Lehrstuhl für Rechnertechnik und Rechnerorganisation /
Parallelrechnerarchitektur der Technischen Universität
München

A Framework for Autonomic Service-based Resource Management Using a Semantic Approach

Houssam Haitof

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr. Thomas Huckle
Prüfer der Dissertation:
1. Univ.-Prof. Dr. Hans Michael Gerndt
2. Univ.-Prof. Dr. Martin Bichler

Die Dissertation wurde am 22.08.2011 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 01.11.2011 angenommen.

Abstract

The management cost of the IT infrastructure can be very high, affecting considerably the IT infrastructure return of investment and rising dramatically the outlay for IT services. Most of this cost can be related to the high complexity of managing a heterogeneous pool of IT resources. In addition, the high dependability of IT management on the human factor makes it more expensive and error-prone. The solution lies in automating complex management tasks using intelligent systems aware of their environment and IT organization management objectives. These intelligent systems would be sophisticated versions of the management applications that already exist. However, the existing management applications are tightly coupled to the resources they are supposed to manage and are quite intolerant to extensions to support other resources, mainly for the use of proprietary protocols and closed standards. Thus, many different management applications need to co-exist, and sharing management data is for the most part impractical for interoperability constrains. Integration of those management applications can be a difficult task involving the development of a number of interfaces and/or wrappers. We do not only need to enhance the existing IT management systems but we should provide a new organizational model for IT resources and IT management that would be "automation-friendly" with the added constraint of being backward compatible with existing IT infrastructure and as non-intrusive as possible.

Our solution is composed of two parts: the Managed Resource Framework (MRF), a framework to model classes and hierarchies of IT physical and logical resources and to generate on demand WSDM-based service representations for rapid composition and deployment, and the Semantic Resource Management Infrastructure (SRMI) composed of structural elements to support the managed resources as well as Specific-Purpose Inference Engines (SPIE) that manage the resources in an autonomic manner using management policies.

MRF is a plug-in based framework that allows to create resource mod-

els and instantiate service representation of those resources. It allows to automatically generate ready-to-deploy service representations of resources from their semantic representations. The objective is to have a computer aided process by which resources can be rapidly instantiated, deployed and managed in a relatively quick and transparent manner for the user. It can generate an archive that is backward compatible with traditional service containers, however, when used in conjunction with a MR Container, software agents can then have access, among other things, to the resource semantic representation, service interface, management interface, client library, subscription-based notification and so on.

Automated management is done through the use of management models that describe a behavioral logic using rules. This logic is then incorporated into reasoners that have access to the resource models and the resource instances. The reasoners infer course of action and initiate the appropriate activities in accordance to the management models. The management models can be independent, inter-related or sometimes contradictory. Mechanism are implemented to manage the interaction between the different management models. Our system presents different levels or scopes where automation is performed. This automation is distributed over a number of specialized Automation Engines. Every automation engine relies on intelligent systems that take care of performing the automation logic.

The ideas presented and discussed in this work are materialized in SEROM (SERvice Oriented Management), a Java implementation of the MRF and the SRMI. We used two real use cases to test our approach, namely FinGrid, the German Financial Grid, and IBM's E2E Automation and Green IT.

Zusammenfassung

In dieser Arbeit präsentieren wir eine Architektur und ein Framework für eine automatische serviceorientierte Ressource-Managementlösung, die einen semantikbasierten Ansatz nutzt. Basierend auf den zwei Prinzipien der Vereinfachung und Automatisierung, ist das Ziel mit Hilfe einer einfachen und praktikablen Lösung die Komplexität zu senken. Unser Lösungsvorschlag besteht aus zwei Teilen, dem MRF und SRMI. Das MRF ist ein Plug-In basiertes Framework, das der Modellierung von Klassen und Hierarchien von physikalischen und logischen IT-Ressourcen dient und zur Generierung von On-Demand Service Representation für eine schnelle Zusammenstellung und Bereitstellung von Ressourcen. SRMI besteht aus strukturellen Elementen, um die zu verwaltenden Ressourcen darzustellen, und stützt sich auf domain-spezifische Inference-Engines, die die Ressourcen anhand eines Management-Modells automatisch verwalten.

Acknowledgments

I would like hereby to thank my supervisor Prof. Dr. Michael Gerndt that gave me the opportunity to work on this research topic under his guidance, I would like to express my gratitude for his trust and encouragement and for the freedom he gave me to pursue my research.

I would also like to thank my referee, Prof. Dr. Martin Bichler, for his time reviewing this thesis and for the opportunity he gave me to have fruitful discussions about my work with his research group.

Further thanks to Prof. Arndt Bode, the leader of our chair, for his support and for being available whenever I needed him.

This work has been supported by an IBM PhD fellowship and by the IBM Center of Advanced Studies with a research grant for Policy-Based Autonomic Service Management. In this regard, I would like to thank the IBM Böblingen Development Laboratory and especially Hans-Dieter Wehle, who invited me as a guest researcher to the laboratory, for his considerable support and the opportunities he gave me to work on exciting projects in Böblingen related to Grid Computing and Green IT. I would also like to thank Martin Reitz and Dominik Jall from the IBM Böblingen Lab. for their support and the valuable discussions we had on Semantic Technologies and Inference Engines internals.

