow object* categories consist of three core elements: *Events*; *Activities*; *Gateways*. *Event*, rappresented as a circle, denotes something that happens. Many types of events are described by BPMN, the principal are the *Start* and *End* event, respectively green and red. *Activity*, describes some kind of work to be performed. *Task* and *Sub-process*, some special cases are in which the former represents an atomic unit of work and the latter is used to involve some self-described process. *Gateways* are used finally, to describe splits and/or joins.

Connecting objects *Connecting objects* are used to make connections between *flow objects*. A line with an arrow is used to describe an execution order, namely *Sequence flow*. *Message flow* (open circle at the start, dashed line and an open arrowhead) describes which messages flow across pools. *Association*, represented with a dotted line, describes a relationship between an artifact and a flow object.

Swimlanes *Swimlanes* are used for categorising activities. *Pool* contains one or more *lanes*. The former is used to differentiate between organisation, instead the latter is used to organise activities accordingly with a performer or a role.

Artifact The last category, *artifact*, permits to add some information to make the diagram more clear. *Data objects* describe which data is produced or is needed. *Group* is just a way to group different activities without affecting the flow of the process. An *Annotation* is used to make comments about the chart.

XPDL

XPDL [16] is a markup language created to ensure interoperability among different workflow management tools in order to handle workflow processes. It was designed to enable the exchange of process definitions, addressing both the graphical and the semantic notations of the relevant workflow. Born as a support for serialization of BPMN constructs, it also incorporates information relating to the graphical representation (e.g. the position of blocks in the workflow). XPDL was developed by the Workflow Management Coalition (WfMC) [15], a consortium formed to define standards for the interoperability of workflow management systems.

XPDL [108] is based on a XML syntax specified by an XML schema. They main elements of the language are *Package*, *Process*, *Activity*, *Transition*, *Participant*, *DataField*, *Type Declaration*.

The *Package* element is a container that holds all the other elements. It could have some *Processes* performed by one or more *Participant*. A set of *Activities* could be declared at *Package* level to be used by processes. It is possible to use standard types to define *DataFields*. In addition, it is possible to declare new types using different mechanisms such as external references or in-line type declaration.

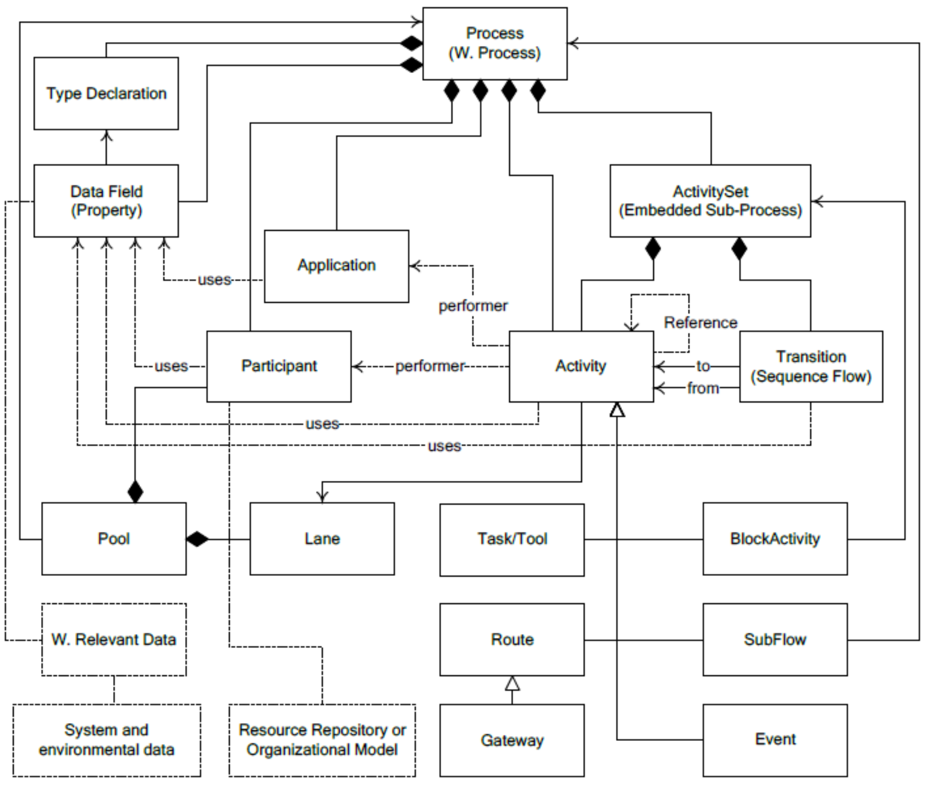


Figure 2.7: *XPDL Process metamodel. Source: [108]*

Process metamodel In Figure 2.7 we see an in-depth representation of *Process* used in the XPDL metamodel. A *Process* is a composition of different elements. A set of elements of *Activity* type is declared inside a *Process*. An *Activity* element is the main block of a workflow definition. It could be of various types such as *Task/Tool* or *SubFlow*. *Task/Tool* defines a set of *Applications* used to specify the interface that should be used to call specific services. A *SubFlow* activity invokes an external self-contained *Process*. A

Route activity is a dummy *Activity* used for routing purposes. *BlockActivities* are used to execute *ActivitySet* that are embedded sub-processes. *Participant* elements are used to specify the entities that execute work.

Activities are connected by *Transitions* that are used to specify the sequence flow. Each of them connect a *From-Activity* with a *To-Activity*. In order to make decisions about the sequence (i.e. which transitions needs to be fired) workflow relevant data are used. They are specified using the *DataFields* element, *Datatypes* defines new types. A *Process*, and therefore also a *SubFlow*, could take parameters in input and specifies parameters as output. A third mode of parameter passing is provided, *INOUT*, in which parameters are modified during the execution.

2.4 Multi-agent system

2.4.1 Definitions

The Multi-agent system (MAS) is a natural and powerful metaphor for conceptualizing, designing and implementing software systems with components, possibly distributed that exhibit properties of autonomy and communication. MAS provides a model more consistent with reality itself. It is commonly used to describe complex systems in which autonomous entities are called upon to solve common objectives through the only means of interaction between them.

Agents are commonly classified by means of some exhibited properties. Wooldridge and Jennings [128] propose four properties that an agent should exhibits:

Autonomy: agents incorporate an internal state, not accessible by other agents, and makes decisions based on his actions to his condition, without the direct intervention of humans or other agents;

Reactivity: the ability to react to environment changes around them, whenever such changes affect their goal;

Pro-activity: the ability to generate events in the environment, start the interaction with other staff, coordinating the activities of different agents stimulating them to produce certain responses;

Social ability: the ability to communicate with other agents, cooperating in pursuit of common objectives, exchanging information and knowledge.

In the last few years the relevant literature recognized the need to explicitly embody the notion of resource in a MAS [98], [90]. A well known approach to address this need is to use the notion of "artifact". Artifacts can be considered as complementary abstractions to agents populating a MAS. While agents are goal-oriented pro-active entities, artifacts are a general abstraction to model function-oriented passive entities. MAS designers employs artifacts to encapsulate some kind of functionality, by representing (or wrapping) existing resources or instruments mediating agent activities (see figure 2.8) [75]. The intent is to encapsulate functionalities and services in suitable first-class abstractions at the agent level [98]. Artifacts could be used for wrapping existing resources and therefore are a suitable model for our purpose. Particularly fitting is the Agent and Artifact model [98], in which an Artifact is structured as a set of operations.

Ricci et al. [99] is proposed *simpA*, a framework built to facilitate the development of concurrent applications built on top of agents, artifacts and workspaces. Workspaces are logical containers. They are used to structure the environment where agents play. *SimpA* is developed using Java and

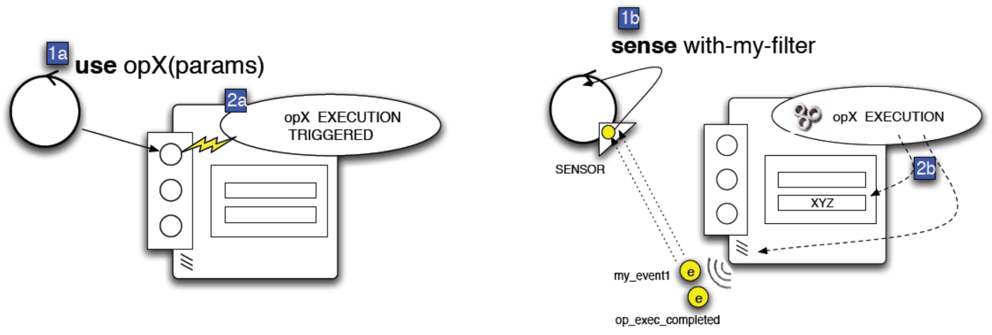


Figure 2.8: *Abstract representation of an artifact (left) and of an agent using an artifact and observing the events generated. Source: [99].*

using extensively the Java Annotation Framework. It is an interpreter of a program specified by an agent. The program contains the descriptions of the activities that the agent needs to execute. Activities could be of two basic types: atomic and structured. Atomic activities are instructions containing actions that could interact with the environment. Structured activity are composition of sub-activities. Atomic activities are declared as methods tagged with the *@ACTIVITY* annotation. Method's body specifies the computational behaviour of the activity.

2.4.2 Agent-based workflow management system

Business Process Management (BPM) is a well-know practice in IT. High quality, mature tools are currently available to manage business processes. However current BPM systems suffer from a number of weakness. The main drawbacks include [27]:

- Limited flexibility
- Inability to cope with dynamic changes
- Inadequate handling of exceptional situations

- Limited ability to predict changes
- Insufficient interoperability

A BPM system could draw great benefit from agent-based methodologies. Some of these advantages are based from the properties of MAS. For example, the agent metaphor allows decentralized ownership and an high degree of concurrency. Moreover agent-based technologies allow the building of distributed and decentralized systems that are closer to the real-world environments.

The enactment of workflows using multi-agent systems has already been proposed in the literature [33, 25]. Result, include the development [53] of workflow management systems based on the popular open source MAS platform JADE [26]. Researchers are currently exploiting agents to amend process integration, interoperability, reusability and adaptability [120]. Societies of software agents could be used to manage and coordinate workflow defined by business processes. Exploiting these methodologies facilitates design process and supports distributed dynamic process management. Agents need to interact and communicate with other agents in the environment to coordinate themselves and control distributed workflows tasks [120]. In order to improve interoperability there is a need for standard semantic constructs.

Chen and Tu [38] proposes an agent-based system using ontology and RFID technology to monitor and control dynamic production flows. The authors describe a whole system composed of several types of agents designed to perform collaborative supports for just-in-time and just-in-sequence production strategies. In the cited architecture all the agents are deployed on a centralized server. Only agents designated to interact with RFID tags are deployed in ad-hoc local computers. The ontology in this case is used to describe RFID tags and thus to render able agents to exchange data coherently.

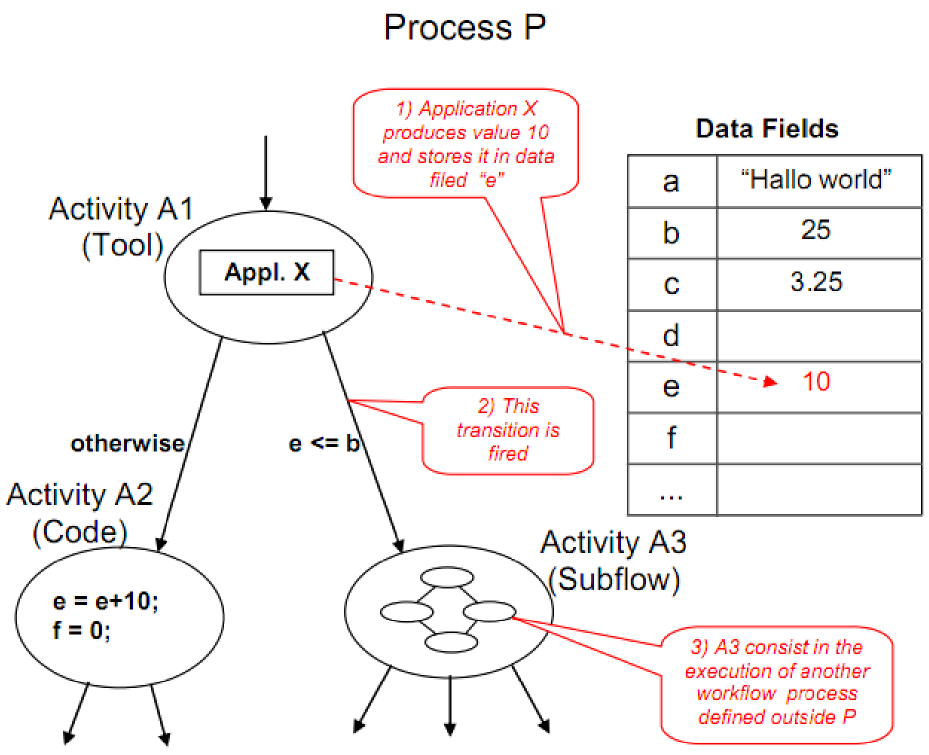

Process P
1) Application X produces value 10 and stores it in data filed "e"
Data Fields
a "Hallo world"
b 25
c 3.25
d
e 10
f
...
Activity A1 (Tool)
Appl. X
2) This transition is fired
otherwise
e <= b
Activity A2 (Code)
e = e+10;
f = 0;
Activity A3 (Subflow)
3) A3 consist in the execution of another workflow process defined outside P

Figure 2.9: *The main elements in the WADE workflow metamodel. A Process is composed of a set of Activities. Each Activity has one or more Transitions with the possibility to add conditions. One ore more Formal parameters are used for defining inputs and outputs. An Activity could be of different types: Tool, Subflow, Code. Source: [36].*

Workflow and Agent Development Environment (WADE) [36] is a software platform proposed as an extension to JADE by the JADE development group itself. JADE provides a middleware in which software agents are able to act and interact by means of FIPA[9] standard protocols. WADE has been developed on top of it, with the implementation of new features for supporting the use of workflows in the deployment of multi-agent applications [36].

[9]http://www.fipa.org/

WADE includes a micro-engine embedded in a set of dedicated agents which are specifically developed for the execution of workflows defined in an extended version of the XPDL metamodel. Doing this, the new engine permits to directly execute the Java code associated to a specific workflow activity. Moreover, this new tool allows to choose and assign secondary agents for the execution of subflows. Additional components have also been defined in WADE so as to manage administration and fault tolerance issues. The main challenge in WADE consists in bringing the workflow approach from the business process level to the level of system internal logics [97]. In other words, the objective is not to support an orchestration of services provided by different systems at high level, but to implement the internal behaviour of single systems.

Figure 2.9 depict Wade workflows shown in a meta-model directly derived from XPDL. The main difference from the standard XPDL meta-model relies on the class *Activity*. Besides the classical activies (*Tool, SubFlow, Route, etc etc*) a *Code Activity* has been added. The *Code Activity* is a peculiar feature of WADE and as such it allows to define Java code to be executed during an Activity.

Chapter 3

Proposed solution

3.1 Architecture

This PhD project tackled the problem presented in Chapter 1 by aiming to close the gap between the biologist's information management system concept on one hand and the software engineering knowledge and technology involved in automating the execution of laboratory protocols on the other hand. To this end, inspired on MDA, we developed a software framework that enables the biologist to directly plan and express his/her protocols without the need to draw from software engineering knowledge and technology, including programming languages, compilers, interpreters, and the like.

The architecture of the solution we envision comprises the following four main constituents:

1. A high-level language as close to the experience and the needs of the biologist as possible, in MDE this is often referred to as "domain specific language";

2. A graphical editor capable of enabling the user to graphically specify the desired protocols with the provided high-level language;

3. A model-to-code compiler able to translate the user protocol in code directly ready for execution on the target platform (the laboratory environment);

4. A run-time environment that understands an executable version of the protocol produced by the said compiler and is able to relate the constraints and needs expressed in the protocol to the actual capabilities of the environment and to accordingly execute multiple protocols in parallel.

The proposed language (1) describes the operational perspective of laboratory protocols using a workflow metaphor expressed in XPDL [108]. To make it better fit our purpose we enriched the XPDL meta-model with concepts drawn from an ontology specialized in biomedical investigations (OBI [40]). We then used the Eclipse Modelling Framework (EMF) to integrate the OBI ontology and the XPDL schema, from which we obtained a new meta-model (nicknamed BioCow after Biology Combined Ontology [and] Workflow). BioCow makes it possible to model workflows in terms of objects and actions specific of our target domain, which arguably meets Objective 1 as specified in Chapter 1.

To meet Objective 2 we used the Obeo Designer[1] to enable the user to graphically specify protocols. This work had a more engineering than scientific nature. However, its final result was important in that it enables the user to produce models of the desired protocols in a manner that constructively guarantees conformance to our meta-model. When the user draw a

[1]http://www.obeodesigner.com/

protocol the editor is able to interpret the ontological constraints and to prevent the violation of them. In fact the editor enable only the symbols that are ontological valid.

Objective 3 was achieved by a classic model-to-text transformation approach for which we used Acceleo[2]. In designing our transformation engine we regarded the execution platform as composed of two distinct parts: the software infrastructure, which we require to be invariant and based on the multi-agent system (MAS) paradigm; and the laboratory equipment, which is the obviously variable part of our target setting. The former is bound to the latter by means of a battery of software drivers. Such drivers are outside of our direct concern and can be written in any programming language as long as they can be wrapped in Java classes that conform to our specification. For our purposes the services provided by those drivers describe the laboratory equipment to the level of detail needed by our model-to-code compiler (this enacts a correct-by-construction approach instead of a construct-by-correction development).

Our execution environment (cf. Objective 4) is built upon a MAS. This choice was motivated by the intuition that a biological laboratory can be regarded as a complex system comprised of a number of heterogeneous, autonomous entities potentially competing for physical resources. The agent metaphor was found to match our needs very well. A further dimension of interest in the agent technology was the ability to exploit the autonomous nature of agents to cope with contingencies, which will be needed to maximize the volume of correct and useful experimental data that can be obtained by automated execution of laboratory experiments.

The remainder of this chapter is as follows. In section 3.1.1 we describe

[2]http://www.acceleo.org/pages/home/en

the whole architecture. In section 3.2 we describe the front-end of our architecture. In section 3.3 we describe the back-end of our architecture.

