Alma Mater Studiorum – Università di Bologna

DOTTORATO DI RICERCA
in Ingegneria Elettronica, Informatica e delle Telecomunicazioni

Ciclo XXI

Settore scientifico disciplinare di afferenza: ING-INF/05

PRINCIPI, METODOLOGIE E PROGETTO DI MIDDLEWARE PER SERVIZI CONTEXT-AWARE

UBIQUITOUS INTERNET MIDDLEWARE: ARCHITECTURE DESIGN AND PROTOTYPE EVALUATION

Presentata da: Samuele Pasini

Coordinatore Dottorato
Chiar.ma Prof. Ing. Paola Mello

Relatori
Chiar.mo Prof. Ing. Maurelio Boari
Chiar.mo Prof. Ing. Antonio Corradi

Esame finale anno 2009

Keywords

Ubiquitous Internet
Distributed services
Middleware
Composition
Orchestration

Contents

Illustration Index

Listing index

Introduction

Technology advances in recent years have dramatically changed the way users exploit contents and services available on the Internet, by enforcing pervasive and mobile computing scenarios and enabling access to networked resources almost from everywhere, at anytime, and independently of the device in use. In addition, people increasingly require to customize their experience, by exploiting specific device capabilities and limitations, inherent features of the communication channel in use, and interaction paradigms that significantly differ from the traditional request/response one.

So-called *Ubiquitous Internet* scenario calls for solutions that address many different challenges, such as device mobility, session management, content adaptation, context-awareness and the provisioning of multimodal interfaces. Moreover, new service opportunities demand simple and effective ways to integrate existing resources into new and value added applications, that can also undergo run-time modifications, according to ever-changing execution conditions.

Despite service-oriented architectural models are gaining momentum to tame the increasing complexity of composing and orchestrating distributed and heterogeneous functionalities, existing solutions generally lack a unified approach and only provide support for specific Ubiquitous Internet aspects. Moreover, they usually target rather static scenarios and scarcely support the dynamic nature of pervasive access to Internet resources, that can make existing compositions soon become obsolete or inadequate, hence in need of reconfiguration.

This thesis proposes a novel middleware approach to comprehensively deal with Ubiquitous Internet facets and assist in establishing innovative application scenarios. We claim that a truly viable ubiquity support infrastructure must neatly decouple distributed resources to integrate and push any kind of content-related logic outside its core layers, by keeping only

management and coordination responsibilities. Furthermore, we promote an innovative, open, and dynamic resource composition model that allows to easily describe and enforce complex scenario requirements, and to suitably react to changes in the execution conditions.

In this thesis, we present middleware design principles and key architectural aspects that permit effective Ubiquitous Internet support. We also provide implementation details and description of typical use cases we have been able to realize, in time, to demonstrate viability of our proposal. Thesis is structured as follows. Chapter 1 introduces aim of the work and guidelines we have considered. Chapter 2 illustrates foundational concepts behind our vision and it introduces the architectural model our work bases on. Chapter 3 deepens the analysis of aspects entailed by the integration of distribute heterogeneous resources, and describes management, communication and interoperation facilities our solution provides. Chapter 4 concentrates on middleware components that pursue effective resource integration and demonstrates how our platform enables support to multiple interfaces, multiple user-service interaction paradigms and suitable content adaptation, while helping to keep problems orthogonal. Chapter 5 points out resource composition challenges we have faced and the composition model we have developed in response, stressing activity orchestration issues in actual business processes as well as the need for dynamic and automatic reconfiguration of resource compositions to fit all-changing scenario requirements. Chapter 6 reports extensive investigation of current attempts to orchestrate computational activities from both final users and services and highlights relevant issues concerned with the Ubiquitous Internet scenario; we propose comparisons between our solution and related work and draw on both current achievements and limitations to motivate our approach. Chapter 7 presents middleware prototype implementation characteristics and debates about technologies we have leveraged and scalability issues. Chapter 8 depicts actual scenarios we have realized over time and shows how we have been able to enforce middleware mechanisms for the sake of administration of middleware itself, by exposing

its core functionalities as ordinary resources to orchestrate. Chapter 9 evaluates prototype performance, overhead, and scalability, Finally, Conclusions summarize design principles and architectural achievements and indicate future research directions.

Chapter 1 – Generalities

Over the last few years, new heterogeneous types of wireless networks and new kinds of devices able to exploit them have become more and more inexpensive and available. Compared to late 70s, when mobile and networked notebook appliances were just "research directions" [Kay77], people have nowadays a plethora of information processing devices at their fingertips and can reach and interact with remote contents via many communication links. We all now own and carry things like mobile phones, handheld devices, personal computers, digital TV set-top-boxes or portable media players, and they exchange data among each other and/or with remote servers through several connection types such as Bluetooth PAN, wired/wireless LAN, ADSL or WiMax powered WAN or even UMTS and satellite links.

Even though today scenario seems to be the result of several concurrent driving factors, connected to both Internet achievements and technological improvements, the overall goal turns out surprisingly clear right from the start. As Alan C. Kay wrote in 1972, talking about a Xerox project at that time:

> *"... Though the Dynabook will have considerable local storage and will do most computing locally, it will spend a large percentage of its time hooked to various large, global information utilities which will permit communication with others of ideas, data, working models, as well as the daily chit-chat that organizations need in order to function. The communications link will be by private and public wires and by packet radio. ..."* [Kay72]

From those times on, technology has evolved and eventually almost fulfilled also Mark Weiser's foundational vision for ubiquitous computing: an environment saturated with pervasive computing and communication capability, yet so gracefully integrated with users that they slightly become

unaware of it, letting computers fade in the background and not demand attention [Wei02].

Even if today machines cannot truly make computing an invisible part of life, devices are little by little integrating with several aspects of the natural human environment. Thus, thanks to a wider and wider set of computing hardware and network infrastructures, users can now exploit remote contents and services almost from anywhere and at anytime and, furthermore, they can experience augmented-life scenarios by combining real-world needs with online-world possibilities. Not only literature [Hen02][Int02][Joh02] but also current reality presents plenty such scenarios: trains are full of laptop users surfing the web or doing instant-messaging, and people buy GPS-enabled mobiles that can leverage online maps and location-based information.

So-called *Ubiquitous Internet* actually represents a great chance to provide highly customized services to further enhance user satisfaction and create new service opportunities. Its evolution towards a global platform for the retrieval, combination and utilization of rich resources is clearly gaining momentum, and relevant applications have been emerging over the last few years [GMaps][Wiki]. Nevertheless, this also entails great challenges, due to different client device capabilities, context conditions and modifications, session management, and software integration issues [Sat01][Sah03]. Providing and consuming services via the Internet still is at its early stage and lacks widely accepted standards for defining service choreographies and semantics. Ultimately, this has prevented global meshes of collaborating Internet resources to appear [Sch07a].

In the fields of Business-to-Business (B2B) and Enterprise Application Integration (EAI), Web Services have experienced great interest as means to realize seamless cross-organizational collaborations, by basing on the principles of Service-Oriented Architectures (SOAs) [Alo03a][SOA]. But on the global side, apart from inflexibility and performance problems, service mashups also suffer from the absence of effective platforms to allow for both human interaction and service composition, able to consider people as “part of

the system" [Chr02]. Focus on user-empowerment and the consideration of the Web as a platform for building systems will certainly facilitate the establishment of global service-orientation, but in the Internet of today users are not usually enabled to draw on more than one "resource" at a time. For instance, iGoogle pages [iGoogle] just represent a first intuitive attempt for a mashup platform, as they base on mere content syndication and limited application functionalities. Very few examples exist that try to enable resource processing and choreography for the (skilled) final users [Pipes][Kapow], though producing "information islands" and applications that are mostly accessible via proprietary portals, rather than actual integration [Sch07b].

In 2005, Tim O'Reilly invented the term *Web 2.0* to describe these kinds of scenario, where a set of Web-based applications are "tied together by a set of protocols, open standards, and agreements for cooperation" [Rei05]. Högg et al. deeply investigate the business model of forty Web 2.0 applications in [Hog07], concluding that they maximize intelligence and added value by means of formalized and dynamic information sharing and creation. Indeed, while conventional SOAs merely aim at interconnecting dispersed business functionalities and facilitating seamless machine-to-machine collaboration, Web 2.0 applications also incorporate human interaction and social aspects, and deal with human-readable content, such as text and pictures.

Both SOA and Web 2.0 enforce reuse and composition of existing resources and promote collaboration of loosely coupled remote services. Despite issues about interoperability and how to model human-intervention, convergence of the two philosophies actually does represent the driving force for the growth of future global SOAs, constituting what is being called the novel *Internet of Services* (IoS) [Sch07c]. So far, such a complex and evolving scenario is being pioneered by several innovative applications and development guidelines, still leading to ad-hoc solutions and heterogeneous ways of facing similar problems several times. Initiatives like Google Mashup Editor [Mashup], for instance, force programmers to mandatorily adopt given technologies to develop services (i.e., AJAX [Gar05]), while framework specifications such as Sun Microsystems Portlet [Pat05] still lack integration

among features they let syndicate. And still final users have to adapt to system behavior to get their 2.0 experience.

1.1 Aim of the work

In our opinion, no matter how powerful the integration platform in use theoretically is, few key elements are crucial to achieve effectiveness in making global resource mashups. First of all, final users must be kept unaware of what is going on behind the scenes: system must support all interaction paradigms they wish to follow – maybe due to personal preferences or terminal capabilities – and not oblige them to behave in a constrained manner. Secondly, developers life should become simpler, rather than more complicated: the business logic they want to pursue is usually complex enough and they certainly do not approve the learning of other software layers.

Software infrastructures that enable our modern information society have to foster the conception, development and provisioning of application scenarios wherein services can meet user requirements in highly efficient and transparent ways, according to preferences that user themselves express or that depend on the inherent nature of the desired interaction type, as well as on current device capabilities and other physical and computational environment information. To tame the growing complexity that such a pervasive computing scenario entails, final users and services that are available via the Internet must remain as much as possible independent of each other. Intermediate software layers, often called *middleware* [Ber96], must intervene to decouple different resources that need to cooperate, in order to consistently and comprehensively tackle the problems that Ubiquitous Internet raises.

From a technical perspective, most challenging issues that Ubiquitous Internet middleware has to address stem from the concepts of *mobility* and *heterogeneity*.

On the one hand, mobility is a fundamental characteristic of modern Internet scenarios: users no longer exploit services only via their desktop PCs

over wired network connections, but demand access via multiple devices, often free to move in space and to connect through different moments in time and different network infrastructures. Providing effective services to this kind of users must adapt to ever-changing device capabilities, as well as take into consideration relevant and dynamic information from their surrounding physical and computational context (e.g., geographical location and available bandwidth). Besides, the opportunity to grant service access to mobile devices also requires suitable session management to avoid users loose information and experience inconsistencies when changing device in use or network address.

On the other hand, heterogeneity relates to intrinsic differences that different types of terminal present in terms of interfaces they provide of users, allowed interaction paradigm, and support of media. To give a short example, getting information of one bank account by visiting the bank Web site via a traditional browser, rather than by performing a phone call to an automatic SIP server extension, can actually consist in leveraging the same bank Web Services, though accessed in extremely different ways. HTTP requests from the browser can convey multiple parameters at once and get complex data in response, such as HTML tables and graphics; on the contrary, phone calls are typically served with nested multiple choice selection menus and they have to cope with them by dialing tones in the correct order, to get limited but detailed information in a voice-synthesized form.

We also claim that support for a given application scenario must not be perceived and considered as some kind of static facility, obtained as the result of human “manual” intervention and programming on the middleware platform. In order to let final users express highly customized preferences and to support dynamic changes in their requirements, middleware must provide mechanisms to deterministically adapt service provisioning to possible varying conditions, hence support automatic reconfiguration.

Final users must be able to specify different means and devices to access desired services, either by indicating explicit choices or by leveraging

middleware capabilities to detect their status and to react properly. Given the current operating conditions and available services, middleware must arrange most suitable type of interaction, content processing and result delivery to satisfy user needs.

Resource integration therefore can only happen in the form of composition of pieces of business logic that altogether define a business process to model the given application scenario, and in the orchestration of that process to accordingly exploit the resources that it entails. Anyway, manual definition of suitable business processes cannot be a solution to Ubiquitous Internet challenges by any means; rather, automatic calculation of such processes is inherently necessary to leverage pervasive computing opportunity to provide value-added services without negative impact on final user experiences. As long as middleware executes autonomously, users can concentrate on their very goal in service exploitation as well as developers can focus on service core logic and undertake little or no additional complexity.

1.2 Guidelines

In a world of pervasive Internet access, people connect with heterogeneous devices and exploit several services, simultaneously in case. Besides, wireless infrastructures let them move freely in space, so that their physical and computational surrounding environment changes continuously. For instance, in a near-tomorrow scenario, university student Arianna has just subscribed to an Internet music service that lets her specify the genre and mood of the songs she would like to listen to and automatically creates a track playlist for her (alike today's Musicovery [Musicovery]). At the university campus, Arianna can exploit free Wi-Fi network coverage to access the service; thus, today she's studying with her earphones on, attached to her smartphone playing online music. Bandwidth is high and the service lets her download contents coded at an elevate bitrate. Later in the afternoon, she decides to go shopping downtown. On her way there, she can keep on listening to music on the smartphone 3G connection; system recognizes that

and reacts by downsizing content bitrate. Back to her student room, she switches on her PC, stops her playlist on the phone and resumes it on the computer, by sending audio to quality speakers. Arianna never stops listening to music nor has to reconfigure things, despite changes in network connection and device she uses. Just as with the mythological red fleece thread of Ariadne, her status and conditions never get lost. Finally, as service plays a song she's particularly fond of, Arianna can leverage the instant messaging service that integrates with the music one to invite one of her online friends to listen to the same song, having the system send data flow to him too.

To tell the truth, this scenario and similar ones are not so distant in future. It is already of no difficult to develop one player per device and a server able to deliver and keep status of song playlists. And desired bitrate may come from one of different song versions or via real-time conversion, and depend on the round-trip time of out-of-band control signaling. And instant messaging user status (e.g., on-line, off-line, busy) may depend on reproduction status (on, off, paused). And data flow forwarding for sharing songs may exploit the same packet circuits already reserved by the instant messaging service. And on, and on, and on. Problems arise, anyway, when it comes to maintain such a system, or add functionalities, or bring existing ones to new kinds of device. As long as scenarios get complex, it is simply unconceivable to let remote services and client software interact directly.

We claim that a truly viable and comprehensive infrastructure for Ubiquitous Internet support must follow a middleware approach [Ber96] and decouple distribute resources that application scenarios involve, to relieve them of the burden of integration. On the one hand, heterogeneous users/clients must be able to access heterogeneous contents/services without worrying about how to invoke each one and how to explicitly influence their behavior; on the other hand, service developers must concentrate only on service business logic, disregarding how users will exploit services to fit their requirements. In other words, final users must be prevented from tedious manual configuration and, at the same time, service developers must not be

concerned with user monitoring and profiling issues or mutual service integration and orchestration problems.

We strongly promote the idea of modeling novel Ubiquitous Internet applications in terms of arbitrarily complex *business processes*, where a distributed and intermediate software layer is in charge to compose resources involved in computation by orchestrating their execution, while providing them with suitable integration facilities.

At the same time, we also argue that complexity and potential relations among different aspects in content processing within the Ubiquitous Internet scenario definitively require a unified perspective approach, in order to keep things as clean and simple as possible, to avoid unnecessary interdependencies and, vice versa, to highlight similarities and unifying abstractions in supporting those aspects. Most current middleware solutions, instead, just focus on providing dedicated features that services and client applications can exploit to face content transformation and aggregation, as well as profiling or monitoring [VoiceXML][Opera]. But as a matter of fact, when the number of functionalities increases and functions have to interact with each other, traditional middleware complexity inevitably grows, making this approach inadequate for facing general application domains. On the contrary, middleware infrastructure should facilitate and reduce resource responsibilities and dependencies, hence promote the concept of disappearing computing and integration.

In our vision, we model both human activities (endorsed by heterogeneous client-side applications) and distributed services (regardless of their implementing technologies) in terms of *resources* we conceive as abstract functionalities we can leverage and provide facilities. This permits to highlight similarities among diverse entity types and to adopt uniform and established ways of representing them within our system.

In particular, we overcome mobility and heterogeneity by means of a well defined resource behavior and lifecycle model. We claim that *proxy* adaptors represent the solution to enforce this model, by making resource

adhere to it via the execution of proxies themselves, and by supplying additional integration features. As Illustration 1 shows, we leverage proxy adaptors to deal with uniform representations of possibly remote and heterogeneous objects to orchestrate; by means of proxies, then, we provide those objects with effective and always available status information and communication capabilities.

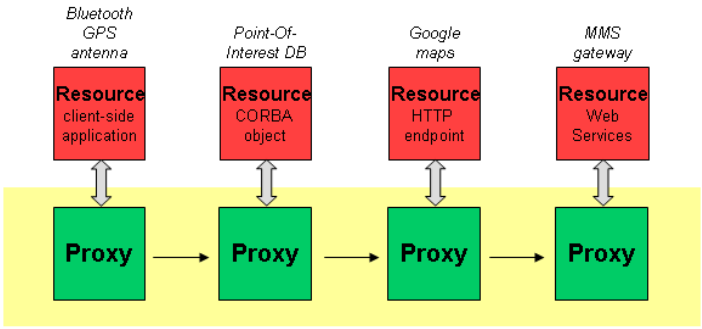

Illustration 1: Proxy adaptors enable resource integration

Besides, we simplify middleware design by endorsing a powerful task delegation strategy that assigns any kind of content-related activity to resources themselves, and leaves to the middleware platform the sole responsibility for their composition, orchestration and management. In details, we adopt *workflow* entities and related patterns [Aal03][Rus08] to gather computational resources into coherent and structured activities that can model concrete Ubiquitous Internet scenarios. Workflow execution represents nowadays a well-established and appreciated practice for organizing distributed functionalities into flows of operations made up of both business logic and control blocks, able altogether to achieve well-defined goals such as those pursued by the business processes in distributed Internet applications.

Illustration 2, below, demonstrates this by arranging diverse resource functionalities, via their corresponding proxies, into a workflow structure that enables content transformation and delivery through different communication channels.

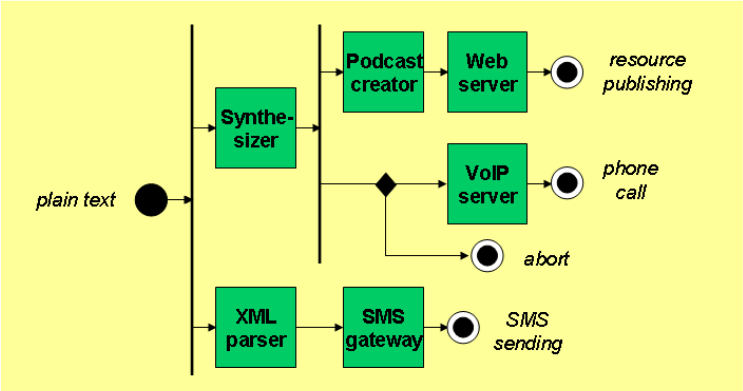

Illustration 2: A sample workflow to realize content transformation and delivery

Chapter 2 – Architecture

Our approach to support Ubiquitous Internet issues strongly relies on the idea to disappearingly integrate final user activities and available distributed services into coherent resource compositions, by means of a middleware coordination platform. At the same time, we abstract resource heterogeneity by means of managed proxy adaptors to potentially introduce any kind of additional integration functionality that business processes from actual application scenarios may require.

By means of proxy-based resource management, we relieve service developers as well as final users of the burden of software integration and let them concentrate on their own needs. Furthermore, thanks to the central role played by our platform in the orchestration of business processes, we endeavor support for changes in user requirements and service conditions in seamless ways, enabling dynamic reconfiguration of their business processes and automatic selection of the resources that participate in them.

2.1 Component model

Most current middleware solutions adopt a layered architectural model and focus on enriching the middleware itself with dedicated features that services and users in turn exploit to face mobile and pervasive computing challenges. But when the number of such features increases, and perhaps they need to interact with each other, middleware complexity inevitably grows, making this approach inadequate for facing the wide domain of Ubiquitous Internet support.

In our opinion, the only viable approach for dealing with increasing complexity consists in simplifying middleware design by leaving it only the core of management and coordination functions, and by moving ubiquity feature logic outside its layers. As a result, the middleware architecture we propose still adopts a layered model, because of the clear definition of

dependencies that it provides, but it permits to simplify the middleware itself by applying a pattern of delegation.

We introduce entities, the proxies, that are responsible for modeling mobile and heterogeneous resource diversity and to provide a unified lifecycle management model; then, we exploit the well-established resource representation that proxies provide to delegate to such kind of non-middleware pieces of business logic all of the content-related activities (e.g., generation, transformation, adaptation, delivery, ...) that otherwise would lead to middleware sophistication. Middleware role hence becomes that of abstracting actual resource distribution and enforce business processes that entail those resources, to pursue the desired Ubiquitous Internet scenarios. By doing so, middleware offers to resource proxies a suitable (but minimal) set of facilities that can overcome mobility and heterogeneity problems, and it provides for effective means of describing resource proxy functional and non-functional characteristics, in order to enable automatic composition of those proxies into business processes that can fit the scenario requirements.

In the following, we report the general architecture schema of our middleware solution (Illustration 3).

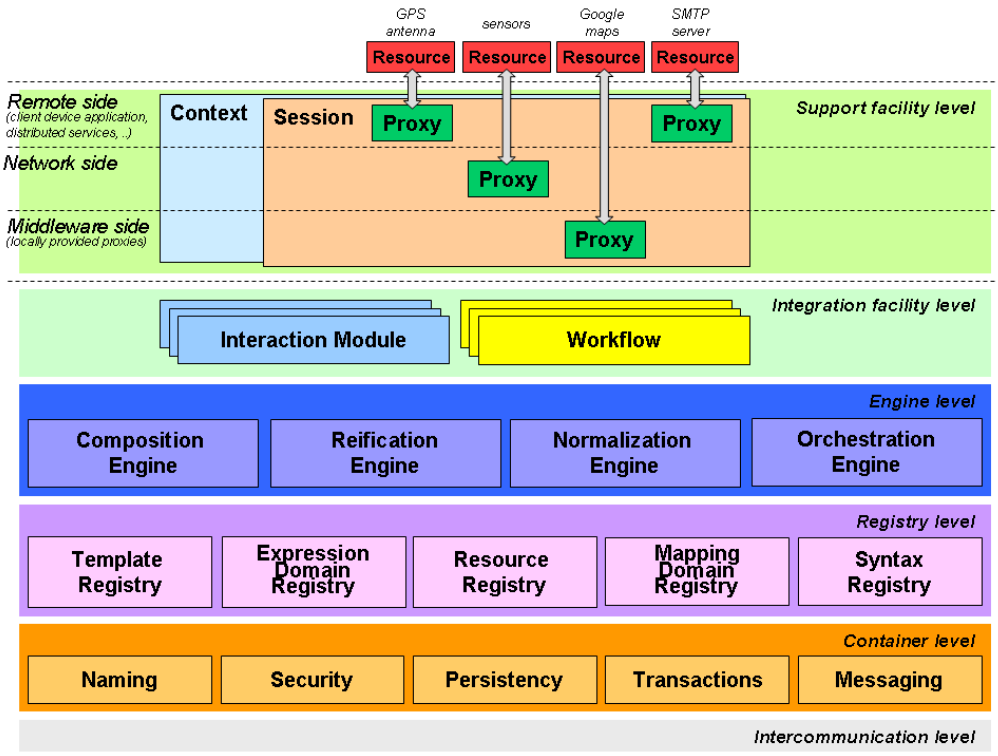

Illustration 3: Middleware architecture

Middleware components divide into separate levels of responsibility, dedicated to well-define and effectively face the diverse aspects of resource communication, management, and coordination. The different parts in the depicted architecture can be briefly described as follows, whereas the most relevant ones, providing resource integration and support facilities, will be stressed in the following Chapters, by deepening the analysis of innovative concepts and design principles they base on. In details:

- ***Resource level:*** resources can be distributed services that are available over the Internet as well as applications running on client devices, sensors, legacy appliances, or whatever;
- ***Proxy level:*** proxies enable resource management and exploitation by the middleware and grant access to its support and integration facilities;

- ***Support facility level:*** middleware maintains context and session information that are always available for direct use by the proxies;
- ***Integration facility level:*** resource interaction with middleware business processes and resource orchestration are both enabled via software components that middleware can dynamically plug-in and exploit to support application scenarios;
- ***Engine level:*** engine components provide implementation of functionalities that are exploited by higher level middleware parts, to face the issues of resource and business process management and actuation;
- ***Registry level:*** registries maintain the knowledge basis for engine operations;
- ***Container level:*** typical features of SOA frameworks are usually provided out-of-the-box by the run-time execution environment, often called the *container*.
- ***Intercommunication level:*** facilities such as remote method invocation, clustering, caching, marshaling, and more, help masquerading actual resource and middleware component distribution.

On top of traditional SOA mechanisms, we conceptually model, represent, and maintain within registries, all pieces of information that characterize current resource composability requirements (*Template* and *Expression Domain Registries*), status *(Resource Registry),* relations *(Mapping Domain Registry),* and formats of data they exchange with middleware *(Syntax Registry)*. Registry components have the sole responsibility of providing the knowledge basis that enable higher level operations.

Engine components, instead, implement the middleware logic that deals with resource lifecycle management *(Reification Engine),* composition into business processes *(Composition Engine),* and invocation according to those

processes *(Orchestration Engine).* Besides, engine level (i.e., *the Normalization Engine)* also provides means to normalize the heterogeneous information that resources communicate to middleware, by translating them to commands that middleware itself can understand.

Integration facilities come in the form of two different flavor of pluggable middleware components. *Interaction Modules* and *Workflows*, respectively, enable resource interaction with middleware-aided business processes and model middleware orchestration of resource-provided logic (in both cases, via their corresponding proxies).

Support facilities are available for direct use by the proxies, via a suitable middleware Application Programming Interface (API), in order to provide them with reliable and effective management of both *Context* and *Session* information;

Finally, *Resources* represent virtually any kind of functionality that middleware is able to manage, compose and orchestrate to foster Ubiquitous Internet application scenarios. Middleware interaction with each resource, anyway, is always mediated by its corresponding *Proxy*, to grant uniform representation and consistent lifecycle management, for the sake of business process modeling and enacting.

2.2 Overall mechanisms

By adopting our architectural model, developers of both client and service software are left free to concentrate on their specific goals, whereas the opportunity to proxy actual computational resources can introduce support for all Ubiquitous Internet issues to consider. Furthermore, this permits integrating also existing client applications and legacy services. For instance, it is possible to intervene on communications from existing browsers via traditional HTTP proxies, to save session information or enable content adaptation, say, to fit current bandwidth; similarly, suitable proxies can mediate requests towards legacy services to enforce specific user preferences or to parametrize them by exploiting middleware-provided information.

This allows to focus the attention on orchestration design, and shifts complexity from the programming of distributed software modules that have to integrate with each other to the sketching of the overall business process they will be involved in. According to this approach, business process architects must draw on resource capabilities to orchestrate suitable business processes; then, provided the role and facilities that proxies can play, architects assign to proxy developers the tasks needed to accomplish their vision. Dealing with existing resources, proxies will pursue integration of legacy assets according to middleware requirements and the described middleware support features; instead, when developing brand new resource types, proxy developers will be able to state their syntactical and semantical behavior and have other programmers apply on it. Finally, to enable final users to become architect of their own resource choreographies too, middleware platform offers simple means to make proxies work together in business processes and have those business processes run, either on middleware or on non-middleware components initiative.

Aside already mentioned proxies, we therefore introduce and leverage additional middleware functionalities to act as glue that makes pieces get along together, keeping at the same time things clean and responsibilities separate, to avoid unnecessary interdependencies. By adhering to definition in [Ber96], they constitute the “general purpose software that sits in between, providing functionalities and facilities that do not tie to any particular scenario” and that “is not an application itself or a specific-purpose service”. In details, the middleware architecture we propose provide means to formally define what a business process is, the goals it pursues and the constraints it has to satisfy, the kind of workflows it entails and the actual resource proxies that take part in them. It permits automatic selection and configuration of the resource proxies to involve in the process and grants safety, by avoiding incompatible resources to be arranged together. When orchestrating a process, then, middleware infrastructure performs resolution of resource proxy invocation parameters, and it enables and supervises message passing among

cooperating proxies. At the same time, middleware monitors process status (e.g., resource availability, context and session information) and it reacts to changes that dissatisfy requirements that have driven its definition. Finally, it provides means to expose workflows of existing business processes as convenient facilities that other resources and processes can invoke, in turn.

We maintain, anyway, that middleware intervention must not be intrusive: neither in terms of the supplied API and the explicit dependencies in code that it entails, nor as far as the set of interaction paradigms that it supports. In our opinion, middleware has not to drive service development, and not even to force behavior of the final users. That is why our system totally disappears in the background, coordinating and orchestrating resources that can be completely unaware of the overall business process they are participating: by supporting communications and by providing integration facilities that achieve location transparency we abstract the actual distributed processing environment to resources that we let compose.

For instance, context information in Illustration 3 is made seamlessly available to all resource proxies of a business process no matter their actual location, as well as means to accept requests for the execution of one business process do not depend on the location of resources involved in that process. As for proxy development, then, we provide simple session and context management API, but do not oblige proxy themselves to implement any other particular programming interface. Rather, we enable integration by means of metadata that proxy developers can provide to describe features, constraints, dependencies and so on (Illustration 4). Finally, thanks to metadata again, we let proxies associate methods they expose to moments of their lifecycle, as managed by the middleware, and map invocation arguments to values that our system can resolve and provide as actual invocation parameters.

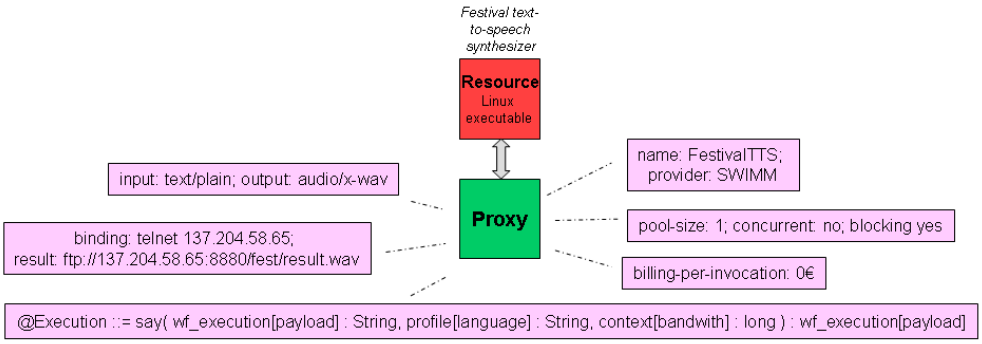

Illustration 4: Example of metadata exposed by a proxy

Involvement of resource proxies in business processes, to realize arbitrarily complex Ubiquitous Internet scenarios, is a consequence of defining workflows that entail invocations of their methods and result passing among them. From their own point of view, proxies are not aware of being interacting with other resource proxies, and not even of being part of any business process.