Further thanks go to my colleagues and the administrative staff at LRR-TUM. I would like to specially thank Dr. Tianchao Li, Dr. Karl Fürlinger and Dr. Edmond Kereku with whom I had the privilege to collaborate on projects. I would also like to mention Dr. Josef Weidendorfer, Dr. Michael Ott, Alin Murarașu and Marcel Meyer for the interesting discussions and support and of course Andreas Hollmann, Anca Berariu, Haowei Huang, Ventsislav Petkov, and Yury Oleynik.

I would also like to thank my friends for their support and the great moments we spent together. They are too many to mention them all here, however I would like to specially mention Dr. Hachim Haddouti, Abdelali Zahi and Fayssal Elmoufatich

Last but not least, warm thanks go to my family for their continuous

support, understanding, encouragement and unconditional love and especially to my wife Amal, the best teammate I ever had.

Houssam Haitof
Munich, Germany
November 2011

Contents

List of Figures

Chapter 1

Introduction

1.1 Overview

IT management complexity is a scourge that every institution has to cope with. In this work we present an architecture and a framework for autonomic service-based resource management using a semantic approach. The objective is to lower complexity using a simple to use and practical solution based on two principles: simplification and automation. The ideas in this work are related to several technologies and research areas that we will briefly present near the end of this introduction.

1.2 IT complexity

The major issue facing IT in companies nowadays is complexity. It spans the IT infrastructure, applications and services as well as the business processes related or based on IT and can hinder to great extent the ability of the company to react to change and being competitive. Higher complexity means higher costs of maintenance and more down time, leading to inefficient use of the IT resources. This can lead to unexpected, or at best, unpredictable system behavior making it near-impossible to offer guarantees based on the IT infrastructure or honoring service level agreements that involves IT. Needless to say that companies with better management of their IT complexity have a significant competitive edge over companies without. Let it be with regards to service guarantees, rapid reaction times, lower management cost or system stability in general. Moreover complex IT leads to more erroneous decisions and the high dependability of IT management on the human factor makes it more expensive and error-prone, add to that the more IT is heterogeneous, the more there is a

need for IT personnel with different and specialized, sometimes obscure IT skills. Last but not least, there is a higher security risk with a more complex or poorly managed IT as many parts of the global architecture would be badly designed, incoherent and susceptible to malicious attacks.

In fact, in a recent study[1], IT complexity was identified as the most significant barrier to business growth. The same study shows that the cost reductions gained from applications streamlining were higher than IT outsourcing or offshoring. Managing complexity then, not only beneficial to the companies but also to the national economies as an optimized IT has a better return on investment than outsourcing!

1.2.1 Origin of IT complexity

IT complexity is rising mainly due to three issues, and these are: the system scalability, the rapid rate of change, and the resources diversity.

1.2.1.1 System scalability

IT systems are always growing, be it for the continuous need to acquire new software or hardware or the growth due to acquisitions or mergers that bring together different companies with different IT systems, management applications and often management cultures. Instead of two poorly managed IT systems, we end up with a bigger mess with more patch work than ever. In such scenarios, the objective of the IT department is to align the business processes of the former companies and little is done towards real integration or alignment of IT management.

1.2.1.2 Rapid rate of change

Although Moore's law only considers the high rate of change in the performance of CPUs, the global rate of change of IT innovation in hardware architectures, software, protocols or communication is simply tremendous. Companies need to invest a lot of effort into adapting and keeping up with the rate of change. This change is driven by the client requests, the competitors pressures or the need to adapt with the other business partners. This never-ending adaptation process generates a considerable amount of coexisting new and legacy systems that are sometimes conflicting and that the IT department is required to support it all, simply because not all of the companies partners have the same rate of IT change.

[1]Source: Findings in A.T. Kearney IT Innovation and Effectiveness Study, 2009

1.2.1.3 Resource diversity

The deployment of different and heterogeneous platforms and solutions engenders a complex pool of IT resources with different management applications, management protocols and even management philosophies. This is mainly due to the fact that IT, while supporting all the business processes, is pushed to adhere to some specific technologies quickly, bypassing the IT department guidelines on systems adoption, if there are any. Add to that the fact that IT became a commodity and that every employee may have their own smart phone or preferred IT environment that the IT department has to integrate in some cases to the company infrastructure.

1.2.2 Lowering the IT complexity

IT complexity cannot be eliminated, but if ignored, it would only grow. The IT complexity needs to be managed to control it under acceptable terms. There are two vectors to manage or even considerably lower IT complexity: managerial and technical. For the managerial aspect, it is important to adopt an IT governance structure for the IT department and to follow IT management best practices such as the Information Technology Infrastructure Library (ITIL) [56, 65, 63, 19, 94] or the Control Objectives for Information and related Technology (COBIT) framework [55]. For the technical aspect, many approaches can be taken to lower the IT complexity with different results such as virtualization, inter-operability tools, process management tools, automation tools, simplification or consolidation and standardization.