3.1.1 Front-end and back-end of our proposed architecture

The system we propose offers a straight path from end-user specification to directly executable protocols. The end-user is provided with an environment in which she can describe her protocols using a language close to her domain knowledge and experience. The user interface of the LIMS supports a domain specific modelling language. The language is built incorporating formal domain knowledge directly into a workflow environment. The end-user deals with the terms of the language and their assembling into protocols by using a graphical editor. The LIMS is therefore composed by the following main building blocks (see Figure 3.1)

1. Front-end
   (a) BioCOW metamodel: it formalizes laboratory knowledge into a graphical language that combines workflow notations and elements of a biological ontology into a language specialised for the modeling of laboratory protocol
   (b) Graphical editor: it supports the language defined by the BioCOW metamodel.
2. Back-end
   (a) Model-to-code transformation: starting from a protocol specified in the BioCOW language it produces an executable program capable of carring the protocol out in a given laboratory

(b) Execution platform: it reads, interprets, checks and enacts the executable version of protocols and deploys them on a MAS platform that encapsulate the laboratory hardware system.

The architecture that we set out to built confirms with the Model Driven Engineering paradigm [54]. The end user (see Figure 3.1) is able to describe the experiment model in her own language. The corresponding formal specification (in the BioCOW meta-model) produced by the Visual Editor is then automatically translated into an executable specification that will be executed by a system of software agents. The product of the transformation is a set of Java classes that can be compiled and used directly in the runtime system. The transformation preserves the requirements and the constrainst specified by the end-users at design time. The transformation generates a system in a meta-model that need not to be directly handled - nor even known - by the end-user, but that preserves the requirements and the constraints postulated at design time.

Our project aims at simplifying the work of laboratory operators. With the drastic increment of formalization and automation that we achieve, the room for man-made errors will be greatly reduced and all the bookkeeping activities will not absorb any more the staff time.

3.2 Front-end: exploiting domain knowledge in a MDA style

We use the MDA paradigm to describe from a high-level more abstract, IT-neutral point of view the entities of a laboratory and their interactions. This higher-level model should be used by a laboratory expert to formally describe her experiment or routine protocol: the product of the user specification is a

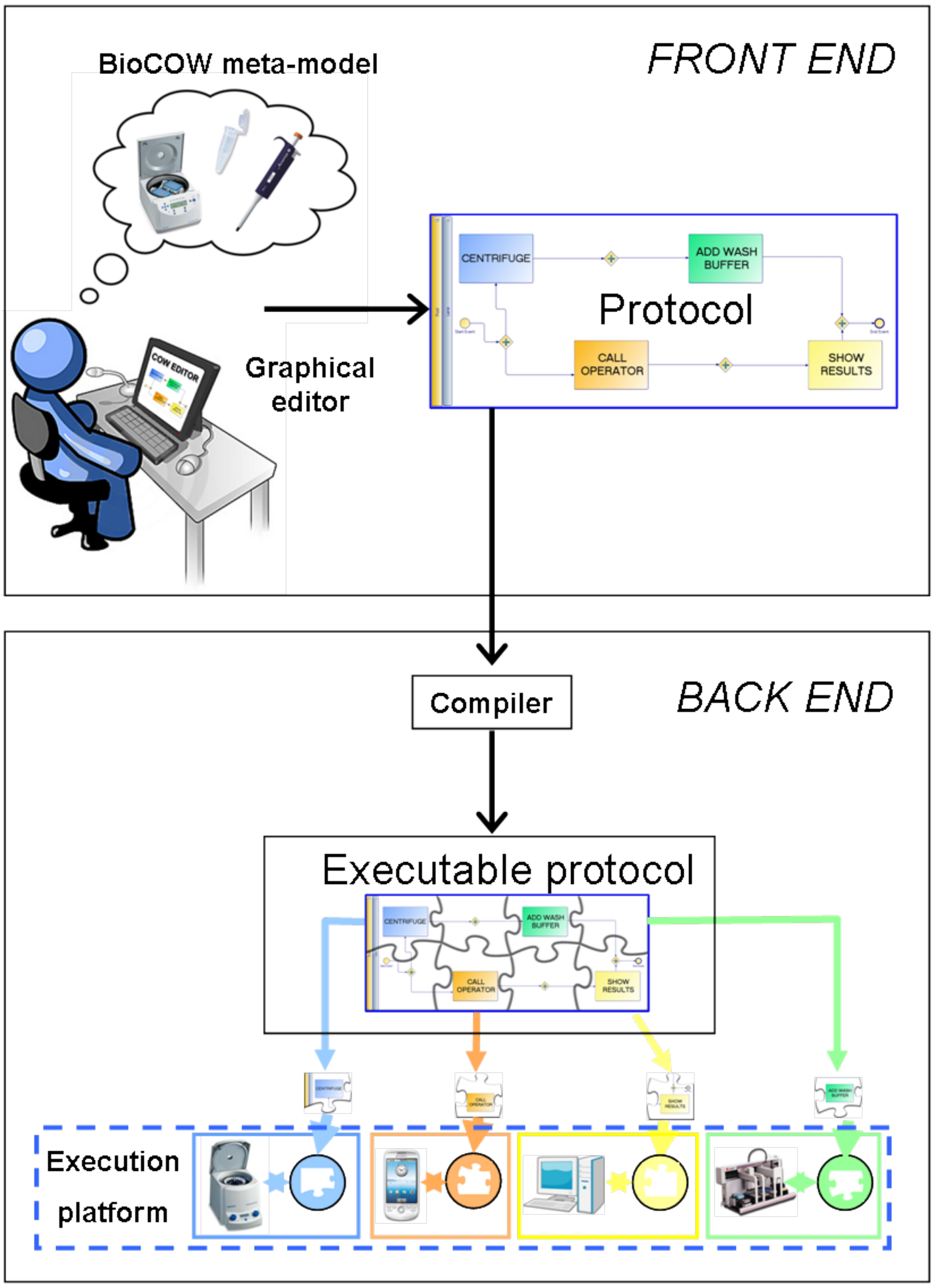


Figure 3.1: *Architecture of our proposed solution.*

model that rests on our metamodel. Subsequently, a fully automated transformation of the model derives executable code for it, ready for execution on a runtime platform that meets our MAS-base specification. The key benefit here is the possibility for the laboratory operator to describe his/her needs - the model of a protocol - using a language that draw from the relevant domain knowledge. Using our infrastructure there is no need for the end-user to learn a programming language for coding an information system to manage the execution steps of the intended protocol and treat the raw data produced from it. Similarly there is no need for the end-user to have probably difficult and expensive interactions with computer scientists for explaining them how to design and customize the needed information system.

In laboratories, and consequently in LIMS, a protocol is a procedural description of the steps necessary to perform an experiment. Protocols are generally expressed in a natural language. Only recently we have seen a trend to coagulate specific aspects of protocols into a more structured form [80]. Writing protocols in a natural language incurs hazards like ambiguity, interpretability, difficulty of automation, implicit knowledge and so forth. We have therefore chosen to promote the notion of protocol to a first-class entity and to adopt more formal denotations for representing and sharing protocols.

In our published work [80] we described an attempt to address those problems by combining the EXACT ontology [115] for representing biomedical protocols with the XPDL meta-model for workflow interchange. By means of an ad-hoc solution, we enabled laboratory staff to intuitively design their protocol by using a standard XPDL editor. Protocols are represented as workflows in the de-facto standard interchange language, incorporating domain knowledge from the EXACT ontology. We subsequently refined our metamodel using the modern OBI ontology..

The front-end of our system is the component destined to interact with the end users and in which the end-user formalizes his protocols. In our system we embed the descriptive knowledge of laboratory ontology inside a workflow prescriptive model, which describes a protocol from an operational point of view. Beginning from the 1980's, the field of computer science started to adopt ontology notions as a way to provide a simplified and well defined view of a specific domain. A workflow instead is a structured definition of a process, used for the automatic management of particular activities. A formalized process involves the definition of activities, their order of execution and their responsible actors [124].

3.2.1 Ontology

As we have seen in Section 2.2.4, OBI is an ontology for the description of biological and clinical investigations [40]. OBI relies on the Basic Formal Ontology (BFO) upper ontology and describes the design of an investigation, protocols and instrumentation, materials used, data generated and analysis performed on it. The ontology is developed to model biomedical investigations, therefore it contains terms for aspects such as:

- biological material, e.g. DNA
- instruments, e.g. centrifuge or thermal cycler
- design and execution of an investigation, e.g. injecting mice with a vaccine to test its efficacy.

An upper ontology is an artefact with the function of "supporting interpretability between domain ontologies to facilitate the share used of data both within and across disciplinary boundaries" [105]. An upper ontology describes concepts of the "Reality" from a high-level of abstraction. BFO is based on the philosophy of realism (also called in this context BFO-realism). It takes on from Aristotle's concept of reality [58, 88].

Since OBI is based on BFO we work with both ontologies. The relationship that we are going to build will be split in two layers. One against BFO and one against OBI. We will use therefore use the whole BFO. This approach prevent disruption in case of changes in the structure of OBI.

3.2.2 Metamodel

XPDL (see section 2.3.3) is a markup language created to ensure interoperability among different workflow management tools. Its main goal is to exchange process definitions, addressing both the topographical and the semantic notations of the relevant workflow. It also incorporates information relating to the graphical representation (e.g. the position of blocks in the workflow).

The meta-model of XPDL involves the definition of activities, the specification of their order of execution and the involved data. The flow of execution is specified through such constructors as sequence, split, join. An elementary activity is an atomic piece of work [101]. An Activity could modify relevant data declared as DataField. In addition to standard types a user could add external types (by means of an XSD declaration or an external reference). It is also possible to declare new complex types directly inside the XPDL file.

In our domain not all the entities of the XPDL metamodel are relevant. In the construction of our metamodel we use only the following elements:

- Process

- Activity
  - SubFlow
  - Route
  - Gateway
  - Task/Tool
- Transition
- Data field
- Type Declaration
- Relevant Data
- Formal Parameter
- Actual Parameter

For brevity in this list we did not include container entities like *Activities* and similar.

In our metamodel we replace the entity *Application*, linked with *Activity* by *Task/Tool* with the new entity *Action*.

3.2.3 BioCOW metamodel

As we are to build a workflow model embedding the OBI ontology, we focused our efforts in defining a precise relation between BFO/OBI and MDE. We did this because an ontology aims at describing a domain of knowledge, therefore it is *descriptive* in contrast with a model which is *prescriptive* [106].

Finding a method to relate MDE (and its various layers of abstraction) to the ontology schema is key to enabling the systematic use of ontology inside the prescriptive models. In order to formally include OBI as component of the XPDL meta-model we built a relation between the classic layers of MDE, BFO/OBI and XPDL.

MDE and XPDL

Figure 3.2 depicts the classic layers of MDE. The workflow components of our formalism are fairly easy to place in this hierarchy. XSD, the XML schema language used to describe XPDL can be positioned at the M3 level (i.e. meta-meta-model). XPDL conforms to a XSD model and therefore lies at the M2 level (i.e. meta-model). A valid XPDL workflow (i.e. a model for the end user) is at level M1. A specific execution of a workflow resides at the ground level M0 (not shown in the figure).

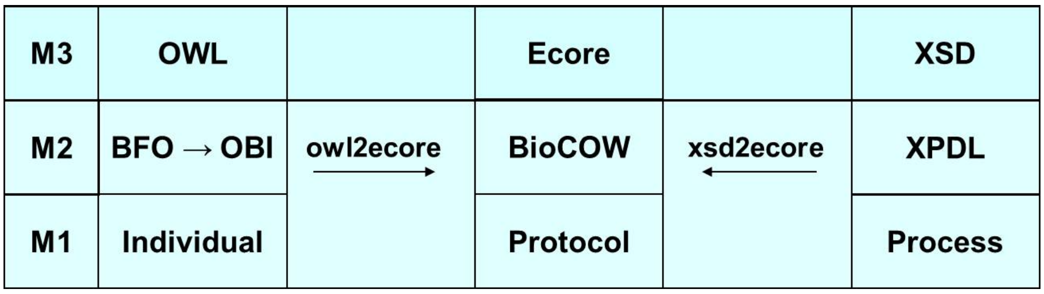


Figure 3.2: *The BioCOW meta-model is built by combining XPDL with BFO/OBI. Both are translated in Ecore by means of model-to-model transformation. A standard XSD to Ecore transformation is used for XPDL. For BFO/OBI we used an existing tool developed by Hillairet Guillaume and Yves [68].*

Let us for example analyse the Application construct of the XPDL meta-model. In XPDL the concept of Activity represents the unit of work. An Application is a particular kind of Activity that describes functionalities offered by legacy systems. In XPDL an Application is invoked by means of a Tool Activity. In terms of Object Oriented programming languages, an

Application can be seen as an interface for a functionality with a name and a list of parameters. We can think of an interface as a sort of “contract” between a class and the outside world. Every parameter is described with a name, a type, and a mode of passing (input, output, mixed). The Application construct is in fact the junction point between the workflow world of XPDL and the ontological world of BFO/OBI.

MDE and BFO/OBI

Before defining a mapping between BFO/OBI and XPDL we need to also relate the former to MDE. BFO is written in OWL, hence, in our schema of interpretation, OWL is at level M3 and BFO at M2. OBI is a specialization of BFO in the dimension of the description of the domain. It is not a specialization in the linguistic dimension proper of the MDE [21]. For that reason, OBI is at M2 in an orthogonal dimension (horizontal instead of vertical in Figure 3.2). A distinct consequence of this choice is that instances of BFO/OBI concepts (in OWL called individuals) are placed at M1. Using this schema of interpretation, individuals are tags that have as referent the real objects that we put at M1.

Mapping

At this point, we have laid out a sufficient basis to relate the parts of BFO/OBI of our interest with the XPDL meta-model to produce a mapping between elements of the two worlds. Table 3.1 presents the resulting mapping. In the first column we see the concepts that we choose to represent inside our metamodel. In our vision the list is exhaustive and general enough to cope with all the needs of a laboratory. The mapping to XPDL is not complete, as we have seen we left out the entities not relevant with our needs. We mapped all the entities of a laboratory against BFO as well as OBI. Figure 3.3 shows the composition of our metamodel. BioCOW is composed using entities from XPDL and terms from BFO/OBI. It is composed using the relevant parts of XPDL and a subset of BFO/OBI.

Laboratory	XPDL	BFO	OBI
Protocol	Process	Generically dependent continuant	Plan specification
Sub-protocol	SubFlow	Generically dependent continuant	Plan specification
Unique single step of a protocol	Task/Tool	Generically dependent continuant	Action specification
Real world (e.g. Illumina sample) or theoretical (e.g. Project) items	Data Type	material entity	material entity
		generically dependent continuant	information content entity
Objects properties	Data Field	specifically dependent continuant	quality

Table 3.1: *Mapping between XPDL and BFO/OBI. The relevant concepts of XPDL are mapped with concepts from BFO and OBI.*

The main concept of *Protocol* is easily mapped to the workflow model by the notion of *Process*. In the XPDL specification [108] a process is defined as a "combination of various activity with a specified flow of execution". An internal process consists of one or more activities, each comprising a logical, self-contained unit of work".

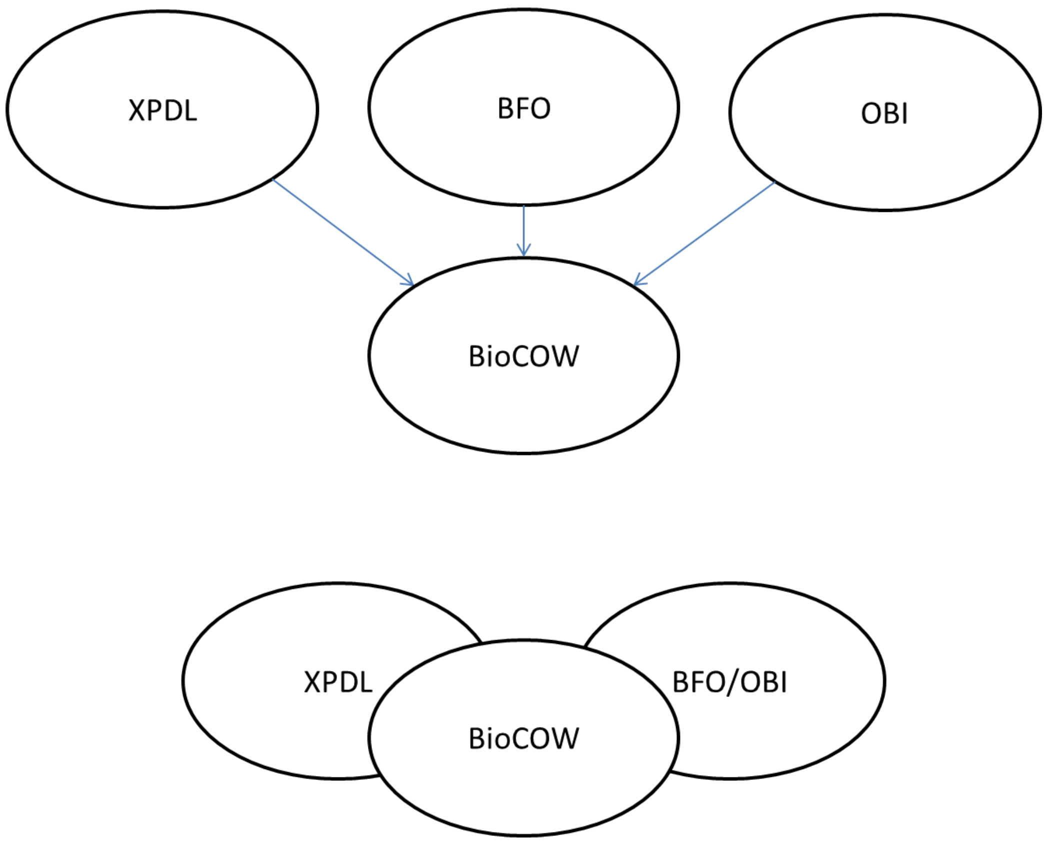


Figure 3.3: *Composition of the BioCOW metamodel.*

We connected this concept with the OBI concept of *Plan specification*, defined as a "directive information entity" that is a "Generically dependent continuant". When concretized it is realized in a process in which the bearer tries to achieve the objectives, in part by taking the specified actions. Plan specifications includes parts such as objective specification, action specifications and conditional specifications. A SubFlow (sub-protocol) is a process itself hence the mapping is the same as for process (i.e. *Plan Specification*).

The second main concept for our effort is the notion of unit of work. In XPDL this is backed by the Activity class, which can be of different kinds. One of those is the Task/Tool class, a service or an application required and invoked by the process. In the XPDL metamodel every tool declares a set of

Applications. We mapped this XPDL concept with the Action specification in OBI, a “directive information entity that describes an action the bearer will take” that is as well subclass of “Generically dependent continuant” of BFO..

Since an Activity is an atomic piece of work that could modify relevant data (declared as DataFields) we mapped both the XPDL concept of DataType and DataField. A Datatype in our model could be, in addition to any standard type, a *material entity* or an *information content entity*. We chose to map a DataField to the concept of *quality*.