When playing a servant role, proxies just perceive invocation by some external client that they can serve by exchanging messages with. As Illustration 5 reports, this is typically what happens with services like a text-to-voice synthesizer or a media file streaming server. The former one, indeed, is clearly a stateless service that supports one-shot request/response message exchange pattern, returning synthesis results upon input arguments; it does not really matter who is requesting service and who will further process its results. The latter one is instead a stateful and connection-oriented service that enables streaming on-demand; middleware orchestration simply makes this possible by commanding its proxy appropriately. It has to be observed that establishing direct connections with clients to download media is inherently part of the streaming server core business logic. There is maybe a subtle distinction between proxy direct interaction and resource direct one, but it is crucial to understand this as a key element to achieve expressiveness and separation of concerns. While proxies have to be kept separate and decoupled, offered simple API when necessary and disappearingly integrated with each other,

resource interaction is instead sometimes strictly necessary and useful and cannot be avoided: on the contrary, it has to be effectively enabled by the middleware.

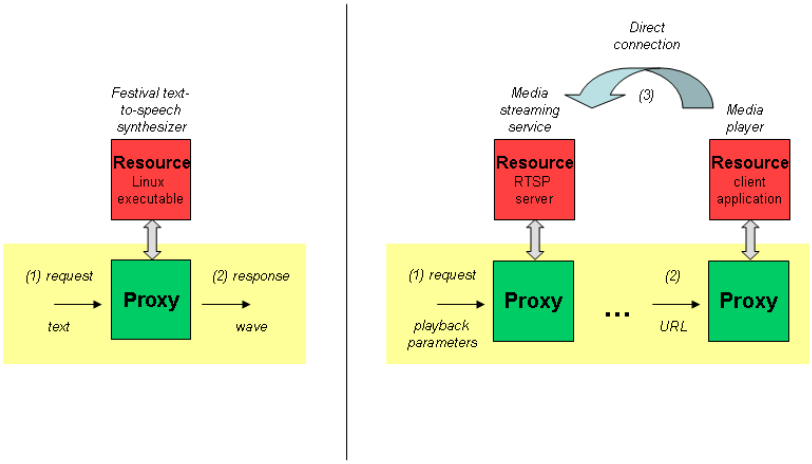

Illustration 5: A text synthesizer service proxy, and a streaming server one

From a totally different perspective, resource proxies that play active roles in processes do not need direct interaction too. This is what happens, for instance, with a news service that causes sending of SMS messages, or with a browser requesting customized news pages. Middleware has only to expose suitable ways of enacting workflows of the desired processes, to support the diverse interaction paradigm that proxies can leverage. As Illustration 6 shows, news service is not interested in results: it just needs one-way message exchange facilities towards the middleware; middleware, in turn, evaluates its message content and enacts a workflow from a business process able to convert news, say from RSS to SMS format, and perform delivery via an available SMS gateway. Conventional browsers, instead, adopt a request/response message exchange pattern, and wait for results. And obviously, there could be also scenarios that expect conversational patterns, connection-oriented simplex, duplex, and publish/subscribe ones, and many others [MomentumA][Gor05].

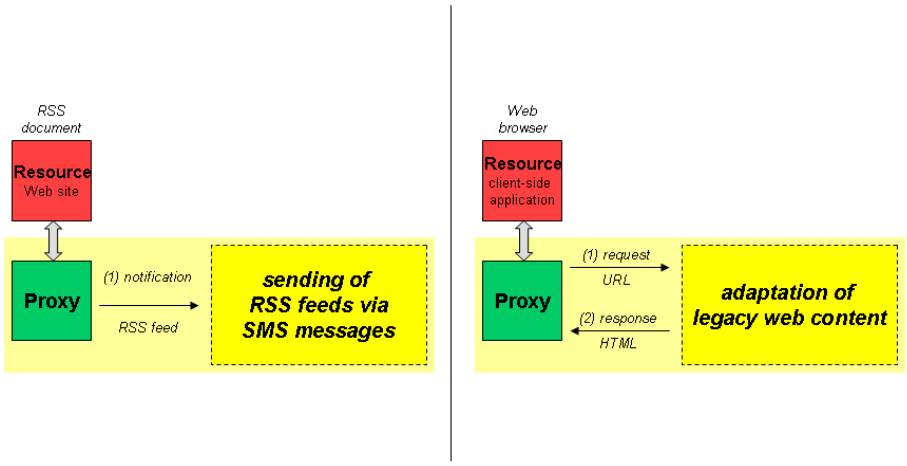

Illustration 6: A news service proxy, and a Web browser one

Rather than modeling and supporting as many interaction paradigms as possible *a priori*, we argue that middleware must be extensible and provide pluggable means of exposing business process workflows, and allow support for additional message exchange patterns in time. Besides, while still considering middleware flexibility and extensibility as crucial requirements, we also claim that composition and orchestration can actually disappear from the user and service point of view, and become automatic, given the set of business process goals and constraints to satisfy.

To deeply investigate workflow-based business process enactment and metadata-based resource support that our system provides, following chapters will stress conceptual model and mechanisms that demonstrate feasibility of our approach. In details, Chapter 3 will stress lifecycle and management model that middleware adopts to provide resource proxies with suitable context and session support and to enable their participation in business processes. Chapter 4, then, will deepen analysis of the integration facilities that permit resource interaction with middleware business processes and modeling of business process logic itself.

Chapter 3 – Resource Reification Model

We provide uniform representations of both client software applications and services by introducing the notion of managed resource proxy, to abstract on their different functionalities and conditions. In our vision, proxies are nothing more than simple means to represent diverse entities that can show analogies and be managed similarly.

The concept of proxies allows assembling elements of Ubiquitous Internet applications in an easy and uniform way, just if they were LEGO® blocks with well-defined characteristics. It does not matter whether they are local or remote, stateless or stateful, available or not: by means of a resource proxy we provide an object that can serve as an endpoint for sending data, to identify the owner of other resources, and as a storage box for saving feature descriptions and information on status. In our system, communications among resources always happen via their proxies, and integration and composition of resources is expressed in terms of integration and composition of proxies.

Besides, proxies undergo middleware management since they enable lifecycle operations, according to a predefined *Resource Reification Model* (RRM). In details, we adopt a 7-steps model that demonstrated to be highly flexible and general: not forcing resources to adhere to it, but mapping to their own lifecycle when due, or enabling additional configuration via their proxies otherwise. Management takes place via the so-called *Reification Engine* component of middleware *Engine level*; resource proxy characteristics are then stored to the *Resource Registry* component in middleware *Registry level.*

Adhering to the 7-steps RRM depicted in Illustration 7 simply requires resource proxies to support the following operations:

- ***Registration***: publication to the system of resource metadata, describing properties that are useful for integration purposes and for resource involvement in real ubiquitous computing scenarios. From this moment on, the resource is potentially available to the system for

orchestrating business processes that comprise it.

- ***Activation***: loading and initialization of a proxy instance for that resource, representing the endpoint to be used to communicate with it. After activating, proxy instance conveys features such as location and availability information of the actual resource, and provides concrete implementation for its business interface.
- ***Configuration***: behavior setup of a proxy instance for a specific business process. Every single business process that middleware orchestrates reserves (and binds to) a particular proxy configuration of each resource it leverages. Resource proxy can directly enact configuration on its corresponding resource, when supported, or permit it by simply storing configuration information for use during actual resource invocation.
- ***Execution***: enactment of actual business logic that resource provides, through its configured proxy, within a particular business process.
- ***Deconfiguration***: discarding of a particular resource configuration; this happens when the system discards the business process that was reserving it.
- ***Deactivation***: discarding of a particular resource proxy; this can happen when no more business processes in the system reserve configurations from that proxy and it always happens in case of failure of the host where the proxy resides and/or in case of deregistration of its corresponding resource (forcing passivation or reconfiguration of the business processes that leverage it).
- ***Deregistration***: deletion of resource metadata; performed in case of resource unavailability or withdrawal by its provider.

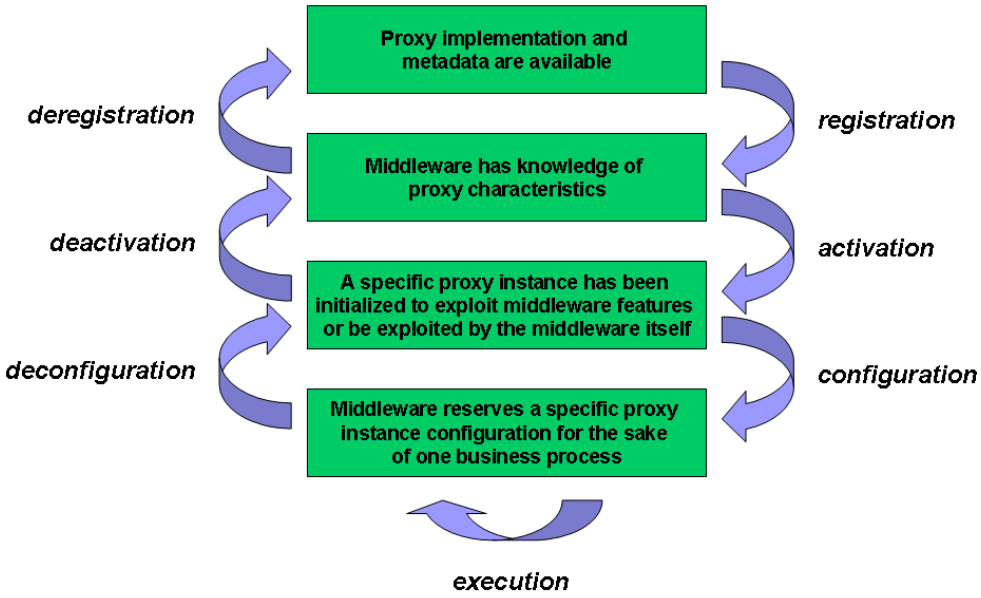

Illustration 7: Resource Reification Model

By means of RRM, system can uniformly treat both users and services as resources to manage and leverage by need, enabling a consistent and uniform abstraction of business process participants.

For instance, providing a CORBA service that converts text to Mp3 files is achieved by registering a resource that performs voice synthesis, as its metadata describe (Illustration 8). Let's now assume that service is stateless, that it can be parametrized in terms of language (influencing word pronunciation) and bitrate quality, and that it is physically located on a German server. Proxy instances can activate on any convenient system node, optimizing business process communications and permitting message reliability and retransmission even if remote service does not natively support that. Furthermore, each proxy can provide different configurations for use in different business processes, for instance “English@320kbps”, or “Italian@160kbps”. As the business process that leverages the Italian configuration needs to synthesize text, proxy stores and forwards its request/response messages and commands the remote service according to its

stored configuration.

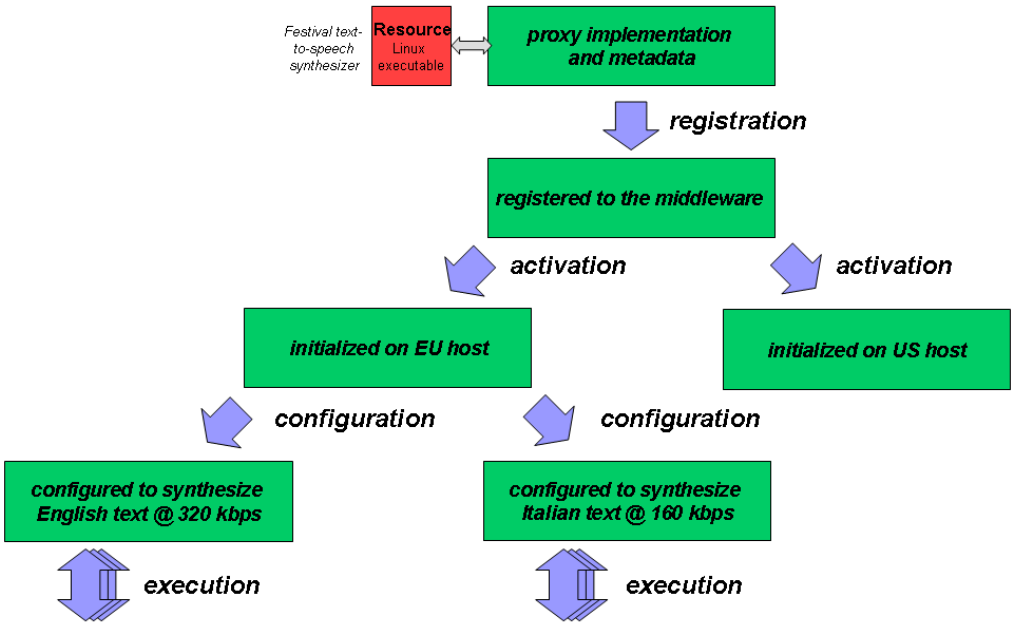

Illustration 8: Reification of a voice synthesis service

On the user side, browsers used to render an online newspaper page are actually resources too. And, in particular, every new supported browser type is a fully-fledged resource that can be registered to the system and perhaps manipulated in terms of supported formats, display resolution, font size, and so on. A user logging onto the newspaper site by means of such a browser commands activation of the corresponding resource proxy. And since every user can have different preferences, maybe depending not only on the browser she's using (e.g., OperaMini [Opera] on her Mobile *versus* Mozilla Firefox [Firefox] on her PC), but also on her current conditions (say the connection type in use: e.g., Wi-Fi versus UMTS on the same mobile phone browser), they can configure their proxies to behave differently in different business processes. Each and every time a user request the online newspaper homepage, the proxy she leverages executes and exploits its configuration to format HTTP responses.

And what if the online newspaper would like to embed spoken versions of its textual news? There you have a hint at what compositions of suitably configured resources in different business processes can achieve.

It is worth insisting on the fact that RRM does not drive resource characteristics, but it instead allows for them, by being as general as possible. Resources that can maintain status and/or be configured are inherently admitted, as *Configuration* step lets different business processes bind to proxy objects that behave in different and customized ways (perhaps also conversational or connection-oriented). In this case, proxy configuration directly "maps" and "is forwarded" to the resource one. Stateless resources are supported too: in case business processes need configuration, proxies will just save configuration on their own and use it to parametrize actual resource *Execution*.

Besides providing a powerful resource abstraction model, RRM also enables fault tolerance and load distribution in simple ways. Indeed, model does not describe a linear sequence of lifecycle steps – with one resource traversing successive states after one another –, but rather it leads to a tree-like generation process, that permits multiple reifications of the same resource as well as the coexistence of reification trees from multiple equivalent resources.

To clarify this, every resource becomes available by means of its metadata *Registration* event. Then, one or more resource proxies perform *Activation*, possibly on different hosts, to concretely represent that one resource in the system and enable communication with it. Resource inclusion in business processes is possible by means of proxy behavior *Configuration*, and the same proxy can provide different configurations in different processes. Finally, configured resource proxies can perform *Execution* several times, upon events, direct invocation or, simply, on their own. As long as the referenced resource is available, system can optimize communication of business processes and status management among different network nodes, and even adopt strategies to migrate proxies and proxy configurations from one node to another in case of local failures. Furthermore, model transparently

enables multiple equivalent resources being registered to the system. Every such resource just provides its own metadata and system lets business processes bind to the proxy of the most available one, on the basis of resource availability information that proxy themselves provide.

3.1 Session management

To support complex and conversational communications, beyond simple request/response message exchange, resources that cooperate within business processes need to preserve status for the operations they are running. Moreover, to let those processes span across time and distributed network nodes, resources also need to establish interaction sessions and to maintain information about them. Session, indeed, can be seen as "temporary confederation of one or more parties" for performing "negotiated and cooperative" activities [Mak94].

As an example, buying at an online shop via the browser consists in successively adding items to an electronic shopping cart, and finally let the remote shop application process it to calculate total costs, enact shipment and update stock. Cart information is usually stored in server memory until order is confirmed, aside information regarding active carts of other users. Every browser, hence, can retrieve and modify its own cart by labeling request messages with the session identifier it has initially agreed on with the server. In this case, cart description and the identifier do represent the session information that browser and server need to collaborate.

Managing session information, anyway, does not just support simple use cases like that, but can actually empower much more complex scenarios. Dealing with Ubiquitous Internet, for instance, also mobility problems arise and integration of resources moving in space and time becomes harder. As a matter of fact, business processes have to allow for device disconnection and reconnection, possibly from different network addresses, and even for user changing the device they use, while maintaining a consistent view of their ongoing activities [Bel03]. Moreover, distributed and fault-tolerant SOA

implementations can expect several service replicas to provide the same kind of service, perhaps varying the one to exploit on the basis of proximity, availability or quality-of-service (QoS) constraints. For instance, back to Arianna music service, it is clear that only suitable session management permits mobility of both terminal (from Wi-Fi network to 3G connection) and user/service resources (from her smartphone to her PC).

We strongly believe that as long as resources cooperate with each other to realize complex scenarios, they also need facilities to deal with status of their interactions and session information scoping to allow simultaneous use of shared resources in multiple processes. For instance, suppose to let users subscribe to a news service where they can choose any kind of RSS news source and where messages are triggered at a predefined moment of day. Sure, some kind of RSS reader is needed to retrieve RSS feeds from news sources. Users who read news via dedicated applications (e.g., Mozilla Thunderbid [Thunderbird]) can keep track of the news they already received via a text file on their own device. In case they exploit some web interface to do so (e.g., Google Reader [GReader]), session can remain on the client device, leveraging browser cookies. But what if users like to get new available feeds via SMS messages? An hypothetical RSS to SMS converter resource has no means to read past messages on the user phone before sending new ones, so it must save session on its side, and maintain separate news histories for different users! Actually, orchestrating business processes out of distributed resources makes effective session management an absolutely crucial issue.

In our model, we leverage proxy entities to associate session information with actual resources and we enforce proxy functionalities to retrieve this information and use it in actual invocation of resource logic. For instance, Illustration 9, below, demonstrates how proxy of the RSS reader service from the previous example can leverage session for storing and reading relevant information for its own execution. Besides, we impose no predefined semantics on session information, but let resource proxy implementations free of making the most suitable usage of system-provided session information, either by forwarding it to final resources when supported, or by using it to

invoke session-agnostic resources accordingly otherwise.

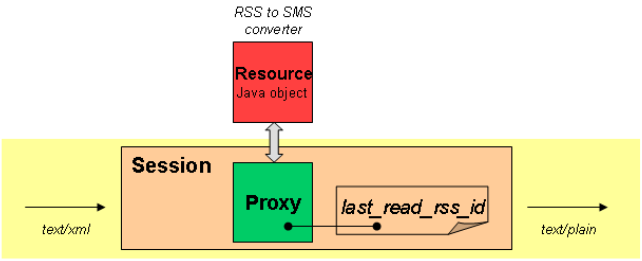

Illustration 9: An RSS reader proxy leveraging session to read and store information

By leveraging RRM, we enable session scoping support for resources that are involved in different business processes. As Illustration 10 reports, this is done by simply applying different session facilities to the different proxy-related states that RRM entails:

- ***lifetime span***: session information that "belongs to" and can be "referenced by" all proxy instances of one resource. In other words, information that is available for execution of business processes of all activated and configured proxy instances for that resource.
- ***proxy instance:*** session information shared among business processes that leverage configurations from one single proxy instance. Although not relevant for the design of business processes, proxy developers are encouraged to store here session information that is relevant for proxy instance activation, so as to enable failover mechanisms.
- ***active configuration***: session information that spans multiple executions of the business process to which one single proxy instance configuration belongs.
- ***current execution***: session information that is valid only within a single execution of the business process to which the proxy instance configuration belongs.

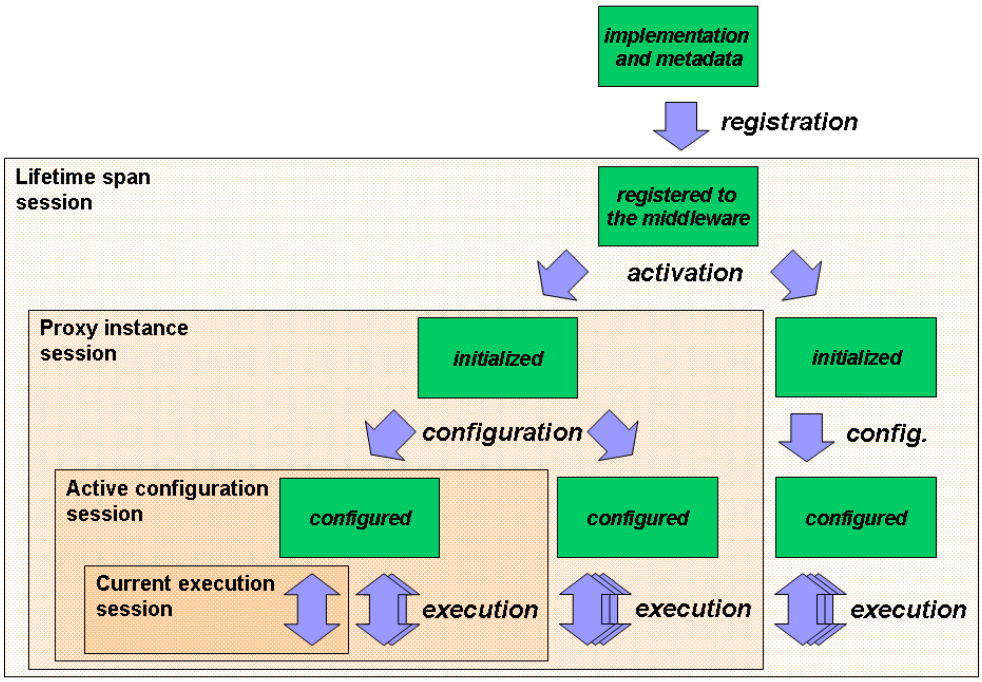

Illustration 10: Different scopes of session information

Examples apply to demonstrate approach achievements (Illustration 11 and Illustration 12). Let us consider Arianna story again, and the possible distributed resources and session information that it entails. The "connect-and-play" business process (or, better, "reconnect-and-resume-playback") obviously expects something like a streaming server that provides media files, and a client-side software module connected to it that decodes stream. Then, a remote playlist manager can enable mobility by holding playlist information and commanding the legacy streaming server as a consequence. "Connect-and-play" execution leads to connection establishment between media server, client, and the playlist manager, to enable download and song playback. Streaming server and the decoder module, in this case, have to connect with each other directly, in order to manage data flow. Incidentally, notice that to overcome problems like NATs, firewalls and alike, they can do that by sending SIP signaling information through their proxies [Pan04]] and exploit proxies themselves to traverse NAT, too.

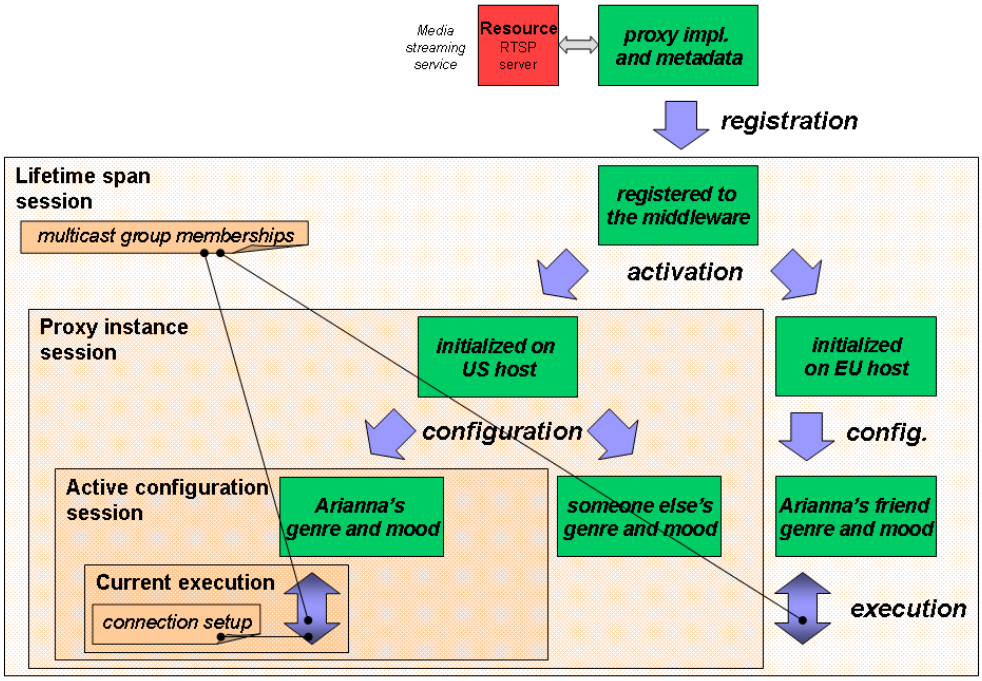

Illustration 11: Session scopes for the streaming server in the Arianna example

Obviously, information that lets server and client keep the connection up is only valid within the "*current execution*" of the "connect-and-play" process. Instead, Arianna's preferences and the current song she is listening to are pieces of information that are essential to resume playback, despite network disconnection/reconnection and device change. In particular, playlist manager keeps the latter one up-to-date, so that it can serve for playing resumption upon every new reconnection; this is "*active configuration*" session information, hence, for use by resources involved in successive "connect-and-play" executions. Finally, the most general session scope is what enables information sharing across same-resource proxies in different business processes. In this case, streaming server proxies might be programmed to act as members of IP-multicast groups. Thus, new friends of Arianna can join her by means of the streaming server proxies that take part in their own "connect-and-play" business processes, simply by having them access her IP-multicast group information, located in the "*lifetime span*" session of proxies of the

unique streaming server resource they are about to share.

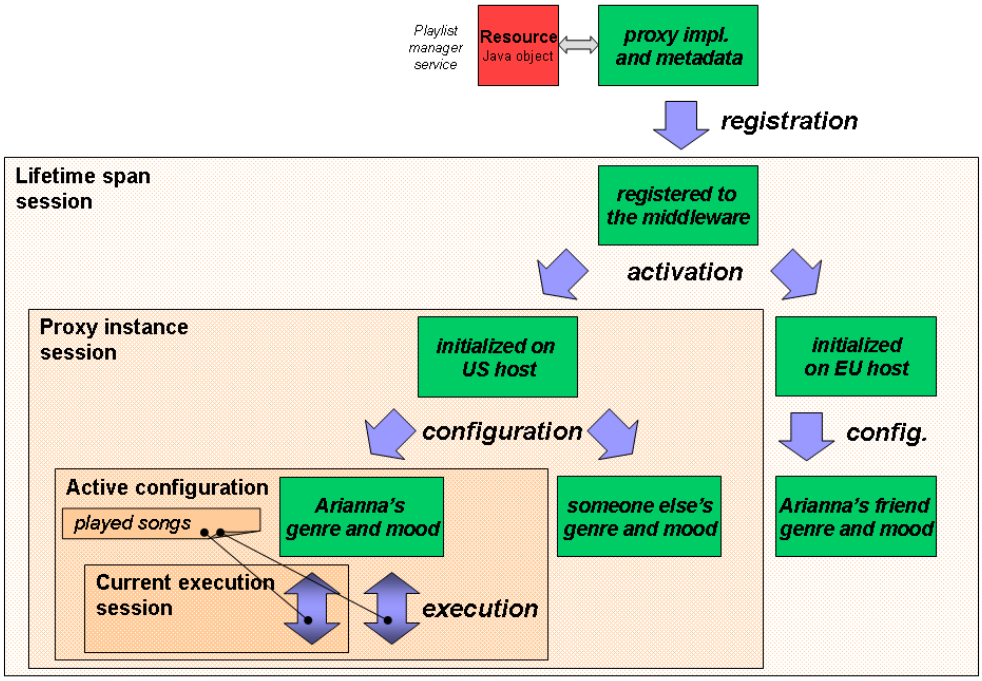

Illustration 12: Session scopes for the playlist manager in the Arianna example

Two more things it is worth to highlight here. First of all, proxies and the session model provide support for all kind of scopes and do not pose constraints on the kind of session data that resources wish to use. There are no predefined data format nor wrapping objects: session scopes are in all similar to reliable tuple spaces where proxies can save interaction status. Secondly, system does not impose the usage of a particular session scope. Proxy instance implementations are free to choose the scope(s) to use on the basis of the desired scenario to enable.

To provide another example, let us consider a user leveraging the browser-based version of the aforementioned RSS news service. Resources taking part in the process are just the user browser and the RSS reader service, and it is possible to create multiple "news-aggregation-set" processes by just specifying different preferences for each one of them. Web pages (or page

fragments) corresponding to different URLs are created to show content from the different processes, and user commands process execution by requesting one of these URLs. Browser message exchange pattern is request/response, and no connection is established: "*current execution*" scope is not used. Then, by leveraging the "*active configuration*" session scope to store preferences, user can run the processes simultaneously and display results at the same time in different browser tabs or page sections, perhaps to embed in other web sites.

3.2 Context-awareness

One of the goals of context-aware computing is to "acquire and utilize information about the context of a device to provide services that are appropriate to the particular people, place, time, events, and so for" [Mor01]. Concrete examples of such service opportunities already are all around us, ranging from conference *vs.* theater *vs.* street profiles of our cellular phones to GPS navigators. Depending on physical, social and computational environment conditions, we can experience different kinds and qualities of traditional services and enable brand new ones, too. Besides, leveraging context also represents a key element in the attempt to seamlessly embed computation facilities in everyday life: indeed, as services become able to adapt to context by themselves, minimal effort is needed on the user part and technology can disappear in the background.

In our view, producers and consumers of context information must not be involved in management and transportation of it, since they are often separate entities (e.g., sensors and monitoring applications) and their roles and responsibilities must remain distinct and focused on their respective goals. To achieve this, we provide configured proxy instances with simple context blackboard functionality that is globally accessible from all proxies that belong to the same business process. Blackboard entries are always available for context consuming resources via their proxies, and at the same time they also allow simple read/write access for context generating ones, such as sensors, client-side monitoring applications, server-side services, or even

infrastructure-side entities (e.g., programmable WLAN access points, GSM base transceiver stations, and so on).

By means of proxies, resources have not to deal with context management directly. For instance, proxies of RFID sensors can just poll such resources in time or be notified by them, depending on sensors API, and then write sensed information to context. Context-leveraging resources, such as an alarm bell to prevent shoplifting, can have their proxies read context information on their behalf and command them accordingly. Similarly, as in Illustration 13 below, several GPS antennas can write coordinates to context by communicating them to their proxies, while a sole geographical application can leverage coordinates from context to draw points-of-interest on a map. Again QoS measurements can take place, leveraging context as a drop box for their results; then, services that can react accordingly will read information from context and make their decision: for instance, to downgrade audio quality when Arianna uses her 3G connection!

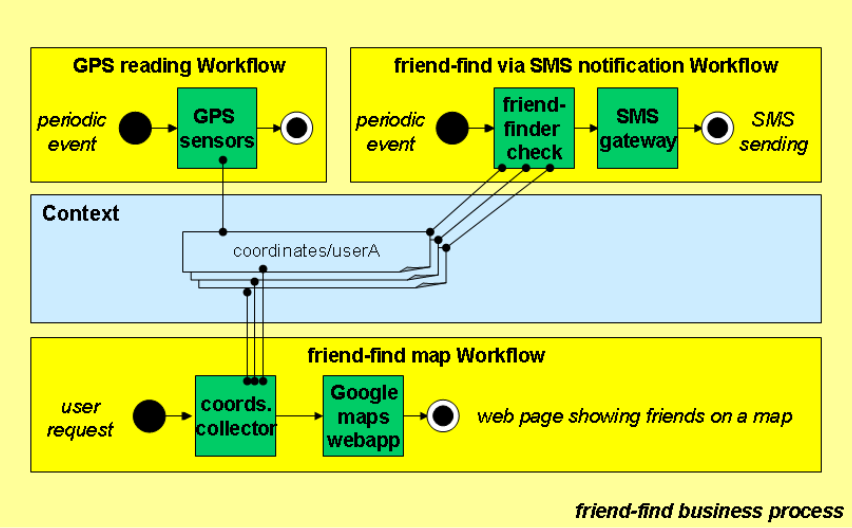

Illustration 13: Context exploitation within a business process

We argue that to achieve effective context support and extensibility in time, the intermediate software layer that is responsible for context management must know nothing about context representation, *a priori*. Hence,

although addressing different semantic issues than session, proxy-aided context handling resolves to nothing more than tuple spaces provisioning, too. Anyway, while purpose of session support is to enable interaction status management from one-single resource point-of-view, context support inherently aims at enabling cooperation of space- and time-decoupled context consumers and producers.