1.2.3 Our proposed solution

It is possible to automate complex management tasks using intelligent systems aware of their environment and IT organization management objectives. These intelligent systems would be sophisticated versions of the management applications that already exist. However, the existing management applications are tightly coupled to the resources they are supposed to manage and are quite intolerant to extensions to support other resources, mainly for the use of proprietary protocols and closed standards. Thus, many different management applications need to co-exist, and sharing management data is for the most part impractical for inter-operability constrains. Integration of those management applications can be a difficult task involving the development of a number of interfaces

and/or wrappers. We do not only need to enhance the existing IT management systems but we should provide a new organizational model for IT resources and IT management that would be "automation-friendly" with the added constraint of being backward compatible with existing IT infrastructure and as non-intrusive as possible.

We use a service orientation model where the service represents a contained set of functionalities related to a resource or class of resources. Services share similar communication protocols among them as well as unified management interfaces allowing other services or software agent to interact with them in a predictable way. Although modeling and describing resources as similar software components, will provide the software agents with access to the *technical* description of the resources and they could then initiate and endure communication, however they will not be able know what the service is actually doing. Another layer is needed to allow software agents to also access or "know" the *semantics* of the resources in a way similar to what a human would do for the purpose of assessing the capabilities and properties of a resource. Describing resources in a well-formed semantic language will not only provide a robust way to manage knowledge about them and their environment but would allow specialized inference engines to act on them to define their behavior in an autonomic manner.

Our solution is composed of two parts, the *Managed Resource Framework (MRF)* is a framework for automatically generating ready-to-deploy service representations of resources from their semantic representations. The objective is to have a computer-aided process by which resources can be rapidly *instantiated*, deployed and managed in a relatively quick and transparent manner for the user. The framework assumes the existence of a semantic representation of a resource written in OWL, an ontology language, that extends a basic resource model, and outputs a deployable service representation called *Managed Resource Archive (MRA)*. The only human intervention during this process would be in the case there were custom capabilities defined in the semantic representation and that lack an implementation. The Managed Resource Archive is a deployable service representation of the resource that is generated by the Managed Resource Framework. The MRA is a Web application formed by a bundle of servlets, classes and other resources and intended to run on *Managed Resource Containers* or on the classical Java servlet containers with, however, a loss of capabilities. Using the Managed Resource Framework, it is possible to generate resource artifacts that constitute the Managed Resource Archive. MRF assumes a target based method by which, only one or several constituent of the MRA could be generated as needed instead

of the monolithic MRA. Important requirements for the MRF are simplicity, practicability and usability. Many solutions that are meant to ease the management of IT end up being too complex and sometimes they add to the hassle of IT management, rendering their functionality moot and their purpose subject to debate.

In the second part of this work, we present an architecture for autonomic service-based management of the MRA services as created using the MRF. The objective is to have an environment where we can test our ideas and the suitability of the MRF. Automated management is done through the use of management models that describe a behavioral logic using rules. This logic is then incorporated into reasoners that have access to the resource models and the resource instances. The reasoners infer course of action and initiate the appropriate activities in accordance to the management models. The management models can be independent, inter-related or sometimes contradictory. Mechanism are implemented to manage the interaction between the different management models.

The ideas presented and discussed in this work are materialized in SEROM (SERvice Oriented Management), a Java implementation of the MRF and use cases for the autonomic service-based management.

1.3 Context

This work was funded by the IBM Böblingen Research Lab with the purpose of coming up with a practical solution for autonomic management of resource applied to two contexts that were of interest to IBM at the time when this work started. Namely Grid Computing with German Financial Grid as use case and End-to-End Automation with Green IT as use case.

1.3.1 The German Financial Service Grid

The Financial Service Grid (FinGrid) [50] is a project funded by the German Federal Ministry of Education and Research to develop a Grid architecture to virtualize services and processes in the financial sector and to build banking Grid services based on an accounting and pricing infrastructure through the development of several prototypes. In this context, we pursue research on the necessary components for a financial Grid to better model an industrialization and pricing scheme. We draw the architecture and implemented the resulting accounting and billing services based on a service oriented model.

1.3.2 End-to-End System Automation for Green IT

End-to-end (E2E) automation is a component of IBM's Tivoli System Automation for Multiplatforms (TSAMP) that deals with the automation of tasks in applications residing in different heterogeneous clusters. There was an interest to use the E2E management model in the context of a Green IT scenario. Green IT, or Green Computing, represents the environmentally aware efforts for an efficient use of computing resources. It has three big components: reducing the use of hazardous materials, improving the recyclability of the computing materials and reducing the energy used by those resources. In our case, we are only concerned with the latter component, namely, reducing energy costs. The idea was to gather energy usage and temperatures from different sensors in the data centers and use this data to move around load from heavily used servers to lesser used and thus "cooler" servers.