Figure 3.4 shows a portion of the class diagram of the BioCOW metamodel. We simplified the XPDL meta-model maintaining the concept relevant for our purpose. Figure 3.4 omits technical details and shows only the protocol relevant concepts. A Protocol is composed by a set of Activities and Transitions between them. An Activity could modify some relevant data of the Protocol declared as variable (Datafields). An Activity could be of three kinds: Route, SubFlow, Action. Only the latter describes a unit of work. Route Activity permits the explicit expression of split or join sequence flow. A SubFlow activity is a node in a process which invokes another protocol.

In the first column of table 3.1 we can see all the main entities of our metamodel. Those entities are general in interpretation and sufficient in quantity enough to describe protocols in our environment. In order to build our metamodel we mapped those concepts to the XPDL metamodel. Since XPDL is richer than our needs not all the entities of our metamodel have a corresponding match. For example the entities *Pool* and *Lane* are left out in our metamodel.

We then matched the relevant concept of our metamodel against BFO and OBI. We were able to match all of them against BFO. Working with OBI instead was more difficult. OBI is a rich ontology and potentially all the terms in it are of our interest. We decided to map only a subset of OBI. We restricted the choice on the higher term on the hierarchy of OBI. For example we map “material entity” and “information content entity” to *DataType*. “material entity” is a class of BFO that OBI inherits. The rationale behind this choice is to preserve our metamodel against future changes in OBI. OBI is in active development, however the main structure should not change, like for example the main branches. Not all the relevant terms of OBI could be included in the current version. We had to omit some useful terms because the technology that we used to translate the OWL formulation of OBI into Ecore is not mature enough to cope with large ontologies like OBI.

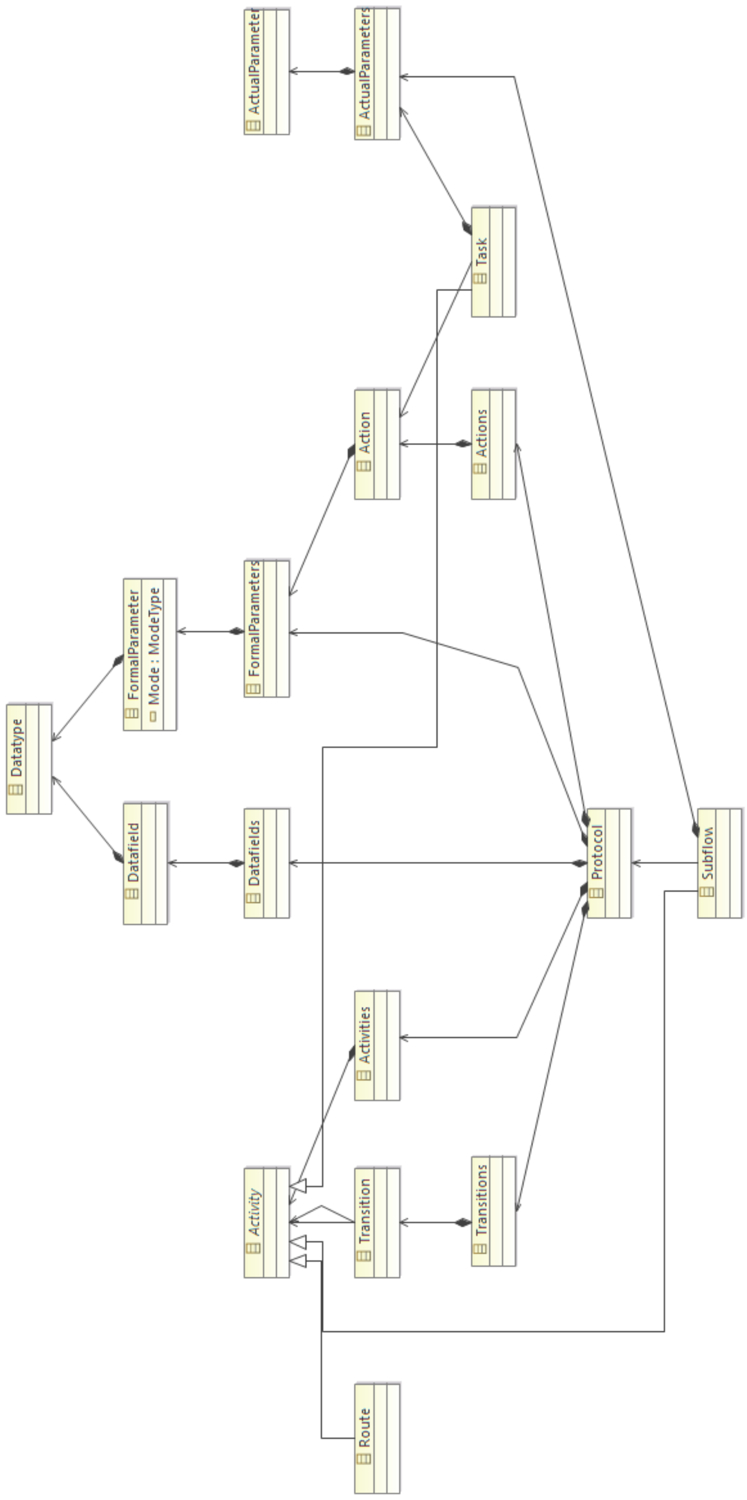


Figure 3.4: *Portion of the class diagram of the BioCOW meta-model.*

3.2.4 Implementation

In order to actually build the described meta-model we used the technology provided by the Eclipse Modeling Framework Project (EMF)[3]. EMF includes a meta-meta-model language (named Ecore) used to describe models and meta-models. It is used internally by the framework to provide runtime support and XMI serialization. EMF provides tools to automatic convert other formats in Ecore. Specifically there is a standard way to translate an XML Schema Definition (XSD) file in the Ecore format. Since XPDL is formulated in XSD we automatically imported it in the EMF.

We had to use a different approach for OBI. We used the approach proposed by [68] described in section 2.3.2. In particular, we translated the whole BFO ontology and the main classes of OBI from OWL into Ecore. The authors of the cited workpropose a set of Eclipse plugins that are able to make a round-trip transformation between OWL and Ecore. The project (named EMF4SW[4]) is not yet mature enough to cope with large and complex ontologies (as OBI). However it is in very active development and it is able to deal with relative small ontologies (as BFO is). We used that project to translate in Ecore some portion of the ontologies of our interest. In particular, we translated the whole BFO ontology and the main classes of OBI.

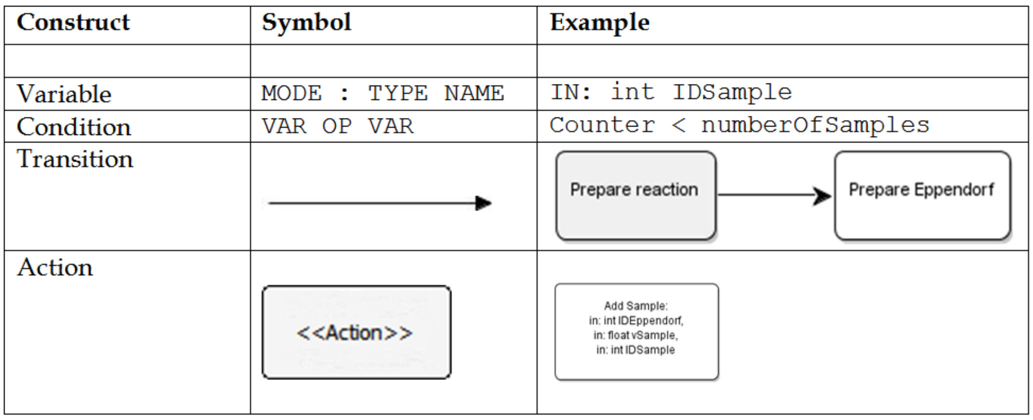
Our BioCOW meta-model is built based on the XPDL meta-model. In order to actually concretize the mapping between XPDL and BFO/OBI we created a new class for every mapped classes. The new class inherits both the XPDL and BFO/OBI classes as specified in the mapping shown in table 3.1. For example, the *BioCOW:Action* class has, as a superclass,

[3]http://www.eclipse.org/emf

[4]http://code.google.com/p/emftriple/

the *BFO:GenericallyDependentContinuant* class. It is worth noting that we have not specialized directly the XPDL meta-model since, as it is, is already richer than we need for our purposes. We therefore only retained the main concepts of XPDL and left out all the surplus details.

Using the resulting BioCOW (Bio-medicine Combined Ontology [and] Workflow) meta-model we are now able to describe laboratory protocols in a formal yet intuitive way. By means of the Obeo designer we are able to build a Graphical User Interface which associates graphical symbols that continue to conform with the workflow metaphor with constructs of the BioCOW meta-model. Obeo designer[5] is a tool that permits to define your own graphical representations using your own meta-models. Figure 3.5 shows the graphical notations used in BioCOW. A variable follows a standard schema of declaration enforcing also the mode of passing (input, output, input-output). A condition could be attached to a transition, paths with condition not satisfied are not followed. A transition specifies the route of the flow of execution between actions. Finally an Action is declared specifying actual parameters.



Construct	Symbol	Example
Variable	`MODE : TYPE NAME`	`IN: int IDSample`
Condition	`VAR OP VAR`	`Counter < numberOfSamples`
Transition	→	Prepare reaction → Prepare Eppendorf
Action	<<Action>>	Add Sample: in: int IDEppendorf, in: float vSample, in: int IDSample

Figure 3.5: *Principal constructs used in BioCOW*

. Principal constructs used in BioCOW.

[5] http://www.obeodesigner.com/

The Protocol Visual Editor allows end-users, expert of biological laboratory domain, to design their experiments by using controlled, well-defined domain terms to describe samples, equipments and experimental actions. End users are not required to have programming skills and the specifications they devise, which on the visual editor are rendered as intuitive workflows elements, are stored in the BioCOW format. The editor, built on the underlying meta-model, is semantically "cultured", and therefore able to interpret the constructs of the meta-model in a domain-specific manner that fit the user intent. Hence, the protocols designed with the editor are syntactically and semantically correct, as the editor prevents the introduction of statements not conforming to the meta-model rules.

3.3 Back-end: translating and executing protocols

3.3.1 Model-to-code transformation

“Programs” (i.e. protocol models) visually designed with the graphical editor in the BioCOW “programming language” (i.e. meta-model) can be used to easily program (i.e. customize) our architecture. Using a model-to-text transformation (i.e., as in MDA) we are able to translate the user protocol in executable code ready for execution into the target platform.

In designing the transformation engine we regarded the execution platform as composed of two distinct parts: the software infrastructure, which we require to be invariant and based on the multi-agent system (MAS) paradigm; and the laboratory equipment, which is the variable part of our target setting. The former is bound to the latter by means of a battery of software drivers. Such drivers are outside our direct concern and can be written in any programming language as long as they can be wrapped in Java classes that conform to our BioCOW profile. For our purposes the services provided by those drivers describe the laboratory equipment to the level of detail needed by our model-to-code compiler. In our laboratory facilities we have set up specific drivers for a centrifuge, a robotic arm, a liquid handling device, a sealer and a thermo-cycler. Centrifuge Actions includes “Centrifuge”, “OpenLid”, “GiveBucket”. We have also tested another set-up with different devices and needs in the laboratory of a commercial organization. In the runtime execution environment we therefore created a layer of abstraction to provide a homogeneous view of the services provided by heterogeneous laboratory machines or operators.

The model-to-text transformation takes as input the model to be implemented and a “description” of the available laboratory equipments in the execution platform. The latter is needed to bind each action used in the modelled protocol with the corresponding real action to be performed by the equipment in the laboratory. The output of the transformation is a Java package composed of a set of Classes describing a protocol ready to be executed by a specific target platform that conform to the JADE/WADE framework. Figure 3.6 shows the layer of our back-end.

The invariant part of our execution platform is based on a Multi-Agent System. A MAS is a system composed of multiple agents assigned with specific tasks. A software agent can be seen as an autonomous, reactive, proactive and social entity [128]. The functionalities of the user system result from the interactions among agents. The adoption of this paradigm was motivated by the consideration that a biological laboratory can be regarded as a complex system comprised of a number of heterogeneous, autonomous entities potentially competing for physical resources. We see the agent metaphor as perfectly matching our needs. A further dimension of interest in the agent technology is the ability to apply autonomous capabilities to cope with contingencies. This capability will be needed to maximize the volume of correct and useful experimental data that can be obtained by automated execution of laboratory experiments.

3.3.2 Execution platform

Moreover, from a technological perspective we needed to handle the physical distribution of the system. Different devices located in different places need to interact with one another. A multi-agent framework deals directly with this issue giving us the freedom work at a higher level of abstraction. For the

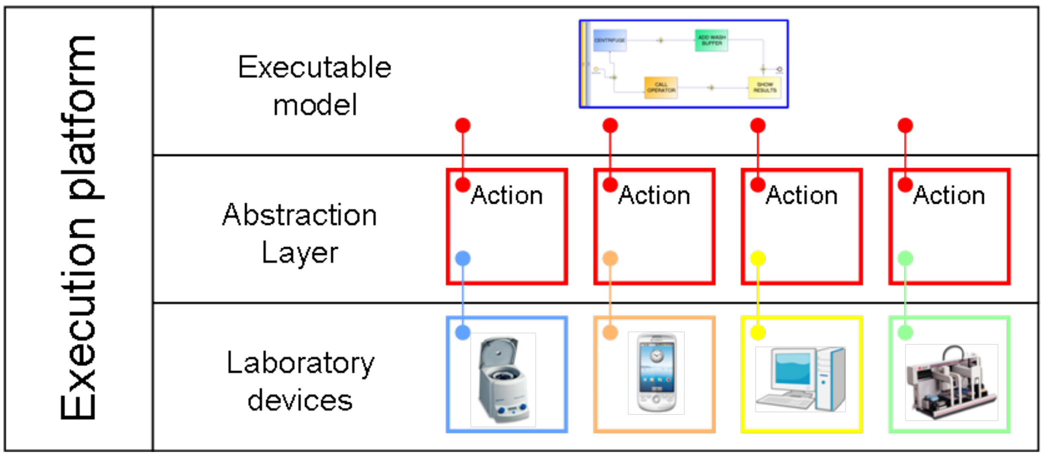


Figure 3.6: *A conceptual overview of the abstractions involved in the back-end layer of our solution. Operations provided by heterogeneous devices are uniformly exposed as Actions for use by the executable model.*

implementation we utilized the well-known JADE framework[6] in combination with the WADE [36] extension. JADE provides a middleware in which software agents are able to act and interact by means of FIPA[7] standard protocols. WADE allows agents to execute workflows using a slightly modified version of XPDL.

In order to effectively exploit these MAS technologies we chose to build our back-end system upon JADE/WADE. Input to this execution platform is the output of our model-to-code compiler. In order to execute compiled versions of the user protocol we had to define an execution model. The execution model describes which construct are available and how the runtime platform deals with them.

[6]http://jade.tilab.com/
[7]http://www.fipa.org/

Executable Model of a protocol

In this subsection we describe the Executable model of a protocol, that we have defined for coding executable protocols in our runtime platform. An essential issue is the necessity of monitoring every protocol during every step of its execution, saving both the information on input and output data and on the executed procedures. It must be noted that the executable protocol is not the one defined by the end-user by means of the Visual Editor and stored in the COW format, but its translation generated by our compiler.

A laboratory protocol can be seen as the composition of precisely defined activities [40]. The executors of the activities could be instruments (e.g. a centrifuge) or laboratory personnel. As an example of the huge quantity of data produced by an activity we could mention the mass of raw data generated by a single DNA sequencing experiment, that is nowadays in the order of the Terabytes. Besides the data, all the procedural steps must be tracked. Returning to the DNA sequencing laboratory example, the protocol would probably require to execute also ”virtual” operations, as opposed to physical like converting the raw data into DNA sequences and subsequently assemble them by means of alignment algorithms.

In the typical protocol we find three kind of activities, depending on their performer:

- those performed by a physical device, like a liquid handler workstation (e.g. Biomek FX);
- those performed by a virtual device, like an assembling software;
- those performed by a human operator, like shaking a plate or taking a sample of DNA through a swab.

Starting from these considerations we defined a general notion of activity, called Action. An Action is defined by a name and a list of parameters. Each Action parameter is characterized by a name, a type and by the mode in which it is passed (read-only, read/write, write-only) (Figure 3.7). An Action is an atomic step in our execution model and could be seen as the simplest instruction that our MAS is able to interpret and execute.

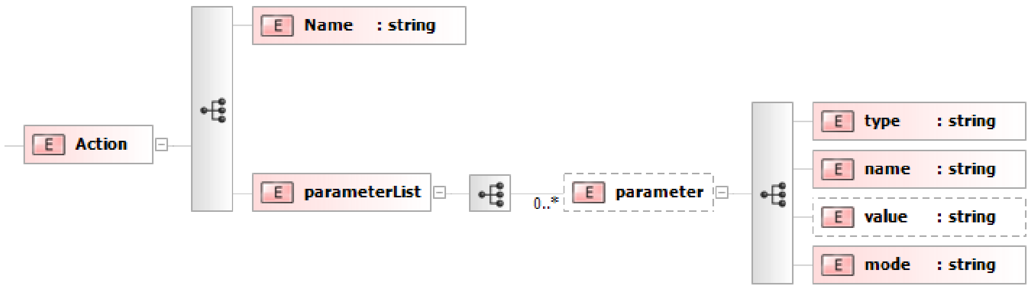


Figure 3.7: *XML schema for an Action.*

Referring again to the programming language metaphor we can assimilate an Action in our Executable model of a protocol to a single machine instruction in a machine code program. A single machine instruction can be directly executed by the processor. Our definition of Action is general enough to cover the three kind of activities above mentioned. Given the heterogeneity and the complexity of the laboratory environment, this solution represents a good trade-off between the need of describing a protocol with enough low granularity and the need of having a common interface for every activity involved.

Listing 3.1 shows the XML document describing a "centrifugate" Action. A centrifugate Action is defined by three parameters. The parameter named "performed", is of Boolean type and its mode is OUT, so that it is actually an output parameter of the action, representing whether the action has been successfully performed. The second and the third parameters are inputs of integer type representing respectively the g-force to be applied in the centrifugation and the centrifugation time.