To demonstrate this, let us go back to the online shop example: cart content and its association with a specific customer identifier are server-side pieces of information, while client browser just holds the identifier one. Back to the passion of Arianna for nonstop music playing, playlist manager intervenes in process to keep track of playlist progress, while media streaming server just plays what it is told to: they don't share information, but each of them deals with the information fragments it needs to work with the other. And the same applies to the socket technology that enables server-client streaming: each endpoint is storing information on its own: there is no “singleton data” describing the established connection.

On the contrary, context information is inherently shared by resources, hence they need common facilities to interact with it. Context scope is set to correspond to the collection of resources belonging to the same business process, because it is within one business process boundaries that context production and consumption take place. Nevertheless, this does not prevent different business processes to leverage the same context-information, since proxy-based RRM trivially supports this scenario, too: there is no need for multiple resources generating the same context-information, but just for multiple configured proxy instances of the actual resource that generates the information.

Chapter 4 – Integration Facilities

The middleware architecture we propose provides users with extremely flexible and extensible ways of accessing contents and services, no matter the communication channel in use, the user interface they choose and the interaction paradigm that it demands, and not even the customized user preferences and inherent device capabilities to be considered.

On the one hand, we delegate application-dependent logic to external resources (e.g., content retrieval, transformation, dispatch, ...), in order to move it outside middleware functionalities and leave only coordination and management responsibilities to the middleware itself. On the other hand, we clearly and neatly separate into diverse software components the concerns of providing convenient user interfaces, supporting different interaction paradigms and orchestrating managed resource proxies to process and transform content in suitable ways.

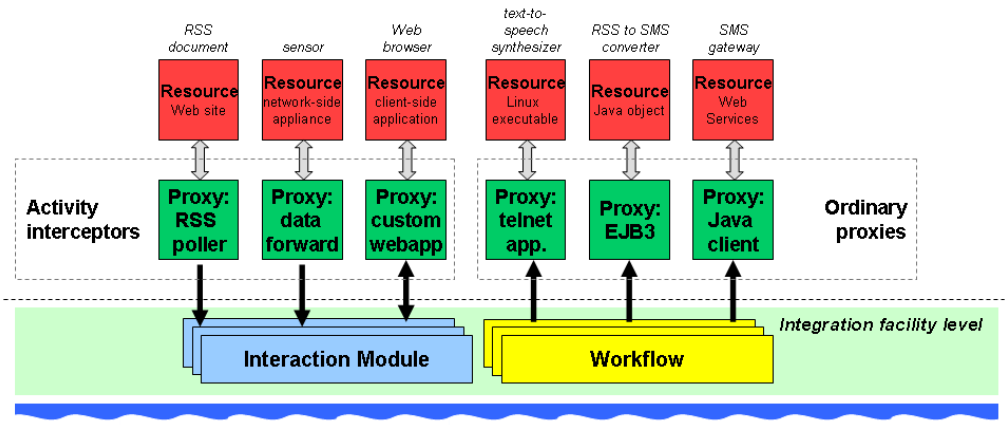

Illustration 14: Integration facility level

As Illustration 14 shows, we introduce workflow entities to describe the business processes of the resource proxies we compose. Furthermore, we denote by the name *activity interceptor* every kind of resource proxy that is able to directly interact with the middleware, via a specific interaction module,

to communicate relevant information about its resource, such as commands and selections on some kind of user interface, sensor measurements, incoming messages through a given service gateway, and so on. In details:

- ***Ordinary Proxies:*** represent managed resource proxies that are not aware of participating to business processes that middleware orchestrates. A part from exploiting middleware context and session management facilities, they just expose suitable methods for invocation, in accordance to RRM lifecycle steps;

- ***Activity Interceptors:*** realize a particular flavor of resource proxy whose goal is to have middleware run previously configured business processes. Heterogeneous resources (i.e., not only client side applications, but also interactive web pages, SMS gateways, and any kind of service) can therefore trigger the execution of one or more of these processes by conveying, through their proxies, explicit requests as well as any kind of information about their ongoing activities;

- ***Interaction Modules:*** support the different communication patterns through which interceptor requests can interact with middleware business processes (e.g., request/response message pairs, request-only ones, conversational patterns, connection-oriented data flow, and so). Besides, Interaction Modules intervene on such requests to analyze the information that they convey and to command middleware facilities accordingly;

- ***Workflows:*** provide the description and support data structures for the business processes that middleware lets define by means of resource composition. By leveraging workflows, it is possible to orchestrate multiple resource proxies to serve an interceptor requests, in order to retrieve, transform and deliver the desired response content according to the most suitable format (e.g., text, audio, ...) and communication channel (e.g., HTTP, SMS, e-mail, digital TV carousel data, etc...).

As well as external resource proxies can register to the system at run-time and take part in novel application scenarios, afore mentioned middleware coordination components can easily plug in by need too, thus allowing incremental support for additional means of interfacing, interaction paradigms and resource compositions.

4.1 Multimodal interfaces

Historically, multimodality relates to permitting different natural input modalities (such as speech, touch, hand gestures, body movements, and more) and coordinating them with corresponding multimedia output [Obr04][Ovi99] [Tur00]. By providing different modal interfaces it is possible to enable users to access the same service from different kinds of device, to gather requests of respective types and to produce suitable results as a consequence, such as contents, side effects, service status modifications, and more.

In our vision, we consider interfaces as fully fledged resources, with associated metadata and proxy objects that can abstract heterogeneity and provide management-, session- and context-related features, according to RRM. Besides ordinary behavior, precise goal of this kind of proxies is to:

- intercept information about ongoing activities on the actual interface in use;
- forward such information to the middleware, along with format description;
- provide results to the actual interface, if expected.

Final users and software developers hence can exploit any kind of interface to interact with the middleware, since it actually constitutes an ordinary resource from the system point of view. Corresponding proxy gathers information from it and then applies for further middleware-aided processing.

To provide some examples, intercepted resources can be remote services as well as web sites, client side applications and user devices in general.

Interceptor implementations range from traditional HTTP proxies (that enable Web navigation on legacy browsers behind firewalls while filtering incoming HTTP requests), to software modules that poll SMS gateways (for incoming messages conveying service requests), digital TV Xlet applications (that react to remote control operations), VoIP server extensions (that deal with tone selections by the users), e-mail daemons, Web Services endpoints and many more.

We believe, anyway, that responsibilities of activity interceptors have to remain as limited as possible, in order to ease their development, deployment, and run-time execution: they are not requested to cope with any kind of activity processing or analysis, but just to forward raw activity data to the middleware. This approach enforces development of highly efficient interceptors, that afford limited computational cost and communication overhead, while avoiding unnecessary integration issues. Moreover, facilities such as authentication and naming – that interceptors would need to evaluate activity information – are not always available at resource proxy level, perhaps due to possible distribution of proxies themselves on client or network nodes where not all middleware platform components are present.

We therefore introduce the concept of "*syntax*" to identify the raw and channel-dependent format of the activity information that each activity interceptor acquires. Syntax indication determines the algorithm through which to normalize corresponding pieces of activity information, in order to extract commands and execution arguments that middleware can exploit to orchestrate business processes.

Every single interceptor can easily provide syntax indication for requests/activities coming from its specific resource and expressed in channel-dependent formats (e.g. HTTP, SMS, e-mail, …) because it simply well-knows the characteristics of data from the resource it is proxy of. Thus, its sole responsibility consists in forwarding pieces of activity information to a suitable interaction module, along with the indication of the syntax to consider for normalizing them. Finally, by exploiting the *Normalization Engine*

component from the middleware *Engine level*, interaction module applies the required syntax-driven algorithm to perform identification and authentication, extract request parameters and select the desired middleware functionality to enact: typically the execution of one or more workflows from a given business process.

To exemplify this, requests typically contain information such as a user-friendly indication of the activity that middleware platform should enact, additional parameters and properties through which to identify the user. For instance (Illustration 15), along with user sending number, an SMS message containing the text *"RSS http://some-news.com/feed.xml 5"* can express the will to obtain the five latest RSS feeds from the given URL. And in the example of Web pages aggregating RSS feeds, syntax for an HTTP GET request for content at URI *"http://more-news.com/aggr?tab=politics"* might be normalized by identifying requester on the basis of the *JSESSIONID* cookie header and the requested resource composition upon the value of *tab* parameter.

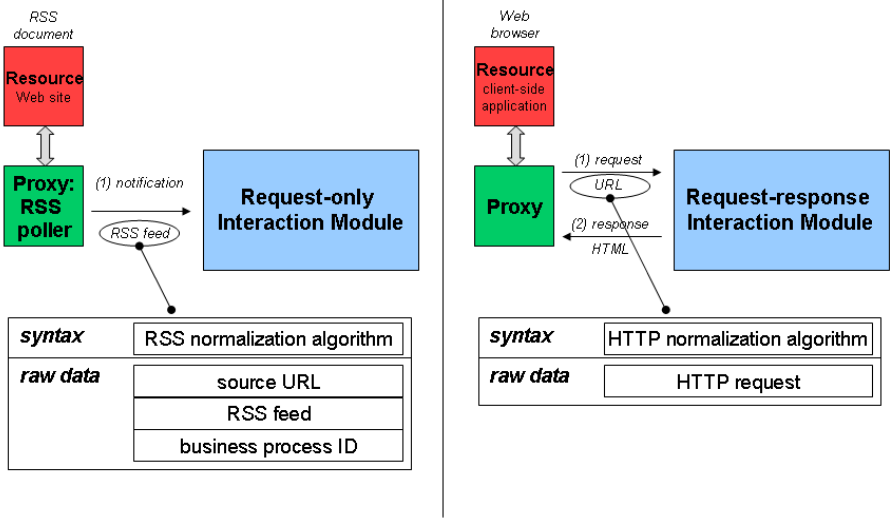

Illustration 15: Syntax labeling

Besides forwarding syntax-activity couples to the appropriate interaction module, some interceptors are also responsible for returning activity results to their own interface-resource, depending on the exploited interaction paradigm. HTTP interceptors, for instance, are used both to receive an HTTP request and to convey its HTTP response.

Moreover, since middleware cannot know every possible algorithm *a priori*, interceptor themselves can teach it new algorithms, by registering syntax name associations with corresponding algorithm implementations. *Syntax Registry* component from the middleware *Registry Level* stores this kind of associations and makes them available for use by the *Normalization Engine*.

Finally, as stated before, interceptors are fully fledged resource proxies from the middleware point of view and they can therefore dynamically plug at run-time.

4.2 Multiple interaction paradigms

Supporting multiple interaction paradigms is a direct consequence of providing multimodal access to services, on multiple media channels. Indeed, as long as different interaction forms and media are available, the pull-type request/response message exchange pattern does not certainly suffice alone, but it is necessary to support also push-type communication patterns, conversational ones, and more.

For instance, an HTML form can pass all request parameters to a given service at once, while exposing that service via phone calls must take care of collecting parameters one-by-one, perhaps by having the user dial her choices on the phone keypad. Again, SMS requests, although able to convey all parameters at once, are inherently decoupled from their responses: a service could either send back an SMS or MMS message or store the user subscription for later response delivery, on event occurrences (e.g., notification of goals during a soccer match!). And finally, orchestrating services into business processes that have some form of human involvement often entails

technological and/or functional issues that can influence human-service interaction paradigm.

In our architecture, interaction modules permit modeling the different interaction paradigms through which it is possible to serve different flavors of activities. To realize this, they exploit middleware engine for normalizing incoming pieces of activity information to extract relevant information and enact the workflows from the business process that they entail, while supporting the given interaction paradigm by realizing all needed communication operations (Illustration 16). In details, interaction modules:

- receive raw information data about ongoing activities, along with the indication of the syntax they adhere to, hence the suitable normalization algorithm;
- perform authentication and identification by means of syntax-dependent identification information;
- translate syntax-dependent content of activity information into normalized commands and execution arguments that middleware can understand;
- exploit these pieces of information to execute workflows that belongs to previously configured business processes.
- handle results of such activities and commands, accordingly to the embodied interaction paradigm.

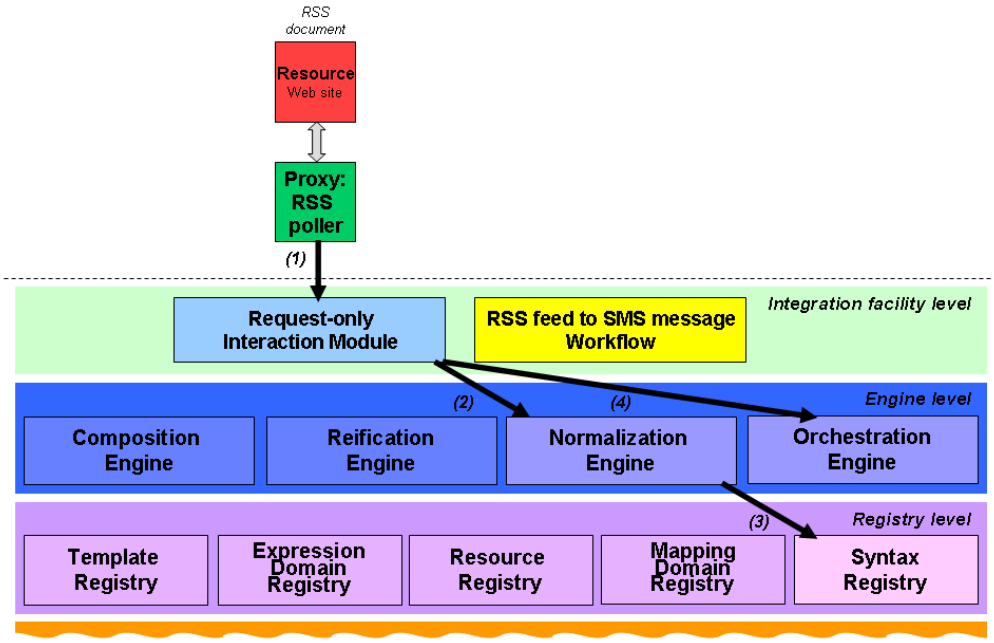

Illustration 16: Interaction module behavior

Thanks to this separation of responsibilities, activities that demand the same kind of interaction paradigm but come in different formats and/or demand different normalization algorithms, can leverage functionalities that are encapsulated in the same modular and reusable interaction module.

To sketch some practical examples, our platform prototype exposes both a pull-based symmetric (request/response) and a pull-based asymmetric (request-only) one-shot interaction module; the former one returns a result through the same interceptor from which the request came, whereas the latter one does not return results at all, meaning that request results (if any) will be delivered through different channels than the request one. We also provide push-type modules, able to monitor and react to virtually any kind of event, for instance time-based ones. Furthermore, we developed symmetric and asymmetric modules for streaming-type continuous interactions.

Finally, consistently with the principle of middleware architecture extensibility, interaction modules are pluggable components in all effects, so as to enable incremental support for additional interaction paradigms.

4.3 Multichannel content adaptation and delivery

Providing multichannel access consists in supporting heterogeneous client applications and devices in order to exploit available services and content information always in the most suitable and consistent manner [Artix] [New05], accordingly to user preferences, communication media in use, and current device capabilities.

For instance, as Illustration 17 shows, by formatting news content into a Web page it is possible to combine text, links and related multimedia content, hence to produce multi-dimensional output at once. Similarly, news can come as video streaming on DVB-T channels, perhaps with text scrolling in the lower part of the screen. On limited devices and/or slow connection types, instead, pictures should be down-scaled and video converted to snapshot images surrounded by plain text. Even more, only text should remain in place to enable SMS delivery and it should be synthesized to perform voice-only communications, such as with VoIP, leading to a linear, mono-dimensional, output type.

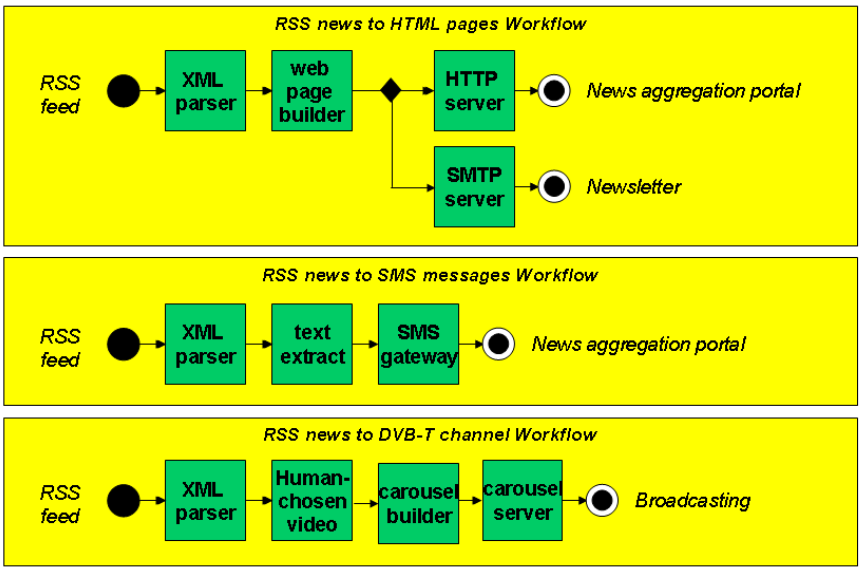

Illustration 17: Multichannel content adaptation

In our approach, we combine functionalities of resources that can elaborate and transform content by defining workflows whose goal is to:

- generate brand new content;
- enrich content being currently processed within the workflow;
- filter content to preserve only relevant or suitable information;
- aggregate pieces of content coming from different workflow branches;
- manipulate content to perform format conversion, transformation from one kind of media to another one, adaptation to device capabilities, and so on;
- deliver content over the desired channel.

Weather forecasts provide the typical example of enabling multichannel access to the same kind of content by means of fine-grained resources whose proxies are arranged into workflows that our middleware orchestrates. For instance, resources that generate content can be weather observation stations that produce METAR reports once an hour [METAR]. METAR format bases on character strings with well-defined characteristics, so conversion to XML data is needed to further process reports in rich applications. A content aggregation service collects XML reports every hour and is followed by a filter selecting weather reports on the basis of current user coordinates. Remaining reports are converted to RSS feeds and then enriched with map images of the interested areas, taken from the Google Maps service. Finally, depending on bandwidth available for the download, final result can be either published “as is” at a certain URI, converted to PDF and sent by e-mail, or enriched with Mp3 tracks from the synthesis of feed textual descriptions, to deliver forecasts over a podcast channel.

This way, users can specify what contents or services to access, in which format and by means of which device and available communication channel. Then, middleware core layer components analyze available service metadata

and user context and requirements, in order to automatically arrange and configure the most suitable transformation flows.

Chapter 5 – Business Process Management

To provide value-added services, leverage new service opportunities, and improve final user experience, the Internet of Services scenario pushes the need to coordinate functionalities from remote and distributed resources. One way this can be done is to expose the business logic of these resources in the form of reusable software modules, and to model business processes that can realize the desired composite applications by means of coordination of operations of modules themselves.

A business process can be defined as the execution of activities from diverse software modules, according to a defined set of rules, to achieve a common goal [Ana04]. In particular, we refer by the term *composition* to the issue of analyzing and selecting the most suitable resource functionalities in order to satisfy a given scenario requirements, whereas we indicate by the name *orchestration* the execution support that middleware provides in order to enact previously configured compositions.

In our model, we use proxy adaptors – as seen – to abstract heterogeneous resource types and execution environments, hence realize a unified and consistent means to deal with diverse software characteristics and to provide additional integration facilities. Thus, resource proxies constitute the actual participants in our business processes, whereas middleware acts as the business process management system that permits modeling, validating, executing and measuring effectiveness of those processes.

Given the description of a desired application scenario and the set of currently available resource proxies, middleware *Composition Engine* component is in charge to create one or more workflow definitions that can altogether pursue the business process goal for that scenario. Then, to serve explicit requests as well as asynchronous events, *Workflow Orchestration Engine* provides all needed facilities to interpret such workflow definitions and enact the activities that they expect.

Finally, each business process in our system binds to a specific set of configurations from the resource proxies that it entails. Indeed, once a workflow definition exists, system invokes RRM configuration methods (when present) on every proxy that participates in it, to reserve specific settings. Following proxy RRM execution steps within that particular process will therefore leverage those settings. In the end, when deleting workflow definitions for the corresponding business process, system releases settings by calling RRM deconfiguration methods (when present) on the proxies.

5.1 Resource composition

Ubiquitous computing calls for dynamic resource composition models, able to cope with changes in user requirements and resource conditions such as location or availability. Variations in user needs as well as in service characteristics can indeed make running compositions less adequate or even useless; they therefore demand support for dynamic reconfiguration to avoid unbearable management burden. Arianna would certainly cancel her online music account if she had to keep up with setup issues every time she changes device or connectivity type!

Within our composition model, resources embody generic pieces of application logic that can be arranged together within business processes, by means of their proxies, to pursue the desired service scenario. We allow the middleware to get knowledge about new or modified resources and to learn how to deal with them by leveraging metadata "*attributes*" that describe resource features. In our model, resource proxy developers are in charge of specifying such attributes and can do that in easy and extensible ways. At the same time, final users willing to exploit distributed resources (as well as expertized process choreographers) can draw on complex aggregations by leveraging intuitive and natural concepts. To achieve this, we adopt a translucent approach: we both guide users/choreographers in the composition creation process by hiding details and complexity, and still remain extremely flexible by unveiling composition mechanisms to metadata providers.

On the complexity hiding side, we enforce a template-based approach to the composition problem, wherein "*templates*" act as models for possible business processes, to fill in with actual resources, and typically represent resource composition schemata that are common to several scenarios.

On the flexibility and extensibility side, we drive resource composition by evaluating composability "*expressions*" that can assert resource compatibility with each other and within the selected template in forms of constraints on acceptable values from their metadata attributes and from resource dynamic characteristics, such as context and session.

According to our model, resource composition to satisfy a given set of requirements resolves to nothing more than expression evaluation and therefore constitutes a deterministic process that can be automatized and performed without human intervention. As a consequence, automatic reaction to changes in scenario requirements and/or resource conditions becomes possible by simply having the middleware re-evaluate those expressions.

Furthermore, by not limiting expression results to mere boolean values, we also enable ranking among valid compositions, via the comparison of their composability scores. And finally, since different expressions can govern different aspects of resource composability, we can choose the ranking policy to adopt by assigning different weights to scores regarding different composability aspects (say "low billing price" versus "high quality of service").

5.1.1 Composition model

Before deepening the description of the overall mechanism that permits calculating resource compositions, following sections analyze the diverse entity types that concur in creating our composition model.

5.1.1.1 Templates

In our vision, outlining a service provisioning scenario by means of distributed resources must be as simple as shaping the corresponding *template*

and indicating features for the actual resources that will take part in it.

Templates represent abstract definitions of business processes. Their goal is to indicate a suitable composition schema and, if needed, to express constraints on the resources that actualize it. To illustrate this, Illustration 18 shows a possible composition schema as a set of empty blocks, representing both control and resource (via their proxies) logic.

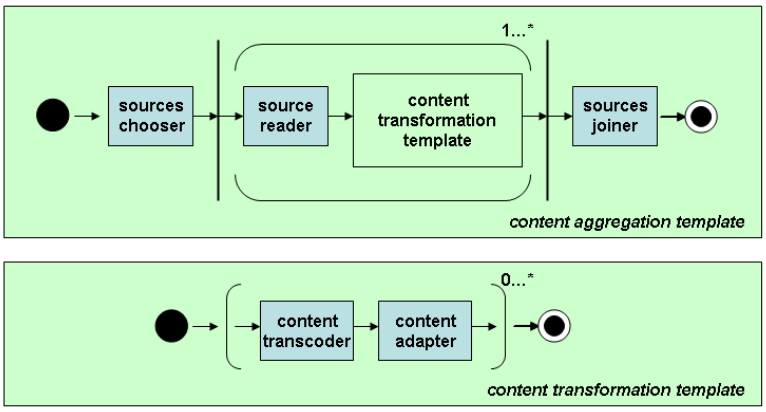

Illustration 18: Composition templates

Actualization of templates with concrete resources is the result of filling in all empty blocks by satisfying both template-required features and all the composability issues that arise, given a set of candidate resources. Illustration 19 provides a snippet of such actualization.

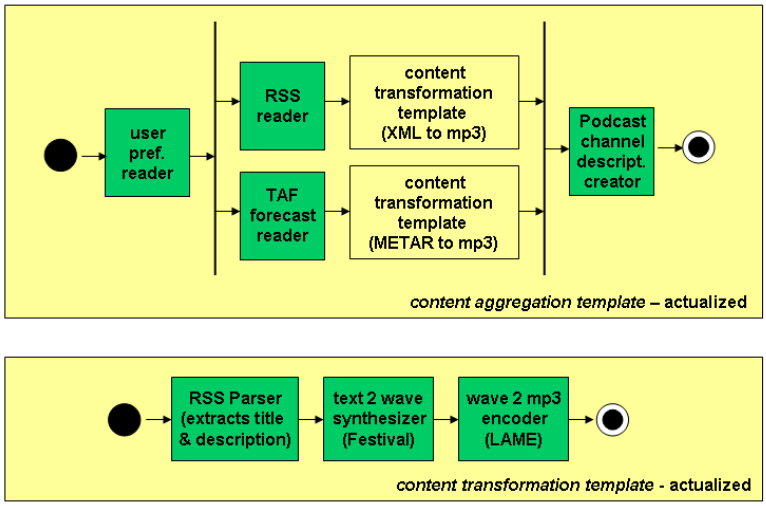

Illustration 19: Actualization of composition templates

Besides, in order to enforce reuse of existing templates (and, possibly, of their already-computed actualizations), every template can be defined in terms of other ones. To clarify this, Illustration 20 provides two complex kinds of composition template.

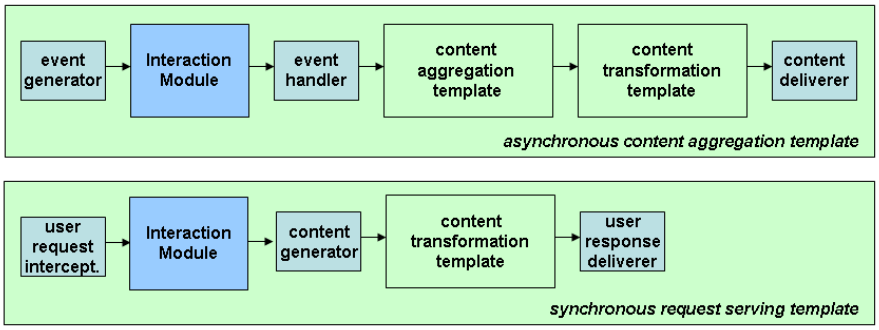

Illustration 20: Reuse of composition templates

Finally, since novel scenarios can require additional composition schemata, novel template definitions can be plugged in at any time in the system.

5.1.1.2 Static metadata attributes and dynamic conditions

Resources provide the actual implementation of business logic like content transcoding, generation, delivery, enrichment, aggregation, adaptation, filtering, and so on. No middleware feature indeed aims at providing this kind of facilities, as this approach would lead to limited flexibility and to overwhelming complexity in API definition and usage. Rather, we enable third-party provided products to do so, by registering their corresponding proxies to the system and by indicating how to integrate them with both middleware capabilities (e.g., messaging, persistence, naming, ...) and with other services (i.e., within composition templates).

To enforce this possibility we leverage both static *metadata* attributes and dynamic information about resource conditions. Resource proxies, indeed, can provide metadata to describe almost any aspect of the resources they represent, without affecting their actual implementation. Besides, to face dynamic aspects of resource composition and orchestration, we enforce middleware support for *context* and *session* management to describe run-time conditions of running resources.

The set of possible values is not predefined, but can expand at any time. For instance, a resource can introduce a new type of metadata in the system by just presenting values for it. As an alternative, it can define it implicitly by imposing constraints on its possible values from interacting services, via the indication of suitable composability expressions.

5.1.1.3 Scenario requirements

Scenario *requirements* convey the particular features and preferences that final users or process choreographers express to select and/or configure actual resources within the composition. In addition, these requirements also indicate the main template that describes the business process of the scenario itself, whose definition may in turn recall those of other finer-grained templates.

As an example, let us consider the “News by SMS” scenario in

Illustration 21 wherein, at a given time of the day, an RSS reader service is triggered to generate news feeds; feeds are then processed to extract news title and description, hence converted to plain text, suitable for SMS distribution. Scenario main template expects a first resource to be configured to retrieve RSS feeds at a given time of the day, then to deliver these feeds via the publish/subscribe middleware interaction module to all interested consuming workflows (say, all subscribed users). Choice of time of the day for firing messages and RSS feeds URL are part of scenario requirements.

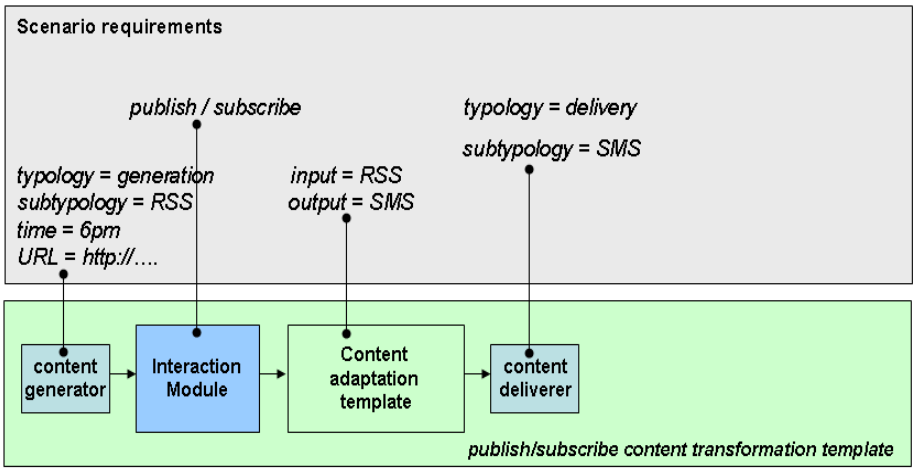

Illustration 21: Main template and scenario requirements for the "News by SMS" application

Consuming workflows are shaped on the basis of the "Content adaptation template", consisting in a sequence of an arbitrary number of resources, each one operating on the result from the preceding one. This finer-grained template requires the first resource to accept content of type RSS feeds and final output to be SMS text, whereas candidate resources pose constraints on their input and output format, thus limiting possible compositions.

5.1.1.4 Composability expressions and domains

Template-driven features, mutual resource compatibility issues, and specific scenario requirements, all formulates in terms of constraints on

metadata attributes and current conditions from the resources that take part in the composition. The evaluation of composability expressions on such values constitutes the only basis for the composition calculus: middleware does not impose any expression *a priori*, but just apply the ones from templates, resources, and scenario, jointly.