1.4 Related concepts

1.4.1 Service-Oriented Computing

In today's world, IT systems and IT processes are becoming more and more complex and more and more interconnected, making IT systems a huge pool of interconnected heterogeneous mini-systems that need to be managed in a coherent and consistent way. Moreover, those systems have to be scalable not only from within, i.e. some component of the system, but also by accepting the integration of external elements. We just described the nightmare for any system administrator: managing a growing pool of interconnected composition of heterogeneous legacy and recent systems. That is a real problem with no easy solution. There are, however, approaches to ease dealing with such a scenario. One of the most known and established approaches is the Service-Orientation principle.

The Service Orientation principle is based on a simple idea that proved its efficiency in dealing with "problems" ranging from organizing a birthday party to waging large scale warfare. This principle was in fact mentioned, among other places, in *Book VI of The Art of War* by Machiavelli. This principle is nothing else than the famous "divide and conquer"! Dividing the problem into small chunks and "dealing" with every chunk alone proved to lower the complexity and hence the difficulty of dealing with problems. Service-orientation follows the same principle with an added bonus, that is to reduce all the smaller chunks to the same rep-

resentation called *service*.

Organizing the system into small and similar chunks solves immediately the heterogeneity problem and helps considerably with the complexity and scalability. Service-orientation as a principle, can be thus defined as a world vision where resources are clearly partitioned and consistently represented. This model applies equally to a task, a solution or an enterprise. Applied to IT, service-orientation would be defined as a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. The last idea in the definition, mainly that resources can be part of different ownership domains, is very important because more and more IT capabilities are used by entities that do not necessarily own them but are nonetheless part of the IT system. Think of, as an example, Cloud Computing, namely remote services like Google App Engine [43] or Amazon EC2 [3].

1.4.2 Grid Computing

Grids consist of a virtual platform for computation and data management using a heterogeneous cluster of computer resources [11]. It enables users and applications seamless access to vast IT capabilities. In his reference paper [39], Foster provides a three-points checklist that a system has to fulfill before it could be identified as a Grid. The first is that [the Grid] *coordinates resources that are not subject to centralized control*. Meaning that not only the Grid resources are geographically distributed but that those resources belong to different administrative domains, i.e. different institutions or departments. Issues like security, usage policies, accounting and billing arise. This conveys the problem of the management of Grids as it becomes very complicated to manage resources that are subject to different usages and regulations. Issuing terms of usage and guarantees for the utilization of the Grid is inconceivable when you have no global vision of the Grid resources nor are they under your direct managerial control. Moreover by their intrinsic nature, the majority of the existing Grid implementations lack the necessary flexibility for resource recomposition and adaptation to changing user requirements and needs making the return of investment (ROI) of a Grid system tied to the time frame of usability of the Grid. A period that can be very short according to today's adaptive enterprise principles.

1.4.2.1 Service-Oriented Architecture and the Grid

Service-Oriented Architecture (SOA) guidelines and web services technologies can be used to construct a solution for a flexible model for Grid management that would tackle the fore-said challenges in the parent section. According to OASIS[2], SOA is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. Some of the main drivers for SOA-based architectures are to *facilitate the manageable growth of large-scale enterprise systems and to facilitate Internet-scale provisioning and use of services*. It revolves around the concept that needs or requirements of one party are met by the capabilities of another. The parties at either ends can be a person or a software agent. Applying SOA principles to the Grid seems to be a natural process. The Grid is composed of a set of distributed resources under the control of different administrative domains and SOA is a model for organizing such system. In fact this amalgam is not the product of pure hazard, the standardization efforts for the Grid are channeled towards the adoption of web services technologies. The most prominent example is the WSRF [6] set of specifications that lay the necessary infrastructure and building blocks for a Service Oriented Grid Architecture.

In SOA, the central mechanism for coupling needs and capabilities are services, defined by the capability of performing work for another, specifying the work offered for another and the offer to perform work for another. The first step to meet SOA objectives is done through decomposition or factoring of complex systems into smaller chunks for more convenient design, implementation and maintenance. Those smaller chunks are what services are supposed to be: small independent components easier to manage and control. SOA architectural models do not preconize the specific use of web services, however they constitute the used de-facto standard.

1.4.2.2 Accessing resources through services

Web services are stateless application components, which are not suitable to describe and interact with Grid resources being logical or physical (servers, storage media,...) that need to maintain a state. Web services use the Simple Object Access Protocol (SOAP) to exchange messages. SOAP constitutes the foundation layer for web services communication and is mainly implemented over HTTP or HTTPS that are per se stateless. Web services need therefore to define custom means to preserve statefulness. It is here where WSRF gets into the picture.

[2]http://www.oasis-open.org

The WSRF set of specifications provides a general solution using web services to an originally specific problem: describing and representing Grid resources. WSRF establish the concept of the Resource Properties document that describes a view of a resource. The resource properties document is referenced in the Web Service Description Language file (WSDL), it describes the physical properties of a concrete resource and provides the possibility to modify those properties through operations on the document. A resource that is described using the resource properties document is called a WS-Resource. Another relevant feature of WSRF is that it brings a solution for management of a WS-Resource lifetime, faults and properties. Rendering Grid resource as WS-Resource and decomposing software components into services is the first step towards an SOA enabled Grid architecture with all the advantages that it can bring such as ease of management, adaptability and automation.