Listing 3.1: XML document for a centrifugate action

```
<?xml version=\"1.0\" encoding=\"UTF-8\"?>
<Action>
  <Name>centrifugate></Name>
  <ontoTag>tagCentrifuge></ontoTag>
  <parameterList>
    <parameter>
      <type>boolean></type>
      <name>performed></name>
      <mode>OUT</mode>
    </parameter>
    <parameter>
      <type>int</type>
      <name>forceApplied</name>
      <mode>IN</mode>
    </parameter>
    <parameter>
      <type>int</type>
      <name>centrifugationTime</name>
      <mode>IN</mode>
    </parameter>
  </parameterList>
</Action>
```

It must be recalled that an Executable Action has a semantic counterpart in the Action concept, formally defined in the laboratory domain ontology of the BioCOW meta-model[80].

In the Executable model, a Protocol is an articulated flow of Actions. A Valid Protocol is a protocol that our runtime environment is able to interpret and to execute. A Protocol in the model could be composed using different Actions available in the runtime environment or loaded from external libraries. In XPDL a process is a structured composition of pieces of works, individually called Activities, which could be of various types [108]. In our system a Valid Protocol is an XPDL compliant model with some minor limitations and differences.

In order to guarantee the correct interpretation of a BioCOW protocol we require that every piece of work must be described by means of an Action concept. In this manner the BioCOW meta-model is semantically enriched to meet our needs. We also want to preserve at runtime the ontological constraints defined at design time. In XPDL the notion of "piece of work" is described by the concept of Activity. Hence we impose that every Activity is allowed to only invoke Actions. In order to satisfy this condition in the Executable model, we placed two restrictions on the XPDL meta-model.

The first one is to limit the types of Activity only to Route and SubFlow. The Route activity performs no work and simply supports routing decisions among the incoming transitions and/or among the outgoing transitions. The SubFlow activity enables the reuse of processes and could be usefully used to encapsulate parts of protocols in self-contained modules.

Second, we provide a specific SubFlow (ExecuteActionW) around an invocation of an Action. The ExecuteActionW SubFlow (Figure 3.8) simply invokes the execution of the Action and checks whether it is performed with or without errors. Actions can be executed only if they are encapsulated within that construct.

Using only Route and SubFlow activities and using the ExecuteActionW SubFlow we can therefore ensure that every piece of work is backed by an Action concept. Below we describe how we have built a MAS runtime system

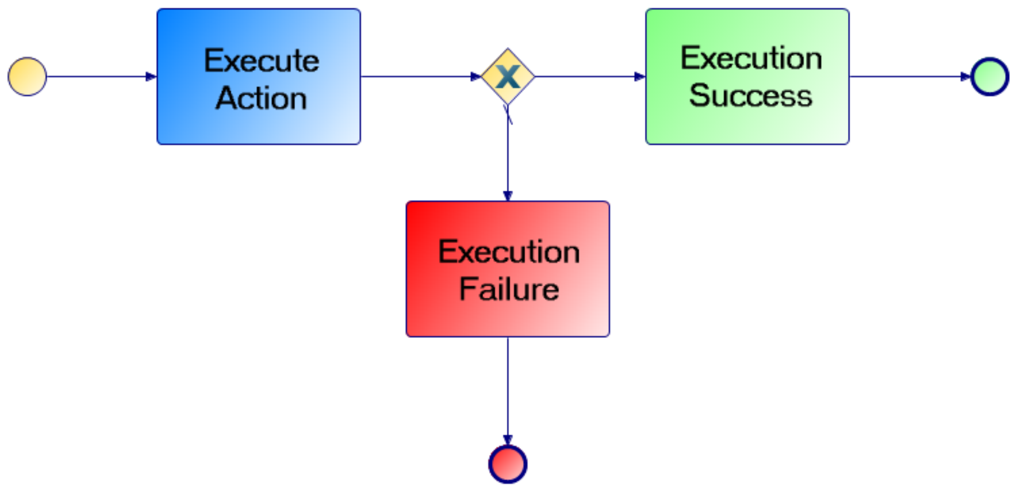


Figure 3.8: *The ExecuteActionW SubFlow that encapsulate the execution of actions.*

able to execute a Valid Protocol.

MAS Runtime Environment

The architecture of the MAS Runtime Environment is designed to closely resemble the laboratory environment, with the additional capability of being able to interpret and execute Actions as described in section 3.3.2. The Executable Model of a protocol involves one main kind of entity. These entities are heterogeneous and distributed resources that actually expose and, on request, perform Actions. We therefore dedicated one class of agent to these entities, the Device Agent (DA). Another distinctive characteristic of the Runtime Environment is an entity that does read an executable protocol and handles its execution. A Protocol Manager agent (PM) is appointed to control this aspect. A user interface agent (APE) is designed for loading new

Agent	Description
Agent Protocol Environment (APE)	allows the loading of new protocols
Protocol Manager (PM)	executes a protocol in the MAS
Device Agent (DA)	controls a resource, physical or virtual environment
Report Agent (RA)	user interface for mobile devices

Table 3.2: *Different classes of agents in our runtime platform.*

protocols in the MAS. A Reporter Agent (RA) is built specifically as a User Interface for mobile devices. Table 3.2 shows the different agents capabilities.

- **DA**: controls a resource (physical or virtual)
- **PM**: executes a protocol in the MAS
- **APE**: allows the loading of new protocols
- **RA**: user interface for mobile devices.

The RA agent is created at the boot of the system. For each resource in the laboratory environment that should be automatically managed from the LIMS, it is then created a DA Agent counterpart. One APE is also created in the boot phase, however two (or more) instances can co-exist without any problem. The same holds for the RA. A PM agent instead is created dynamically on user demand, and it is responsible for the execution of a particular protocol. On completion of the protocol, the PM agents automatically end its life cycle and is removed from the system.

Device Agent (DA)

We use a combination of a *Driver* and an Agent to make available in the MAS a service that can be executed by a physical or virtual resource. A *Driver* in our model actually handles the communication with a legacy resource such as a centrifuge or a robotised station. Since our model is inspired on the A&A model [90], our Driver is structured in terms of Actions. The similarity with the model lies in the fact that a Driver represents a resource in the MAS environment (Artifact in the A&A model). The main difference is that we strictly bind an instance of a Driver with exactly one instance of a DA. As a consequence, every request of Actions must be posted to a specific DA that acts as a proxy to the driver and therefore to the resource. A Resource exposes a set of Actions, one for every functionality. In the Centrifuge example the Centrifuge is the resource itself, and it is described by means of the functionalities it exposes and hence by a set of different Actions, i.e. the actions it can actually perform (e.g. ”centrifugate” or ”open lid”). A DA is responsible for executing the individual actions, therefore it needs to know how to physically communicate with the resource it encapsulate. The DA also needs to communicate with other agents so as to satisfy any incoming requests for its functionalities. We therefore structured a DA in two layers as depicted in Figure 3.9. The bottom layer is responsible for the communication with the resource using a specific driver. The top layer carries out the normal agent duties behaviour and social capabilities.

The scope of the bottom layer is to fetch and store metadata, using a resource specific driver. In the development of such a complex and heterogeneous system like a biological laboratory, the design of a new driver can become a hard bottleneck. Hence we spent some effort to simplify the process of driver creation. In our approach, a driver could be any piece of Java code. This choice enables the reuse of legacy code as well as direct interfacing with the instrument. The only added requirement for a developer is to declare which services the driver does expose. This is done via the Java annotation

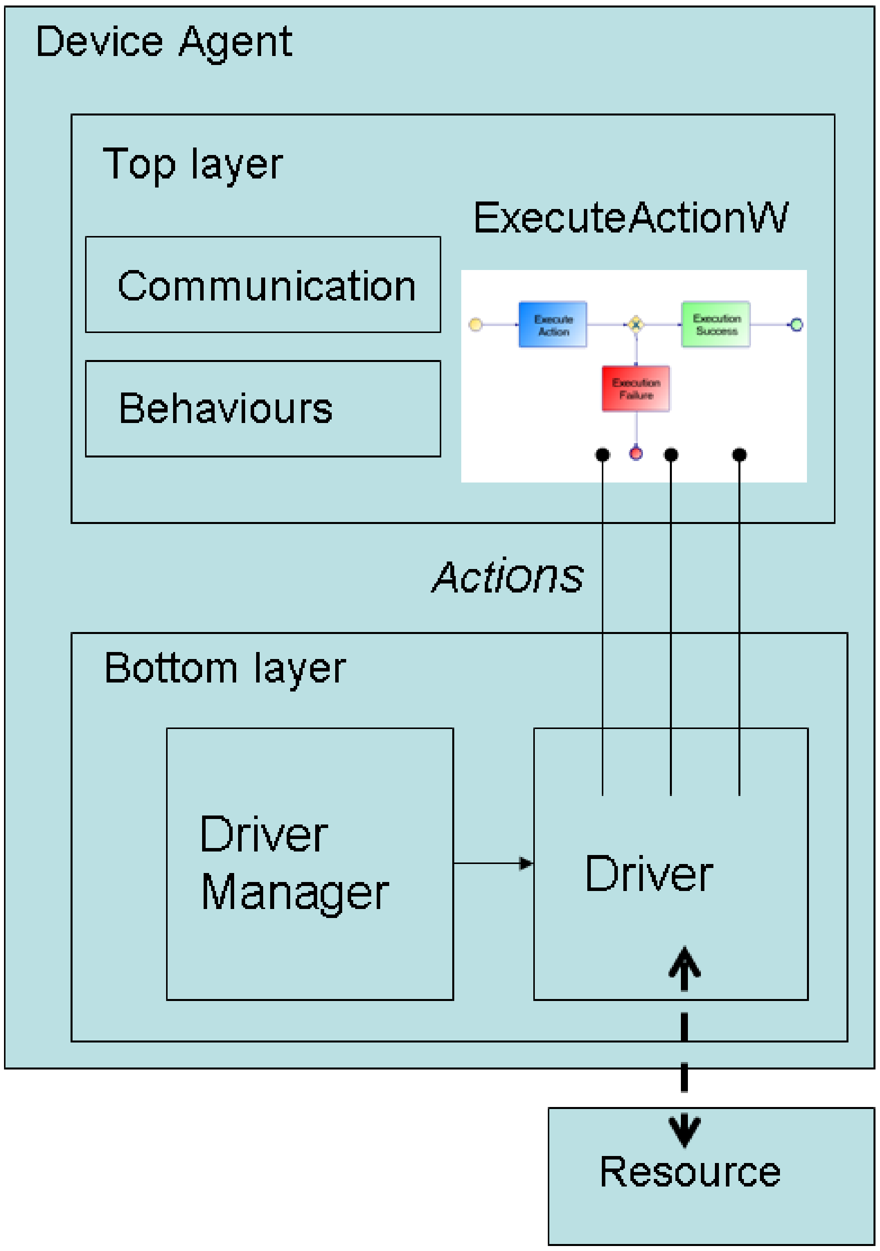


Figure 3.9: *Layers of the Device Agent.*

mechanism, which allows to add metadata to the code. We provide a set of annotations like @Action and @Par. Every method that is exposed as a first class entity in the system (Action) must be annotated with the @Action tag. In case of parameters, the @Par tag should be used. As illustrated in Listing 3.2 the method *centrifugate* is promoted to the level of an Action entity in the MAS. Two parameters are declared plus an extra one for the return value of the method. The XML document of Listing 3.1 is actually created from the annotated *centrifugate* Java method of Listing 3.2.

Listing 3.2: Example of a method annotated with an @Action tag

```
@Action(ontoTag = "tagCentrifuge", returnName = "performed")
public boolean centrifugate(
    @Par(name = "gforce", mode = Mode.IN) int rpm,
    @Par(name = "sec", mode = Mode.IN)int sec
){
//communication with the centrifuge
}
```

Using a driver manager the Device Agent is able to load and extract the metadata for a driver that fulfills these requirements. During the initialization phase the agent loads the driver, analyzes the metadata and creates a set of Action objects compliant to our model. The set of these objects provides the descriptions of the capabilities of the Device agent. In the last step of the initialization phase the DA register itself (with the exposed capabilities) in the MAS.

The top-layer of the device agent is responsible for the interaction with other agents in the MAS, responding to request for Actions. Its main capability is to execute the ExecuteActionW SubFlow (see Figure 3.8). If a PM agent wants to execute an action available on the interfaced device he should first retrieve the corresponding Action object querying the yellow pages. At that point, the PM should request to the DA to perform the ExecuteActionW SubFlow using as parameter the Action object and the actual parameters (if any) of the action. The DA then tries to execute the action, communicating with the resources by means of the driver. If any error occurs, the caller is notified. In case of no errors, the resulting output parameters are filled in the Action object and the caller is notified.

Protocol Manager agent (PM)

The Protocol Manager (PM) agent is responsible for the correct execution of a protocol. It incorporates the capabilities to execute a restricted XPDL protocol (according to section 3.3.2). Since the restriction imposed on XPDL in our Executable Model are minor, a normal WADE agent can be used without problems. In order to develop a protocol directly in the MAS system it is therefore possible to use the WOLF tool [35]. However, in the future, we intend to translate a protocol, structured in the BioCOW meta-model, *directly* in Java code fit for use in JADE/WADE. On the launch of a new protocol a new PM is created. The first step performed by a PM is to check whether the protocol can run on the current environment. The PM tries to verify the avalaibility of every Action used in the protocol before actually starting execution. Only if that control is successful then the execution of the protocol can take place. When the PM agent encounters an Action invocation, it first check, which DA is actually able to perform it. The answer depend on the actual state of the resource (the resource could be already in use or could be broke). The search is performed using the classic yellow pages system of JADE. Then, the agent delegates the execution of the ExecuteActionW SubFlow to the proper DA. The standard WADE mechanism used to enact distribuited workflow execution is applied. If multiple protocols require the same action, the requests are queued and acted upon by the DA. The requests are then served on a FIFO base. In the future, using a separate scheduler, more complex policies will be usable.

Agent Protocol Environment (APE)

The APE agent provides a user interface (UI) to laboratory operators in order to load new protocols. A protocol is enclosed in a package that contains three different categories of elements:

- main protocols as well as the sub-protocols used in them;
- local resources like images or spreadsheet files required by the activities of the protocols;
- specific external libraries to provide utility function like PDF documents generation.

APE loads a package and does visualize its content to the operator. It then extracts all the resources from the package and does deploy them into the runtime system. It is also responsible for creating a new PM agent and to charge it with the execution of the loaded protocol.

Reporter Agent (RA)

A Reporter Agent has been built specifically to handle requests from mobile devices that provide GUIs to laboratory operators. We currently support Android [2] based mobile devices using the peer-to-peer approach proposed in [123]. The RA is able to query the system and to provide information about the state of a sample processed in the laboratory. It interacts with the other agents of the runtime system and queries the database in order to determine detailed information like:

- the customer order that activates the laboratory analysis;

- the type of the biological analysis in which the sample is involved;
- the current phase of processing reached by the sample;
- the relationship with other samples produced in the laboratory for the same customer order.

Finally, after collecting all pieces of information, the RA is able to produce a report and to send it to the operator's GUI on their mobile device.

3.4 Results obtained for each element of the system

Our system consists of four parts:

1. A high-level language that draws both from a domain-specific ontology and a workflow metamodel;

2. A graphical editor built upon the above language;

3. A model-to-code compiler that translate the language written using the graphical editor in code suitable for the runtime platform;

4. A runtime platform;

The high-level language, BioCOW, is currently at a prototype stage. The mapping provided is sufficiently detailed to be used in a real-environment. However with the technology we used we were not able to implement the whole mapping in our technological stack. We therefore limited ourselves to map just a fraction of the whole ontology. In particular we mapped all the BFO ontology and only the main branches of the OBI ontology. That limit does not impair the methodology as the development of the tools used is currently active and there is a vivid interest in the community to explore such possibilities.

The graphical editor built using the Obeo designer was developed for demonstration purpose and not with a commercial intent. The current stage of the implementation is suitable for build valid BioCOW models. All the main concepts are mapped to graphical symbols. However it is not stable enough for industrial use. The procedure used to draw relies on the expertise of the developer. An end-user would not be able to effectively use the system. Even more the editor has not been tested and critical bugs should be expected.

The model-to-code compiler as well has been built for demonstration purpose only. The current stage of the implementation covers the main constructs and it is able to translate BioCOW models in valid packages for the execution on the runtime system. However it has not yet been used extensively and therefore some bugsmay still be encountered in actual use.

The runtime system has been validated and used in an industrial environment (BMR Genomics). It should be considered a beta release and the majority of the features are already implemented. It is stable for real world using and the development is currently active. It comprises API for building protocols, API for develop drivers as well as a small set of built-in drivers. It has been tested over the main operating system (Windows, Linux and Mac OS X).

Chapter 4

Evaluation

In this chapter we provide some elements for an evaluation of the work described in this dissertation. Since our system is divided in two layers (front-end and back-end) we need two different kind of expertise. Therefore, in collaboration with domain experts, we tried our prototype in two different use cases.

The front-end of our architecture is the layer that deals with the end-user. Hence, on the ground of the partnership with CRIBI[1], we engaged researchers from the field of molecular biology in using our specification language. Drawing from the researcher experience, and working directly with them, we used our tools to develop a real-world protocol. We then compared the produced protocol.

For the back-end part we worked in collaboration with BMR Genomics[2] a company involved in the sequencing fields that offers sequencing services for third parties. BMR services range from sequencing for researcher project to paternity tests for private individuals. During the course of the project we

[1]http://www.cribi.unipd.it/
[2]http://www.bmr-genomics.it/

deployed our back-end system in their business processes. Thus we had to directly relate our runtime system to their legacy protocols and environment.

Section 4.1 presents the front-end use case; Section 4.2 and 4.3 the back-end one.

4.1 Specification language

The experiment has been structured in sequential four phases:

1. Introduction to the expert of the intent of the system and its way of modeling
2. Choice of a sample protocol to implement in the system
3. Development of the model of the chosen protocol
4. Evaluation of the developed model against given criteria.

The outcomes expected of the experiment were of two categories.

- That our system is capable of facilitating the production of a well-formed protocol formalized
- That our language and method achieve good marks in the dimensions of expressivity, economicity and ambiguity.

In the first stage of the experiment we introduced the user to the ideas behind the model. We described to the laboratory expert the high-level architecture and the logic of the system, its advantages and its current limits. In particular, we presented the new approach to protocol formalization based on workflows, listing in a concise way all the constructs currently available in the visual modelling language (see Figure 3.5). We explained that we wanted to maximize the expressivity of the user specification language. In this "learning" phase we introduced three workflow patterns chosen among those described in [124]: Parallel Split; Synchronization; Structured Loop. These patterns do not contribute to the language expressivity but considerably reduce the user effort. Attaching a graphic symbol to each of these patterns adds to the economicity of the language (as does for example, the single word "rainbow" in contrast with the dictionary description "An arc of spectral colors, usually identified as red, orange, yellow, green, blue, indigo, and violet").