As seen, each of these entities can indicate its own set of constraints to satisfy (in case, leading to discard a candidate resource itself if no valid composition is possible, given its constraints). Expressions, anyway, always evaluate against values that have to be correctly specified. Therefore, to ease resource description on the side of resource proxy developers, we do not consider single composability expressions, but group semantically related expressions within so-called *domains* that can represent composability constraints at a higher abstraction level.

Besides collecting related expressions, domains also declare the name of attribute values needed for evaluation, their value type, and allow for testing. Domains hence represent a shared knowledge base that resource proxy developers can refer to, in order to provide feature descriptions that are suitable for evaluation. Eventually, when calculating definition of an actual composition, requirements, templates and candidate resources themselves specify what domains to apply on current metadata.

To exemplify this, a trivial domain we have leveraged several times in real scenarios consists in the MIME datatype compatibility one. This domain is made up of one single expression, that bases on `outputMime` and `inputMime` attributes of composed resources. The expression just asserts that a resource (e.g., an RSS feed aggregator service) must provide an `inputMime` attribute value that is compatible with the `outputMime` one from the resource that produces the data it will further processes (e.g., an RSS feed reader service), within the composition. Hence, developers of resource proxies to compose with each other can leverage MIME type compatibility domain to agree on the metadata attributes to specify. In an all similar way, they can refer to other well-known domains to express data transport issues, aspects such as synchronous/asynchronous behavior, the ability to accept just

one input data payload to process at a time or more (think of content aggregating resources), as well as other syntactical or semantical constraints.

Knowledge of new expression domains can be registered to the middleware at run-time, enabling incremental support for additional resources, templates and requirements in general, by supporting the additional constraints that they entail.

5.1.1.5 Roles

By defining the resource composition schema of an application scenario, a composition template also defines the *roles* that resources play within the schema. Role concept enables evaluation of composability expressions against attributes from actual resources, since it permits indicating which resource should provide which attribute value. Indeed, as expressions apply to attributes of resources that candidate to play roles that template indicates, evaluation simply consists in substituting formal expression arguments with actual values from those resource attributes, according to the role that each one candidates to play.

Recalling the previous MIME type example, MIME type composability domain expects attributes `'inputMime'` and `'outputMime'` to be provided from resources that candidate to compose with each other. Hence, by leveraging the roles of content `'producer'` and `'consumer'`, its sole expression formulates the following constraint:

```
producer.outputMime == consumer.inputMime
```

Roles, anyway, do not tie to any particular composability domain, but several domains can refer to the same role set, each one to formulate its own constraints. For instance, to express direct composability between sibling resources in a content distribution process (e.g., streaming server and connected client of Arianna example), transport type composability domain might express constraints such as:

```
producer.outputProtocol == consumer.inputProtocol
```

Or, again (in a short form, by assuming method definitions as being provided elsewhere by the domain itself):

```
producer.codec isSupportedBy consumer.knownCodecs
```

Similarly, roles do not event tie to any particular composition template, but several templates can expect resources playing the same role, in different composition schemata.

Trivially, direct resource composition such as that of the streaming server and its connected client does not leverage workflow execution for result passing between composed resources. On the contrary, an RSS feed reader simply returns content to its invoker (i.e., the middleware), that will pass it over to the next resource in the composition flow. Clearly, as Illustration 22 shows, these two resource couples are part of different resource composition schemata; anyway, corresponding templates can both leverage the roles of content `'producer'` and `'consumer'` to formulate constraints.

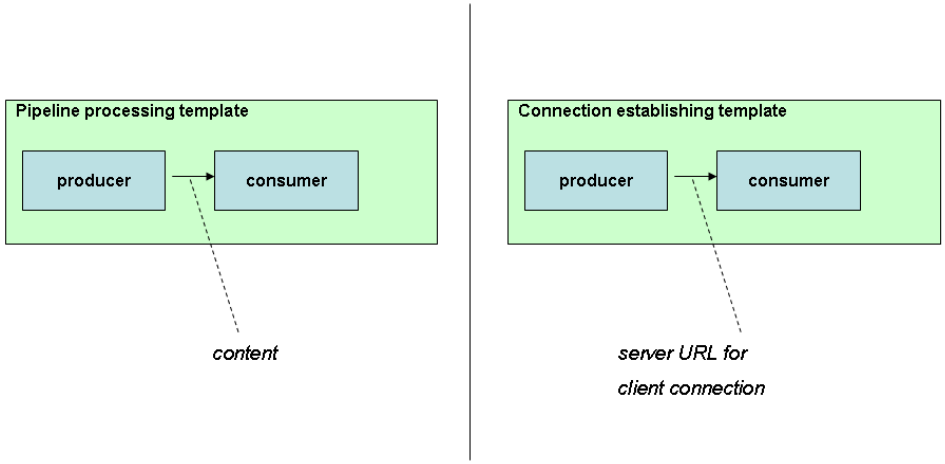

Illustration 22: Different templates, same roles

Summarizing, roles as well as composability expression domains realize a knowledge base that composition players in the system share. Hence, by referencing the same roles within templates, scenario requirements, and resource compatibility constraints, it is possible to determine the resources

providing the most suitable metadata values for the sake of composition in the given scenario.

Moreover, since the actual roles to consider in a composition process are entailed by entities that can dynamically add and/or register to the middleware (i.e., templates, resources, requirements), we do not even assume any a priori knowledge of roles, but let those entities define any new role they may need by just introducing its corresponding and unique noun.

5.1.1.6 Scores

Expression evaluation produces not only boolean results (meaning that composition actualization is acceptable), but values potentially of any type. Thus, by leveraging non-boolean results as composition *scores* it is possible to enable ranking and automatic choice among several possible composition actualizations.

By basing on scores, scenario requirements can state the particular kind of ranking to perform, perhaps reflecting user-specific preferences. Indeed, a composition will typically show more than just one score value (e.g., number of services, computational load, billing costs, ...) and there is no way to tell which one should prevail, *a priori*. Requirements, hence, are also in charge of indicating weights for each score type.

Middleware can therefore autonomously calculate the most suitable composition that satisfies the composition request from a particular set of requirements, given the resources that are currently registered to the system. Scores that requirements do not mention are simply ignored.

Alike roles and expression domains, scores too realize a kind of knowledge that middleware does not provide, but that entities can introduce and share with each other. Indeed, scores do not tie to any particular domain, but can be the result of expressions from several different ones. Hence, every kind of score also defines a function to aggregate values of its own score type, coming from the evaluation of multiple expressions and a comparison function to judge on compositions that show different values for the same kind of score.

5.1.2 Composition calculus

To summarize previous sections, our composition model requires:

- resource proxies to provide static metadata attributes and to leverage middleware session and context support to describe dynamic resource conditions;
- composition templates to declare roles;
- scenario requirements to indicate the main composition template to realize;
- scenario requirements, composition templates and candidate resources to address domains of composability expressions to evaluate;
- candidate resources to enter expression evaluation by playing the role they are being considered for, within the selected template;
- middleware to evaluate composability expressions to determine possible resource compositions;
- middleware to leverage composition scores to rank possible compositions and to select the most suitable one.

Illustration 23 below reports the overall schema of the composition calculus actors.

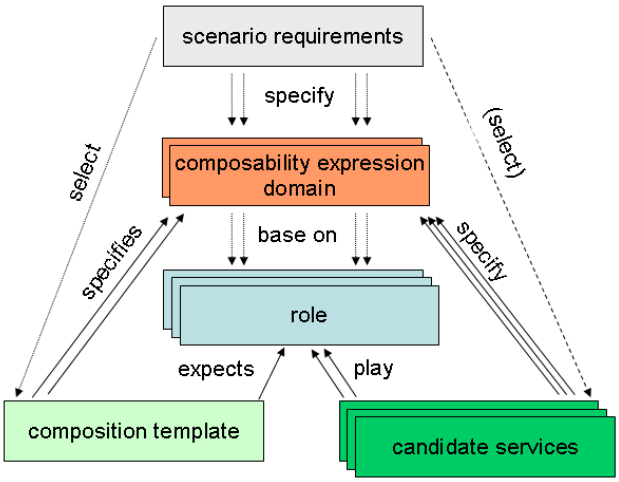

Illustration 23: Overall schema for the composition calculus

5.1.2.1 Representation

For the sake of integration with our middleware, resource proxies typically provide general information, such as:

- name, provider, version;
- lookup and invocation mechanism (e.g, EJB3, WebServices, CORBA, ...);
- expected invocation parameters and how they map to middleware entities (e.g., argument #1 in signature corresponds to tuple labeled `'XXX/YYY'` in context description);

To enable mutual composability, then, resources have not to adhere to any particular information format, but simply to indicate:

- a set of attribute names and values;
- the composability domains that express conditions to successfully compose with other resources, given their own metadata attributes;
- the composability domains that express conditions upon which

resource execution can be performed (entailing information from the session of the resource itself or the context of its business process).

To exemplify this, Listing 1 reports the values of metadata from a typical content generation service, capable of extracting weather forecasts from METAR messages [METAR]:

```
<?xml version="1.0" encoding="UTF-8"?>

<properties>

   <comment>
     Service to read location-aware METAR messages from a given url
   </comment>

   <!-- Framework-integration metadata-->

   <entry type="fwk" name="general">
     name=MetarReader;provider=Swimm;version=1.0.0
   </entry>
   <entry type="fwk" name="deployment">
     mechanism=EJB3;jndihost=137.204.58.65; jndiport=1099;
     jndiname=metar-app/ReaderBean/remote;
     interface=it.swimm.impl.generation.METAR.ReaderRemote;
     clientlib=MetarAPP-client-lib.jar;method=read
   </entry>
   <entry type="fwk" name="mapping">
     args=request/url,user/context/location/coordinates
   </entry>

   <!-- Service-composability metadata-->

   <entry type="cmp" name="typology">
     type=generation
   </entry>
   <entry type="cmp" name="load">
     avg=low
   </entry>
   <entry type="cmp" name="billing">
     fee=0.001c
   </entry>
   <entry type="cmp" name="datatype">
     outputmime=text/plain;outputformat=METAR
   </entry>
   <entry type="cmp" name="semantics">
     pull=true;push=false;before=none;after=one
   </entry>

   ...

</properties>
```

Listing 1 – Sample of service metadata

Metadata are simple name/value pairs and they obey no particular

format. The first three entries in the listing are middleware-specific ones and let the service declare, for the sake of invocation, that it runs as an `'EJB3'` component on host `'137.204.58.65'` with the JNDI name of `'metar-app/ReaderBean/remote'`. Besides, it expects two arguments: the `'URL'` (extracted from the user request) where to read METAR messages and the current user geographical position (as mapped to the `'/user/context/location/coordinates'` element in context). Composability metadata, instead, just represent the fact that "as far as a given expression domain is considered, the service provides a certain set of attributes". For instance, according to `'datatype'` domain, the service formats its results as `'METAR'` and their MIME type is `'text/plain'`. Keys `'datatype'` and `'arguments'` are just the domain names referring the expressions that tell about service suitability and composability with the other resources in the composition.

In our model, an expression domain defines as:

- a unique name;
- a set of expressions;
- a set of roles that its expressions base on;
- a set of attributes that its expressions expect.

To achieve implementation simplicity, every domain is also associated to the URL where its XML definition is published (alike locations of XML schema definitions). As soon as an entity – be it a resource, a template or a set of requirements – entails a new expression domain, the system can achieve knowledge of that particular domain by simply downloading its definition from the corresponding URL.

To provide a brief example, Listing 2 reports an expressions excerpt from the `'datatype'` domain. As the text suggests, these expressions can be used to assert mutual resource compatibility within a composition template that expects the roles of `'consumer'` and `'producer'` :

```
...

<expression domainName="datatype" type="boolean">
  producer.outputformat == consumer.inputformat
</expression>

<expression domainName="datatype" type="boolean">
  consumer.inputmime isSupersetOf producer.outputmime
</expression>

...
```

Listing 2 – Sample syntax rules for producer and consumer roles

As for scenario requirements, they simply:

- indicate the main composition template;
- can impose required features to the resources to compose;
- define the ranking criteria that govern the election of the best template actualization, in case multiple ones are possible.

Listing 3 provides a brief XML example of a requirements description:

```
<?xml version="1.0" encoding="UTF-8"?>

<requirements>
  <user fwk="swimm">31231</user>
  <template name="pushAggregation"/>
  <properties>
    <entry type="cmp" name="delivery">channel=MMS</entry>
  </properties>
  <ranking>
    <score weight="1.5">billing</score>
    <score weight="1">performance</score>
  </ranking>
</requirements>
```

Listing 3 – Simplified scenario requirements description

Listing 4, finally, reports a sample definition for the `'billing'` type score:

```
<?xml version="1.0" encoding="UTF-8"?>

<score name="billing">
   <format type="java.lang.Double"/>
   <compare>&gt;</compare>
   <aggregate>+</compare>
</score>
```

Listing 4 – Example of score definition

5.1.2.2 Evaluation

We can think of solving the composition problem for a given application scenario by simply producing a map of roles and corresponding actual resources where every role of the composition template is played by one resource and all expressions from requirements, templates and resources are satisfied.

When filling in map entries, to accept a given resource in a composition role it is necessary that all indicated expressions successfully evaluate against all other entities already in place: candidate resources already in the composition, the composition template, and the scenario requirements. The same applies to the expressions from the other resources that have already proposed as candidates for other roles in that composition, as well as to expressions specified by the composition template and the scenario requirements: they must of course remain valid as new resources are accepted as candidates.

To demonstrate a possible implementation of the solution to the composition problem, Listing 5 reports an almost self-explanatory imperative formulation of the algorithm that, given the above actors, leads to the election of the most suitable composition to meet an application scenario requirements.

```
Composition compute(Requisites requisites, Resource[] available_resources) {

  // Step 1.1 – Expression from requisites and templates, individually

  List< Set< Map<Resource,Role> > > list_of_resources2roles_maps;

  foreach domain in domain_union(
       requisites.domains, requisites.template.domains
    )

    foreach expression in domain.mandatory_expressions

      list_of_resources2roles_maps.add(

        /* applies expression to the possible role-resource pairs, saving every
        allowed combination as resources2roles map, returning the set of the
        possible maps */
        evaluate(expression, requisites.template.roles, available_resources)

      );

  // Step 1.2 – Intersection of results from individual expressions

  Set< Map<Resource,Role> > resources2roles_maps =

    /* keeps only the maps that are present in all list items
    (i.e., allowed by all expressions) */
    intersection( list_of_resources2roles_maps.entries );

  // ------------------------------------------------------------

  // Step 2 – Rules from the candidate-to-roles resources

  /* note: a cloned structure is used to avoid removing entries from
  a data structure that is being iterated */
  Set< Map<Resource,Role> > allowed_resources2roles_maps =
    clone( services2roles_maps );

  /* requirements- and template- allowed maps of resources to roles
  associations are validated against expressions from the resources */
  foreach map in resources2roles_maps
    foreach domain in domain_union( map.keyset )
      foreach expression in domain.manadatory_expressions

        /* evaluation is skipped if current map has already been discarded */
        if ( map in allowed_resources2roles_maps )

          /* same behavior and result type as of
          evaluate( expression, req.template.roles, candidates )
          but with already-known resources-to-roles associations */
          if ( evaluate( expression, resources2roles_map ) == null )

            /* failure leads to discarding the current map */
           allowed_resources2roles_maps.remove( map );

  // ------------------------------------------------------------

  // Step 3.1 – Scoring

  List<Composition> allowed_compositions;
```

```
  foreach map in allowed_resources2roles_maps {

    Composition composition = new Composition( requisites, map );
    /* domains of an actual composition are the union of those
    from resources in the map, requisites and template */
    foreach domain in composition.domains
      foreach expression in domain.scoring_expressions
        composition.assign( score( expression, composition.map ) );
    allowed_compositions.add( composition );

  }

  // Step 3.2 – Ranking

  Composition best_composition =
    rank( requisites.criteria, allowed_compositions );

  // ------------------------------------------------------------

  // Step 4 – Monitoring

  foreach property in best_composition.monitored_properties
    Middleware.monitor( property.value, property.expression );

  // ------------------------------------------------------------

  // Step 5 – Allocation

  Middleware.register( best_composition );

  return best_composition;

}
```

Listing 5 – Imperative formulation for the composition calculus

Every candidate resource that plays a role in the composition adds its own expression to evaluate. This leads to a tree of possible choices where nodes correspond to incremental actualizations of the available roles. The first resource being considered for a role in the composition template becomes root of one possible tree. At any depth, to accept a resource in the tree as a player for a vacant role, the expressions it entails must be satisfied, as well as the expressions from the rest of resources already in the tree.

Actual implementation of the `evaluate()` function explores resource trees depth-first and stops upon finding a given (configurable) number of acceptable composition actualizations to rank and choose from.

Optimization strategies start filling the role that probably has the lowest number of available candidates (we called it “per-role early pruning”) and

consider candidates in the order they bring the lowest number of new expressions to the system (named as “information base greediness”).

When all roles are filled, all entailed expressions need to be satisfied. Theoretically, there is no conceptual distinction among those coming from scenario requirements, composition template or candidate resources. Nevertheless, expressions from scenario requirements and composition template are present in all trees and permit to discard immediately the ones with unfit resources. Thus, it is smart to process them first: a service “not providing attributes for” or “not satisfying” a requirements- or template-driven expression can never be a candidate.

Finally, algorithm code also permits to dynamically react to variations in resource conditions that may entail business process reconfiguration. Indeed, as previously shown, expressions can refer to both static metadata attributes and dynamic session and context characteristics. In the latter case, middleware registers “*monitor*” entities to watch on changes of their values, in order to re-evaluate corresponding expressions accordingly and trigger business process reconfiguration in case, as the next section discusses.

5.2 Process orchestration

Resource composition constitutes the basis for the execution of arbitrarily complex business processes, entailing both control and business logic, wherein the middleware orchestrates resource proxies to accomplish the goals of a given application scenario.

According to RRM, all resource proxies within a business process can expose suitable methods for the sake of configuration and deconfiguration and leverage suitable metadata to advertise such functionalities. Upon calculation of a resource composition, the middleware looks up a proxy instance for each and every resource that takes part in the composition itself and invokes the configuration method that it provides (if any). Similarly, upon deletion/modification of a composition, the middleware recalls the same proxy instances to invoke corresponding deconfiguration methods (if any). Between

these two moments, resource proxies primary business logic executes as many times as the system needs to orchestrate the business process to which it belongs. Illustration 24 below exemplifies this, in the case of the business process for receiving RSS news via SMS messages.

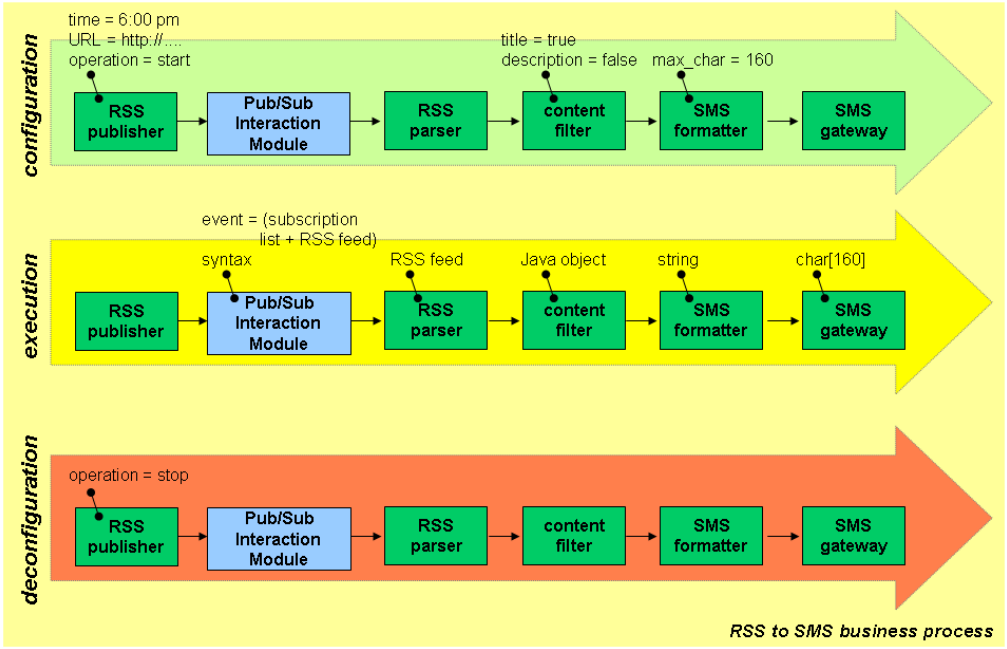

Illustration 24: Invoking configuration, execution, and deconfiguration methods on resource proxies within a business process

As previously described, business processes bind to resource configurations because the precise resource proxy instance that takes part in a business process execution corresponds to the one that has provided configuration for that process. Thus, it is in charge of resource proxies to maintain separate business process configurations and, in case, interact with middleware session and context facilities accordingly. As for the rest, resource proxies are completely unaware of collaborating within complex compositions: they do not directly interact with each other, but just provide results to middleware requests (i.e., invocations of methods that expose their primary business logic) or demand middleware operations themselves.

At run time, by basing on build time definitions of resource composition, middleware orchestrates business processes by exposing one or more suitable interceptor resources and by registering one or more workflow definitions. Upon final user activities and/or system events, resource interceptors stimulate the interaction modules that correspond to the interaction paradigm that they enforce, demanding execution of suitable resource workflows (Illustration 25).

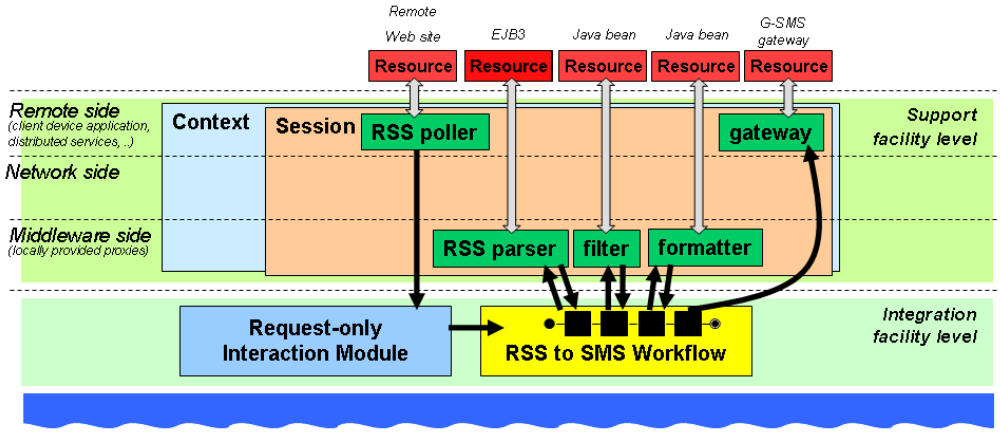

Illustration 25: RSS service proxy interacts with middleware via the Request-only Interaction Module to demand the orchestration of "RSS to SMS" workflow

5.2.1 Parameter resolution

Workflow definitions enforce activity sequences wherein each resource operates on the results from the previous ones in the flow. Anyway, in real IoS scenarios, distributed resources and functionalities typically expect several parameters in addition to the main payload to elaborate, to influence behavior, result type, authentication, billing, and many more aspects.

Invocation of resource proxy methods therefore demands a scrupulous match between formal and actual parameters that they expect, by basing on both resource-provided metadata and scenario-related preferences. Besides, resolution of part of these values can happen at build-time, to directly hard-coded them to the workflow definition, whereas other ones necessarily refer to

properties that only are available (or significant) at run-time.

Values of properties that do not change over time, such as user identity, composition-related preferences, and many others, can become inherent part of the workflow description. This helps saving system resources and improves the overall run-time performance when executing workflows. On the contrary, values for remaining parameters, that depend on present conditions at workflow execution time, must be dealt with at run-time, upon corresponding resource invocation, while the middleware orchestrates workflow business and control logic.

Invocation values map to several possible domains of data within our middleware, depending on both explicit user/choreographer preferences and resource characteristics that metadata convey. By means of metadata, indeed, developers can parametrize resource behavior upon user profile data, session and context information, network infrastructure conditions, features such as addresses or status of serving nodes, and a lot more, and indicate whether resolution must happen at build-time (composition calculus) or run-time (process orchestration). Besides, to enable resource configuration driven by final user preferences, it is in charge of resource metadata also to specify the set of possible values from data domains and the choice criteria to adopt.

Resource metadata achieve this, by specifying on each formal parameter to resolve for actual resource invocation:

- the data domain to consider;
- the precise property name to read or the value sets to choose from;
- the actual choice criteria to enforce.

For instance, in order to support automatic delivery of customized breaking news via podcast, content transformation workflow involves services like several RSS readers and a voice synthesizer, among the others. By leveraging this kind of workflow, every user in the system can configure her own personal podcast channel and download content in Mp3 format from it. Since RSS documents consist in XML data that syndicates content feeds, each

reader executes upon the indication of *a)* the URI of the RSS source to analyze and *b)* the identifier of the last feed that the current user already retrieved from that source (to avoid returning the same content several times). Instead, voice synthesis service accepts as arguments *a)* the current text to reproduce and, in addition, *b)* the user language to adopt for text analysis and *c)* the desired output bitrate.

User explicitly selects the sources of content that she desires, by specifying URLs for the corresponding feeds at time of resource composition creation. Thus, possible values undergo build-time resolution and are chosen from the set of known URLs that each RSS reader service advises in the so called *service* data domain (depending on supported RSS version, character encoding, or, merely, commercial agreements between service provider and news publisher). On the contrary, identifiers of past RSS feeds are part of *session* domain and only relevant at run-time. To clarify this, Listing 6 in the following reports a snippet of the actual metadata that an RSS reader resource proxy provides, limitedly to the method it indicates for RRM execution step:

```
<?xml version="1.0" encoding="UTF-8"?>

...

<methods>

  ...

  <method rrm-step="execution" name="readFromChannel">

    <argument name="url"
      description="location of the XML descriptor of news">

<!-- user is presented the whole set of possible URLs from the indicated
mapping field of the specified domain; her preferences are hard-coded to the workflow
definitions for the business processes this resource will take part in -->

      <resolution>build-time</resolution>
      <domain>service</domain>
      <mapping>/ACME/rss-reader/URL</mapping>
      <choice>user</choice>
      <default>http://swimm0.ing.unibo.it/blog/rss.php</default>
    </argument>

    <argument name="lastRead"
      description="identifier of the most recent already read feed">

<!-- all rss reader instances store here the association between the-URL-they-
read-from and the-last-read-feed-id, in an array-like structure. Middleware cannot
know which one to choose, so each service will get the whole array and filter the
```

```
sole id it is interested in. -->

            <resolution>run-time</resolution>
            <domain>session</domain>
            <mapping>/ACME/rss-reader/lastfeed</mapping>
            <choice>service</choice>
            <default>null</default>
          </argument>

        </method>

        ...

      </methods>

      ...
```

Listing 6 – Metadata for execution method of the RSS reader service

Similarly, text to synthesize represents the main payload being processed by the text synthesis service; hence, it belongs to the *execution* scope domain of the current workflow, which is unavailable at build-time. User language is part of the user *profile* information that is known to the middleware since user account creation. And finally, depending on service metadata, audio output quality can either rely on build-time *QoS* agreement (that user has paid for) or relate to run-time available bandwidth from the current *context* information (to enable download over slow connections as well). Listing 7 illustrates how the text synthesis service proxy exposes suitable parameter mapping:

```
<?xml version="1.0" encoding="UTF-8"?>

...

<methods>

  ...

  <method rrm-step="execution" name="synthesize">

    <argument name="text" description="the textual content to process">
      <resolution>run-time</resolution>
      <domain>execution</domain>
      <mapping>/PAYLOAD</mapping>
      <choice>none</choice>
      <default>null</default>
    </argument>

    <argument name="language" description="the language determining rules to
       adopt for text analysis and to determine pronunciation of word tokens">
```

```
      <!-- notice: this parameter directly maps to user profile information, but
other pieces of metadata also impose composability expressions, to prevent selection
of this service in case of unsupported languages -->

            <resolution>build-time</resolution>
            <domain>profile</domain>
            <mapping>/language</mapping>
            <choice>none</choice>
            <default>en</default>
          </argument>

          <argument name="quality"
            description="a parameter influencing the final output bitrate">
            <resolution>build-time</resolution>
            <domain>QoS</domain>
            <mapping>/festival/bitrate</mapping>
            <choice>user</choice>
            <default>64kbps</default>
          </argument>

        </method>

        ...

      </methods>

      ...
```

Listing 7 – Metadata for the execution method of the text synthesis service

Data domain that resource proxies specify can be well-known domains that middleware inherently provides (i.e., *session, context, execution*, ...) as well as additional domains for supporting traditional real life scenarios (e.g., *profile*, ...) or specific tasks (e.g., *service*, *QoS* as well as *network, middleware,* and so on).

In all cases, from the moment middleware accepts registration of any data domain implementation, it then supports transparent access to its entries for both build-time and run-time resolution moments. In particular, at time of creation of a composition, resolution happens at once for all build-time parameters of all resources that take part in it and selected values are directly written to the definitions of its corresponding workflows, to improve actual resource invocation. At workflow execution time, instead, middleware resolves run-time parameters resource by resource, and it leverages values from the workflow definition to assign remaining parameters.

5.2.2 Result passing

Though quite a trivial issue from a theoretical point of view, result passing permits to support business process execution by coordinating invocation of resources that participate to workflows. Indeed, dependency constraints, sequences of operations on a same data payload, parallel branches, error handling and conditional executions driven by result characteristics are all typical problems that arise when commanding invocation of independent resources that have to cooperate with each other.

Traditionally, workflow engines deal with coordination and data treatment by interpreting formal descriptions of the business processes they have to enact, and by providing a suitable execution environment for method invocation and data exchange among all resources entailed by a composition. Every workflow describes actions to enforce on specific resources as well as control logic that determines the order of operations, time dependencies, data transportation, and so. Interpreter evaluates such instructions to arrange a work flow of successive activities to orchestrate.

Thread safety of multiple simultaneous interpreters guarantees concurrent execution of multiple processes, as well as forks, branches, and joins are possible by splitting up a single workflow in multiple subparts, to assign to different interpreters, each one providing its own execution scope.

In our middleware architecture, as seen, resource proxies never directly interact with each other. Hence, in order to cooperate and exchange partial results, they either enforce the actual resources they manage to intercommunicate with each other (e.g., in Arianna story, the case with streaming) or demand handling of such results to who actually orchestrates their execution. It is therefore middleware responsibility not only to invoke resources according to a given workflow definition and parameter resolution strategy, but also to properly handle their results, in case, and to transmit them to successive stages of the running workflow.

Besides, to deal with huge resource distribution (such as with distributed and replicated services), middleware features location transparency while

orchestrating and forwarding results among them. Resources are unaware of their invoker location and do not influence result destination, neither in terms of consuming resources nor in terms of transport mechanisms.

Finally, to support human actors who participate to workflow activities (i.e., the final users), middleware also provides durable and reliable result passing between resources. This way, it is possible to allow for passivation and resumption of long-running processes, in order to save computational capacity.

For instance, in traditional enterprise scenarios as well as in more typical IoS ones (e.g., online order processing, instant messaging, download of web pages followed by form submission, and so), human involvement may lead to long lasting workflows, where inactivity time exceeds actual processing time. To overcome this, invocations by the middleware to resource proxy methods can enforce a blocking policy as well as exploit a callback mechanism to prevent waiting for results.

5.2.3 Automatic reconfiguration

Dealing with modern IoS scenarios, where user conditions can vary in extremely dynamic ways, automatic and efficient composition (re-)calculation can become really effective only by monitoring relevant user characteristics to learn when and how to perform it.