1.4.3 Autonomic Computing

Autonomic computing as coined by IBM [59] refers to self-managed distributed IT resources. The term was inspired from the human nervous system, where the body responds to its environment without conscious acts from the person. Tasks like breathing, sweat regulation or cellular reparation are performed by the body without it being instructed to. Applied to IT, autonomic computing aims towards computing systems that manage themselves based on high-level guidelines from human administrators. The ultimate goal is to lower the complexity of managing IT. Autonomic computing does not aim at eliminating the role of the IT professionals [85], but instead, it is supposed to relief them and to free them to concentrate on their core business.

Autonomic computing has eight defining characteristics [53] that are represented in four self-management areas:

self-configuration: systems should be able to configure themselves following high-level guidelines to adapt to new situations such as installing new software components without system disruption.

self-optimization: systems should be able to optimize themselves depending on the situation such as spawning new processes if augmented load is sensed or turning off some sections of the system when they are not used.

self-healing: systems should be able to heal and recover from failures or other situation that would disrupt its normal behavior. The system

should be able to discover the source of the problem and circumvent it.

self-protection: systems should be able to to protect themselves from unauthorized access, viruses, tempering and other security breaches. Protection can also mean physical protection.

All these characteristics require at least the following capabilities of a self-managed system:

- awareness
- ability to analyze
- ability to plan
- ability to affect its state or its environment state

We think that a self-managed system has to be aware of its state and its environment and be able to analyze the situation and plan accordingly. The plan needs then to be applied to the system itself or to its environment.

1.4.4 Semantic approach

By semantic approach we mean the use of the technologies rooted in the semantic web as coined by Tim Berners-Lee when he defined it as "a web of data that can be processed directly and indirectly by machines" [13]. To achieve this objective he proposed to include semantic elements to the traditionally syntactical, hyperlink-based world wide web. The idea to incorporate machine-readable elements to allow software components to “understand” these additions went way beyond its application in the world wide web to other aspects of IT. A domain of application is resource management where it is often not enough for the management applications to know the technical aspects to communicate or command the managed application. Allowing management applications to have a better understanding of its environment would empower them to interact with it in ways not possible otherwise. The semantic web has a number of mature technologies used to provide a formal description of concepts for the purpose of describing and handling the resources represented by these concepts. This topic will be discussed in more details in chapters 2 and 3.

1.5 Methodology and structure of this work

The methodology used in this work is based on the design science paradigm that has the objective to [52]: "*... create and evaluate IT artifacts intended to solve identified organizational problems. Such artifacts are represented in structured form that may vary from software, formal logic, and rigorous mathematics to informal natural language descriptions*". The design science methodology has two design processes: build and evaluate [70]. Research is done through the building of artifacts and their evaluation in regards to the business needs. We follow the framework of the design science as outlined in [52] where we use an observational, experimental, and descriptive evaluation method of the work.

In chapter 2 we give an introduction to the knowledge representation and reasoning as well as to ontologies and the engineering of ontologies. In chapter 3 we present and discuss the ideas of this work and the system organization as well as the related work. In the chapters 4 and 5 we present the two results of this work, namely the Managed Resource Framework and the Semantic Resource Management Infrastructure. In chapter 6 we present the resulting prototype and we conclude in chapter 7 with a summary and future work discussion.

Chapter 2

Knowledge Representation and Ontology Engineering

2.1 Overview

Knowledge Engineering is a research area in artificial intelligence that deals with issues related to knowledge such as knowledge *acquisition*, *representation* and *application*. Knowledge representation is an important field that aims at representing knowledge using formal symbols for the purpose of reasoning over this knowledge and generating implicit knowledge through inferential procedures that are a formalization of the reasoning notion. Several formalisms for knowledge representation have been suggested. Most notable are *rule sets* [27], *generalized graphs* [36] and *predicate logic* [62]. We will present different approaches for knowledge representation and concentrate more on the logic representation.

2.2 Knowledge representation problem

The notion of knowledge representation is not complex, and rather simple to explain. Knowledge representation deals with capturing a view or a state of the world in some language or communication medium. It is however important to emphasis the need for capturing such view. The majority of applications that make use of knowledge representation, need to capture some knowledge for later manipulation, to reason on such knowledge, or both. From that we can derive that the first issue that we are confronted with is the *knowledge representation language*. There are two dimensions to a knowledge representation language, a *syntactical* dimension and an *inferential* dimension. The syntax of a language is the set of rules

by which we can construct valid sentences (called expressions in our context) in that language and it governs the way knowledge can be stored explicitly. The inferential dimension is how readily the symbols in the knowledge base can be manipulated to generate new (and valid) knowledge, called implicit knowledge, from the already existing explicit knowledge. This operation is performed by a component called an *inference engine*. The inference engine uses a set of predefined rules to generate the new knowledge that are called *inference rules*. An example of such a rule, based on *modus ponens,* would be: "if φ then ψ" , applied to a knowledge base containing the expression φ, it would infer that ψ is the case. There are a number of other issues related to the representation language and the knowledge representation model used. Some of those issues are presented below [92]:

Expressive adequacy

The expressive adequacy deals with the expressiveness of the representation language. It answers the questions: How expressive should the language be? Can all the aspects of the knowledge be captured using the language? How easily this can be done? The most important thing to note here is that the higher the expressiveness of a language is, the easier it is to represent complex concepts and knowledge with it. However, this expressiveness goes in parallel with the complexity of the inference process, to the point that the inference can take a large amount of computation and even be *undecidable*, or in other words, not guaranteeing a result.