We asked the domain expert to write the protocol of interest using the proposed language constructs, first coding "by hand" in the graphical language of workflows. The design process was performed on paper without the support of any graphical editor. Since the graphical editor is built over the meta-model it does not add specific constructs and therefore expressivity. Hence the choice of performing the experimentation on paper, instead that directly on our graphical, does not have an impact on our experimentation. We decided to use that approach for constraints of time and to avoid the impediment caused by the technological non maturity of our editor.

Figure 4.1 depicts a representation of the protocol that had to be transferred into our platform. The figure shows that our meta-model is expressive enough to describe the operational parts of laboratory protocols.

Every block in the protocol represents an Action. The Action "Add Sample" for example adds a precise quantity (*vSample*) of the sample (*IDSample*) in the specified micro-centrifuge Eppendorf tube (*IDEppendorf*). Since a DataType could be backed also by an ontological term we used *deoxyribonucleic acids*[3] (DNA) for the sample (*IDSample*). An Action could

[3]Full id: http://purl.org/obo/owl/CHEBI#CHEBI_16991

have zero or more parameters, each of which could be backed by an appropriate ontological term.

In general-purpose workflow languages the end-user meaning is attached a posteriori to symbols and association. Conversely, basing the workflow meta-model on the domain specific ontology guarantees maximally consistent use of the modelling language offered to the end user. This occurs because every symbol of the workflow language now has an a priori attached meaning well anchored to the domain ontology.

In the protocol we can also see one of the three proposed patterns: the Structured Loop. The meaning here is to prepare *numberOfSamples* samples as specified by the *Prepare Eppendorf* action. The two conditions attached to the transitions specify when to either repeat or stop the action.

It is worth noting that in the first learning round the biologist produced a very specialized protocol instance with ad-hoc parameters and patterns. Obviously, the protocol in question was only capable of describing the contingent needs of that particular protocol. Once the expert learned how to use the language to its full potential, he became able to design a more general version of his own original protocol, therefore earning larger reuse potential This shows that we met the economicity goal because we can synthesize a single artifact which can express a whole range of protocols.

4.1.1 Evaluation

The resulting protocol (Figure 4.1) has shown that the proposed BioCOW meta-model enables the biologist to describe his protocols. In our experiment a biologist without previous knowledge of our system has been able to express a protocol routinely used in his experimentation. The learning curve proved to be fairly low. In a couple of learning cycles we saw the domain expert

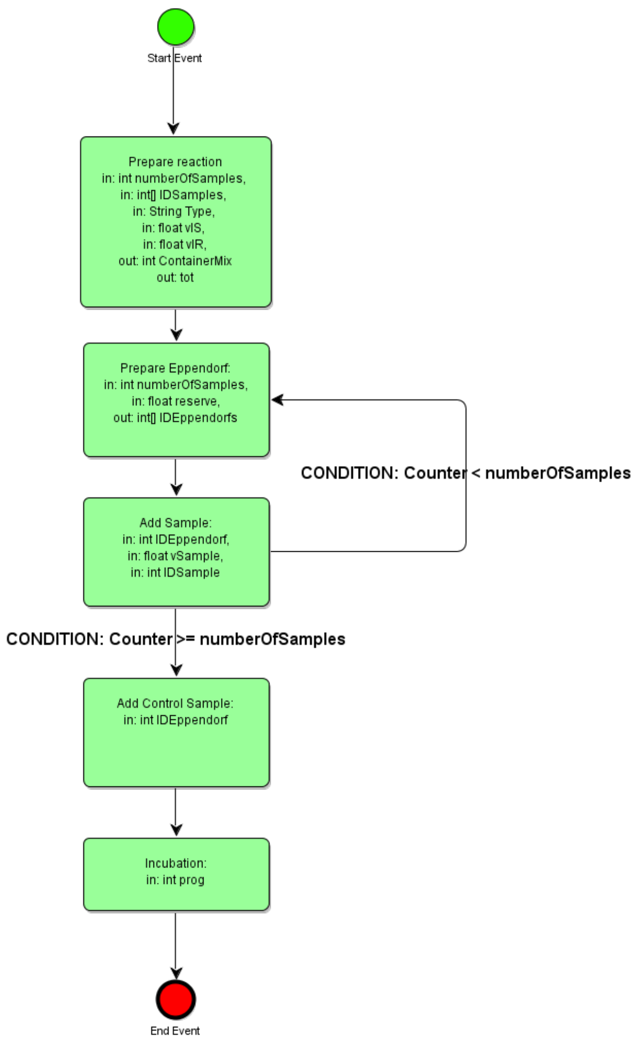


Figure 4.1: *Protocol*

become able to generalize a protocol producing a more general template for it. Thus we can safely say that the proposed BioCOW meta-model is sufficiently expressive for the envisioned needs.

The second dimension of evaluation, economicity, takes advantage of the produced template. The first version of the protocol was comparatively similar to a classic protocol description in terms of quantity of syntactic constructs used. The language proposed is quite simple and does not provides complex constructs able to describe composite behaviours in a concise way. However, constraining the domain expert to use just a small set of constructs has provided an unexpected result. The domain expert has naturally recognized and developed a common pattern and produced a template for a variety of protocol experiments. Hence the experiments also produced a valuable feedback in the economicity dimension.

For the last dimension, ambiguity, the ontology for the reason described above helps and drives the realization of a more precise protocols. However, since the validation has been made without the use of a graphical tool, we were not able to take full benefit from some technological enhancement such as for example autocompletion. Writing protocols by hands has been made by careful manual search of the correct ontologicals terms to use. This drawback could be obviously overcome using a graphical editor able to adequately interpret the BioCOW meta-model entities.

Another point of interest is the non-ambiguity of the interpretation of the model. Since the model needs to be translated in an executable form, no ambiguity is admissible in the interpretation of the constructs (e.g. how to execute actions and how to evaluate conditions to fire transitions). As our language is comparatively simple no ambiguous statements could be produced in the written model. To confirm its correctness has been checked manually by experts of the execution platforms.

How complex constraints expressed at design time could be preserved at runtime is currently under investigation. For example, a biologist could express a condition in which a sample needs to be processed under a specific temperature (e.g. a centrifugation at 4 degrees). An execution platform needs to ensure that this specific constraint will be met during the execution.

A laboratory experiment presents itself to the execution environment with a list of Actions that compose it. The execution of the experiment is then enacted if and only if all those Actions can be supported. At present, we accomplish this simply as a static acceptance check. In the future we want to augment this with the dynamic capability of handling contingencies. To this end, the autonomous nature of agents could be of great help.

4.2 Bioinformatic pipeline

In order to assess the validity of our platform we tested it against a set of bioinformatics analysis. We developed a pipeline used in our laboratory to analyse the raw data produced by the DNA sequencer. We describe here the pipeline and the development of that pipeline inside our runtime platform.

4.2.1 Pipeline: alignment of RNA sequences

The aim of the pipeline is to align RNA sequences (reads) of grapes against a reference genome and to produce an alignment, recorded in the *SAM* format. A *SAM* (Sequence Alignment/Map) format is a generic format for storing large nucleotide sequence alignments [78]. *SAMTools* is a suite of programs able to manipulate that format. We also want to visualize the data in a *GBrowse* [4] (a genome viewer). Input of a GBrowse is a *BAM* file that is a binary version of a SAM.

In order to produce a BAM file we need to align our sequences against a reference genome. In our pipeline we start using sequences produce by the SOLiD (Sequencing by Oligonucleotide Ligation and Detection) produced by *Applied Biosystems* (ABI). Data produced by the SOLiD sequencer is not directly saved in DNA sequences. Instead the *color space* is used [91].

[4]http://gmod.org/wiki/GBrowse

The color space sequences are usually saved in the *csfasta* (color space fasta) format. A quality file *qual* is provided as well in order to assess the quality of the sequence. Those two formats (*csfasta* and *qual*) are then combined in the format *fastq*. The *fastq* format contains both the sequence and the quality. In our pipeline we used a tool produced by our team (*csfasta2fastq*) able to merge those two formats.

We could then align the sequence against a reference genome. We used *Pass* as tool [37]. *Pass* is able to work directly in color space. Output of Pass is a SAM file that is subsequently sorted and converted in a binary file (BAM format).

4.2.2 Development

The following steps involved in the pipeline:

- csfasta2fasta
- Map reads
- sam2bam
- bam sorting
- indexing

Each step has as its counterpart a specific tool that could be launched by command line. We therefore develop a simple driver able to launch a process like a command line. A more interesting approach would be to developed a specific driver for every program. That approach would be more time consuming but it will enable the possibility to describe with more details the parameters of the single programs.

The resulting driver, *CLIBioinfoDriver*, has been developed in Java. The only *Action* exposed is *execute* that takes as parameter the command line to be executed and returns true in case of success or false otherwise.

We subsequently prepared an environment with the driver and developed our pipeline. In figure 4.2 we can see the resulting pipeline. Every box of the Pipeline is an actual invocation of the *execute Action* exposed by the driver.

csfasta2fasta This step merges a csfasta file with his quality file. It takes as input a csfasta and a qual files. A fastq file is returned as output.

Map reads This step maps the reads against a given genome. It takes as input a fastq file. Every reads is then aligned against the reference genome. A SAM file is given as output. Pass, the alignment tool, is able to work directly in color space. It must be noted that Pass is a resource consuming program. In the example tested we worked with a small subset of data (only two chromosomes instead of the whole genome). We tested using a laptop with 4 gigabytes of RAM. The amount of RAM was barely enough and the execution lasted around twelve hours. Usually, specific workstations are used for this kind of works. However, our platform is general enough to cope with this situation. We used a simple driver able to launch programs using the command line. In the case of the workstation it would be necessary to write just a different driver able to command the workstation. The remainder of the Pipeline would not need to be changed in that case.

in that case it would not be changed.

sam2bam Conversion of the SAM file in a BAM file. Since a SAM file is text using a binary conversion we are able to save around 30% of disk space. The conversion is carried out by the SAMTools.

bam sorting It creates a new file with the data sorted by chromosome position. The sorting is carried out by the SAMTools.

indexing It creates a index file with extension needed by the *GBrowse*.

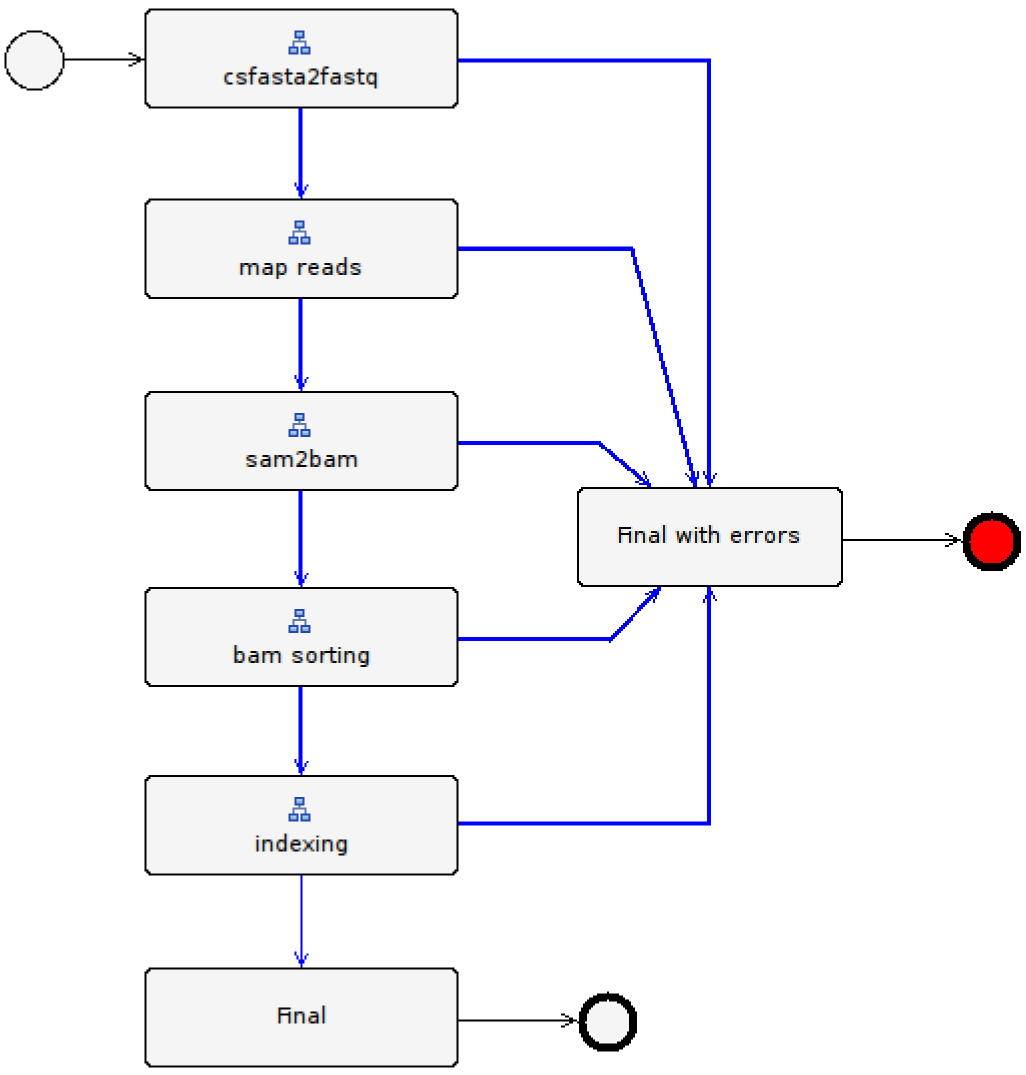


Figure 4.2: *Bioinformatic pipeline: RNA-seq.*

4.3 Demo case: paternity test

The Paternity Test aims at establishing if a man is the biological father of an individual. A customer, willing to perform the test, places an order through a website. Afterward, DNA samples belonging to the individual and to the supposed child are collected, usually by mean of buccal swabs, and sent to the analysis laboratory. When the material reaches the laboratory, some biological analysis can actually be executed, according to the protocol described in Figure 4.3. Through several sub-protocols, the samples are processed and, at every step, transformed into specific types of succeeding samples. In the final steps of the protocol, by DNA sequencing techniques, some data results are obtained. The DNA sequencing output is then used to compute the profile of the individuals involved in the specific test and finally a medical report that explain the results is produced by an expert.

Each action of the protocol is currently activated manually by a laboratory operator, following the workflow. In different phases of the process the operator is bounded to fill some digital resources and execute some bioinformatics analysis.

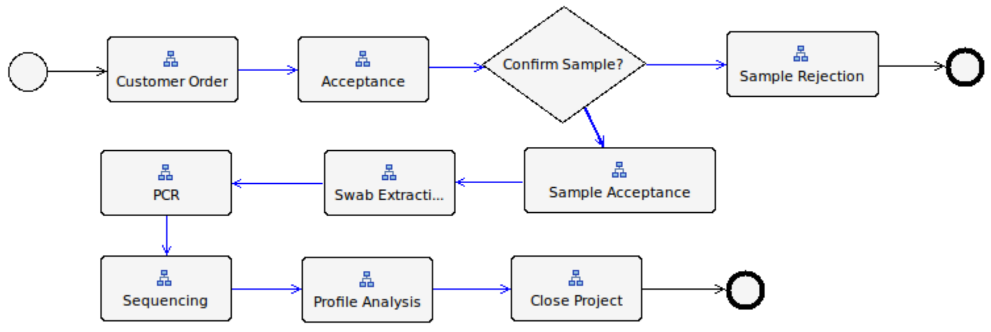


Figure 4.3: *The protocol formally describing the Paternity Test*

[The protocol formally describing the Paternity Test.]

In order to test the potential of our system the protocol described above has been formalized in the BioCOW meta-model. Every depicted activity block could represent either a normal SubFlow or an Action invocation by means of the ExecuteActionW SubFlow.

Figure 4.4 shows a subprotocol that describes the steps involved in the PCR SubFlow. In it we can see a use of the centrifugate action of Listing 3.1. In the PCRCycle Action the DNA material is amplified by means of the Polymerase Chain Reaction so that its quantity becomes sufficient for the following steps of the analysis. It can be noticed that the protocols does include not only the physical processing of samples but also the management of the produced data and of the history of the sample (e.g. by mean of the DBReg Action, that interacts with a database). Doing so make it possible to support existing legacy systems without changing their structure.

It is worth underlining that since the PCR sub-protocol is self-contained in the SubFlow is possible to reuse it in other contexts without writing a single line of code. This drastically reduces the time needed for the implementation of new protocols.

Using the proposed approach an explicit knowledge of the concepts involved in the protocol exists in the system. The MAS is therefore able to interpret this knowledge and to act correctly depending on the real environment. In the case study of the paternity test only the tracking activities have been totally automated. The operator is therefore notified when he can start the physical steps, to be executed from a device. Nevertheless, with proper drivers and proper hardware, physical actions could be automated . The system notifies the next steps to be performed. In the case study the operator is notified to perform a PCR on some specific samples. After the sequencing phase an automatically analysis is performed and the results are delivered to the laboratory operator.

In our test case a total of 31 activities were included to define the paternity protocol (included the sub-protocol *SwabExtraction, PCR, Sequencing and Analysis*). Using our model we automated 12 of those activities. Once automated these activities become transparent to the end user and they could be also easily reused in other protocols with minimal effort.

With respect to the initial requirement of traceability, automation and integration our test case shows promising results. The requirement of traceability is easily guaranteed, and all the related - and heavy - duties are now transparent to the end user.

The second requirement of automation is met. In our test case only some activities have been automated. The bottleneck is the legacy environment and the development of the drivers. However, also without producing drivers for the specific hardware, we automated 12 of the 31 initial activities (38%).

The last requirement is met under the constraint to produce specific drivers for the specific devices used in the laboratory.