Our composition model enables this by means of a particular kind of composability expressions, called *monitors*. Monitors not only evaluate when the middleware first calculates a composition definition, but they also register to the system the resource characteristics to observe and re-evaluate upon their changes. When a monitored resource characteristic changes, compositions that depend on it may become no more valid, depending on the result of monitors re-evaluation. Hence, they are forced to check their own validity again and to recalculate their own definitions in case of failure.

It is fundamental to notice that composition check and (in case) recalculation occur at time of changes in monitored values, and not when the

user demands results from that resource composition. Hence, reconfiguration is proactive and brings little or no impact on user experience.

Listing 8, in the following, completes the METAR service metadata example by arguing on “non-nullable” values, data format constraints and allowed ranges (e.g., Bologna metropolitan area).

```
<?xml version="1.0" encoding="UTF-8"?>

<properties>
  <comment>
    Service for reading location-aware METAR messages published at a given url
  </comment>

  ...

  <!-- Monitoring metadata-->

  <entry type="mon" name="notnullable">
    props=user/context/location/coordinates
  </entry>

  <entry type="mon" name="allowedformat">
    props=user/context/location/coordinates,LatLong
  </entry>

  <entry type="mon" name="allowedrange">
    props=user/context/location/coordinates,[44.55,11.17]/[44. 44,11.42]
  </entry>

</properties>
```

Listing 8 – Metadata for monitoring characteristics

Whether the final user has no valid position or she is outside the service scope, the service itself must be substituted by another one (maybe not location-driven – e.g., forecasts for the whole user's country – or related to another geographical area and perhaps at another billing cost). If substitution is not possible, composition becomes unavailable until middleware succeeds again in calculating its expressions (e.g., new available services or changes in user coordinates arise).

Chapter 6 – Related Work

To define middleware features, we strongly enforce concepts form the general structure of a Distributed Processing Environment (DPE) as exposed in [Cha95], and endorse best practices and integration strategies described in Rod Johnson famous book on enterprise application design and development [Joh03]. In details, likewise TINA-C specification in [Cha95], we promote the idea of abstracting the current distributed processing environment to cooperating resources, by offering communication and interoperation facilities that can provide location transparency. Furthermore, we advocate a business process management and coordination role for the middleware itself, rather than making it a sort of content-related facilities provider with which resources have to deal directly. Middleware intervenes on middleware-unaware resources and orchestrates their integration and execution. Thus, complexity shifts from software design issues to business process modeling and middleware disappears in the background while it manages resource functionalities.

Usage of proxy entities to abstract resource location and to enable technology agnostic interaction is a well-know software design pattern [Gam94]. Though others adopt proxies as a means to pursue integration of heterogeneous distributed legacy assets [Ber04], we argue that leveraging proxies to provide uniform and consistent resource lifecycle management and to provide Ubiquitous Internet related facilities is an original contribution from our work.

6.1 Session

Session related issues are being heavily debated in SOA and enterprise software communities and several standards [Kri97][Pan04][Sch02] and proposals [Ueh01a][Roh97][Ueh01b] are emerging to provide viable solutions.

Seam project from the JBoss group [Kin08] promotes a framework architecture where the run-time environment that is in charge of enacting resource business logic also provides the session management facilities that are needed to orchestrate resources themselves into complex business processes. Furthermore, as validity of session information can undergo different constraints on different kinds of business processes, framework also enables differentiated session scoping for different pieces of session information. In details, by focusing on rapid development of Web applications, Seam framework provides session contexts that can tie to a single request/response message pair, to all requests from a single client for a particular Web page, or to a conversational flow spanning across multiple pages, as well as to one business process entailing multiple software components, or to the entire application.

By studying a set of target applications, also [Abr96] derives the description of a set of functional scopes to provide effective session facilities to distributed applications. Although from a different perspective – that is to say, abstracting session management for the application programmer rather than enabling use case driven composition of resources –, proposed reference model claims to differentiate session details that are provided to diverse business participants: final users, application as a whole, distributed cooperating functionalities and their coordination protocols.

[Haa97] emphasizes the problems of session establishment and service continuity as session participants distribute over different – and even mobile – network nodes. Separating resources with intermediate software layers is claimed to ease solutions for both mutual discovery, hence initiating interactions, and state information retention/transferring among resources. In our vision, configured proxy instances realize part of such in-between software, in effect. And the uniform resource abstraction they provide also constitutes the basis for a uniform approach on session management issues.

6.2 Context

Context Toolkit from Salber at al. [Sal99] is generally considered as one of the most important milestones in work on context-aware ubiquitous computing. Authors observe several technological efforts on sensing and interacting with physical context of people's activities, and highlight the need for exploring realistic scenarios and location-dependent services in easy ways. Article also crafts a new operational definition of context, in terms of the actors and information sources involved in creating and leveraging it: context is "any information that can be used to characterize the situation of entities [...] that are considered relevant to the interaction between a user and an application, including the user and the application themselves. Context is typically the location, identity, and state of people, groups, and computational and physical objects". To endorse this definition, authors provide a suitable toolkit to build context-aware applications, after the premise that combined toolkit components can determine a contextual state by capturing, transforming and aggregating raw information. Thus, they also insist on aspects such as context representation, management, integration in the computer world and exploitation in software.

Nowadays, context is actually a broad topic and it involves approaches from several disciplines, ranging from computer science to cognitive and social sciences. For example, [Eri02] investigates chances of building robust context-aware systems that will rarely fail to react appropriately to context-related events; artificial intelligence techniques are criticized, due to difficulties in capturing relevance differences in people experiences. [Gre01] emphasizes the inherently dynamic nature of context information, continuously varying and changing as long as interaction proceeds; author claims that it may be difficult to limit possible contextual states *a priori* and also to determine what information is necessary to infer one of these states, as well as to automatically enact appropriate actions on it. On the contrary, [Che06] tries to model a formal way to define context descriptions pertaining to service requesters and providers by means of ontologies and [Sva01] even

appeals to phenomenology to develop foundational understanding of context-awareness as it was done with aspects of human activities and interactions.

We strongly agree on concepts from [Hon01], that argues on facing context-aware computing by means of an underlying service infrastructure, made up of a pervasive intermediate software layer, thanks to which much of the work of collecting and processing context information can be decoupled from the application itself. We believe that benefits from a similar approach lead straightforward to loosely-coupled resources, able to leverage context to differentiate their behavior without directly coping with retrieval and transformation problems. Furthermore, we agree on considerations in [Win01], where different architectural approaches are compared for building context-aware systems; conclusions assert that a blackboard-based approach shows more flexibility than using software components to model context domain.

6.3 Multimodal and multichannel access

Research on multimodal interfaces, multichannel access and interaction paradigms have so far evolved almost separately: for instance, multichannel platforms too often focus on adapting contents to devices, but do not easily integrate with different interfaces from the one initially expected. In a similar way, multimodal frameworks enable development of effective multimodal applications, but do not easily integrate with existing services or different standards from those they adopt. Although requirements for integration of different modalities of natural input/output are commonly acknowledged, the proposed solutions and frameworks tend to have vertical approaches and focus only on specific and fixed sets of interaction modalities or application domains.

Typical platforms target, for instance, e-learning [Shi07], medical consultation [Aka98] or crisis management [Sha03]. Although some general purpose multimodal frameworks [Mmi][Rav03][VoiceXML][Opera][IBM] have been proposed, again they are limited to a set of predefined interaction

modes (specially auditive ones) and therefore still lack concrete and widespread adoption.

As for content multichannel access, instead, legacy systems are usually built with one delivery channel in mind and need re-engineering to enable access via multiple channels; typically, this is done by exposing functionalities as software services and adopting SOA strategies to compose them [Jef08], either implementing a channel-agnostic communication system [Zim05] or a channel-adaptive one [Com04].

6.4 Standard tools for enterprise integration

We commonly refer to services as "self describing, open components that support rapid, low-cost composition of distributed applications", providing "a distributed computing infrastructure for both intra- and cross-enterprise integration and collaboration" [Pap03]. SOA approaches promote the encapsulation of application logic within independent service modules that expose well-defined interfaces, to act as service contracts and specify behavior and interaction details [Ort05]. Service composition techniques enable the creation of brand new valued-added services on top of existing ones and offer abstractions and tools to achieve this goal. Finally, orchestration is often referred to as the act of executing business processes that are defined in form of service compositions, by dealing with the aspects of message passing and identification, invocation sequences and branching logic [MomentumB].

Current service composition platforms usually provide models and languages to define complex business processes and suitable execution environments to enact them. Being developed by BEA, IBM, Microsoft and SAP (among the others), BPEL [Cur03] has emerged over time as an XML-based definition language to "support process-oriented service composition by means of interaction with a Web Services subset to achieve a given task" [Mil04]. Especially in the field of open source software, other relevant attempts tried to enable business process management out of distributed computational resources [Koe04] and/or message routing and transformation

[Camel]. Eventually, BPEL established as a *de facto* standard for process definition and gained support from orchestration engines of other vendors.

Initially, BPEL lacked to support human involvement in service-oriented architectures, wherein business activities invoke services to perform various tasks of their processes and human intervention plays a central role, too. To provide means to model human tasks and to enact services that deal with human actors playing particular roles in the overall process, technologies have been proposed for integrating people interaction with BPEL processes, such as BPEL4People [Agr07a] and the related WS-HumanTask [Agr07b] standards. Anyway, these kinds of specification mainly define syntax and semantics element and introduce a technological-dependent perspective, based on languages and tools, rather than a model-driven one. This forces adaptation of existing implementations (realized by both industry and academia in the meantime) to comply with the standards themselves. As [Hol08] states, "to reduce migration and maintenance costs, adaptation to such technology standards should be easy to perform: while concepts of a system may not change, new technology may introduce new syntax elements and may modify semantics. Therefore it is desirable to have conceptual representations within a system that have only the necessary dependencies on foundational technology". By adopting the pattern-based approach in [Aal03] to describe these requirements, [Rus08] criticizes BPEL achievements; similarly, [Hol08] argues that such a technological-dependent perspective should be replaced with a model-driven approach capable of expressing system concepts at a higher level of abstraction.

6.5 Models for service composition

Several B2B success stories regard middleware adoption as a a comprehensive integration platform for resource composition and process orchestration. For instance, IBM WebSphere Message Broker [WebSphere] is a leading commercial product to connect existing IT system to an SOA messaging backbone, realizing distributed processing and transactions.

Recently, enterprise service bus architectures (ESB) have emerged to expose service functionalities on a shared message bus and to enable orchestration of business processes on top of message flows [Rad09][Woo06]. Open source initiatives are gaining momentum too, dealing with ESB implementation technologies [Mule][ServiceMix].

Most of these solutions, anyway, mainly target static scenarios – such as organizations and optimization of existing business processes – where long-lasting requirements rarely demand service reconfiguration/substitution or expect new services to become available in time for use in existing processes [Alo03b]. As a consequence, they leverage tools for assisting humans in manual creation of service compositions and neatly separate build-time and run-time moments. Networking facilities and the opportunity to provide users with a huge number of services via the Internet, as well as the evolution towards mobile and ubiquitous computing scenarios have clearly made these assumptions obsolete. Indeed, frequent changes in user requirements (e.g., typology and features of the device in use) and service variations (e.g., temporary unavailability or brand new services being published) can easily cause current compositions to become less adequate or even invalid [Boa07]. Given these premises, the lack of support for dynamic and automatic reconfiguration becomes a crucial issue in realizing global mashups of services and final-users.

Research tries to tackle these problems mostly by focusing on a semantic approach. It is argued that for composition platforms to dynamically arrange and compose resources, they should describe them from both a syntactic and a semantic standpoint. For instance, dealing with Web Services, this entails coupling traditional WSDL descriptors with additional semantic-related annotations. So far, a wide set of solutions have been proposed to enable this, ranging from custom application-driven formalization of resource features [Cha06] to the definition of ontology standards and meta-languages for expressing them [OWL][Mar04][Her04][Rom05]. Ontologies, in particular, realize “formal and explicit specifications of shared conceptualizations” [Ber01] and can be created by domain experts to provide

relevant vocabulary for describing properties and relations. Thus, by formalizing concepts, values and meanings used to semantically describe resources, the task of retrieving and dealing with the desired functionalities can be improved and automatized.

Some solutions [Kal07][Fuj06] rely on a graph-based composition model that leverages semantic descriptions to dynamically generate paths among available services, to satisfy user requirements. We disregard this kind of approach, due to the difficult in predicting and ranking actual compositions out of multiple valid paths, as well as in formulating requirements that involve intermediate graph nodes (such as supported service preferences, besides initial and final states) or the overall resulting composition (such as QoS constraints).

Other approaches enforce composition models that base on rules to check for service compatibility and to rank possible compositions, leveraging semantic information to infer service degree of interoperability. [Nar07] separates requirements into two parts, functional and extra-functional, regarding commitments on the overall service composition and constraints on the behavior of individual services, respectively. Authors then concentrate on defining modular requirements that can be processed along the execution of their adaptive workflow model. [Med05] insists on the benefits of differentiating composability rule levels according to syntactic, semantic and qualitative degrees. Besides, they introduce the notion of partial composability and highlight the problem of relative weights of composability results from different rule levels. Finally, [Med03] illustrates four conceptually separate phases in automatic service composition, from *specification* of requirements to features *matchmaking*, service *selection* and final *generation* of the composition description; an ontology-based framework to support formulation and processing of semantic information is provided.

Though these approaches sometimes miss the right abstraction level, as they concentrate on too vertical and domain-specific descriptions, nevertheless we strongly agree on the benefits of rule-driven composition and of leveraging separate rule sets to model different composability issues. Anyway, we also

maintain that rule evaluation framework has to allow for addition of new concepts, values and rules at any time, to guarantee service set and application domain extensibility. In fact, as [Sch07b] criticizes, “semantic islands” are only useful to a limited degree.

Chapter 7 – Prototype Implementation

For the implementation of our middleware prototype, we have adopted the most relevant *de facto* standards, both in terms of technologies and development tools. Given the active and lively user community, and the availability of several free and/or open source support projects, most references in the following relate to the Java language and its Java Enterprise Edition (JEE) APIs and facilities [JEE]. However, design guidelines and principles we have depicted so far grant our architectural proposal real independence of the underlying software infrastructure and execution environment.

7.1 Intercommunication, container and registry levels

To avoid redesigning from scratch solutions for persistence, resource naming, component pooling, and so on, development has strongly relied on the adoption of an application server infrastructure. In particular, among JEE solutions, we have chosen to exploit the open source application server implementation from the JBoss Group, now part of Red Hat Middleware, Inc. [JBoss], due to the out-of-the-box implementation that it provides for most JEE API specifications and its support to custom extensions of its core functionalities.

As Illustration 26 in the following shows, middleware *Intercommunication, Container,* and *Registry* levels heavily leverage some major JEE facilities, such as Java Naming Directory Interface (JNDI) specification as for resource and component naming [JNDI], Java Authentication and Authorization Service (JAAS) to deal with security management [JAAS], Java Persistency API as a simple programming model for entity persistence [Bis06], Java Transaction API (JTA) to coordinate parties involved in possibly distributed transactional operations [JTA], and Java Message Service (JMS) to support persistent and reliable message

exchanges among system parts [JMS]. As a database support, we exploit MySQL server [MySQL], an outstanding open source database application with support for transactions and master-slave replication. Besides, to effectively persist middleware entities (in the form of Java objects) on a relational database, we leverage Hibernate leading open source persistence framework [Hibernate] for Object-Relational Mapping (ORM). Finally, application server provides transparent support for Java Remote Method Invocation (RMI) among system components [RMI] as well as efficient caching and clustering mechanisms.

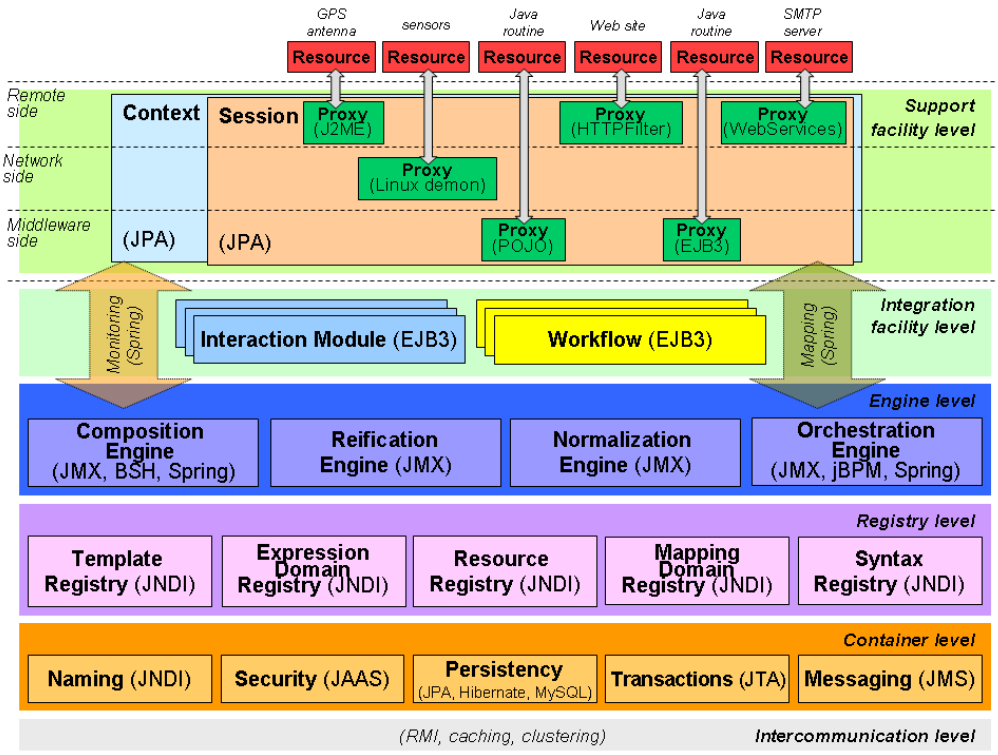

Illustration 26: Middleware implementation technologies

Application server clustering facility, in particular, has allowed us to achieve load balancing and middleware scalability in very easy and transparent ways. We have therefore avoided typical client-server solutions and limitations and easily provided a distributed and scalable solution with

almost no effort, but deployment configuration and administration. Components up to the whole platform can indeed be easily replicated and/or moved over the network nodes where application server instances are running, while sharing the same knowledges and bases of information. By doing so, we have managed to keep most of the computation at the middleware side, thus posing virtually no constraints on client device capabilities and remote servers hosting the services being exploited. This has resulted in a highly powerful and lightweight approach for integrating resources, that can be ultimately old and legacy ones, too.

7.2 Engine level

Components in the *Engine* level extend traditional application server facilities by leveraging JBoss support for the Java Management eXtension (JMX) specification [JMX]. JMX objects, in particular, permit to develop software components that can both execute autonomously within the application server environment – by running and controlling threads, holding in-memory data, and so on – and serve requests from other, same or higher level, components. Though other promising standard specifications are emerging, and we will consider them in the near future – e.g., the Open Service Gateway initiative (OSGi) [OSGi] –, JMX technology currently constitutes the main way through which our middleware prototype implementation deals with resource management problems, in terms both of control on object dependencies and policy enforcement.

Reification Engine is a custom Java component, exposed as a JMX service, that accepts registration of resource proxy metadata and implementations, by leveraging the underlying *Resource Registry*. Besides, it is also responsible for maintaining information about RRM status of each resource proxy in the system and to enforce RRM steps.

Normalization Engine is a custom Java component, too. As a JMX service, it accepts registration of syntaxes and corresponding normalization algorithms, by leveraging the underlying *Syntax Registry*. It is then the

interpreter of such algorithms, thus the transformer of actual raw intercepted activity data into middleware commands.

Composition Engine is in charge of evaluating composability expressions (from the *Expression Domain Registry*) within composition template schemata (from the *Template Registry*) in order to satisfy a given scenario requirements, by producing an actual composition. To ease expression formulation and adoption, hence metadata provisioning by proxy developers, we model composability expressions in the form of Java language ordinary expressions. Metadata enter evaluation as properties from the resource proxy objects that play the desired roles, being selected by means of ordinary "getter" and "setter" methods. To grant flexibility and extensibility, expression domains are not compiled to Java classes, but undergo evaluation by means of a run-time interpreter. In details, our implementation builds on top of the BeanShell lightweight scripting interpreter [BeanShell].

Finally, *Orchestration Engine* employs an open source and third-party provided framework to describe and enact the workflows that realize the business processes within our system. Also from the JBoss group, the Java Business Process Management (jBPM) platform [jBPM] realizes a powerful workflow execution engine, able to support even passivation and resumption of long-lasting processes, for instance in case of currently unavailable resources or human direct intervention in intermediate content transformations. jBPM workflows can either be described via the jBPM-specific Java Process Definition Language (jPDL) or the standard BPEL composition language. By choosing the first-one for the sake of clarity and simplicity, Listing 9 reports a sample from an "RSS-to-mail" workflow:

```
<?xml version="1.0" encoding="UTF-8"?>

<process-definition name="rsstomail_workflow">

  <start-state name="start">
    <transition name="begin" to="state_1"/>
  </start-state>

  <state name="state_1">
    <transition name="service_1" to="state_2">
      <action name="rss_srvc" class="it.swimm.workflow.jbpm.Ejb3Handler">
        <typology>content_generation</typology>
```

```
        <subtypology>rss_reader</subtypology>
        <deployment>
          java.lang.String:servicetype:EJB;java.lang.String:host: localhost;
          java.lang.Integer:jndiport:1099;
          java.lang.Str ing:jndiname:RSSReaderService/local
        </deployment>
        <method>downloadNews</method>
        <arguments>
          java.lang.String[]:urls:request(/PARAMS/urls)
        </arguments>
        <return>
          java.lang.String:rss:execution(/PAYLOAD)
        </return>
      </action>
    </transition>
  </state>

  <state name="state_2">
    <transition name="service_2" to="state_3">
      <action name="rss2txt_srvc" class="it.swimm.workflow.jbpm.Ejb3Handler">
        <typology>content_adaptation</typology>
        <subtypology>text_converter</subtypology>
        <deployment>
          java.lang.String:servicetype:EJB;java.lang.String:host: localhost;
          java.lang.Integer:jndiport:1099;
          java.lang.Str ing:jndiname:RSS2TextService/local
        </deployment>
        <method>extractNews</method>
        <arguments>
          java.lang.String:in:execution(/PAYLOAD)
        </arguments>
        <return>
          java.lang.String:out:execution(/PAYLOAD)
        </return>
      </action>
    </transition>
  </state>

  <state name="state_3">
    ...
  </state>

    ...

</process-definition>
```

Listing 9 – Workflow sample in jPDL language

As description shows, orchestration of services performing content generation (RSS reading), adaptation (RSS feed to plain text conversion) and delivery (e-mail sending) is as simple as performing traditional EJB3 lookups and invocations. jPDL listing reports metadata needed to complete this task (deployment information, method names and argument mapping to existing properties, if needed) and assigns actions to an object of custom class

`it.swimm.workflow.jbpm.Ejb3Handler`, we have developed on purpose to manage invocation of methods on EJB3 components. *Composition Engine* saves workflow description to middleware persistence layer, for later execution by the *Orchestration Engine*, and associates it to the name "rsstomail_workflow".

7.3 Integration and support facility level

Interaction Modules and *Workflows* come in the form of version 3 Enterprise JavaBeans (EJB3) [EJB]. In particular, according to current EJB3 specification, interaction modules are Session-type EJBs, that execute upon activity interceptor initiative, whereas workflows are Entity-type EJBs, bearing business process definition, partial computation results and status. This inherently provides for scalability support at the application server level, thanks to container-managed pooling, caching, and clustering mechanisms.

By exploiting the different flavors of Session-type EJBs, interaction modules manage to support different interaction patterns, such as the request-only one (via Message-Driven Beans), the request-response one (via Stateless Beans), conversational ones (via Stateful Beans) and combinations of them, up to supporting multi-party interactions (via Singleton Beans, from the most recent EJB3.1 specification release). As for workflows, JBoss support for distributed transactions and clustered data cache, through Hibernate, enables entity management through different application server nodes, hence load balancing and fault tolerance.

Context and *Session* blackboards are provided to resource proxies in terms of a simple API that proxy themselves can leverage to gain access to reliable data storage, independently of their actual location. API implementation currently bases on Java Persistence API, too. Remote access to non-Java software modules, such as Win32 client applications managing GPS measurements or Linux network gateways tracking device connections, rely on custom adapters to deal with API implementation and data marshaling towards actual Java Persistence layer. We have evaluated the viability of other

approaches, too, such as in-memory databases and cross-platform support frameworks; experiments have been conducted, in particular, on Oracle Coherence [Coherence], a proprietary in-memory distributed data grid for clustered applications and application servers.

Monitoring of relevant properties from context, session, and other data domains (that composability expressions mandate to assert validity of existing business processes), as well as mapping of proxy invocation arguments (that need resolution to perform workflow orchestration) are crosscutting concerns that spans multiple middleware components and levels. We have enforced Aspect-Oriented Programming (AOP) techniques to manage with them, by associating execution of monitoring and parameter resolution routines to relevant middleware activities entailing such properties, such as value modifications and resource proxy invocations. Current AOP support leverages SpringFramework facilities [Joh05]: Spring AOP benefits of low complexity, in that it supports runtime configuration to weave aspects – i.e., execution of crosscutting functionalities – into execution of methods from other objects. We have investigated different solutions, in order to smoothly integrate the Spring lightweight container within the JBoss application server infrastructure, and eventually permitted a synergistic coexistence of the two environments.

7.4 Resource proxies

Resource proxies undergo different forms and implementing technologies, depending on the actual business processes to support. In time, to demonstrate our approach viability, we have realized a number of actual application scenarios and developed corresponding proxies for the resources that they involve.

We have adopted several different solutions, often arranged together within the same workflow to orchestrate rich business processes. To provide some examples, we have typically exploited WebServices and EJBs to model proxies for remote services, as Listing 9 showed. For instance, proxy for the voice synthesis service bases on a Stateless Session Bean component, able to

communicate via telnet to a remote Festival server, i.e., the actual resource to integrate. SMTP server resource for e-mail sending, instead, integrates by means of a simple Plain Old Java Object (POJO) proxy, exposed as a platform agnostic Web Services interface that expects additional invocation arguments to customize messages. Besides, other content transformers, such as an XML parser for RSS documents, and a picture-downsizer to improve web navigation on slow connections, are just POJOs that directly run at middleware side.

As for activity interceptors, we have managed to intercept HTTP requests by means of both coarse- and fine-grained web components. HTTP proxies as well as HTTP filters permit us to intervene on legacy requests, for the wide Internet area and local web site resources, respectively. Instead, specific web application components (often in the form of JEE Servlets and JSP pages) let us accept requests that provide well-defined headers, cookies and parameters. Other stand-alone applications, running on local or remote network nodes, allow us to monitor incoming requests, for instance on SMS gateways or VoIP servers, whereas Windows Mobile or Java Micro Edition (JME) applications provides functionalities to deal with peripherals on mobile devices, for instance to read current GPS coordinates and send them as UDP datagrams to a suitable, request-only, interaction module endpoint.

Chapter 8 – Some Scenarios

The middleware prototype we have developed covers the discussed core architectural levels and a few basic types of interaction modules, featuring support for the most common interaction paradigms such as request-only, request-response, publish/subscribe, and conversational ones. By leveraging registries, it allows for dynamic addition of composition templates, as well as of composability expression domains, syntax normalization algorithms, and property domains to map invocation arguments of resources that participate in business processes. Finally, it can be extended not only in terms of resource proxies that it lets integrate and workflows that it can enact as a result for the composition calculus, but also with novel types of interaction modules.

After providing it with knowledge of an initial set of composability domains and quite a numerous set of resource proxies, we have thoroughly tested it in several different scenarios, to stress critical Ubiquitous Internet problems and enforce novel Ubiquitous Internet applications.

8.1 Campus Web site

A typical use case is with one student that can access the Internet by means of her personal smartphone, either by exploiting a slow GPRS connection or a faster Wi-Fi one, and wants to read Web pages from the campus Web site. Furthermore, college provides a news service she is particularly interested in, a shared student calendar with indication of campus events and a blog service where students can comment on aspects of campus life, music, politics, and so on.

We expose service configuration facilities via a dedicated Web application, to let students express their preferences. Student we observe has chosen:

- to subscribe to the campus news service;

- to receive news title via SMS messages on her phone as soon as news become available;

And, furthermore:

- to have daily e-mail reports of the full content of the news of the day;
- to enrich every daily report with calendar events regarding next seven days, starting from the present date;
- to aggregate to this report also contributions from the blogs of two friends of her.

As for Web browsing, then, she has chosen:

- when surfing the campus Web pages through a GPRS connection, to have middleware resize pages to fit her device screen and reduce dimension of image files to save bandwidth;
- when surfing the campus Web pages through a Wi-Fi connection, to just have middleware resize pages.

These requirements point out important aspects our middleware supports. First of all, user can exploit both synchronous and asynchronous interaction paradigms, via the HTTP request/response message exchange pattern and the news service publish/subscribe one, respectively; indeed, some functionalities obey a “pull”-type provisioning model and are only useful when she connects to the campus site; other ones, instead, realize “push”-type content provisioning and must be running even when she is not logged onto the system. Secondly, changes in the user context can cause service compositions to change accordingly, at run-time, such as when the shift from GPRS to Wi-Fi connection type occurs.

The Web configuration interface we expose (Illustration 27) is in charge of collecting user preferences and to assemble them into a corresponding set of requirements that middleware Composition Engine can leverage to compute a suitable resource composition. In the case we consider, preferences also entail constraints that require different business processes, according to different

context conditions. Of course, user neither directly indicates composability expressions nor provide requirement description on her own, but she selects intuitive GUI controls that achieve the desired effect by adding corresponding composability expressions to her scenario requirements.

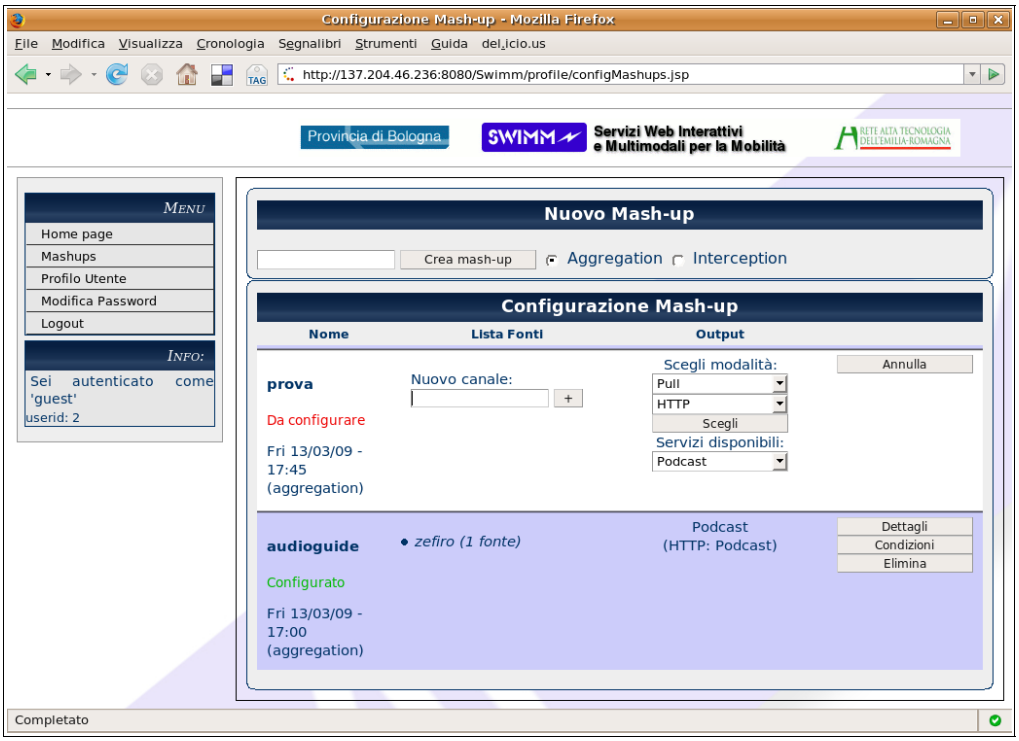

Illustration 27: Graphical user interface for middleware configuration

According to RRM, middleware activates (if needed) all resource proxies that participate in the business processes that satisfy current scenario requirements, and it configures them accordingly. On the contrary, middleware does not reserve resource proxy configurations for non-running processes.