Reasoning efficiency

The reasoning efficiency is a general representation problem in computer science: it is of little importance that an inference computation terminates (or any computation for that matter) contrasted to the fact that this computation terminates in acceptable times. There is always a trade-off between the expressive adequacy and the reasoning efficiency as mentioned earlier: Typically, the more the language is expressive, the more computation over objects represented using that language takes time. Choosing the *acceptable* balance between reasoning efficiency and expressiveness is a complex decision that depends of course on the definition of "acceptable" and on the computational resources available to the inference engine.

Primitives

Primitives are the atomic elements of the representation language. They are used in composing more complex expressions and formulas. Choosing the proper primitives for the language is of course important and affects both the expressiveness and the reasoning efficiency.

Meta-representation

The meta-representation relates to the knowledge about knowledge. It deals with representing the knowledge in a knowledge base and with representing the structure of that knowledge.

Incompleteness

Incompleteness is a complex research topic that deals with knowledge that is missing and how should the inference engine deal with that. What if the unknown if it became to be known should contradict what we believe to be true (the existing knowledge)? Shall we even allow such knowledge?

Real-world knowledge

The real-world knowledge is another complex research topic that deals with formalization of notions like *beliefs*, *desires* and *intentions* and with avoiding paradoxes related to self-referential propositions.

2.2.1 Levels of knowledge representation

Brachman discussed in his work on semantic networks five different levels of knowledge representation [37], the implementation level, the logical level, the epistemological level, the conceptual level and the linguistic level. This organization is, however, general enough to be broadened to encompass other knowledge representation formalisms.

The implementation level

Deals with the implementation of the necessary constructs needed to use the knowledge representation language on a computer. At the syntactical level, the choice of the data structures to hold and organize the expressions is the most important issue. At the inferential level, the concern is with the implementation of algorithms that are used to infer facts from the knowledge base.

The logical level

Deals with the logical properties of the representation language at the syntactical level. Especially about the meaning of formalisms that are used in expressions. A famous example is the following: φ is-a ψ. What does "is-a" in that context mean? Does it mean that every φ is a ψ? Or that φ have all or some properties of ψ (inheritance)? Both are valid, but it is desirable that the meaning of "is-a" to be unique and consistent. Another issue is the expressive power of the language as stated earlier. To know what knowledge can be represented and what cannot. At the inferential level, the issue are the logical properties of the inference and especially the soundness of the inference rules. Soundness, in logics, is the property that the rules preserve the *truth* in the system. We say that a system is sound if and only if its inference rules derive only expressions that are correct with respects to the system's semantics. Etymologically, the Germanic word *Sund* as in *Gesundheit* is the origin of the word sound. Something is sound, thus, it is healthy!

The epistemological level

Is concerned with the *type* or *class* of primitives and inference strategies to be used, without specifying those primitives or strategies. Epistemology or *theory of knowledge* is a philosophical branch that studies the nature of knowledge or the meta-knowledge. In this context, the epistemological level is not concerned with the actual primitives of the knowledge but rather how those primitives should be organized and to which class should they belong. Same goes for the inference strategies. As an example, the inference strategies for theorem proving would be different from those for medical diagnostics regardless of the strategies themselves.

The conceptual level

Deals with the actual primitives of a knowledge representation language and the actual inference strategies. While in the epistemological level we are concerned with the types of primitives and inference strategies, such as *semantic networks* for instance, where we decide that concepts would be represented as nodes and relationships as arcs between nodes and inferences would be drawn from those arcs, at the conceptual level we could specify that "is-a" would be a specific arc representing the inheritance property.

The linguistic level

Is the last level in Brachman work. It deals with the way a particular knowledge is represented in a given knowledge representation formalism. Our discussion here is in general about the knowledge representation formalism, so this level is not of particular importance for us in this work.

We should make a note here about the importance of the knowledge domain for all the forementioned levels. The knowledge representation scheme or model depends heavily on what domain we are dealing with. In the mathematical domain, it would be easy to represent mathematical concepts and inferences (mathematical rules) in a logical language, however this is less straight forward when we deal with other domains such as medicine for instance where the exception is the rule.

2.3 Approaches to knowledge representation

There exist several approaches to knowledge representation. We present here the most frequently used ones: semantic networks, production rule systems and logic.

2.3.1 Semantic networks

The "modern" use of networks to represent knowledge on a computer system begun with the seminal paper of Quillian (1968) [89] on the first semantic memory model that covers both general knowledge and word knowledge. All these networks share the property of describing relationships between concepts using links between nodes that represent those concepts. Semantic networks belong to the generic family of *structured objects* as defined by Nilsson [82]. This family contains other formalism such as Frames and Object-Oriented systems. Structured objects represent knowledge using elements similar to the nodes and arcs of graph theory or the slots and fillers of record structures [57].