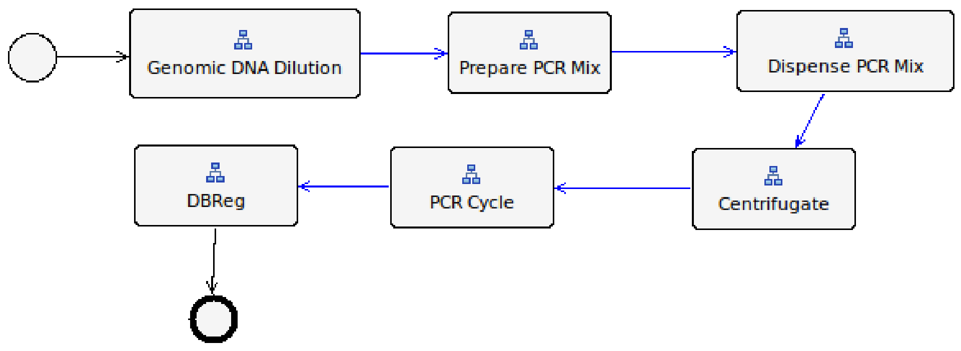


Figure 4.4: *The PCR subprotocol of the paternity protocol.*

Chapter 5

Discussion and conclusion

In 1977, a team lead by Frederick Sanger sequenced the first DNA-based genome (Phage F-X174). The sequence, only 5 thousand nucleotide bases, opened a new era for natural sciences. In 2000, after ten years of effort, two independent projects announced the sequencing of the human genome. A big technical and methodological leap was needed to sequence all the 3 billions of bases, 6 orders of magnitude larger than the first sequence of 5.000 nucleotides. The last 10 years have also seen a surge of very solid interest from both the scientific and the industrial communities in the new emerging opportunities. The second generation of sequencing machine, 454, Solid, Illumina, significantly dropped the cost for base pair (bp). A third generation is expected in few a years from now. In the meanwhile, experiments have been made to directly read the strand of DNA, the success of which will constitute the forth generation solution.

However this increasing productivity inevitably entails a rise in complexity. New challenges, both for the data produced and the laboratory protocols (commonly referred to as procedures) used, arise from that progress. The raw data produced by the laboratory equipment have to be refined so as to be post-processed, which poses the so-called assembly problem. The procedures, which change very rapidly to keep pace with the progress of the sequencing techniques, must be understood by domain experts. In addition to producing potentially huge volumes of data, the procedures themselves must be aware of the progress of execution, for, general intelligence as well as contingency. The economic value of their execution, in terms of both resource efficiency and product efficacy obviously benefits from a controlled and stable quality of execution. This goal is best achieved by automation. However the current state of the art offers no standards to this effect. Moreover, since the sequencing technology is moving fast a solution based on a monolithic infrastructure (commonly referred to as information management system, IMS) is the worst possible answer to those needs. In thinking of a fitting solution we must also bear in mind that, at present, IMS are most often directly realized by the biologists themselves.

Biomedical analyses are becoming increasingly complex, with respect to both the type of the data to be produced and the procedures to be executed. This trend is expected to continue in the future. The development of information and protocol management systems that can sustain this challenge is therefore becoming an essential enabling factor for all actors in the field. The use of custom-built solutions that require the biology domain expert to acquire or procure software engineering expertise in the development of the laboratory infrastructure is not fully satisfactory because it incurs undesirable mutual knowledge dependencies between the two camps. We propose instead an infrastructure concept that enables the domain experts to express laboratory protocols using proper domain knowledge, free from the incidence and mediation of the software implementation artefacts. In the system that we propose this is made possible by basing the modelling language on an authoritative domain specific ontology and then using modern model-driven architecture technology to transform the user models in software artefacts ready for execution in a multi-agent based execution platform specialized for biomedical laboratories.

In this thesis we proposed an architecture that aims to close the gap between the biologist's IMS concept and the software engineering knowledge and technology involved in automating the execution of laboratory protocols. To this end, inspired on the Model-Driven Architecture paradigm, we developed a software framework that enables the biologist to directly plan her protocols without the need to draw from software engineering knowledge and technology, including programming languages, compilers, interpreters, and the like.

The architecture of the solution we proposed comprises the two main layers. A front-end layer with:

- A high-level language as close to the experience and the needs of the biologist as possible;
- A graphical editor capable of enabling the user to graphically specify the desired protocols with the provided high-level language;

A back-end layer with

- A model-to-code compiler able to translate the user protocol in code fit for the target execution platform;
- A run-time environment that understands an executable version of the protocol produced by the compiler and is able to relate the constraints and needs expressed in the protocol to the actual capabilities of the environment and to accordingly execute multiple protocols in parallel.

BioCOW, our high-level language, describes the operational perspective of a laboratory protocol using a workflow metaphor expressed in XPDL, a markup language created to ensure interoperability among different workflow management tools in order to handle workflow processes. To make it better fit our purpose we enriched the XPDL meta-model with concepts drawn from an ontology specialized in biomedical investigations, OBI. OBI addresses the need for controlled vocabularies not only for the experimental data annotation but also for the representation of investigations in the Biological and Biomedical Sciences. We then used the Eclipse Modeling Framework (EMF) to integrate OBI ontology and the XPDL schema, from which we obtained a new meta-model (nicknamed BioCOW after Biology Combined Ontology [and] Workflow). BioCOW make it possible to model workflows in terms of objects and actions specific of our target domain. The mapping provided is detailed enough to be used in a real-environment. We tested it against three real use cases drawn from academic and industrial scenarios. However, the technological stack used is not mature enough to implement the whole mapping between OBI and our metamodel. Therefore we implemented a partial view of the map. We mapped all BFO ontology but only the main branches of the OBI ontology.

The graphical editor has been built for demonstration only. With it we were able to build valid BioCOW models. The relevant domain-specific protocol concepts are mapped to graphical symbols. The model-to-code compiler as well has been built for demonstration purpose. The current stage is able to translate valid BioCOW models in executable code for the runtime system.

The runtime system has been validated and used in industrial (BMR Genomics) and academic (CRIBI) environments. The majority of the features are implemented and tested. It comprises a set of API for building protocols and develop drivers. A growing set of drivers is currently being in development.

To assess our platform we tested it in an incremental way. We worked with three different use cases. With the the first use case we engaged a researcher to write a protocol using our metamodel (see section 4.1). The protocol used was a typical *wet-lab* protocol. The evidence collected suggests that the language covers the needs of the researcher. The researcher was able to express a protocol of choice with our language. This validation covers the front-end part of our architecture.

We tested our platform also against a bioinformatic protocol (pipeline) to analyse data, in contrast with the first use case in which the protocol produced data (see 4.2. In this test we implemented the pipeline directly on the runtime platform, hence using only the workflow view of our metamodel. We then executed the protocol describing the pipeline in our runtime platform. With this test case we validated our interpretation of what is a protocol inside a biological laboratory. Our architecture proved flexible and capable to deal with a set of new operations (bioinformatics tools) requiring only the development of a single extra driver. The tests were carried out in an academic (CRIBI) environment using real data.

Finally we tested in an industrial environment the runtime platform (see 4.3). We developed a protocol used as paternity validation. The protocol developed is a real protocol currently used at the BMR Genomics. We developed the drivers necessary to interface our system with the specific laboratory environment. Our approach proved to be flexible enough to cope with a different set of use case in a robust way. Since our platform relies on a simple definition of drivers we were able to easily incorporate legacy systems in use at BMR. In this way is possible to develop a graceful transition from the legacy system to our system. BMR Genomics is currently upgrading his systems using our platform.

Our long term vision depicted a picture in which a biologist is able to concentrate on the research goal rather than on side details. With the work of this thesis we wanted to enable the first steps of this long term vision. We developed a metamodel that draw directly from an domain ontology. Using this metamodel a researcher is able to develop protocols and to execute them in an automated way. However the work on this field is not completed yet. We can see some directions of work that need to be pursued in the future.

The first is a technological one. Our work proved that a similar approach is feasible but with some drawbacks. The major limitation is the initial amount of work that must be done to provide a minimal working tool. Another drawback is the current state of the tools able to deal with ontology and metamodels. This field is comparatively new and the tools are insufficient. Further work would be required to complete the mapping provided from a technological point of view. The current maturity of the tools is not mature enough to translate real world ontology (written in OWL) inside the Eclipse EMF framework..

A second axis is about the reasoning that we want to enable. In order to provide a system able to make automatic reasoning and inferences we need the domain knowledge and the knowledge in a structured way. In our domain the required knowledge is already expressed in a formalized way (OBI).

Appendix A

List of scientific publications

The author contributed to the following scientific publications during the timeframe of his PhD:

1. A. Maccagnan, M. Riva, E. Feltrin, B. Simionati, Tullio Vardanega, Giorgio Valle, Nicola Cannata: "Combining ontologies and workflows to design formal protocols for biological laboratories". *Automated Experimentation*, Volume 2, No. 3, April 2010 [80].

2. A. Maccagnan, T. Vardanega, E. Feltrin, G. Valle, M. Riva, and N. Cannata: "A multi-agent system for the automated handling of experimental protocols in biological laboratories". *In Proc. of the 11th WOA 2010 Workshop, Dagli Oggetti Agli Agenti*, September 2010 [81].

3. A. Maccagnan, N. Cannata, G. Valle and T. Vardanega: "Mapping OBI and XPDL to a MDE framework for laboratory information processing". To appear *in proc. of the Fourth International Conference on Information, Process, and Knowledge Management*, February 2012 [82].

Appendix B

Example of drivers

B.1 Command line driver

Listing B.1 shows a simple driver to launch process using the command line. The actual code that performs the action starts at line 27. We can see the method "execute" that is annotate with *Action*. Using this strategy the system is able to recognize the method and extract and XML describing it. Listing B.2 shows the XML conform with the Action schema (see Figure 3.7) for the only method annotated "execute". Must be noted that only on the methods annotated with *Action* are analysed. Such XML is subsequently used to invoke the corresponding action during the protocol execution.

Listing B.1: Command line driver.

```
  package org.farm.drivers.clibioinfo;

3 import java.io.BufferedReader;
  import java.io.InputStreamReader;

  import org.farm.driverutils.FarmDriver;
  import org.farm.driverutils.annotations.Action;
  import org.farm.driverutils.annotations.Par;
  import org.farm.driverutils.annotations.Par.Mode;
  import org.farm.driverutils.exceptions.FarmDeInitException;
  import org.farm.driverutils.exceptions.FarmInitException;
  import org.farm.driverutils.exceptions.FarmTestException;

  public class CLIBioinfoDriver extends FarmDriver {

    @Override
    public void init() throws FarmInitException {}

    @Override
    public void deInit() throws FarmDeInitException {}

    @Override
    public boolean test() throws FarmTestException {
      return true;
    }

    @Action(ontoTag = "CLIBioinfo", returnName = "Success")
    public static boolean execute(
        @Par(name = "CLI", mode = Mode.IN) String cli) {

      Runtime rt = Runtime.getRuntime();
      Process pr;
      try {
        pr = rt.exec(cli);
        BufferedReader input = new BufferedReader(new
           InputStreamReader(pr.getInputStream()));
        String line=null;
        while((line=input.readLine()) != null) {
          System.out.println(line);
        }
        int exitVal = pr.waitFor();
        if (exitVal >= 0){
          return true;
        }
      } catch (Exception e) {
        return false;
      }
      return false;
    }
  }
```

Listing B.2: The resulting "Action" of the "execute" method.

```
<?xml version=\"1.0\" encoding=\"UTF-8\"?>
<Action>
  <Name>execute</Name>
  <ontoTag>CLIBioinfo</ontoTag>
  <parameterList>
    <parameter>
      <type>boolean</type>
      <name>Success</name>
      <mode>OUT</mode>
    </parameter>
    <parameter>
      <type>class java.lang.String</type>
      <name>CLI</name>
      <value>DUMMY-VALUE</value>
      <mode>IN</mode>
    </parameter>
  </parameterList>
</Action>
```

B.2 BiomexNX Driver

Listing B.3 shows a more elaborated driver. A Biomek NX is a liquid dispenser able to handle nanoliters. *Beckman Coulter Inc.*[1] is the industrial vendor. A Biomek NX is controlled using the proprietary software provided by Beckman developed in Visual Basic. We used a bridge library (Com4j[2]) in order to use the proprietary software in Java. Due to limitation of such library we developed a client-server driver in order to run the bridge in a different Java Virtual Machine. The code listed shows the client side of our driver.

Listing B.3: Biomek NX Driver.

```
package org.farm.drivers.robots.biomekNX;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.net.Socket;
import java.net.UnknownHostException;
import javax.swing.JOptionPane;

import org.farm.driverutils.FarmDriver;
import org.farm.driverutils.PropertiesManager;
import org.farm.driverutils.annotations.Action;
import org.farm.driverutils.annotations.Par;
import org.farm.driverutils.annotations.Par.Mode;
import org.farm.driverutils.exceptions.FarmDeInitException;
import org.farm.driverutils.exceptions.FarmInitException;
```

[1]https://www.beckmancoulter.com
[2]http://com4j.java.net/

```
import org.farm.driverutils.exceptions.FarmTestException;
import org.farm.driverutils.exceptions.LoadDriverException;


public class BiomekNXDriver extends FarmDriver {

  private static final long serialVersionUID =
      -4326807377835319383L;
  private static final String SERVICE_ADDRESS = "localhost";
  private static final int SERVICE_PORT = 14189;
  private static final int SLEEP_TIME = 1500;
  private static String configFile = "drivers/BiomekNXDriver/
      BiomekNXDriver.conf";
  private BufferedReader in = null;
  private PrintStream out = null;
  private Socket socket = null;
  private boolean connectedToMachine;

  @Override
  public void init() throws FarmInitException {
    System.out.println("initializing BiomekNX...");
    PropertiesManager pManager;
    try {
      pManager = new PropertiesManager(configFile.toString());
      String connectedValue = pManager.getProperty("connected
         ");
      if (connectedValue.compareTo("false") == 0){
        connectedToMachine = false;
      } else {
        connectedToMachine = true;
      }
    } catch (IOException e) {
      throw new FarmInitException();
    }
  }

  @Override
  public void deInit() throws FarmDeInitException {
    System.out.println("deinitializing BiomekNX...");
  }



  private void startService() throws Exception {
    try {
      Runtime.getRuntime().exec("cmd /c start drivers\\
         BiomekService\\startService.bat");
      Thread.sleep(SLEEP_TIME);
    } catch (InterruptedException e1) {
      e1.printStackTrace();
      throw new Exception(e1);
    } catch (IOException e2) {
      e2.printStackTrace();
      throw new Exception(e2);
    }
  }

  @Override
  public boolean test() throws FarmTestException {
    if (!connectedToMachine) {
      return true;
    }
    boolean result = false;
    try {
      startService();
    } catch (Exception e1) {
      return false;
    }
    try {
      socket = new Socket(SERVICE_ADDRESS, SERVICE_PORT);
      in = new BufferedReader(
          new InputStreamReader(socket.getInputStream()));
      out = new PrintStream(socket.getOutputStream(), true);
      out.println("isDeviceUp");
      String response = "";
      while(response.compareTo("completed")!= 0) {
        response = in.readLine();
        if (response.compareTo("completed")!= 0) {
          if (response.compareTo("deviceUp") == 0) {
            result = true;
          }
        }
      }
      out.close();
      in.close();
    } catch (UnknownHostException e) {
      e.printStackTrace();
      result = false;
    } catch (IOException e) {
      e.printStackTrace();
      result = false;
    }
    return result;
  }

  @Action(ontoTag = "executeOperationNX", returnName = "void")
  public void executeOperationNX (
      @Par(name = "operationName", mode = Mode.IN) String
          operationName) throws Exception {
    if (!connectedToMachine) {
      return;
    }
    JOptionPane.showConfirmDialog(null, "Vuoi procedere con l’
        esecuzione?","",JOptionPane.INFORMATION_MESSAGE);
    startService();
    try {
      socket = new Socket(SERVICE_ADDRESS, SERVICE_PORT);
      in = new BufferedReader(
          new InputStreamReader(socket.getInputStream()));
      out = new PrintStream(socket.getOutputStream(), true);
      out.println(operationName);
      String response = "";
      while(response.compareTo("completed")!= 0) {
        response = in.readLine();
        if (response.compareTo("completed")!= 0) {
          System.out.println(response);
        }
      }
      out.close();
      in.close();
    } catch (UnknownHostException e) {
      e.printStackTrace();
      throw new Exception(e);
    } catch (IOException e) {
      e.printStackTrace();
      throw new Exception(e);
    }
  }

  @Action(ontoTag = "plateTransferNX", returnName = "void")
  public void plateTransferNX() throws Exception {
    if (!connectedToMachine) {
      return;
    }
    JOptionPane.showConfirmDialog(null, "Vuoi procedere con l'
       esecuzione?","",JOptionPane.INFORMATION_MESSAGE);
    startService();
    try {
      socket = new Socket(SERVICE_ADDRESS, SERVICE_PORT);
      in = new BufferedReader(
          new InputStreamReader(socket.getInputStream()));
      out = new PrintStream(socket.getOutputStream(), true);
      out.println("Rack_tubini96-384_demo_LIMS");
      String response = "";
      while(response.compareTo("completed")!= 0) {
        response = in.readLine();
        if (response.compareTo("completed")!= 0) {
          System.out.println(response);
        }
      }
      out.close();
      in.close();
    } catch (UnknownHostException e) {
      e.printStackTrace();
      throw new Exception(e);
    } catch (IOException e) {
      e.printStackTrace();
      throw new Exception(e);
    }
  }

  @Action(ontoTag = "ethanolBSDispensationNX", returnName = "
     void")
  public void ethanolBSDispensationNX() throws Exception {
    if (!connectedToMachine) {
      return;
    }
    JOptionPane.showConfirmDialog(null, "Vuoi procedere con l'
       esecuzione?","",JOptionPane.INFORMATION_MESSAGE);
    startService();
    try {
      socket = new Socket(SERVICE_ADDRESS, SERVICE_PORT);
      in = new BufferedReader(
          new InputStreamReader(socket.getInputStream()));
      out = new PrintStream(socket.getOutputStream(), true);
      out.println("etanolo_blu_sali_demo_LIMS");
      String response = "";
      while(response.compareTo("completed")!= 0) {
        response = in.readLine();
        if (response.compareTo("completed")!= 0) {
          System.out.println(response);
        }
      }
      out.close();
      in.close();
    } catch (UnknownHostException e) {
      e.printStackTrace();
      throw new Exception(e);
    } catch (IOException e) {
      e.printStackTrace();
      throw new Exception(e);
    }
  }

  @Action(ontoTag = "ethanol70DispensationNX", returnName = "
     void")
  public void ethanol70DispensationNX() throws Exception {
    if (!connectedToMachine) {
      return;
    }
    JOptionPane.showConfirmDialog(null, "Vuoi procedere con l'
        esecuzione?","",JOptionPane.INFORMATION_MESSAGE);
    startService();
    try {
      socket = new Socket(SERVICE_ADDRESS, SERVICE_PORT);
      in = new BufferedReader(
          new InputStreamReader(socket.getInputStream()));
      out = new PrintStream(socket.getOutputStream(), true);
      out.println("etanolo70_demo_LIMS");
      String response = "";
      while(response.compareTo("completed")!= 0) {
        response = in.readLine();
        if (response.compareTo("completed")!= 0) {
          System.out.println(response);
        }
      }
      out.close();
      in.close();
    } catch (UnknownHostException e) {
      e.printStackTrace();
      throw new Exception(e);
    } catch (IOException e) {
      e.printStackTrace();
      throw new Exception(e);
    }
  }
}
```

Appendix C

List of abbreviations

COTS	Commercial-off-the-shelf
LIMS	Laboratory Information Management System
BFO	Basic Formal Ontology
IFOMIS	Institute for Formal Ontology and Medical Information Science
OBO	Open Biomedical Ontologies
NCBO	National Center for Biomedical Ontology
OBI	Ontology for Biomedical Investigations
OWL	Web Ontology Language
RO	Relation Ontology
NCBI	National Center for Biotechnology Information
IAO	Information Artifact Ontology
MDE	Model-driven engineering

BPMN Business Process Modelling Notation

XPDL XML Process Definition Language

MAS Multi-agent system

MDA Model-driven architecture

MOF Meta-Object Facility

OMG Object Management Group

BPM Business Process Management

MAS Multi-agent system

WADE Workflow and Agent Development Environment

CRIBI Centro ricerche interdipartimentale biotecnologie innovative

Bibliography

[1] Gene ontology. URL http://www.geneontology.org/.