Thus, when user is not logged in, middleware does not need Web content adaptation process and it does not reserve any proxy configuration for it. Indeed, composition calculus fails since constraint on user authentication

status makes no actual proxy composition satisfy the scenario requirements. Nevertheless, monitoring of user authentication status takes place all the same, to enable reactions to property changes that may lead to feasibility.

Web content adaptation workflows are saved to system entities only in case user logs in to the middleware, via the campus Web site; by that time, middleware selects participating proxies as a function of actual user connectivity type.

Business processes that relate to news forwarding via SMS messages and mail delivery, instead, correspond to workflows that must always be available and ready to run at any given moment, to serve the events of "news publication" and "mail sending-time reached".

Finally, monitoring of user authentication status and connectivity type permits workflows construction to be pro-active: compositions are not created when the user actually exploits them, but as soon as her characteristics vary. This has proven to work very well as a solution to the trade-off between responsiveness and average computational load.

8.1.1 Notification of news availability

Considering campus Web site as a legacy resource to integrate within business processes, availability of campus news represents the actual resource activity that campus site proxy must intercept and forward to the middleware for further processing. Thus, campus site proxy can be, in effect, an RSS reader application.

Since campus Web site – of course – is not interested in processing results, message exchange pattern between proxy and the middleware can leverage the request-only interaction paradigm to just trigger execution of the workflow that realizes the desired content processing.

Anyway, more than a user may share interest in the same campus news, despite they indicate content processing through different workflows. For the sake of these business processes, hence, campus Web site proxy accepts configuration arguments in the form of identifiers of workflow subscriptions

that wish to receive campus news.

As Illustration 28 shows, to enforce one-to-many content distribution, campus site proxy forwards news raw data to the middleware, along with the collection of all subscription identifiers for that content, and message syntax indication to normalize data to middleware commands. On the middleware side, we provide an appropriate publish/subscribe interaction module that can leverage subscription identifiers to enact as many different workflows as required.

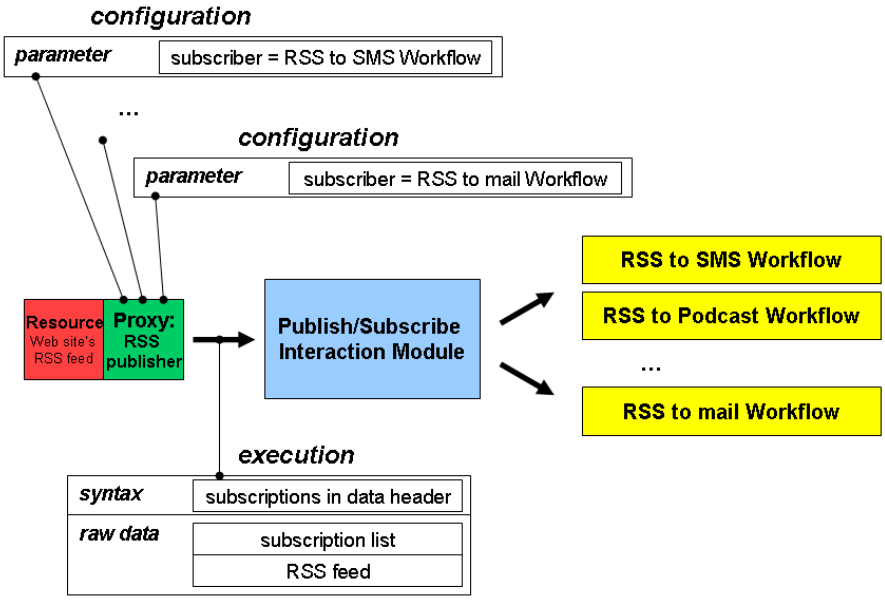

Illustration 28: Different configurations of the same resource proxy in different business processes to perform publish/subscribe interaction

Finally, in the case we consider, by supposing updates of campus site just consist in publication of RSS feed documents, we enable SMS sending by means of a simple workflow that analyzes those documents, extracts RSS feed titles and gathers them in a text message, finally delivered via a GSM gateway (Illustration 29).

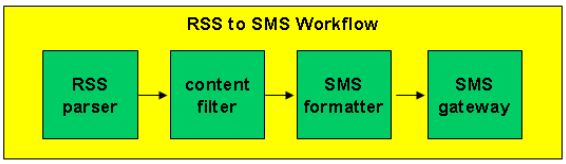

Illustration 29: RSS to SMS Workflow

8.1.2 Scheduled content aggregation delivery

User preferences we consider permit to exemplify publish/subscribe mechanisms once again. Indeed, since user has chosen to also get campus news in the form of daily e-mail reports, another business process of hers entails workflow subscription to the campus "news publication" event. By simply leveraging a buffering service, such a workflow enables collection of RSS news for later processing and delivery, perhaps at a given time of the day.

The whole business process that models "scheduled content aggregation and delivery via e-mail" consists of two separate workflows, as Illustration 30 reports. The first one, as seen, subscribes to RSS updates to store campus news RSS documents to a temporary buffer. The second one, instead, executes upon scheduled events to retrieve contents from that buffer, aggregate it to content from other sources, and finally perform e-mail sending.

By intercepting activities of system clock (as an actual resource), a "timer service" proxy permits to model scheduled business processes. It reserves business process configuration by accepting the "identifier of the workflow to enact" and the "desired daytime to trigger execution" as parameters of its RRM configuration method. When the given time comes, then, it self-performs RRM execution and forwards to the middleware request-only interaction module the raw data that describes the current event (including workflow-to-enact indication), along with the appropriate syntax that lets middleware normalize message, hence identify and run the desired workflow.

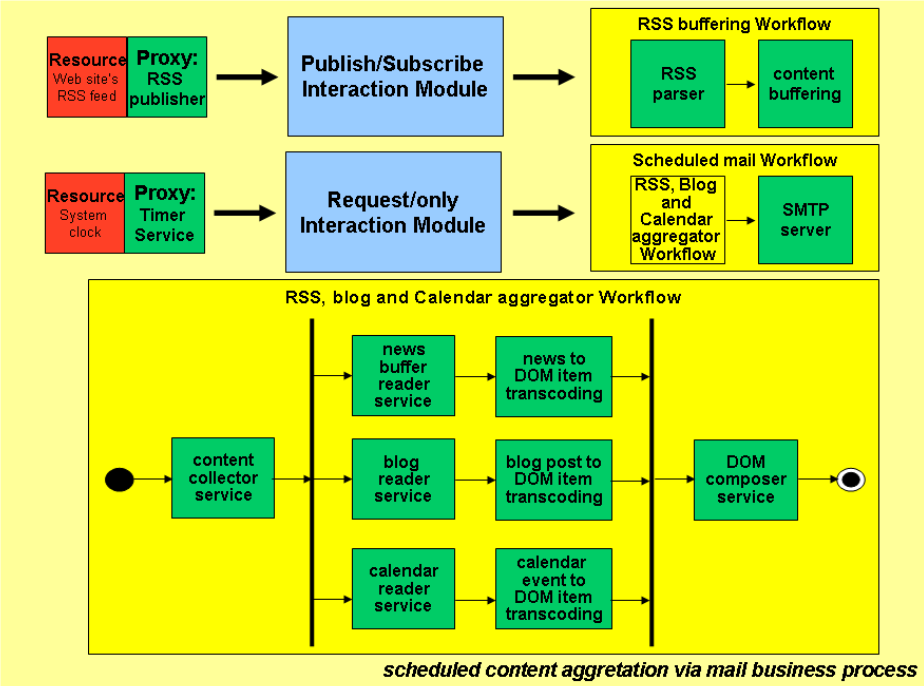

Illustration 30: Scheduled content aggregation and delivery

8.1.3 Web content adaptation

As the user logs in to the campus Web site, middleware determines a suitable resource composition to provide Web content adaptation.

This time, activity interceptor proxy consists in a simple HTTP filter component that campus Web server associates to all pages from the campus Web site. Filter allows configuration by means of "user identity" and "workflow identifier" arguments. Then, when user requests a page from the campus site, filter intervenes on her browser HTTP request to operate as follows:

- it extracts the client IP from the request data and stores it to the business process context blackboard;
- it modifies the request by adding a further HTTP header, indicating the workflow that middleware must enact for that user (as specified at

configuration time);

- it forwards the modified HTTP request to the middleware request/response interaction module, once again along with syntax information.

In the very beginning, user connectivity type is unknown; composability expressions therefore behave as if we were in the worst-case. Practically speaking, this is done by preventing selection of picture down-sampling service if connection type certainly is from a broadband Internet Service Provider (ISP). Thus, by assuming that user exploits a GPRS connection at first, resource composition by the middleware (Illustration 31) expects:

- a first resource proxy, to submit client IP address information in the business process context to an IP database, in order to learn about approximate client location and Internet Service Provider (ISP), and store such information to context, too;
- a second proxy, to analyze the HTTP request headers and save relevant device information to the business process context blackboard, such as user-agent characteristics and client device capabilities (e.g., screen resolution), by leveraging the WURFL specification file [Pas08] as its own resource;
- following service in the workflow, to actually serve the user request and save the resulting HTML content to the current workflow execution payload, for further processing;
- a forth proxy, to exploit an ImageMagick-based [ImageMagick] service in order to reduce the size in kilobytes of page images (via down-sampling), and to modify the HTML payload accordingly (by linking modified image versions, instead of the original ones);
- last stage in the workflow, finally, to modify the body style of the HTML page payload, in order to fit user device screen resolution, according to device information in the business process context.

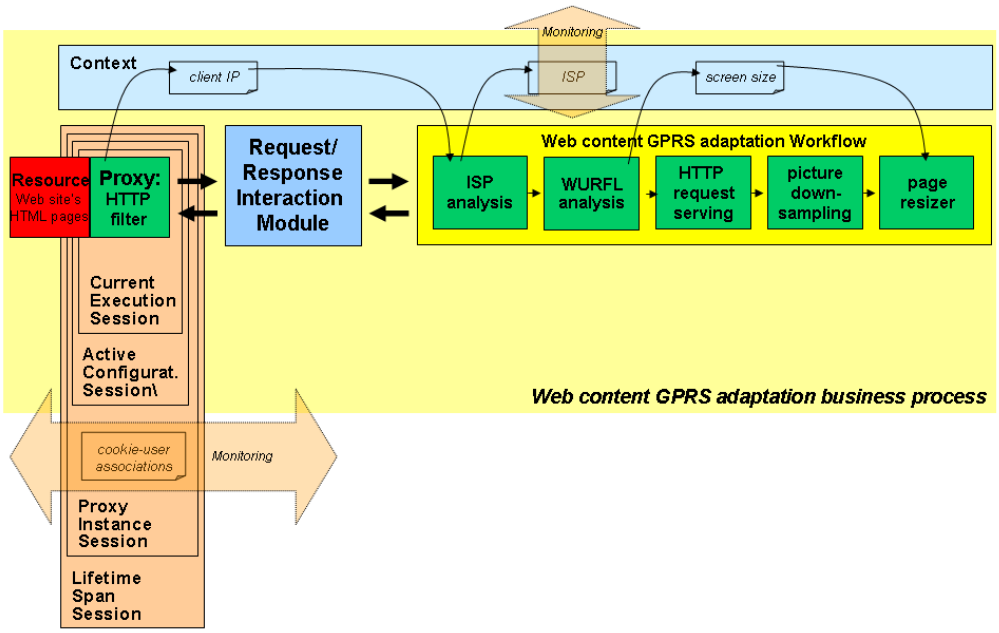

Illustration 31: Web content adaptation (GPRS case)

Given this workflow characteristics, as soon as user connectivity type changes, context properties that relate to her IP address and ISP are changed accordingly. In particular, middleware performs monitoring on the latter value to prevent picture down-sampling in case ISP is recognized to be a broadband provider. When this happens, it forces composition calculus to evaluate scenario requirements again, hence substitutes the current web content adaptation business process with its Wi-Fi version (Illustration 32).

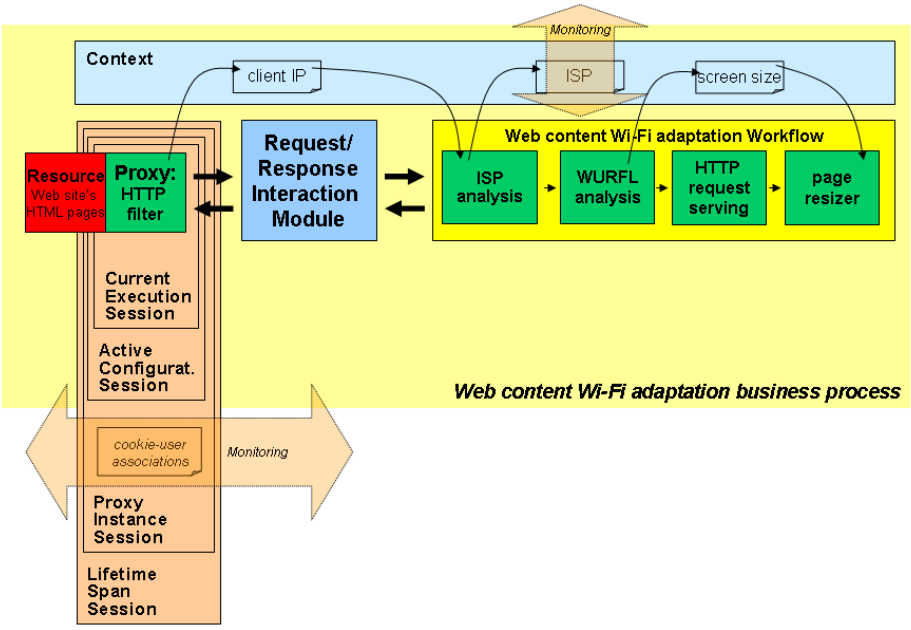

Illustration 32: Web content adaptation (Wi-Fi case)

It has to be observed that HTTP filter has no means to directly recognize identity of a specific user upon her HTTP requests. Indeed, although requests convey session cookies, filter cannot directly exploit such cookies to tell the precise user identity, hence select her corresponding workflow to command: session cookies just distinguish different users, but do not provide identity information.

To overcome this, when user requests a page from the campus Web site first, she is presented a login form to fill in. By also belonging to the "Web identification process" depicted in Illustration 33, HTTP filter proxy intercepts this form submission and forwards data to the "Web login" Workflow, to evaluate credentials, authenticate users, and – most important – append user identity information to the response.

Before returning results to the final user, filter reads (and removes) explicit user identity indication from the response and leverages its own proxy instance session scope to save the association between user identity and the current HTTP session cookie from the Web server.

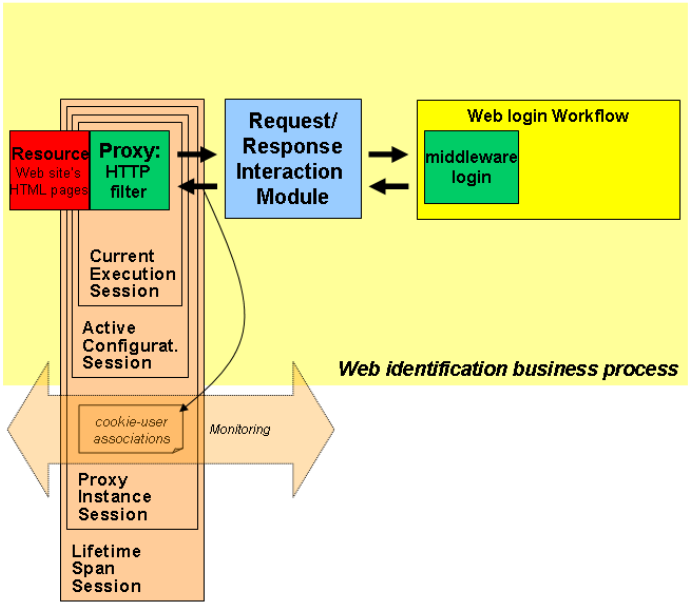

Illustration 33: Web identification

Session-managed user identity information enables monitoring of user authentication status by the middleware itself, hence configuration and deconfiguration of her Web content adaptation process, as she logs in and out. Besides, given the chosen session scope, cookie-identity association is available to all business processes in which HTTP filter participates; therefore, it can survive business process reconfiguration (for instance, from GPRS to Wi-Fi adaptation), just as traditional browser cookies do.

From this moment on, by leveraging cookie-identity association, HTTP filter manages to tell user identity upon HTTP requests and choose the corresponding workflow to demand orchestration of.

8.2 Personal podcast channel

"Personal podcast channel" belongs to a number of Ubiquitous Internet application scenarios we have developed, in partnership with an enterprise consortium to provide pervasive services in the field of tourism. Precisely, aim

of this scenario is to enable travelers to download tourist guide excerpts as Mp3 tracks on their mobile devices, in podcast format [Podcast].

In simple words, *podcast files* are XML files that resemble the RSS format and embed links to multimedia resources, such as audio files, videos, and so. *Podcast channel* is the term in use to indicate the podcast file URL, i.e., the Internet location from where final users can download its content, usually via an ordinary HTTP request. Subscribing to a podcast channel, hence, consists in saving the podcast file URL to a client application that can automatically recognize updates and download podcast tracks at regular intervals (e.g., Apple iTunes [iTunes] or Mozilla Songbird [Songbird]) .

Within the "personal podcast channel" scenario, we enable personalization of the podcast content by leveraging additional user-specific information to filter and download customized data. In details, scenario requires:

- user to communicate her current geographical position to the middleware, by leveraging a GPS device connected to her PC;
- user to subscribe to a given podcast channel by means of Apple iTunes application, from her PC;
- middleware to provide a business process that intercepts iTunes request for the podcast file, and arranges a customized podcast content by leveraging user geographical position, geographically-related excerpts of a tourist guide, and a voice synthesis service;
- finally, user to exploit Apple iTunes to save podcast Mp3 tracks to her iPod device.

The choice to enforce legacy iTunes application as an inherent part of the scenario depends on the supposed habits of final users: scenario is targeted to yachtsman tourists, used to leverage a notebook PC when on-board, and to carry just an iPod music player with them when visiting the country.

As Illustration 34 shows, process realization bases on two different types

of activity interceptor and two different workflows for content processing.

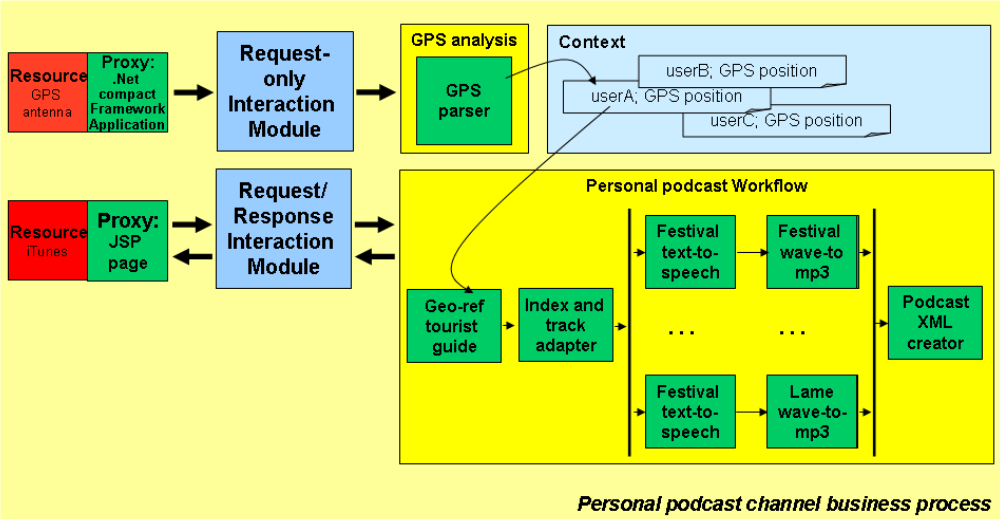

Illustration 34: Personal podcast channel

As for GPS positioning, we have leveraged Microsoft .Net Framework [dotNet] to develop a simple client application, running in the background of the user PC, that reads coordinates via serial port commands on the device GPS module and sends them as UDP datagrams to the middleware. On the middleware side, we have developed a UDP front-end for the message queue that we associate to middleware request-only interaction module, to let it receive and normalize UDP datagrams. Middleware commands in the datagrams convey user identity and coordinates, and demand running the "GPS analysis" workflow.

Proxy for the podcast subscriber application consists in a simple JSP page running at a given URL, such as:

http://137.204.46.234:8080/Podcast/init.jsp

Interception of podcast channel requests from iTunes happens by simply leveraging this address as the starting point to provide the podcast service. JSP logic handles requests targeted to this URL, and forwards them to the middleware request/response interaction module, for processing by the

“Personal podcast” Workflow.

To let middleware identify the user, hence write and read her correct GPS position in the business process context, we provide each user with a different identification parameter: such value must be provided to both the GPS application (at its startup), and the podcast subscriber one (as a URL query fragment to append to the podcast/JSP page URL):

`C:\gpsdemon.exe` **`userA`** *(or via the application GUI)*

*http://137.204.46.234:8080/Podcast/init.jsp?**id=userA***

In case user cannot run the GPS application, JSP page can be invoked from a traditional browser, too: response is an interactive map of Southern Italy (Illustration 36), where user can click on her approximate position and get a popup message with the corresponding podcast URL to provide to iTunes (Illustration 35).

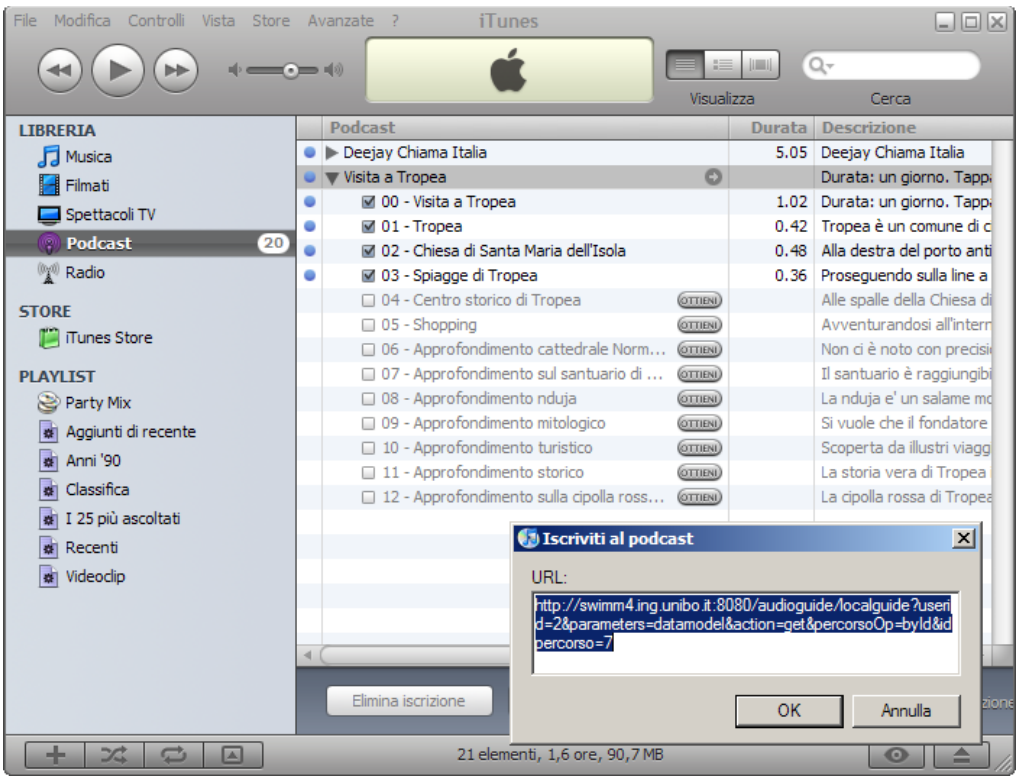

Illustration 35: Podcast subscription with iTunes

Illustration 36: Access to the personal podcast channel via a traditional Web browser

Finally, by requesting the given URL, user obtains her podcast customized content as the result of the following sequence of activities (Illustration 34):

- a first resource proxy retrieves user coordinates from the process context and invokes the tourist guide Web Services accordingly;
- a custom Java routine analyzes guide items, assemble them to an ordered list and make mutual references explicit by appending suitable predefined text to each item (i.e., *"to get more information on... go to track number ..."*);

- a text-to-speech application, namely Festival [Festival], synthesizes item text to wave files (and saves them to a cache to improve following executions on the same data);
- an audio converter, namely Lame [LAME], transforms wave files to Mp3 tracks (and saves them to a cache);
- finally, guide items and Mp3 tracks are arranged together to create a suitable podcast XML descriptor file.

8.3 Middleware configuration

The Web configuration interface in Illustration 27, through which we mask to final users the burden of assembling scenario requirements, is not actually an *ad hoc* application that directly accesses and commands middleware components to achieve its results. Rather, such a Web application and any other graphical tool – we have exploited to manage and configure middleware – are all resource proxies that we leverage to intercept management and configuration activities in different forms, and to command the middleware accordingly.

Indeed, alike with conventional resource activities/requests, it is possible to label management and configuration requests with suitable syntax indications too, and to command middleware behavior accordingly.

Illustration 37, in the following, briefly reports the mechanism through which an activity interceptor proxy demands middleware orchestration of a certain workflow:

- explicit requests or information about current activities are intercepted on the actual resource;
- proxy labels raw activity data with the corresponding syntax information, in order to describe their format to the middleware, and let it understand how to behave as a consequence;
- raw activity data are sent to the middleware via the interaction module

component that realizes the needed interaction paradigm;

- by leveraging *Normalization Engine*, middleware analyzes the activity information to identify its origin (typically, the requesting user), extract parameters (if any), and translate raw data into middleware commands;
- finally, if activity demands orchestration of one or more workflows from a given business process, middleware exploits the *Orchestration Engine* to suitably serve the request.

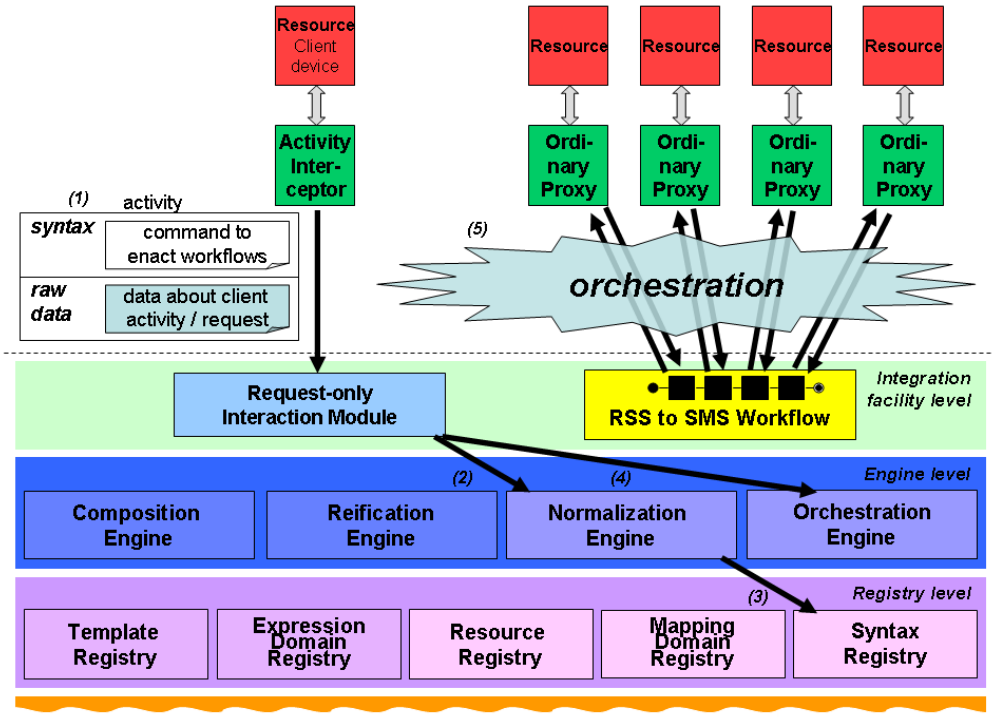

Illustration 37: Activity interception to command workflow orchestration

Given these premises, following paragraphs will demonstrate how it is possible to leverage similar mechanisms for commanding other middleware behaviors than the process orchestration one, hence performing management and configuration operations; for instance, in case of registration of a novel resource proxy and of creation of a novel resource composition.

8.3.1 Resource proxy registration

Resource proxy registration consists in submitting to the middleware:

- the metadata that describe resource/proxy characteristics, for the sake of composition within business processes;
- the actual implementation of the resource proxy (or, at least, a facade [Gam94] to command it), for the sake of activation, configuration and execution by the middleware, according to its reification model.

Submission can leverage a simple request/response interaction paradigm in order to command middleware *Reification Engine* to store metadata and proxy implementation in the *Resource Registry* (Illustration 38).

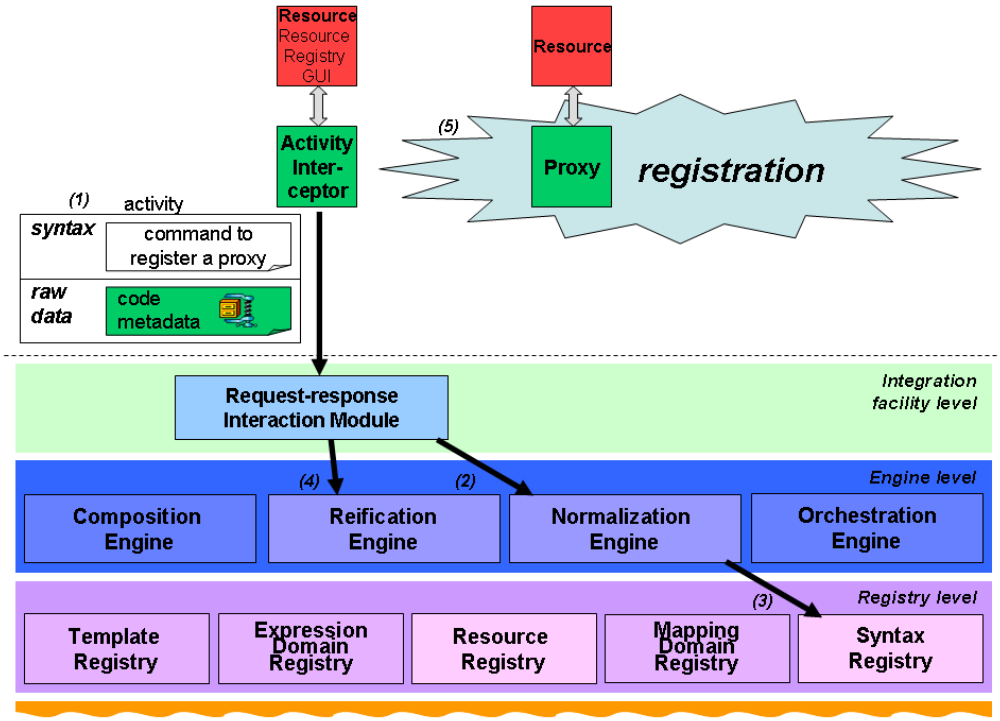

Illustration 38: Activity interception to command resource proxy registration

To easily enable registration and deregistration of resource proxies, we have therefore developed a simple Web application that reports the list of

currently available resource proxies, and permits both adding new entries (by submitting their code and metadata, in the form of *.jar* archives), and deleting existing ones. Illustration 39, below, reports a snapshot of such application.