Figure 2.1 is an illustrative example that shows a semantic network of a vulgarized snapshot of the tree of life for the mamals cats and mice. It presents concepts in ovals linked with labeled arrows. From the figure we can read that "Penny is an instance of Cat" that "Cat is a Filidae" and that "Filidae is a Carnivora" for instance. Note that we cannot conclude that Penny is from the order Carnivora from the figure alone as there is no direct link between the two. The information that the "is a" link is transitive

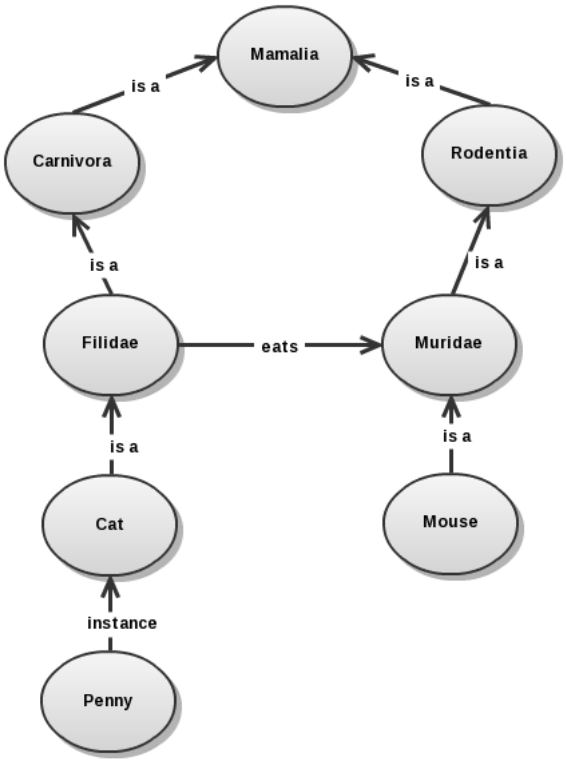

Figure 2.1: A Semantic Network

needs to be formalized somehow. Same with the fact that Penny can eat mice that is not straight forward from the network shown in the figure. We need more information, especially that the link "is a" is an inheritance, meaning that "Cat" inherits all the properties of "Filidae". Only then we can say that the carnivore Penny can eat a mouse!

There is no way (from the network alone) to specify such properties for the links or to have more control on the concepts such that the concept "Filidae" is mutually exclusive with the concept "Muridae". This lack of semantics, let alone formalisms, lead many practitioners and researchers to formalize semantic networks using other representation formalisms, especially predicate logic.

2.3.2 Production rule systems

Another way to represent knowledge is the use of rules that have the form **IF** *antecedent* **THEN** *consequent*. The rule is *satisfied* when the antecedent holds, which leads to the execution of the consequent. The antecedent is also called the *body* of the rule and the consequent is called the *head* of the rule. The production rule system consists of three components: *working memory*, *production memory* and the *inference engine*.

The inference engine is a reasoning system that keeps its actual knowledge in a database-like structure called the working memory. The working memory gets updated in real-time with the changes in the system state. The inference engine's task is a three steps cycle [15]:

- matching and rule selection: recognize the rules that are applicable.
- resolution: resolve conflict among the resulting rules.
- firing: act accordingly by changing the working memory and firing the appropriate actions.

It is called inference engine because it matches facts with the rules to *infer* actions. The facts are stored in the working memory whereas the rules are stored in what is called production memory. Facts maybe added or removed from the working memory at run-time depending on the data received. A system with a large number of rules and facts may find at a certain time several rules to be true for a specific working memory state. Chances are that some of these rules would conflict (shutdown the database server vs. keep a high availability for premium clients to access the database). The inference engine needs then to implement conflict resolution strategies.

Following are examples of production rules[1] written in natural language:

1. **IF** something is golden customer **THEN** it is also a customer
2. **IF** something is platinum customer **THEN** it is also a golden customer
3. **FACT** the event "Event_0X" is a VM assignment
4. **IF** there is a VM assignment event for a golden customer and resource usage of less than 240 hours **THEN** the cost per second is 0.0002

The same rules reproduced in a formal logical language more suited for computations:

1. $customer(?c) \ :- \ golden(?c)$
2. $golden(?c) \ :- \ platinum(?c)$
3. $event(?e)$
4. $eventCost(?e, 0.0002) \ :- \ event(?e) \wedge customer(?c) \wedge LessThan(?u, 240 * 60 * 60) \wedge VMAssignement(?e) \wedge golden(?c)$

Rules (1) and (2) are examples of inheritance. Rule (3) is a simple fact whereas rule (4) is a more developed rule. The sign $:-$ indicates an inverse implication, meaning that you should read the rules from right to left or, if you prefer to read them for left to right, you need to know that you have to start with the head of the rule then the conditions that would make the head hold.