[2] Android platform. URL http://code.google.com/android.

[3] Bpmi. URL www.bpmi.org.

[4] Journal of visualized experiments. URL http://www.jove.com.

[5] Jawe based process editor. URL http://www.jped.org.

[6] Minimum information about a microarray experiment. URL http://www.mged.org/Workgroups/MIAME/miame.html.

[7] Metabolomics standards initiative. URL http://msi-ontology.sourceforge.net/.

[8] myexperiment. URL http://www.myexperiment.org/.

[9] Nature methods, . URL http://www.nature.com/nmeth/index.html.

[10] Nature procotols, . URL http://www.nature.com/nprot/index.html.

[11] The object management group. URL www.omg.org.

[12] Proteomics identifications database. URL http://www.ebi.ac.uk/pride/.

[13] Protocol-online. URL http://www.protocol-online.org.

[14] Science advisory board. URL http://www.scienceboard.net/.

[15] Workflow management coalition. URL http://www.wfmc.org/.

[16] Xml process definition language. URL http://www.wfmc.org/xpdl.html.

[17] Steering the future of computing. *Nature*, 440(7083):383, March 2006. doi: 10.1038/440383a. URL http://dx.doi.org/10.1038/440383a.

[18] Abran, Alain, Bourque, Pierre, Dupuis, Robert, Moore, James W., and Tripp, Leonard L. *Guide to the Software Engineering Body of Knowledge - SWEBOK*. IEEE Press, Piscataway, NJ, USA, 2004 version edition, 2004. ISBN 0769510000. URL http://www.swebok.org/ironman/pdf/SWEBOK_Guide_2004.pdf.

[19] Richard Arndt, Raphaël Troncy, Steffen Staab, Lynda Hardman, and Miroslav Vacura. Comm: Designing a well-founded multimedia ontology for the web. In *ISWC/ASWC*, pages 30–43, xx 2007. URL http://10.1007/978-3-540-76298-0. 10.1007/978-3-540-76298-0_3.

[20] Robert Arp and Barry Smith. Function, role, and disposition in basic formal ontology. In *Available from Nature Precedings*, page xx, 2008. URL http://hdl.handle.net/10101/npre.2008.1941.1.

[21] Uwe Assmann, Steffen Zschaler, and Gerd Wagner. Ontologies, Metamodels, and the Model-Driven Paradigm. *Ontologies for Software Engineering and Software Technology*, pages 249–273, 2006. doi: 10.1007/3-540-34518-3_9.

[22] George Avery, Charles McGee, and Stan Falk. Product review: Implementing lims: A how-to guide. *Analytical Chemistry*, 72(1):57 A–62 A, 2000. doi: 10.1021/ac0027082.

[23] Pierre Baldi, G. Wesley Hatfield, and Wesley G. Hatfield. *DNA Microarrays and Gene Expression: From Experiments to Data Analysis and Modeling*. Cambridge University Press, 1 edition, September 2002. ISBN 0521800226. URL http://www.worldcat.org/isbn/0521800226.

[24] Jonathan Bard and Seung Rhee. Ontologies in biology: design, applications and future challenges. *Nature reviews. Genetics*, 5:213–222, Mar 2004. ISSN 1471-0056. URL http://www.nature.com/nrg/journal/v5/n3/abs/nrg1295.html. 10.1038/nrg1295.

[25] Ezio Bartocci, Flavio Corradini, and Emanuela Merelli. Enacting proactive workflows engine in e-science. In *International Conference on Computational Science (3)*, pages 1012–1015, 2006.

[26] Fabio L. Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing Multi-Agent Systems with JADE (Wiley Series in Agent Technology)*. Wiley, April 2007. ISBN 0470057475.

[27] F. Bergenti, G. Caire, D. Gotta, D. Long, and G. Sacchi. Enacting bpm-oriented workflows with wade. In *Atti del 12 Workshop dagli Oggetti*

agli Agenti (WOA) Progettazione ed analisi di sistemi complessi mediante modellazione e simulazione basate su agenti, Calabria (Italia), 2011.

[28] H.C Birnboim and J Doly. A rapid alkaline extraction procedure for screening recombinant plasmid dna. *Nucleic acids research*, 7(6):1513, 1979.

[29] Olivier Bodenreider and Robert Stevens. Bio-ontologies: current trends and future directions. *Briefings in bioinformatics*, 7:256–274, Sep 2006. URL http://dx.doi.org/10.1093/bib/bbl027. 10.1093/bib/bbl027.

[30] Matteo Bordin and Tullio Vardanega. Correctness by construction for high-integrity real-time systems: a metamodel-driven approach. In *Ada-Europe'07: Proceedings of the 12th international conference on Reliable software technologies*, pages 114–127, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-73229-7.

[31] Christopher Brewster, Kieron O'Hara, Steve Fuller, Yorick Wilks, Enrico Franconi, Mark Musen, Jeremy Ellman, and Simon Shum. Knowledge representation with ontologies: The present and future. *IEEE Intelligent Systems*, 19:72–81, Jan 2004. URL http://www2.computer.org/portal/web/csdl/doi/10.1109/MIS.2004.1265889. 10.1109/MIS.2004.1265889.

[32] Ryan Brinkman, Melanie Courtot, Dirk Derom, Jennifer Fostel, Yongqun He, Phillip Lord, James Malone, Helen Parkinson, Bjoern Peters, Philippe Rocca-Serra, Alan Ruttenberg, Susanna-Assunta Sansone, Larisa Soldatova, Christian Stoeckert, Jessica Turner, Jie

Zheng, and the OBI consortium. Modeling biomedical experimental processes with obi. *Journal of Biomedical Semantics*, 1(Suppl 1): S7, 2010. ISSN 2041-1480. doi: 10.1186/2041-1480-1-S1-S7. URL http://www.jbiomedsem.com/content/1/S1/S7.

[33] Paul A. Buhler and José M. Vidal. Towards adaptive workflow enactment using multiagent systems. *Inf. Technol. and Management*, 6 (1):61–87, 2005. ISSN 1385-951X. doi: http://dx.doi.org/10.1007/s10799-004-7775-2.

[34] Jean Bzivin, Vladan Devedzic, Dragan Djuric, Jean-Marie Favreau, Dragan Gasevic, and Frederic Jouault. An m3-neutral infrastructure for bridging model engineering and ontology engineering. In Dimitri Konstantas, Jean-Paul Bourrires, Michel Lonard, and Nacer Boudjlida, editors, *Interoperability of Enterprise Software and Applications*, pages 159–171. Springer London, 2006. ISBN 978-1-84628-152-5. URL http://dx.doi.org/10.1007/1-84628-152-0_15. 10.1007/1-84628-152-0_15.

[35] G. Caire, M. Porta, E. Quarantotto, and G. Sacchi. Wolf - an eclipse plug-in for wade. In *WETICE '08: Proceedings of the 2008 IEEE 17th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 26–32, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3315-5. doi: http://dx.doi.org/10.1109/WETICE.2008.57.

[36] Giovanni Caire, Danilo Gotta, and Massimo Banzi. Wade: a software platform to develop mission critical applications exploiting agents and workflows. In *AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, pages

29–36, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.

[37] Davide Campagna, Alessandro Albiero, Alessandra Bilardi, Elisa Caniato, Claudio Forcato, Svetlin Manavski, Nicola Vitulo, and Giorgio Valle. Pass: a program to align short sequences. *Bioinformatics*, 2009. doi: 10.1093/bioinformatics/btp087. URL http://bioinformatics.oxfordjournals.org/content/early/2009/02/13/bioinformatics.btp087.abstract.

[38] Ruey-Shun Chen and Mengru (Arthur) Tu. Development of an agent-based system for manufacturing control and coordination with ontology and rfid technology. *Expert Systems with Applications*, 36(4): 7581 – 7593, 2009. ISSN 0957-4174. doi: DOI:10.1016/j.eswa.2008.09.068. URL http://www.sciencedirect.com/science/article/B6V03-4TNWGV3-2/2/bb69503c218efc2e1f7969a62d208738.

[39] J. I. Clark, C. Brooksbank, and J. Lomax. It's all GO for plant scientists. *Plant Physiol*, 138(3):1268–79, 2005. URL http://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?cmd=prlinks&dbfrom=pubmed&retmode=ref&id=16010001.

[40] Mélanie Courtot, William Bug, Frank Gibson, Allyson L. Lister, James Malone, Daniel Schober, Ryan Brinkman, and Alan Ruttenberg. The owl of biomedical investigations. In Catherine Dolbear, Alan Ruttenberg, and Ulrike Sattler, editors, *OWLED*, volume 432 of *CEUR Workshop Proceedings*, page xx. CEUR-WS.org, 2008.

[41] Antonio de Nicola, Michele Missikoff, and Roberto Navigli. A software engineering approach to ontology building. *Information Systems*, 34:

258, xx 2009. URL http://dx.doi.org/10.1016/j.is.2008.07.002. 10.1016/j.is.2008.07.002.

[42] D. De Roure and J. A. Hendler. E-science: the grid and the semantic web. *Intelligent Systems, IEEE*, 19(1):65–71, April 2005. doi: 10.1109/MIS.2004.1265888. URL http://dx.doi.org/10.1109/MIS.2004.1265888.

[43] David de Roure and Carole Goble. Software design for empowering scientists. *IEEE Software*, 26:88–95, Jan 2009. URL http://www2.computer.org/portal/web/csdl/doi/10.1109/MS.2009.22. 10.1109/MS.2009.22.

[44] Ewa Deelman, Dennis Gannon, Matthew Shields, and Ian Taylor. Workflows and e-science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25:528, xx 2009. URL http://dx.doi.org/10.1016/j.future.2008.06.012. 10.1016/j.future.2008.06.012.

[45] K. Degtyarenko, P. D. Matos, M. Ennis, J. Hastings, M. Zbinden, A. McNaught, R. Alcantara, M. Darsow, M. Guedj, and M. Ashburner. ChEBI: a database and ontology for chemical entities of biological interest. *Nucleic Acids Res*, 2007. URL http://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?cmd=prlinks&dbfrom=pubmed&retmode=ref&id=17932057.

[46] M. Deng, Z. Tu, F. Sun, and T. Chen. Mapping Gene Ontology to proteins based on protein-protein interaction data. *Bioinformatics*, 20(6):895–902, 2004. URL http:

//eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?cmd=prlinks&dbfrom=pubmed&retmode=ref&id=14751964.

[47] Dragan Djuric. The tao of modeling spaces. *Journal of Object Technology*, 5:125–147, 2006.

[48] Ping Du and Joseph A. Kofman. Electronic laboratory notebooks in pharmaceutical r&d: On the road to maturity. *Journal of the Association for Laboratory Automation*, 12(3):157 – 165, 2007. ISSN 1535-5535. doi: DOI:10.1016/j.jala.2007.01.001.

[49] M. Dumontier and R. Hoehndorf. Realism for scientific ontologies. *Front. Artif. Intell. Appl. Frontiers in Artificial Intelligence and Applications*, 209:387–399, 2010. URL http://www.worldcat.org/oclc/609654404.

[50] Jean-Marie Favre and Tam Nguyen. Towards a megamodel to model software evolution through transformations. In *SETRA Workshop, Elsevier ENCTS*, volume 127, pages 59–74, 2004.

[51] Heike Fiegler, Richard Redon, and Nigel Carter. Construction and use of spotted large-insert clone dna microarrays for the detection of genomic copy number changes. *Nat. Protocols*, 2:577–587, Mar 2007. ISSN 1750-2799. URL http://www.nature.com/nprot/journal/v2/n3/abs/nprot.2007.53.html. 10.1038/nprot.2007.53.

[52] Stanley Fields. Proteomics: Proteomics in genomeland. *Science*, 291:1221–1224, Feb 2001. URL http://www.sciencemag.org/cgi/content/full/291/5507/1221. 10.1126/science.291.5507.1221.

[53] G. Fortino, A. Garro, and W. Russo. Distributed workflow enactment: an agent-based framework. In *Atti del 7 Workshop dagli Oggetti agli*

Agenti (WOA) Sistemi GRID, Peer-to-peer e Self-, Catania (Italia)*, 2006.

[54] R France and B Rumpe. Model-driven development of complex software: A research roadmap. *International Conference on Software Engineering*, Jan 2007. URL http://portal.acm.org/citation.cfm?id=1254709.

[55] D. Gaševic, D. Djuric, and V. Devedžic. *Model Driven Engineering*, page 125. 2009. doi: 10.1007/978-3-642-00282-3_4.

[56] GenoLogics. Selecting a lims for the next-generation genomics lab: Five capabilities a preconfigured system should deliver (and why). Technical report, GenoLogics Life Sciences Software Inc., 2011.

[57] Royston Goodacre, Seetharaman Vaidyanathan, Warwick B. Dunn, George G. Harrigan, and Douglas B. Kell. Metabolomics by numbers: acquiring and understanding global metabolite data. *Trends Biotechnol*, 22:245–252, May 2004. ISSN 0167-7799. URL http://dx.doi.org/10.1016/j.tibtech.2004.03.007. 10.1016/j.tibtech.2004.03.007.

[58] Pierre Grenon and Barry Smith. Snap and span: Towards dynamic spatial ontology. *Spatial Cognition & Computation: An Interdisciplinary Journal*, 4(1):69–104, 2004. doi: 10.1207/s15427633scc0401_5.

[59] T. R. Gruber. Towards principles for the design of ontologies used for knowledge sharing. In N. Guarino and R. Poli, editors, *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Deventer, The Netherlands, 1993. Kluwer Academic Publishers. URL citeseer.ist.psu.edu/gruber93toward.html.

[60] John H. Gennari Hao Li and James F. Brinkley. Model driven laboratory information management systems. In *AMIA Annual Symposium Proceedings*, pages 484–488, 2006.

[61] M. Harris and H Parkinson. Standards and Ontologies for Functional Genomics: Towards Unified Ontologies for Biology and Biomedicine. *Comparative and Functional Genomics*, 4(1):116–120, 2003. doi:10.1002/cfg.249.

[62] M. A. Harris, J. Clark, A. Ireland, J. Lomax, M. Ashburner, R. Foulger, K. Eilbeck, S. Lewis, B. Marshall, C. Mungall, J. Richter, G. M. Rubin, J. A. Blake, C. Bult, M. Dolan, H. Drabkin, J. T. Eppig, D. P. Hill, L. Ni, M. Ringwald, R. Balakrishnan, J. M. Cherry, K. R. Christie, M. C. Costanzo, S. S. Dwight, S. Engel, D. G. Fisk, J. E. Hirschman, E. L. Hong, R. S. Nash, A. Sethuraman, C. L. Theesfeld, D. Botstein, K. Dolinski, B. Feierbach, T. Berardini, S. Mundodi, S. Y. Rhee, R. Apweiler, D. Barrell, E. Camon, E. Dimmer, V. Lee, R. Chisholm, P. Gaudet, W. Kibbe, R. Kishore, E. M. Schwarz, P. Sternberg, M. Gwinn, L. Hannick, J. Wortman, M. Berriman, V. Wood, N. de la Cruz, P. Tonellato, P. Jaiswal, T. Seigfried, and R. White. The Gene Ontology (GO) database and informatics resource. *Nucleic Acids Res*, 32(Database issue):D258–61, 2004. URL http://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?cmd=prlinks&dbfrom=pubmed&retmode=ref&id=14681407.

[63] Michael Hecker, Tharam Dillon, and Elizabeth Chang. Privacy ontology support for e-commerce. *IEEE Internet Computing*, 12:54–61, Mar 2008. URL http://www2.computer.org/portal/web/csdl/doi/10.1109/MIC.2008.41. 10.1109/MIC.2008.41.

[64] John P. Helfrich. Thin lims, thick lims — new it implementation strategy for cgmp quality informatics. Technical report, VelQuest Corporation, 2008.

[65] Leakha Henry, Kerrie Ramm, Qian-Hao Zhu, and Narayana Upadhyaya. Rgmims: a web-based laboratory information management system for plant functional genomics research. *Molecular Breeding*, 22: 151–157, 2008. ISSN 1380-3743. 10.1007/s11032-008-9160-z.

[66] Martin Hepp, Pieter De Leenheer, and Aldo De Moor. *Ontology Management: Semantic Web, Semantic Web Services, and Business Applications (Semantic Web and Beyond)*, volume 7. Springer, 1 edition, November 2007. URL http://www.springerlink.com/content/978-0-387-69899-1.

[67] Randy C Hice. Web-based lims: The indelible mark on the face of pharmaceutical informatics. *Innovations in Pharmaceutical Technology*, 25: 32–34, 2009.

[68] Bertrand Frdric Hillairet Guillaume and Lafaye Jean Yves. Bridging emf applications and rdf data sources. In *Proceedings of the 4th international workshop on Semantic Web Enabled Software Engineering (SWESE) at ISWC'08*, pages 26–40, 10 2008.

[69] Horrocks, I., Schneider, Patel P., and van Harmelen, F. From shiq and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.2.7039.

[70] Da Wei Huang, Brad T. Sherman, and Richard A. Lempicki. Systematic and integrative analysis of large gene lists using david bioin-

formatics resources. *Nature protocols*, 4:44–57, Dec 2008. ISSN 1750-2799. URL http://www.nature.com/nprot/journal/v4/n1/abs/nprot.2008.211.html. 10.1038/nprot.2008.211.

[71] Lederberg Joshua and Mccray Alexa. ”ome sweet ”omics–a genealogical treasury of words. — accessmylibrary - promoting library advocacy. *The Scientist*, April 2001. URL http://www.accessmylibrary.com/coms2/summary_0286-719248_ITM.