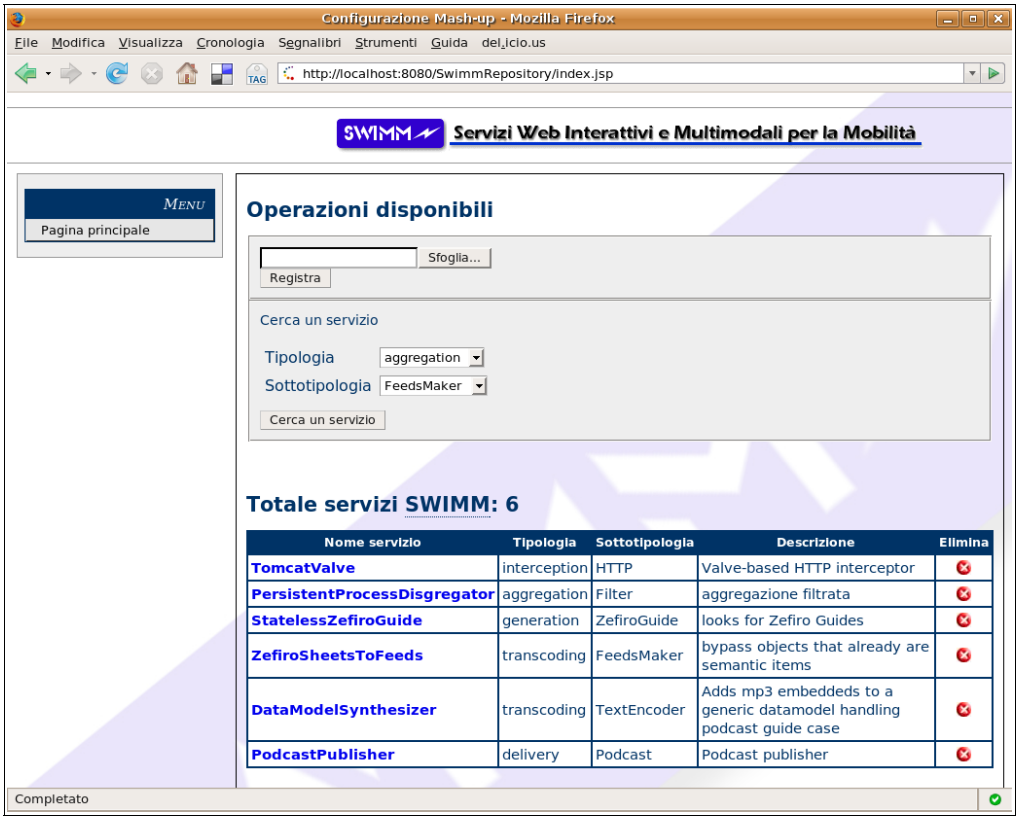

Illustration 39: Resource proxy registration by means of a Web application

8.3.2 Creation of a novel resource composition

To enable effective setup of novel resource compositions, hence to create and save their corresponding workflows to the middleware, it is necessary to present final users with simple choices and selections on a friendly graphical interface. Scenario requirements are then created step by step, by letting users express their own preferences in terms of device to exploit for a given service scenario, type of content adaptation, output media format, and so on.

Anyway, in several circumstances, operations that users can perform depend on (or are mandate by) their previous choices. For instance, upon selection of a given resource, it may be necessary to configure its invocation parameters according to user explicit preferences (as seen) or selection from a given list of possible values. Again, when user selects a given resource for her application scenario (e.g, the SMS gateway as the communication channel to receive content), other ones may become no more useful or valid and their selection can be prevented or disabled (e.g., picture-related services, alternative output channels, and more).

As Illustration 40 shows, configuring a resource composition therefore requires a conversational interaction paradigm with the middleware, in order to let it process partial preferences and filter available choices to the final user, until all scenario requirements are in place. Intercepted activities consist hence in incremental sets of scenario requirements and corresponding syntaxes, that a conversational interaction module leverages, via the middleware *Normalization Engine,* to enforce composition calculus by the C*omposition Engine*.

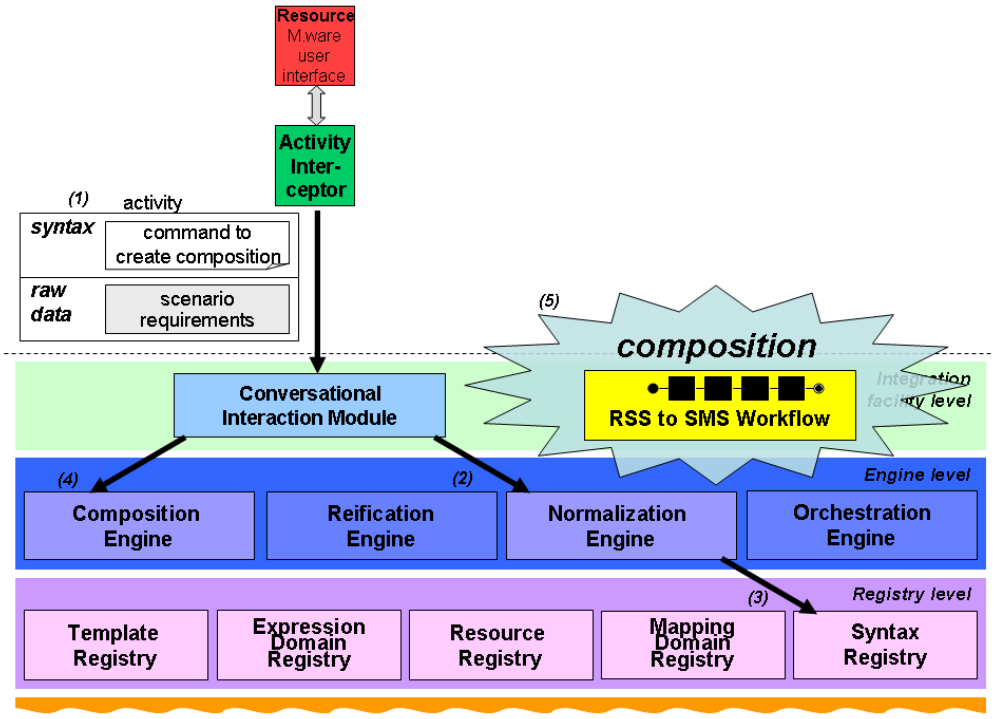

Illustration 40: Activity interception to create a novel resource composition

Illustration 41 and Illustration 42, below, present two more snapshots from the Web application interface for configuring novel resource compositions. In particular, Illustration 41 refers to an intermediate choice of configuration parameters that user has to perform before proceeding. Illustration 42, instead, reports the final result of the conversation, wherein middleware confirms that composition calculus was successful and also advices user on how to exploit the new resource composition. In details, advice message is nothing more than a particular, textual, kind of composition score. Though of course of no use for the sake of composition ranking, composition template permits to create it by indicating a composability expression domain, wherein placeholders in a given statement are substituted by properties from the services that play roles in the composition.

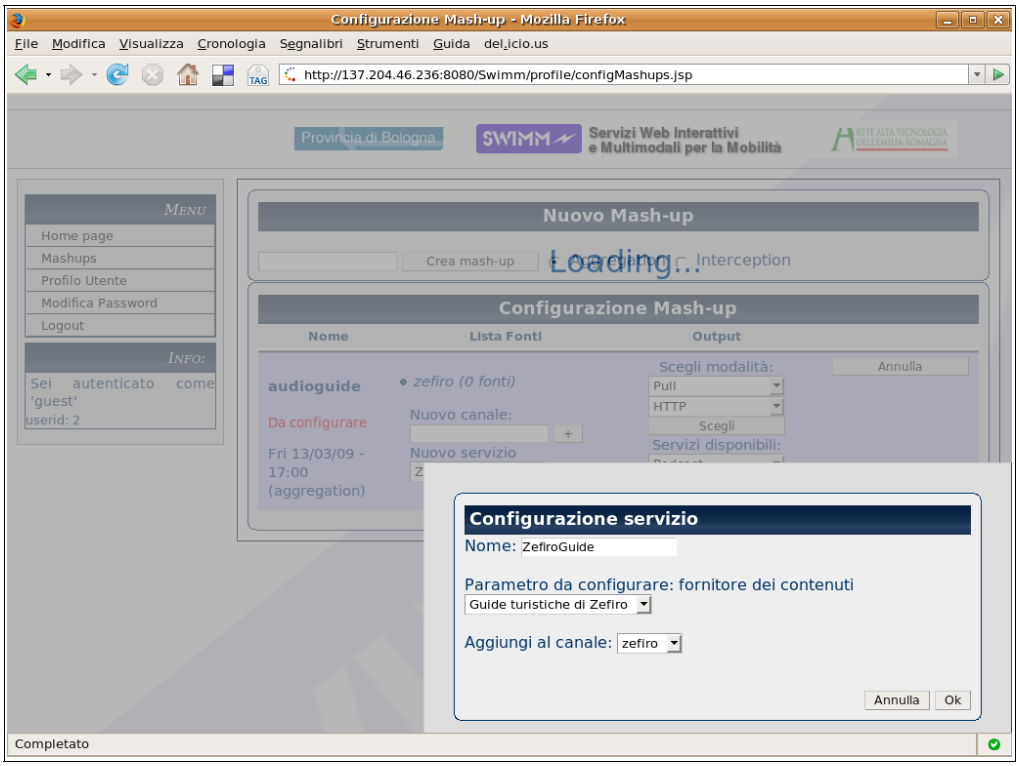

Illustration 41: Choice of configuration parameters for a novel resource composition

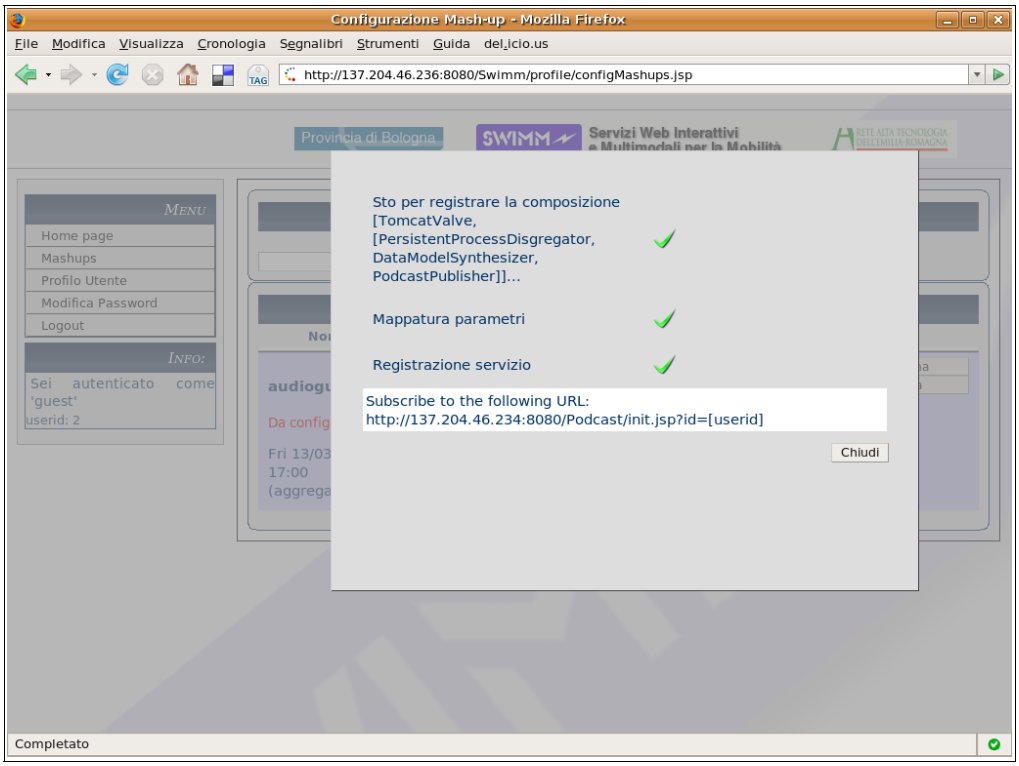

Illustration 42: Result of the creation of a novel resource composition

Chapter 9 – Performance evaluation

To verify the feasibility of our approach, as for computational overhead, impact on system resources of nodes that run middleware components, and system scalability as well, we have intensively stressed the software components involved in the scenarios depicted so far.

We have collected relevant measurements about performance and resource exploitation on both middleware nodes and hosts running resource proxies (and support facilities) only. Since results were similar in quality, independently of the considered scenario, in the following we concentrate on describing single reports in terms of coordination overhead, performance scaling, and memory occupation.

9.1 Coordination overhead

Overhead tests aim at demonstrating how coordination by the middleware impacts the overall execution time in serving user requests. Considered scenario relates to the Web content adaptation example, where middleware orchestrates a set of resource proxies – dealing with the analysis of context conditions, actual content retrieving and successive transformations of it – to let the final user download Web pages that fit her device screen and connectivity type.

To separate middleware overhead contribution from actual service request time, tests leverage workflows from two different business processes sharing the same scenario requirements. In the first process, services actually perform valuable operations such as downloading Web content on the behalf of user, and manipulating images. In the second process, instead, workflows exploit “fake” versions of those services, that perform no time- consuming operations and that immediately return control to the middleware, to just entail its overhead in terms of invocation and coordination. Furthermore, tests point out how our prototype implementation manages to transparently exploit some

relevant application server facilities – such as resource pooling and caching – while serving multiple requests that involve the same kind of middleware components and resource proxy entities.

Tests come as a series of request burst-cycles at very small time intervals, resembling actual scenarios of intense middleware exploitation by final users. AOP techniques let us register suitable observers to both interaction modules and actions entailed by the workflow components, in order to keep track of the elapsed time to normalize requests and enact corresponding workflows. A modified version of the HTTP filter that serves as the browser proxy, is in charge to capture the initial HTTP request from the browser, and to forward several replicas of it to the middleware, in the form of request burst-cycles.

Testbed consists in two identical workstations, say A and B, each one equipped with a 3,06 GHz Intel Pentium4 CPU, 2 Gigabytes of RAM and linux operating system, kernel 2.6.15. Workstation A hosts the middleware central components, from *Intercommunication* and *Container levels* up to the *Integration* and *Support facility* ones, whereas workstation B runs the actual resource proxies for the content-related services and grants them access to middleware *Support facility level.*

Illustration 43 reports average performance results for a a 50-series of request burst-cycles, issued at 100 millisecond time intervals from each other:

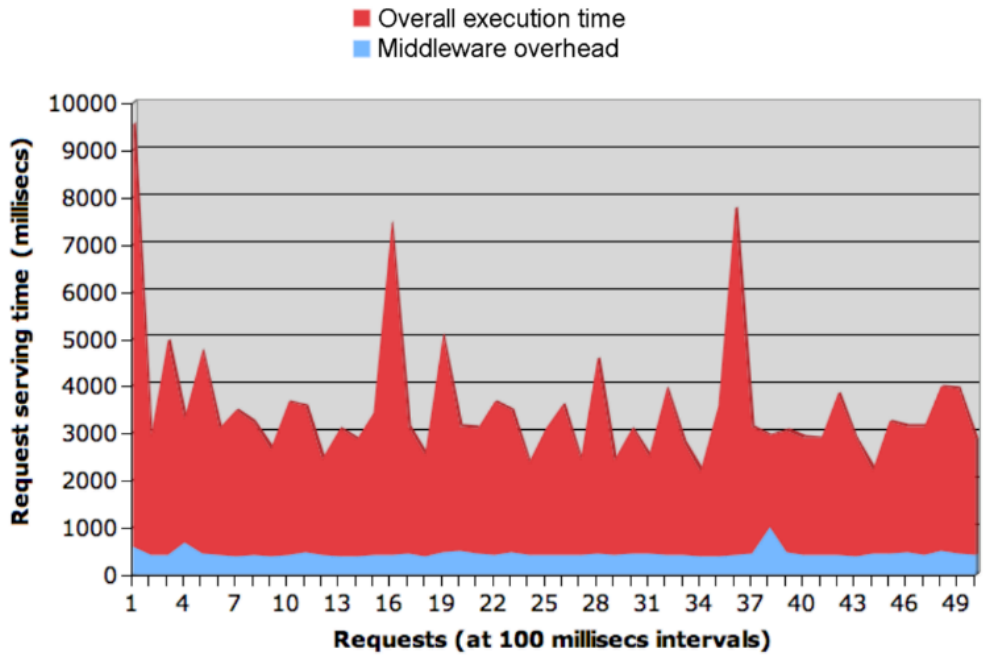

Illustration 43: Web content adaptation burst requests, average serving time

We can observe that heavy system load causes the overall service provisioning to run significantly slow, up to 9 seconds on service startup and regular garbage collection occurrences (i.e., the peeks in the figure). Anyway, this is partially due to network connection establishment and download time when fetching actual Web content and, most important, middleware overhead rarely exceeds 500 milliseconds per request (about 10% to 14% of total time), in order to perform syntax-driven request normalization, workflow resolution, and service orchestration.

Finally, we chose to implement tests in the form of request burst-cycles (slightly spread over time, though partially in overlap), rather than by issuing lots of completely concurrent requests, to both prevent “denial-of- service” effects and to enable EJB container facilities. Indeed, thanks to technology and implementation considerations, system is able to scale well on increasing request numbers, and to impose a nearly constant average overhead. This is possible by leveraging component replicas that the application server provides

within pools, and by exploiting in-memory cache replicas of both persistent objects (such as recently read workflow descriptions) and remote component stubs (such as remote proxy ones).

9.2 Scalability

Two major factors determine the middleware overhead we have experienced in the previous test. On the one hand, remote method invocations on distributed resource proxies involve establishing connections between middleware *Orchestration Engine* and resource proxies themselves. On the other hand, middleware orchestration intervenes in proxy invocation by also performing run-time resolution of part of their execution parameters, as seen. Whereas connection setup type is an intrinsic consequence of coordinating distributed software functionalities and proportionally grows as the number of resource proxies increases, middleware run-time parameter resolution – if not dealt with effectively – can seriously affect the overall performance of business processes.

In particular, at time of resource proxy invocation, parameter resolution can either happen sequentially, one by one resolving all expected values, or in parallel, by exploiting concurrent threads to operate simultaneously. After experimental results (Illustration 44), the first solution proves to work well, especially on business processes that are made up of limited numbers of resource proxies. On the contrary, as the number of proxies increases, sequential implementation of the resolution routines does not sufficiently scale.

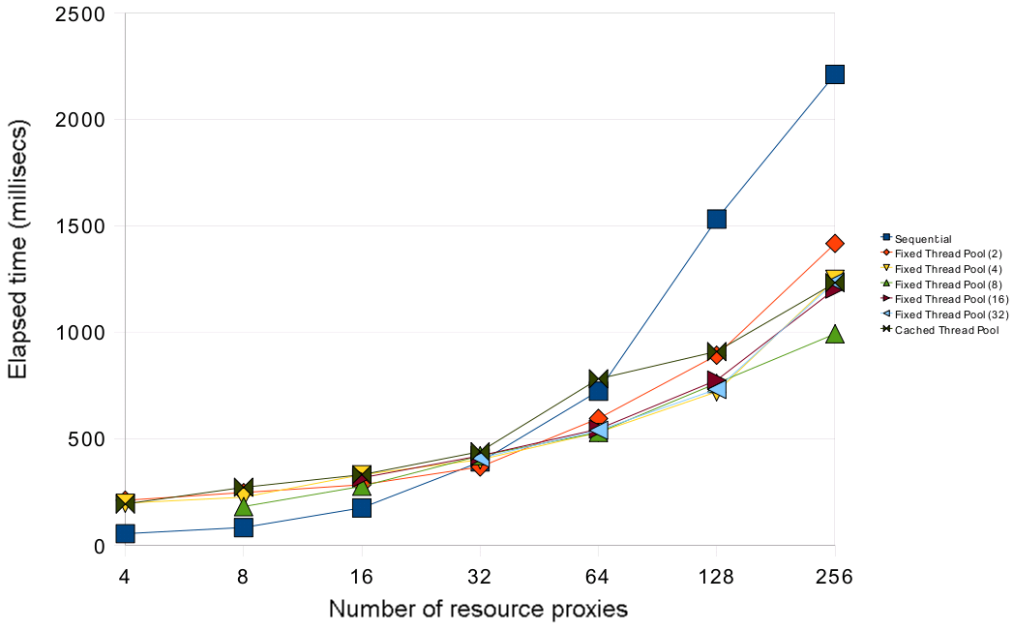

Illustration 44: Elapsed time to perform run-time parameter resolution

We have therefore developed an alternative, concurrent, implementation for the resolution routine, able to leverage execution threads from a pool of of either fixed or varying dimension. Illustration 44 also reports performance of such concurrent implementation, according to different thread pool sizes. Experimental results show that whereas concurrency permits better scaling in business processes of more than 32 resource proxies, overhead from the pool management itself makes this kind of solution far less convenient in simple business processes.

9.3 Memory occupation

Middleware memory occupation largely benefits from the choice of orchestrating business processes by means of workflow activities. Workflows, indeed, determine the execution of flows of operations that are inherently organized in a pipeline form, wherein each pipeline stage (leveraging a particular resource proxy, in our case) gets immediately available after performing its own piece of work, with no need to wait for the whole process

to complete, before accepting further requests.

On the contrary, traditional systems that leverage monolithic servant objects, usually need to handle requests one by one or, to increase parallelism, to instantiate a number of servant object replicas, usually managed within a pool. Anyway, by doing so, memory occupation becomes a crucial problem for these kind of systems. Application server containers, in particular, generally let specify a maximum pool size and dynamically create and destroy servant replicas, according to current system load.

Our middleware solution, by organizing servant objects in workflows, manages to further increase parallelism with little or no additional memory usage, hence to tolerate heavier system load. In details, by dividing request serving into separate pipeline-like stages, we enable reuse of already exploited proxies to serve successive requests before completing current business processes, and – most important – without requiring the application server container to instantiate additional object replicas.

Experimental results demonstrate the effectiveness of our approach by comparing memory occupation in different load conditions. In details, we consider the "Personal Podcast Channel" scenario and issue HTTP requests for the podcast channel that corresponds to a given geographical position. Workstations A and B from previous tests realize this testbed environment, too.

Under all circumstances, HTTP response (bearing the podcast XML descriptor) is returned within 3 seconds to the requesting application: text synthesis leverages cache for the audio files, hence simulation stresses once again middleware orchestration logic and, in particular, its memory needs for creating the remote object stubs that let it invoke actual distributed resource proxies.

Illustration 45 , below, reports memory occupation in case of a series of separate podcast requests. Requests are served one by one: we wait for response to each request before issuing a new one, thus they do not overlap. Container instantiates needed object in the Java heap, and destroy them, after a

little lingering, as they are no longer needed by our middleware applicative logic. Memory occupation remains nearly constant throughout test execution.

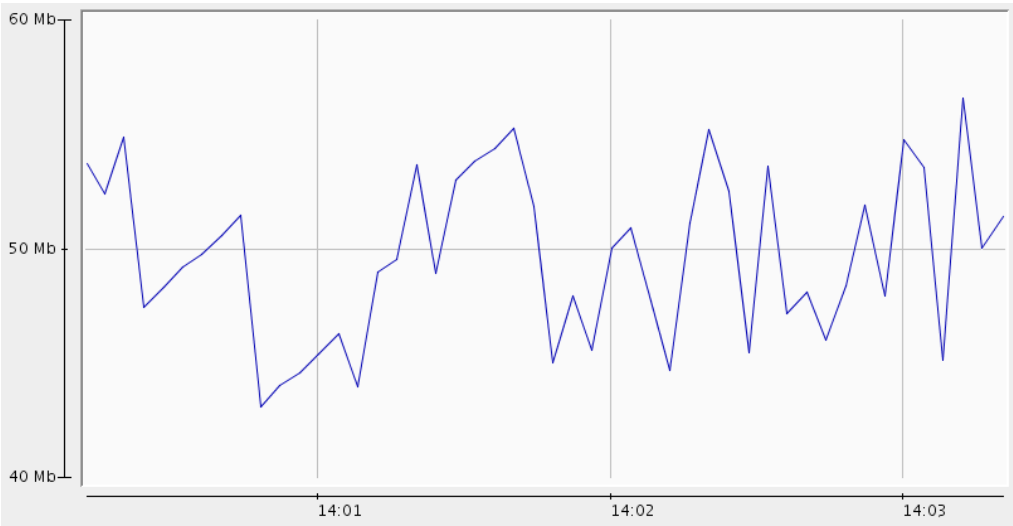

Illustration 45: Memory usage in case of non-overlapping podcast requests

In Illustration 46, instead, we report memory occupation for a series of partially overlapping podcast requests. Workflows permit reusing part of the proxy stub replicas that are already in the application server heap. Thus, despite higher values, especially in the central part of test execution, memory occupation does not experience critical growth.

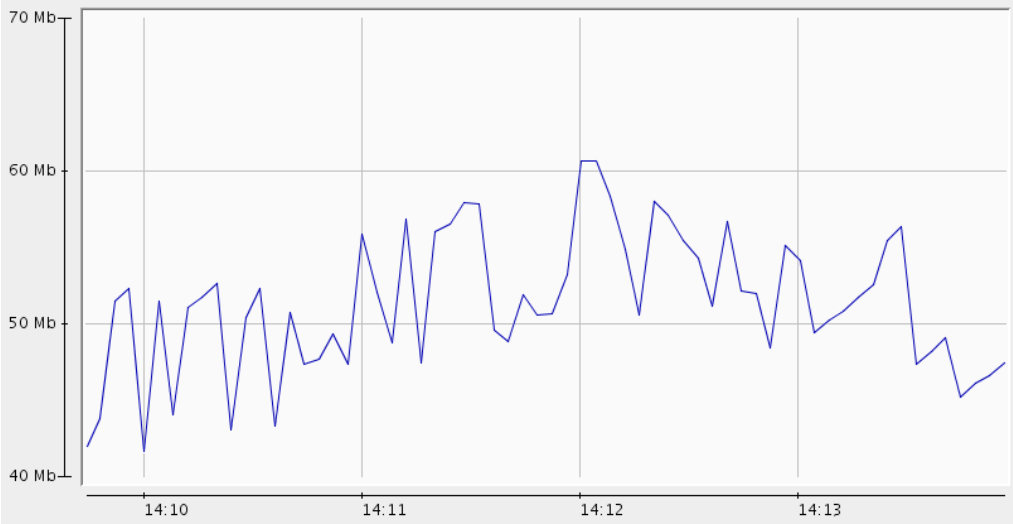

Illustration 46: Memory usage in case of partially overlapping podcast requests

Finally, we stress middleware operations by issuing request burst-cycle series; Illustration 47, reports memory usage we obtain, and compares it with Java heap dimension that server presents when serving no requests at all. In this case, too, despite higher memory peaks, memory occupation continues to take advantage of the efficient reuse of system resources that workflow organization permits.

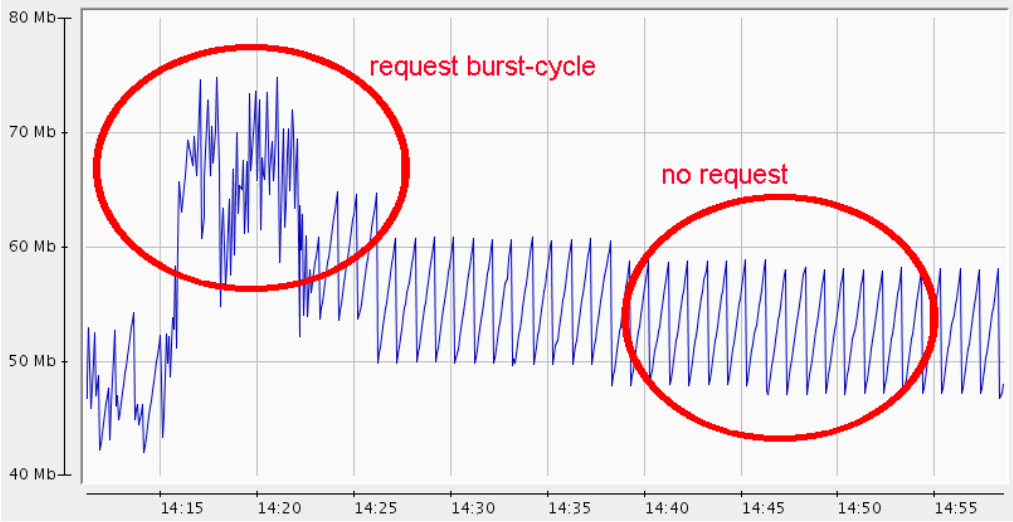

Illustration 47: Memory usage in case of request burst-cycles, and in case of no request

Conclusions

Ubiquitous Internet presents incredible opportunities to provide innovative and value-added services to users, by leveraging and reusing existing resources as well as by developing brand new functionalities.

Nowadays, indeed, people connect via traditional PCs as well as PDAs, smartphones, network-enabled multimedia players, or even digital TV appliances, and they do so through both wired and wireless network infrastructures, 3G operators, Bluetooth data link, and lots of other connection types. They ask for moving across different networks and staying connect through different terminals in a seamless way, while keeping their communication session consistent. As well, services and contents should base on user-specific information, and adapt to her preferences and physical or computational environment. Furthermore, interfaces and interaction paradigms should tailor to device support and characteristics, to consistently enable Web browser access, rather than service exploitation via SMS, VoIP phone calls, and more. Finally, it should be possible to reuse existing resources in simple and effective ways to build new applications, or to dynamically reconfigure current ones, according to runtime characteristics.

As a matter of fact, research achievements tend to evolve separately and often lead to *ad hoc* solutions and too vertical approaches, that focus only on specific application domains. Solutions exist, for instance, that either enable content adaptation but still miss multimodal interface capability, or that permit context-awareness while lacking effective session management. In the field of Business-to-Business and Business-to-Consumer integration, most promising solutions adopt service oriented architectures; by promoting modularity and reuse of software components, indeed, service abstraction can highly empower the creation of complex and value-added applications. Nevertheless, configuration of such applications usually demands explicit human-intervention, hence fall short of potentials for dynamic reconfiguration.

As the result of in-depth analysis of state-of-the-art proposals, formulation of theoretical design principles, and experimental verification of their viability, this thesis work has described an innovative approach to comprehensively deal with Ubiquitous Internet challenges. This dissertation has argued that an effective solution to support Ubiquitous Internet scenarios must follow a middleware approach, decouple final users and services to exploit and uniformly treat them in the form of resources to integrate. At the same time, it must push any kind of content-related logic outside its core layer, by keeping only management and coordination responsibilities. That succeeds in making the middleware design clearer and neater, and in enforcing its adoption to support actual scenarios.