[72] F. Jouault and J. Bezivin. Km3: a dsl for metamodel specification. *Lecture Notes In Computer Science*, 4037:171–185, 2006. URL http://atlanmod.emn.fr/www/papers/KM3-FMOODS06.pdf.

[73] Elisabeth Kapsammer, Horst Kargl, Gerhard Kramler, Thomas Reiter, Werner Retschitzegger, and Manuel Wimmer. Lifting metamodels to ontologies - a step to the semantic integration of modeling languages. In *In Proceedings of the ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems (MoDELS/UML 2006*, pages 528–542. Springer, 2006.

[74] Ross D. King, Jem Rowland, Stephen G. Oliver, Michael Young, Wayne Aubrey, Emma Byrne, Maria Liakata, Magdalena Markham, Pinar Pir, Larisa N. Soldatova, Andrew Sparkes, Kenneth E. Whelan, and Amanda Clare. The Automation of Science. *Science*, 324(5923):85–89, 2009. doi: 10.1126/science.1165620.

[75] Rosine Kitio, Olivier Boissier, Jomi Fred Hübner, and Alessandro Ricci. Organisational artifacts and agents for open multi-agent organisations: ”giving the power back to the agents”. In *Proceedings of the 2007 international conference on Coordination, organizations, institutions, and*

norms in agent systems III, COIN'07, pages 171–186, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 3-540-79002-0, 978-3-540-79002-0. URL http://dl.acm.org/citation.cfm?id=1791649.1791664.

[76] Ekat Kritikou. Watch and learn. *Nat Rev Mol Cell Biol*, 8:4, Jan 2007. ISSN 1471-0072. URL http://www.nature.com/nrm/journal/v8/n1/full/nrm2097.html. 10.1038/nrm2097.

[77] Patrick Lambrix, Manal Habbouche, and Marta Pérez. Evaluation of ontology development tools for bioinformatics. *Bioinformatics*, 19:1564, xx 2003. URL http://dx.doi.org/10.1093/bioinformatics/btg194. 10.1093/bioinformatics/btg194.

[78] Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, Richard Durbin, and 1000 Genome Project Data Processing Subgroup. The sequence alignment/map format and samtools. *Bioinformatics*, 25(16):2078–2079, 2009. doi: 10.1093/bioinformatics/btp352. URL http://bioinformatics.oxfordjournals.org/content/25/16/2078.abstract.

[79] Ling Liu and M. Tamer Özsu. Encyclopedia of database systems, 2009. URL http://www.worldcat.org/isbn/9780387496160.

[80] Alessandro Maccagnan, Mauro Riva, Erika Feltrin, Barbara Simionati, Tullio Vardanega, Giorgio Valle, and Nicola Cannata. Combining ontologies and workflows to design formal protocols for biological laboratories. *Automated Experimentation*, 2(1):3, 2010. ISSN 1759-4499. doi: 10.1186/1759-4499-2-3.

[81] Alessandro Maccagnan, Tullio Vardanega, Erika Feltrin, Giorgio Valle, Mauro Riva, and Nicola Cannata. A multi-agent system for the automated handling of experimental protocols in biological laboratories. In Andrea Omicini and Mirko Viroli, editors, *Proceedings of the 11th WOA 2010 Workshop, Dagli Oggetti Agli Agenti, Rimini, Italy, September 5-7, 2010*, volume 621 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010.

[82] Alessandro Maccagnan, Tullio Vardanega, Giorgio Valle, and Nicola Cannata. Mapping obi and xpdl to a mde framework for laboratory information processing. In *The Fourth International Conference on Information, Process, and Knowledge Management - eKNOW 2012*, 2012.

[83] R. Mack and M. Hehenberger. Text-based knowledge discovery: search and mining of life-sciences documents. *Drug Discov Today*, 7(11 Suppl): S89–98, 2002. URL http://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?cmd=prlinks&dbfrom=pubmed&retmode=ref&id=12047886.

[84] Andreia Malucelli, Daniel Palzera, and Eugénio Oliveiraa. Ontology-based services to help solving the heterogeneity problem in e-commerce negotiations. *Electronic Commerce Research and Applications*, 5:29, xx 2006. URL http://dx.doi.org/10.1016/j.elerap.2005.08.002. 10.1016/j.elerap.2005.08.002.

[85] R.D. McDowall. Future trends in lims. *American pharmaceutical review*, 8(6):10 – 15, 2005.

[86] John A. McGiven, Iain J. Thompson, Nicola J. Commander, and

Judy A. Stack. Time-Resolved Fluorescent Resonance Energy Transfer Assay for Simple and Rapid Detection of Anti-Brucella Antibodies in Ruminant Serum Samples. *J. Clin. Microbiol.*, 47(10):3098–3107, 2009. doi: 10.1128/JCM.00919-09.

[87] Stephen Muggleton. 2020 computing: Exceeding human limits. *Nature.*, 440:409–410, Mar 2006. ISSN 1476-4687. URL http://www.nature.com/nature/journal/v440/n7083/full/440409a.html. 10.1038/440409a.

[88] Fabian Neuhaus, Pierre Grenon, and Barry Smith. A formal theory of substances, qualities, and universals. In Achille C. Varzi and Laure Vieu, editors, *In Achille Varzi and Laure Vieu, editors, International Conference on Formal Ontology in Information Systems (FOIS'04)*, Frontiers in artificial intelligence and applications, v. 114, pages 49–59. IOS Press, 2004. ISBN 9781586034689.

[89] Tom Oinn, Matthew Addis, Justin Ferris, Darren Marvin, Martin Senger, Mark Greenwood, Tim Carver, Kevin Glover, Matthew Pocock, Anil Wipat, and Peter Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics (Oxford, England)*, 20:3045–3054, Nov 2004. URL http://dx.doi.org/10.1093/bioinformatics/bth361. 10.1093/bioinformatics/bth361.

[90] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. Artifacts in the a&a meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17(3):432–456, 2008. ISSN 1387-2532. doi: http://dx.doi.org/10.1007/s10458-008-9053-x.

[91] Brian D. Ondov, Anjana Varadarajan, Karla D. Passalacqua, and Nich-

olas H. Bergman. Efficient mapping of Applied Biosystems SOLiD sequence data to a reference genome for functional genomic applications. *Bioinformatics*, 24(23):2776–2777, December 2008. ISSN 1460-2059. doi: 10.1093/bioinformatics/btn512. URL http://dx.doi.org/10.1093/bioinformatics/btn512.

[92] R. Pavlis. The paperless laboratory: Realities and expectations. *Innovations in Pharmaceutical Technology*, 29:35–37, 2009.

[93] Asuncion G. Perez, Oscar Corcho, and Mariano F. Lopez. *Ontological Engineering : with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web. First Edition (Advanced Information and Knowledge Processing)*. Springer, July 2004. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.124.3549&#38;rep=rep1&#38;type=pdf.

[94] Erik Pettersson, Joakim Lundeberg, and Afshin Ahmadian. Generations of sequencing technologies. *Genomics*, 93(2):105 – 111, 2009. ISSN 0888-7543. doi: DOI:10.1016/j.ygeno.2008.10.003.

[95] S. Philippi and J. Kohler. Addressing the problems with life-science databases for traditional uses and systems biology. *Nat Rev Genet*, 7(6):482–8, 2006. URL http://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?cmd=prlinks&dbfrom=pubmed&retmode=ref&id=16682980.

[96] Michael Pidd. *Tools for Thinking: Modelling in Management Science*. Wiley, 3 edition, February 2009. ISBN 0470721421. URL http://www.worldcat.org/isbn/0470721421.

[97] Agostino Poggi and Paola Turci. An agent-based bridge between business process and business rules. In *Decimo Workshop Nazionale Dagli Oggetti agli Agenti*, 2009.

[98] Alessandro Ricci, Mirko Viroli, and Andrea Omicini. The a&a programming model and technology for developing agent environments in mas. In *ProMAS'07: Proceedings of the 5th international conference on Programming multi-agent systems*, pages 89–106, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 3-540-79042-X, 978-3-540-79042-6.

[99] Alessandro Ricci, Mirko Viroli, and Giulio Piancastelli. simpa: An agent-oriented approach for programming concurrent applications on top of java. *Science of Computer Programming*, 76(1): 37 – 62, 2010. ISSN 0167-6423. doi: DOI:10.1016/j.scico.2010.06.012. URL http://www.sciencedirect.com/science/article/B6V17-50G0631-1/2/abdeaa8467f0d325ad4968ddb5f8ae18. Selected papers from the 6th International Workshop on the Foundations of Coordination Languages and Software Architectures - FOCLASA'07.

[100] Paolo Romano. Automation of in-silico data analysis processes through workflow management systems. *Brief Bioinform*, 9(1):57–68, January 2008. ISSN 1477-4054. doi: 10.1093/bib/bbm056. URL http://dx.doi.org/10.1093/bib/bbm056.

[101] Nick Russell, Arthur H. Hofstede, David Edmond, and Wil M. der Aalst. *Workflow Data Patterns: Identification, Representation and Tool Support*, volume 3716, chapter Chapter 23, pages 353–368. Springer-Verlag, Berlin/Heidelberg, 2005. ISBN 3-540-29389-2. doi: 10.1007/11568322_23.

[102] Nick Russell, Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, and David Edmond. *Workflow Resource Patterns: Identification, Representation and Tool Support*, pages 216–232. Springer Berlin, Heidelberg, 2005. ISBN 978-3-540-26095-0. doi: 10.1007/11431855_16. URL http://dx.doi.org/10.1007/11431855_16.

[103] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall series in artificial intelligence. Prentice Hall, second edition, December 2002. ISBN 0137903952. URL http://www.worldcat.org/isbn/0137903952.

[104] Christopher S. Sandlin, Rudolph C. Johnson, Leigh Swaim, and David L. Ashley. Laboratory information management system for emergency response: Validation and quality assurance of analytical methodologies. *Journal of the Association for Laboratory Automation*, 14(3): 126 – 132, 2009. ISSN 1535-5535. doi: DOI:10.1016/j.jala.2009.02.001.

[105] Stefan Schulz, M. Boeker, and H. Stenzhorn. How granularity issues concern biomedical ontology integration. *Studies in health technology and informatics*, 136:863–8, 2008.

[106] E. Seidewitz. What models mean. *Software, IEEE*, 20(5):26 – 32, sep. 2003. ISSN 0740-7459. doi: 10.1109/MS.2003.1231147.

[107] Arash Shaban-Nejad, Olga Ormandjieva, Mohamad Kassab, and Volker Haarslev. Managing requirement volatility in an ontology-driven clinical lims using category theory. *Int. J. Telemedicine Appl.*, 2009:1–14, 2009. ISSN 1687-6415. doi: \url{http://dx.doi.org/10.1155/2009/917826}.

[108] Robert Shapiro and Mike Marin. *Workflow Management Coalition Workflow StandardProcess Definition Interface– XML Process Definition Language*. The Workflow Management Coalition, 99 Derby Street, Suite 200 Hingham, MA 02043 USA, October 2008.

[109] Chad B. Shawn, Shawn Bowers, Matthew B. Jones, Bertram Ludäscher, Mark Schildhauer, and Jing Tao. Incorporating semantics in scientific workflow authoring. In *In Proceedings of the 17th International Conference on Scientific and Statistical Database Management (SSDBM'05)*, 2005. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.60.7517.

[110] Vladimir Shulaev. Metabolomics technology and bioinformatics. *Briefings in Bioinformatics*, 7:128, xx 2006. URL http://bib.oxfordjournals.org/cgi/content/full/7/2/128. 10.1093/bib-/bbl012.

[111] B. Smith, W. Ceusters, B. Klagges, J. Kohler, A. Kumar, J. Lomax, C. Mungall, F. Neuhaus, A. L. Rector, and C. Rosse. Relations in biomedical ontologies. *Genome Biol*, 6(5):R46, 2005. URL http://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?cmd=prlinks&dbfrom=pubmed&retmode=ref&id=15892874.

[112] Barry Smith. Beyond concepts: Ontology as reality representation. In *In Achille Varzi and Laure Vieu, editors, International Conference on Formal Ontology in Information Systems (FOIS'04)*, Frontiers in artificial intelligence and applications, v. 114, pages 73–84. IOS Press, 2004.

[113] Barry Smith, Michael Ashburner, Cornelius Rosse, Jonathan Bard,

William Bug, Werner Ceusters, Louis J. Goldberg, Karen Eilbeck, Amelia Ireland, Christopher J. Mungall, OBI Consortium, Neocles Leontis, Philippe Rocca-Serra, Alan Ruttenberg, Susanna-Assunta A. Sansone, Richard H. Scheuermann, Nigam Shah, Patricia L. Whetzel, and Suzanna Lewis. The obo foundry: coordinated evolution of ontologies to support biomedical data integration. *Nature biotechnology*, 25(11):1251–1255, November 2007. ISSN 1087-0156. doi: 10.1038/nbt1346. URL http://dx.doi.org/10.1038/nbt1346.

[114] Larisa Soldatova and Ross King. An ontology of scientific experiments. *Journal of The Royal Society Interface*, 3:795, xx 2006. URL http://dx.doi.org/10.1098/rsif.2006.0134. 10.1098/rsif.2006.0134.

[115] Larisa Soldatova, Wayne Aubrey, Ross King, and Amanda Clare. The exact description of biomedical protocols. *Bioinformatics (Oxford, England)*, 24:i295–303, Jul 2008. URL http://bioinformatics.oxfordjournals.org/cgi/content/short/24/13/i295. 10.1093/bioinformatics/btn156.

[116] Andrew Sparkes, Wayne Aubrey, Emma Byrne, Amanda Clare, Muhammed Khan, Maria Liakata, Magdalena Markham, Jem Rowland, Larisa Soldatova, Kenneth Whelan, Michael Young, and Ross King. Towards robot scientists for autonomous scientific discovery. *Automated Experimentation*, 2(1):1, 2010. ISSN 1759-4499. doi: 10.1186/1759-4499-2-1.

[117] Gernot Stocker, Maria Fischer, Dietmar Rieder, Gabriela Bindea, Simon Kainz, Michael Oberstolz, James McNally, and Zlatko Trajanoski. ilap: a workflow-driven software for experimental protocol develop-

ment, data acquisition and analysis. *BMC Bioinformatics*, 10(1):390, 2009. ISSN 1471-2105. doi: 10.1186/1471-2105-10-390.

[118] Michael R. Stratton, Peter J. Campbell, and P. Andrew Futreal. The cancer genome. *Nature*, 458(7239):719–724, April 2009. ISSN 0028-0836. doi: 10.1038/nature07943. URL `http://dx.doi.org/10.1038/nature07943`.

[119] Andrew Tolopko, John Sullivan, Sean Erickson, David Wrobel, Su Chiang, Katrina Rudnicki, Stewart Rudnicki, Jennifer Nale, Laura Selfors, Dara Greenhouse, Jeremy Muhlich, and Caroline Shamu. Screensaver: an open source lab information management system (lims) for high throughput screening facilities. *BMC Bioinformatics*, 11(1):260, 2010. ISSN 1471-2105. doi: 10.1186/1471-2105-11-260.

[120] Charles V. Trappey, Amy J.C. Trappey, Ching-Jen Huang, and C.C. Ku. The design of a jade-based autonomous workflow management system for collaborative soc design. *Expert Systems with Applications*, 36(2, Part 2):2659 – 2669, 2009. ISSN 0957-4174. doi: DOI:10.1016/j.eswa.2008.01.064. URL `http://www.sciencedirect.com/science/article/B6V03-4RV7Y9W-2/2/f933cced0e8af5448692818153bb1648`.

[121] Petr V. Troshin, Chris Morris, Stephen M. Prince, and Miroslav Z. Papiz. Laboratory information management system for membrane protein structure initiative from gene to crystal. *Molecular Membrane Biology*, 25(8):639–652, 2008. doi: 10.1080/09687680802511766.

[122] Mike Tyers and Matthias Mann. From genomics to proteomics. *Nature*,

422:193–197, Mar 2003. ISSN 0028-0836. URL http://dx.doi.org/10.1038/nature01510. 10.1038/nature01510.

[123] Marco Ughetti, Tiziana Trucco, and Danilo Gotta. Development of agent-based, peer-to-peer mobile applications on android with jade. *Mobile Ubiquitous Computing, Systems, Services and Technologies, International Conference on*, 0:287–294, 2008. doi: http://doi.ieeecomputersociety.org/10.1109/UBICOMM.2008.72.

[124] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51–51, July 2003. ISSN 09268782. doi: 10.1023/A:1022883727209. URL http://www.workflowpatterns.com/documentation/documents/wfs-pat-2002.pdf.

[125] Thea Van Rossum, Ben Tripp, and Denise Daley. SLIMSa user-friendly sample operations and inventory management system for genotyping labs. *Bioinformatics*, 26(14):1808–1810, 2010. doi: 10.1093/bioinformatics/btq271.

[126] C. Voegele, S.V. Tavtigian, D. de Silva, S. Cuber, A. Thomas, and F. Le Calvez-Kelm. A Laboratory Information Management System (LIMS) for a high throughput genetic platform aimed at candidate gene mutation screening. *Bioinformatics*, 23(18):2504–2506, 2007. doi: 10.1093/bioinformatics/btm365.

[127] David L. Wheeler, Tanya Barrett, Dennis A. Benson, Stephen H. Bryant, Kathi Canese, Deanna M. Church, Michael DiCuccio, Ron Edgar, Scott Federhen, Wolfgang Helmberg, David L. Kenton, Oleg Khovayko, David J. Lipman, Thomas L. Madden, Donna R. Maglott, James Os-

tell, Joan U. Pontius, Kim D. Pruitt, Gregory D. Schuler, Lynn M. Schriml, Edwin Sequeira, Steven T. Sherry, Karl Sirotkin, Grigory Starchenko, Tugba O. Suzek, Roman Tatusov, Tatiana A. Tatusova, Lukas Wagner, and Eugene Yaschenko. Database resources of the National Center for Biotechnology Information. *Nucleic Acids Research*, 33(suppl 1):D39–D45, 2005. doi: 10.1093/nar/gki062. URL http://nar.oxfordjournals.org/content/33/suppl_1/D39.abstract.

[128] Michael Wooldridge and Nicholas R. Jennings. Intelligent Agents: Theory and Practice. *Knowledge Engineering Review*, 10:115–152, 1995. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.55.2702.

[129] A. F. Yousef, I. M. Baggili, G. Bartlett, M. D. Kane, and J. S. Mymryk. Lina: A laboratory inventory system for oligonucleotides, microbial strains, and cell lines. *Journal of the Association for Laboratory Automation*, Mar 2010. ISSN 1535-5535.