On the one hand, client devices must be able to access heterogeneous services and contents without worrying about how to suitably request them, and service developers must concentrate only on improving service business logic, disregarding how users will actually exploit it to fit their own requirements. On the other hand, it is possible to simplify the design of Ubiquitous Internet middleware by assigning to external and pluggable resources all the facilities that relate to content processing and transforming, while introducing workflow entities to effectively compose and orchestrate them.

Our proposed model enables the integration of distributed resources via proxy entities that can abstract their heterogeneity; besides, model can be easily extended in terms of supported interaction paradigms and composition schemata, to cope with novel scenarios and ubiquity support aspects. Encouraging results and middleware employment in several actual use cases have proven to demonstrate viability of our approach.

Future research work will consider adoption of alternative industry standards and frameworks for the implementation of current middleware parts, and will deeply investigate the opportunity to distribute middleware components to heterogeneous and unclustered network nodes, including client devices, to promote migration and replication of system parts and resource

proxies on such nodes, for the sake of performance, scalability and fault-tolerance.

With a more detailed focus, we are considering enforcing Enterprise Service Bus (ESB) support for middleware components and resource proxies. ESB technology represents nowadays the *de facto* standard for orchestrating business-to-business applications out of distributed services. We strongly believe that ESB resources available through the Internet can really bring to Ubiquitous Internet applications a higher level of functionalities and opportunities: from e-commerce, to cross-enterprise Business-to-Consumer processes, and more.

Furthermore, we share interest in the Open Service Gateway initiative (OSGi), to leverage the increasing computational capability of client devices, both to develop new and more powerful resource proxies, and to enforce the execution of middleware parts directly on the client side. OSGi is today a leading framework for modular software development and deployment; it enables over-the-air download of software components, and leverages characteristics of the current execution environment to select most suitable service implementations. Manufacturers from mobile phone companies to automotive ones participate in the OSGi project, with the aim to spread adoption of framework-enabled devices and foster pervasive computing application scenarios.

Finally, we intend to exploit distribution and replication of both middleware components and resource proxy instances to promote communication efficiency, load balancing and fault tolerance. In details, we want to leverage the opportunity of migrating proxy configurations to different network locations, in order to enforce proximity policies while executing corresponding business processes, to improve system scalability by suitably distributing proxy configurations themselves, and to permit failover by re-enabling configurations of broken proxies on different nodes.

Acknowledgments

I would like to express my gratitude to all those who gave me the possibility to complete this thesis.

I want to thank my advisors, Maurelio Boari and Antonio Corradi for their patience, moral support, and trust.

I'm grateful to those who have encouraged me to go ahead with my PhD, and to those who have not, too.

I am deeply indebted to my present and former colleagues at the SWIMM project and, later on, at the "via Nosadella" laboratory.
I wish to thank them all for their help, support, suggestions, confidence, interest and valuable hints.

Finally, I would like to give my special thanks to my wife, my parents, and all my family members, whose patient love - especially patient - enabled me to complete this work.

March 17th, 2009.

Publications

M. Boari, E. Lodolo, S. Monti, S. Pasini, "Middleware for Automatic Dynamic Reconfiguration of Context-Driven Services", 11th Symposium on Computers and Communications (ISCC'06), IEEE , Pula, Italy, June 2006.

M. Boari, E. Lodolo, S. Monti, S. Pasini , "Progetto SWIMM (Servizi Web Interattivi e Multimodali per la Mobilità). Risultati ed applicazioni", AICA national conference, Milano-Mantova , Italy, September-October 2007.

A. Corradi, E. Lodolo, S. Monti, S. Pasini, "User-Centric Emergency Management: a Disappearing Middleware Approach", Wireless Rural and Emergency Communications Conference (WRECOM'07), Rome, Italy, October 2007.

M. Boari, E. Lodolo, S. Monti, S. Pasini, "Middleware for Automatic Dynamic Reconfiguration of Context-Driven Services", Microprocessors And Microsystems Journal, Issue 32, pages 145-148, Elsevier, November 2007.

M. Boari, A. Corradi, E. Lodolo, S. Monti, S. Pasini, "Coordination for the Internet of Services: a user-centric approach", 3rd International Conference on Communication System Software and Middleware (COMSWARE'08), Bangalore, India, January 2008.

A. Corradi, E. Lodolo, S. Monti, S. Pasini, "A user-centric composition model for the Internet of Services", 13th Symposium on Computers and Communications (ISCC'08), IEEE, Marrakesh, Morocco, July 2008.

A. Corradi, A. Landini, E. Lodolo, S. Monti, S. Pasini , "Integrating Service Composition with Mobile Code Technologies", 2nd International Workshop on Distributed Agent-based Retrieval Tools (DART'08), Cagliari, Italy , September 2008.

S. Monti, S. Pasini, A. Corradi, E. Lodolo, M. Boari, “An eXtensible middleware for Multichannel, Multimodal, and Multipattern services (X3M)”, 5th International Workshop on Next Generation Networking Middleware (NGNM'08), Samos Island, Greece, September 2008.

A. Corradi, F. Di Marco, S. Monti, S. Pasini, “Facing Crosscutting Concerns in a Middleware for Pervasive Service Composition”, accepted for publication, 14th Symposium on Computers and Communications (ISCC'09), IEEE, Port El Kantaoui, Tunisia, 2009.

A. Corradi, E. Lodolo, S. Monti, S. Pasini, “Dynamic reconfiguration of middleware for Ubiquitous Computing”, submitted to the 3rd Workshop on Adaptive and DependAble Mobile Ubiquitous Systems (ADAMUS'09), London, England, 2009.

Bibliography

[Aal03] W. M. P. Van Der Aalst, A. H. M. Ter Hofstede, B. Kiepuszewski, A. P. Barros , "Workflow Patterns", Distributed and Parallel Databases, Volume 14 Issue 1, Pages 5-51, Springer Netherlands, July 2003. DOI 10.1023/A:1022883727209.

[Abr96] Abramowski, S.; Klabunde, K.; Konrads, U.; Newnest, K.; Tjabben, H.; "Multimedia session management", Intelligent Network Workshop, 1996. IN '96., IEEE 21-24 April 1996. Digital Object Identifier 10.1109/INW.1996.539694.

[Agr07a] A. Agrawal, M. Amend, M. Das, M. Ford, C. Keller, M. Kloppmann, D. König, F. Leymann, R. Müller, G. Pfau, K. Plösser, R. Rangaswamy, A. Rickayzen, M. Rowley, P. Schmidt, I. Trickovic, A. Yiu, and M Zeller, "WS-BPEL Extension for People (BPEL4People)", version 1.0, 2007.

[Agr07b] A. Agrawal, M. Amend, M. Das, M. Ford, C. Keller, M. Kloppmann, D. König, F. Leymann, R. Müller, G. Pfau, K. Plösser, R. Rangaswamy, A. Rickayzen, M. Rowley, P. Schmidt, I. Trickovic, A. Yiu, and M Zeller, "Web Services Human Task (WS- HumanTask)", version 1.0, 2007.

[Aka98] M. Akay, I. Marsic, A. Medl, G. Bu, "A System for Medical Consultation and Education Using Multimodal Human/Machine Communication", IEEE Transactions on Information Technology in Biomedicine, Vol. 2, No. 4, December 1998.

[Alo03a] G. Alonso, F. Casati, H. Kuno, V. Machiraju, "Enterprise application integration", in: "Web Services: Concepts, Architectures and Applications", Springer-Verlag, 2003, ISBN 354044008.

[Alo03b] G. Alonso, F. Casati, H. Kuno, V. Machiraju, "Service composition" in "Web Services: Concepts, Architectures and Applications", Springer-Verlag, June 2003.

[Ana04] A. Anagol-Subbarao, "J2EE Web Services on BEA WebLogic", Prentice-Hall, October 18, 2004. ISBN-10: 0-13-143072-6.

[Artix] IONA Technologies, "Using Artix and Service-Oriented Architecture for Multi-Channel Access", http://www.iona.com/devcenter/ artix/articles/0304soa.pdf, February 2008.

[BeanShell] "BeanShell – Lightweight scripting for Java", available from http://www.beanshell.org/.

[Bel03] Bellavista, P.; Corradi, A.; Montanari, R.; Stefanelli, C.; "Dynamic binding in mobile applications", Internet Computing, IEEE Volume 7, Issue 2, March-April 2003 Page(s):34 – 42. Digital Object Identifier 10.1109/MIC.2003.1189187.

[Ber01] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web", Scientific Am., vol. 284, no. 5, pp. 34-43, May 2001.

[Ber04] Girma Berhe, Lionel Brunie, Jean-Marc Pierson , "Modeling service-based multimedia content adaptation in pervasive computing ", Proceedings of the 1st conference on Computing frontiers , CF '04. ACM, April 2004.

[Ber96] P. A. Bernstein, "Middleware: a model for distributed system services ", Communications of the ACM, Volume 39 Issue 2, February 1996.

[Bis06] R. Biswas, "The Java Persistence API - A Simpler Programming Model for Entity Persistence", available from http://java.sun.com/developer/technicalArticles/J2EE/jpa/, May 2006, © 2009 Sun Microsystems, Inc.

[Boa07] M. Boari et al., "Middleware for automatic dynamic reconfiguration of context-driven services", Microprocessors and Microsystems, No. 32, Pages 145-160, September 2007.

[Camel] "Camel Manual 2.0", Apache, 2008. Available from http://activemq.apache.org/camel/manual/camel-manual-2.0-SNAPSHOT.pdf.

[Cha06] A.T.S. Chan, Y. Zheng, "Coordinated composition of services for adaptive mobile middleware", "Proceedings of the 11th IEEE Symposium on Computers and Communications", Page(s): 789 – 794, 26-29 June 2006. Digital Object Identifier 10.1109/ISCC.2006.55.

[Cha95] M. Chapman, S. Montesi, ''Overall Concepts and Principles of TINA – Version 1.0'', Telecommunications Information Networking Architecture Consortium, 17 February 1995. Available from: http://www.tinac.com/specifications/documents/overall.pdf.

[Che06] Chen, I.Y.L.; Stephen J.H. Yang; Yang, S.J.H.; Ubiquitous Provision of Context Aware Web Services, Services Computing, 2006. SCC '06. IEEE International Conference on, Sept. 2006 Page(s):60 - 68. Digital Object Identifier 10.1109/SCC.2006.110.

[Chr02] Christopher Lueg, "Representations in Pervasive Computing", Proceedings of the Inaugural Asia Pacific Forum on Pervasive computing, 2002.

[Coherence] "Oracle Coherence", http://www.oracle.com/technology/products/coherence/index.html.

[Com04] M. Comerio, F. De Paoli, S. Grega, C. Batini, C. Di Francesco, A. Di Pasquale, “A service re-design methodology for multi-channel adaptation”, Proceedings of the 2nd International Conference on Service Oriented Computing, ICSOC 2004, November 2004.

[Cur03] F. Curbera, S. Weerawarana, et al., “Business Process Execution Language for Web Services Version 1.1”. Specification, BEA Systems, IBM Corp., Microsoft Corp., SAP AG, Siebel Systems, 2003.

[dotNet] “Microsoft .NET Framework”, http://www.microsoft.com/net/, © 2008 Microsoft.

[EJB] “Enterprise JavaBeans Technology”, http://java.sun.com/products/ejb/, Copyright 1994-2009 Sun Microsystems, Inc.

[Eri02] Thomas Erickson, Some problems with the notion of context-aware computing, Communications of the ACM, Volume 45, Issue 2, February 2002. Pages: 102 - 104. ISSN:0001-0782.

[Festival] “The Festival Speech Synthesis System”, http://www.cstr.ed.ac.uk/projects/festival/, Centre for Speech Technology Research, Edinburgh, © The University of Edinburgh.

[Firefox] “Firefox web browser: Faster, more secure, & customizable”, http://www.mozilla.com/en-US/firefox/, 2009 © Mozilla Foundation.

[Fuj06] K. Fujii and T. Suda, "Semantics-based Dynamic Web Service Composition," the International Journal of Cooperative Information Systems (IJCIS), special issue on Service-Oriented Computing, Vol. 15, No. 3, pp. 293-324, Sep. 2006.

[Gam94] E. Gamma, R. Helm, R. Johnson, J. M. Vlissides, “Structural Patterns” in “Design Patterns: Elements of Reusable Object-Oriented Software”, Addison-Wesley, 1994. ISBN: 0-201-63361-2.

[Gar05] J. J. Garrett, “Ajax: A New Approach to Web Applications”, http://www.adaptivepath.com/ideas/essays/archives/000385.php, Adaptive Path, February 18, 2005.

[GMaps] “Google Maps”, http://maps.google.com, © 2009 Google.

[Gor05] R. Gorrieri, C Guidi, R. Lucchi, "Reasoning About Interaction Patterns in Choreography", from "Formal Techniques for Computer Systems and Business Processes", Pages 333-348. Springer Berlin / Heidelberg, 2005. DOI: 10.1007/11549970, ISBN: 978-3-540-28701-8.

[Gre01] Saul Greenberg , “Context as a Dynamic Construct ”, Human-Computer Interaction, Volume 16, Issue 2 - 4 February 2001 , pages 257 – 268 . DOI: 10.1207/S15327051HCI16234_09.

[GReader] “Google Reader”, http://www.google.com/reader. © 2008 Google.

[Haa97] Oliver Haase , “Session maintenance ” from “The Handbook of Mobile Middleware ”, P. Bellavista, A. Corradi, CRC Press, 2007 . ISBN 0849338336, 9780849338335.

[Hen02] Henricksen, K., Indulska, J., and Rakotonirainy, A. (2002). Modeling context information in pervasive computing systems. In Mattern, F. and Naghshineh, M. (eds) Proceedings of the International Conference on Pervasive Computing 26–28 August 2002, Zurich, Switzerland. Springer LNCS 2414, pp. 167–180.

[Her04] I. Herman, R. Swick, D. Brickley, “Resource Description Framework (RDF)”, W3C Semantic Web Activity, 2004. Available from: http:// www.w3.org/RDF/.

[Hibernate] “hibernate.org – Hiberante”, http://www.hibernate.org/, © Copyright 2006, Red Hat Middleware, LLC.

[Hog07] R. Högg et al., “Overview of Business Models for Web 2.0 Communities”, Proc. Gemeinschaften in Neuen Medien, Technische Universität Dresden, 2006, pp. 23-37.

[Hol08] T. Holmes, H. Tran, U. Zdun, S. Dustdar, “Modeling Human Aspects of Business Processes - A View-Based, Model-Driven Approach”, European Conference on Model Driven Architecture, Foundations and Applications (ECMDA-FA), 2008. Springer Berlin / Heidelberg . Volume 5095/2008 . from book “Model Driven Architecture – Foundations and Applications ”. DOI 10.1007/978-3-540-69100-6 , ISBN 978-3-540-69095-5 , DOI 10.1007/978-3-540-69100-6_17 , Pages 246-261.

[Hon01] Hong. J. I., Landay J. A., "An infrastructure approach to context-aware computing", Human-Computer Interaction, Volume 16, Issue 2 - 4 February 2001 , pages 287 - 303.

[IBM] International Business Machines Corporation, “Why IBM? – Leadership in Multimodal”, http://www-306.ibm.com/software/pervasive/multimodal/, 2006.

[iGoogle] “iGoogle”, http://www.google.com/ig, © 2009 Google.

[ImageMagick] “ImageMagick: Convert, Edit, and Compose Images”, http://www.imagemagick.org/script/index.php, © 1999-2009 ImageMagick Studio LLC.

[Int02] Intille, S.S., “Designing a home of the future”, Pervasive Computing, IEEE, Volume 1, Issue 2, April-June 2002, Page(s): 76 - 82. Digital Object Identifier 10.1109/MPRV.2002.1012340.

[iTunes] “Apple – iTunes”, http://www.apple.com/it/itunes/overview/, Copyright © 2009 Apple Inc.

[JAAS] "Java Authentication and Authorization Service (JAAS) Reference Guide", available from http://java.sun.com/javase/6/docs/technotes/guides/security/jaas/JAASRefGuide.html.

[JBoss] "Community driven open source middleware", available from http://www.jboss.org/, 2009.

[jBPM] "JBoss jBPM", available from http://www.jboss.com/pdf/jb_jbpm_04_07.pdf, © 2008 Red Hat Middleware, LLC.

[JEE] "Sun Java EE 5 - Overview", available from http://java.sun.com/javaee/technologies/javaee5.jsp, © 2009, Sun Microsystems, Inc.

[Jef08] C. Jefferies, P. Brereton, M. Turner, "A Systematic Literature Review of Approaches to Reengineering for Multi-Channel Access", 12th European Conference on Software Maintenance and Reengineering, pp. 258-262, April 2008.

[JMS] "Java Message Service (JMS)", available from http://java.sun.com/products/jms/, © 2009 Sun Microsystems, Inc.

[JMX] "Java Management Extensions (JMX) Technology", available from http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/, Copyright 1994-2009 Sun Microsystems, Inc.

[JNDI] "Java Naming and Directory Interface (JNDI)", available from http://java.sun.com/products/jndi/, © 2009 Sun Microsystems, Inc.

[Joh02] Johanson, B.; Fox, A.; Winograd, T., "The Interactive Workspaces project: experiences with ubiquitous computing rooms ", Pervasive Computing, IEEE , Volume 1, Issue 2, April-June 2002, Page(s):67 – 74 . Digital Object Identifier 10.1109/MPRV.2002.1012339.

[Joh03] Rod Johnson, "Design techniques and coding standards for J2EE projects", from "Expert One-on-One J2EE Design and development", Wrox, 2003. Available from: http://www.theserverside.com/tt/articles/content/ RodJohnsonInterview/JohnsonChapter4.pdf.

[Joh05] Rod Johnson, "Introduction to the SpringFramework", available from http://www.theserverside.com/tt/articles/article.tss?l=SpringFramework, May 2005.

[JTA] "Java Transaction API (JTA)", available from http://java.sun.com/javaee/technologies/jta/index.jsp, © 2009 Sun Microsystems, Inc.

[Kal07] Kalasapur, S.; Kumar, M.; Shirazi, B.A.; "Dynamic Service Composition in Pervasive Computing", Parallel and Distributed Systems, IEEE Transactions on; Volume 18, Issue 7, July 2007 Page(s):907 – 918. Digital Object Identifier 10.1109/TPDS.2007.1039.

[Kapow] "Kapow Mashup Server Product Family", available from: http://www.kapowtech.com/products/products.aspx, © 2008 Kapow Technologies.

[Kay72] A. Kay, "A Personal Computer for Children of All Ages," Draft, Xerox Palo Alto Research Center, 1972.

[Kay77] A. Kay and A. Goldberg, "Personal Dynamic Media", IEEE Computer, vol. 10, no. 3, Mar. 1977, pp. 31-42.

[Kin08] G. King et al., "Seam - Contextual Components. A Framework for Enterprise Java", version 2.1.0.SP1, ed. S. Kittoli, © 2008, Red Hat Middleware LLC. Available at http://docs.jboss.com/seam/2.1.0.SP1/reference/en-US/pdf/seam_reference.pdf.

[Koe04] J. Koenig, "JBoss jBPM – White paper", 2004. Available from http://www.jboss.com/pdf/jbpm_whitepaper.pdf.

[Kri97] D. Kristol and L. Montuli, "HTTP State Management Mechanism," RFC 2109, Network Working Group, 1997.

[LAME] "LAME MP3 Encoder", http://lame.sourceforge.net/.

[Mak94] V. W. Mak, "Session management for distributed multimedia applications", 5th IEEE COMSOC International Workshop on Multimedia Communications, Pages: 6/2/1 – 6/2/5, May 1994. Digital Object Identifier 10.1109/IWMC.1994.601228.

[Mar04] D. Martin et al., "OWL-S: Semantic Markup for Web Services", W3C Member Submission, 22 November 2004. Available from: http://www.w3.org/Submission/OWL-S/.

[Mashup] "What is Google Mashup Editor?", http://code.google.com/gme/, © 2009 Google.

[Med03] Brahim Medjahed, Athman Bouguettaya, Ahmed K. Elmagarmid, "Composing Web services on the Semantic Web", The VLDB Journal — The International Journal on Very Large Data Bases , Volume 12 Issue 4 , November 2003. Springer-Verlag, New York, Inc.

[Med05] Medjahed, B.; Bouguettaya, A.; "A multilevel composability model for semantic Web services ", Knowledge and Data Engineering, IEEE Transactions on ; Volume 17, Issue 7, July 2005. Page(s):954 – 968 . Digital Object Identifier 10.1109/TKDE.2005.101.

[METAR] "METAR Data Access", http://weather.noaa.gov/weather/metar.shtml, 2007.

[Mil04] Milanovic, N.; Malek, M., "Current solutions for Web service composition ", Internet Computing, IEEE ,Volume 8, Issue 6, Nov.-Dec. 2004 Page(s):51 – 59 .Digital Object Identifier 10.1109/MIC.2004.58.

[Mmi] W3 Consortium, “W3C Multimodal Interaction Framework”, W3C Note, http://www.w3.org/TR/mmi-framework/, May 2003.

[MomentumA] “Message Exchange Pattern”, http://www.serviceoriented.org/message_exchange_pattern.html, © 2006 Momentum s.r.l.

[MomentumB] “Web Service Orchestration (WSO)”, © 2006 MomentumSI. Available from: http://www.serviceoriented.org/web_service_orchestration.html.

[Mor01] Thomas P. Moran; Paul Dourish ; “Introduction to This Special Issue on Context-Aware Computing”, Journal of Human-Computer Interaction, Volume 16, Issue 2 - 4 February 2001 , pages 87 – 95 . DOI: 10.1207/S15327051HCI16234_01.

[Mule] MuleSource Inc., “Mule 2.x Getting Started Guide”, 2008. Available from: http://mule.mulesource.org/display/MULE2INTRO/Home.

[Musicovery] “Musicovery – The first intuitive webradio”, Press Kit, available from http://musicovery.com/pressRelease/PressKitMUSICOVERY.doc, © 2007 musicovery.com.

[MySQL] “MySQL - The world's most popular open source database”, http://www.mysql.com/, © 1995-2008 MySQL AB, 2008-2009 Sun Microsystems, Inc.

[Nar07] N. C. Narendra, Bart Orriens , “Modeling web service composition and execution via a requirements-driven approach ”, SAC '07: Proceedings of the 2007 ACM symposium on Applied computing , March 2007, ACM.

[New05] E. Newcomer, G. Lomow, “Understanding SOA with Web Services”, Addison Wesley. ISBN 0-321-18086-0, 2005.

[Obr04] Z. Obrenovic, D. Starcevic, “Modeling Multimodal Human-Computer Interaction”, IEEE Computer, Vol. 37 , No. 9, September 2004.

[Opera] Opera Software ASA, “Opera Multimodal Browser”, http://www.opera.com/products/verticals/multimodal/index.dml, 2001.

[Ort05] E. Ort, “Service-Oriented Architecture and Web Services: Concepts, Technologies and Tools”, Sun Microsystems Inc., April 2005. Available from: http://java.sun.com/developer/technicalArticles/WebServices/soa2/.

[OSGi] “OSGi Alliance - Main / OSGi Alliance”, http://www.osgi.org/Main/HomePage, Copyright © 2009 OSGi™ Alliance.

[Ovi99] S. L. Oviatt, “Ten myths of Multimodal Interaction”, Communications of the ACM, pp. 74-81, November 1999.

[OWL] W3 Consortium, “OWL Web Ontology Language Overview”, W3C Recommendation. Available from: http://www.w3.org/TR/owl-features/.

[Pan04] Ai-Chun Pang; Jyh-Cheng Chen; Yuan-Kai Chen; Agrawal, P.; "Mobility and session management: UMTS vs. cdma2000", IEEE Wireless Communications, IEEE, Volume 11, Issue 4, Aug. 2004 Page(s): 30 – 43. Digital Object Identifier 10.1109/MWC.2004.1325889.

[Pap03] M. P. Papazoglou and D. Georgakopoulos, “Service Oriented Computing”, Comm. ACM, vol. 46, no. 10, 2003, pp. 25–28.

[Pas08] L. Passani, “WURFL”, http://wurfl.sourceforge.net/, © 2008 Luca Passani.

[Pat05] S. Patil, “What is a Portlet”, O'Reilly, http://www.onjava.com/pub/a/onjava/2005/09/14/what-is-a-portlet.html, September 14, 2005.

[Pipes] “Yahoo Pipes”, http://pipes.yahoo.com/pipes/, © 2008, Yahoo! Inc.

[Podcast] “Podcast”, availabale from http://en.wikipedia.org/wiki/Podcast, 14 March 2009.

[Rad09] T. Rademakers, J. Dirksen, “Implementing a process engine in the ESB”, from “Open source ESBs in action”, Manning, 2009.

[Rav03] T. V. Raman, G. McCobb, R. A. Hosn, “Versatile Multimodal Solutions. The Anatomy of User Interaction”, XML Journal, Vol. 4, No. 2, Apr. 2003.

[Rei05] T. O’Reilly, “What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software”, 2005, available from: www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html.

[RMI] “Remote Method Invocation Home”, available from http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp, Copyright 1994-2009 Sun Microsystems, Inc.

[Roh97] Roh, J.R.; Kook, K.M.; "A switchable session management for the distributed multimedia-on-demand system", Protocols for Multimedia Systems - Multimedia Networking, 1997. Proceedings., IEEE Conference on, 24-27 Nov. 1997 Page(s):102 - 111. Digital Object Identifier 10.1109/PRMNET.1997.638886.

[Rom05] D. Roman et al., “Web Service Modeling Ontology ”, “Applied Ontology ”, Volume 1, Number 1/2005 , Pages 77-106, IOS Press . ISSN: 1570-5838.

[Rus08] N. Russell, W.M.P. van der Aalst, “Evaluation of the BPEL4People and WS-HumanTask Extensions to WS-BPEL 2.0 using the Workflow Resource Patterns”, Technical report, Queensland University of Technology, Brisbane, 2008.

[Sah03] Saha, D.; Mukherjee, A., “Pervasive computing: a paradigm for the 21st century ”, IEEE Computer , Volume 36, Issue 3, March 2003, Page(s):25 – 31 . DOI 10.1109/MC.2003.1185214.

[Sal99] D. Salber, A.K. Dey and G.D. Abowd, “The Context Toolkit: Aiding the development of context-enabled applications”, in Proceedings of the SIGCHI conference on Human factors in computing systems, Pittsburgh, Pennsylvania, United States . Pages: 434 - 441 , 1999. ISBN:0-201-48559-1.

[Sat01] M. Satyanarayanan, “Pervasive Computing: Vision and Challenges”, IEEE Personal Computing, Aug. 2001, pp. 10–17.

[Sch02] H. Schulzrinne et al., "SIP: session initiation protocol", IETF RFC 3261, Available from: http://www.ietf.org/rfc/rfc3261.txt, June 2002.

[Sch07a] C. Schroth, O. Christ, “Brave New Web: Emerging Design Principles and Technologies as Enablers of a Global SOA”, in Proceedings of the 2007 IEEE International Conference on Services Computing (SCC 2007), Salt Lake City, USA, 2007.

[Sch07b] Schroth, Christoph, "The internet of services: Global industrialization of information intensive services," Digital Information Management, 2007. ICDIM '07. 2nd International Conference on , vol.2, no., pp.635-642, 28-31 Oct. 2007.

[Sch07c] Schroth, C.; Janner, T., “Web 2.0 and SOA: Converging Concepts Enabling the Internet of Services” , IT Professional, Volume 9, Issue 3, May-June 2007, Page(s):36 – 41. Digital Object Identifier 10.1109/MITP.2007.60.

[ServiceMix] The Apache Software Foundation, “Apache ServiceMix 3.x Users' Guide”, February 2008. Available from: http://servicemix.apache.org/users-guide.html.

[Sha03] R. Sharma, M. Yeasin, N. Krahnstoever, I. Rauschert, G. Cai, I. Brewer, A.M. Maceachren, K. Sengupta, “Speech–Gesture Driven Multimodal Interfaces for Crisis Management”, Proceedings of the IEEE, Vol. 91, No. 9, September 2003.

[Shi07] T.K. Shih, T. Wang, C. Chang, T. Kao, D. Hamilton, “Ubiquitous e-Learning With Multimodal Multimedia Devices”, IEEE Transactions on Multimedia, Vol. 9, No. 3, April 2007.

[SOA] “Reference Model for Service Oriented Architecture 1.0”, Committee Specification 1, available from: http://www.oasis-open.org/ committees/download.php/19679/soa-rm-cs.pdf, August 2nd 2006.

[Songbird] "Songbird – Open source music player", http:// www.getsongbird.com/, © 2005-2009 Pioneers of the Inevitable.

[Sva01] D. Svanaes, "Context-aware technology: a phenomenological perspective". Human-Computer Interaction, Volume 16, Issue 2 - 4 February 2001 , pages 379 – 400.

[Thunderbird] "Thunderbird - Reclaim your inbox", http://www.mozilla.com/thunderbird. 2008 © Mozilla Foundation.

[Tur00] M. Turk, G. Robertson, "Perceptual User Interfaces (Introduction)", Communications of the ACM, pp. 33-35, March 2000.

[Ueh01a] Uehara, S.; Mizuno, O.; Kikuno, T.; "Development of session management mechanism for cellular phone with WWW connection", Software Engineering Conference, 2001. APSEC 2001. Eighth Asia-Pacific, 4-7 Dec. 2001 Page(s):345 - 348.

[Ueh01b] Uehara, S.; Mizuno, O.; Kikuno, T.; "An implementation of electronic shopping cart on the Web system using component-object technology", Object-Oriented Real-Time Dependable Systems, 2001. Proceedings. Sixth International Workshop on, 8-10 Jan. 2001 Page(s):77 - 84. Digital Object Identifier 10.1109/WORDS.2001.945116.

[VoiceXML] VoiceXML Forum, "XHTML + Voice Profile 1.2", VoiceXML 2.0 Recommendation, http://www.voicexml.org/specs/multimodal/x+v/ 12/spec.html, Mar. 2004.

[WebSphere] IBM, "WebSphere Message Broker: Delivering business value through a universal enterprise service bus.", White paper, December 2007. Available from: ftp://ftp.software.ibm.com/software/ websphere/integration/wbimessagebroker/WSW14004-USEN-00_WMB_ESB_WP_1206.pdf.

[Wei02] M. Weiser, "The Computer for the 21st Century", Scientific American, Sept., 1991, pp. 94-104; reprinted in IEEE Pervasive Computing, Jan.-Mar. 2002, pp. 19-25.

[Wiki] "Wikipedia", http://www.wikipedia.org, Wikimedia Foundation, 2009.

[Win01] Terry Winograd, "Architectures for Context", Human-Computer Interaction, Volume 16, Issue 2 - 4 February 2001, pages 401 – 419. DOI: 10.1207/S15327051HCI16234_18.

[Woo06] Bobby Woolf, "ESB and Workflow", Weblog, 31 March 2006. Available from: http://www.ibm.com/developerworks/blogs/ page/woolf?entry=esb_and_workflow.

[Zim05] O. Zimmermann, V. Doubrovski, J. Grundler, K. Hogg, “Service-oriented architecture and business process choreography in an order management scenario: rationale, concepts, lessons learned”, Companion To the 20th Annual ACM SIGPLAN Conference on Object- Oriented Programming, Systems, Languages, and Applications, pp. 301-312, ACM Press, October 2005.