Production rules may seem at first glance similar to more classical conditional statements that we find in programming languages, however there are two distinctions between the two. The first is that in the production rules, the antecedent is expressed as a pattern rather than a boolean expression. This can be emulated in the programming languages by using a boolean function in the conditional statement that would compute the complex pattern. However, the second distinction is a real and important one. The control flow in the production system is defined solely by the inference engine. Contrast that to the programming languages where the control flow is passed sequentially or based on special constructs (e.g. GOTO). This, on the other hand, does not give guarantees on the order of executions of the rules in a production rule system.

[1]These rules are adapted examples from the FinGrid Accounting and Billing project [50].

2.3.3 Logic

One of the earliest definitions of logic is that logic is *"the tool for distinguishing between the true and the false"* (Averroes). Logic is used to study the correctness of inferences. A correct inference is a *truth* preserving inference, meaning that if the premises is correct, then the conclusion drawn from this premises must be also correct. However, building a formal system of correct inference requires that we construct it in a formal language, which leads us to the definition of a formal language first. In fact, in mathematics and computer science, logic is the study of inferencial properties of formal languages. There are three aspects of a declarative language (the kind we use for representing knowledge), the *syntax*, the *semantics* and the *pragmatics*. Constructing a formal language amounts to defining strings of symbols that constitute the well-formed expressions or sentences. This aspect of logic is referred to as syntax. In C++ for instance the expression `int x(-10u);` is a valid syntax whereas in C it is not. Validity of expressions is important but it is not the finality, we want to have truth preserving inference. Truth is a semantic notion related to the meaning of the expressions. In other words we need to interpret the meaning of the expressions to assert their validity. The expression `int x(-10u);` declares a variable `x` of type `int` and assigns to it the *unsigned* value of `-10`. It may look invalid if we use our common sense, but the forementioned expression is valid and compiles under gcc 4.4.4 as it follows the semantics of C++. The pragmatics aspect is concerned with the use of the expressions of the language and specifically the logical symbols that affect the interpretation of the non-logical symbols like the logical consequence or conjunctivity for instance.

Knowledge-based systems can be viewed at a symbolic level, or a knowledge level [81]. Representation language formalism lies at the knowledge level, where we are concerned with the expressive adequacy of the language and its entailment relation. Logic being the study of entailment and rules of inference, the tools of formal symbolic logic are ideally suited for a knowledge representation system [15]. A popular logical language is the first-order logic (FOL). Although we use a subset of FOL in this work, it is of interest to have a short introduction of FOL here. We hope to introduce some important concepts necessary for the comprehension of this chapter.

2.3.4 First-order logic (FOL)

2.3.4.1 Syntax

The first thing that we define for FOL is its syntax. We do this by defining the vocabulary then the formation rules.

Definition 2.1:
The vocabulary of FOL consists of:

1. constants $\{a_1, a_2, \ldots\}$
2. variables $\{x_1, x_2, \ldots\}$
3. connectives $(\wedge, \vee, \rightarrow, \exists, \forall \, and \, \neg)$
4. n-ary predicate symbols, n being positive integer $\left\{P_1^n, P_2^n, \ldots\right\}$

Concerning the formation rules, the following definition presents the FOL's allowed constructs used to form new expressions.

Definition 2.2:
Let ϕ and ψ be variables, c_i a constant, x_i a variable and P_j^n an n-ary predicate, then the constructs:

$(\neg\phi)$
$(\phi \wedge \psi)$
$(\phi \vee \psi)$
$(\phi \rightarrow \psi)$
$P_j^n \, (c_1 \ldots c_n)$
$\exists x_j \left[\phi \left[c_i / x_j\right]\right] \; and \, \forall x_j \left[\phi \left[c_i / x_j\right]\right]$
where $\phi \left[c_i\right]$ is a sentence containing the constant c_i and,
where $\phi \left[c_i / x_j\right]$ is the substitution of c_i with the occurrences of x_j
Are valid sentences.

You should note that the above definitions define a class of languages rather than a specific language and this because the first definition states that there should be constants, variables and predicates (the vocabulary) at the epistemological level, i.e. without specifying what are those constants, variables and predicates. For creating and actual language, we need to act at the conceptual level by specifying a vocabulary or the constants and predicates that would represent our domain knowledge.

2.3.4.2 Semantics

The semantics deal with defining the meaning of the logical expressions in FOL so we can define what would be a valid consequence. Which in itself is the formalization of correct inference. However, we cannot explain the meaning of every sentence in FOL for the simple reason that FOL allows non-logical symbols with their meaning changing depending on the domain and the interpretation of knowledge. For example, the object "keyboard" can mean different things whether our context is IT or music instruments. So what we do is to specify the meaning of the expressions as *a function of the interpretation of the predicate symbols* [15]. Such specification is achieved in the following way: for any predicate P of arity 1, P^1, there exist a set of objects D that satisfy this property. As examples, if P^1 is *happy* for instance then D is the set of the happy people, or if P^1 is the predicate *dualCore* then D is the set of dual core processors or machines depending on the domain knowledge. As a generalization, for n-ary predicates P^n, D would be the set of n-objects sets that satisfy the property P^n. The predicate *parentOf* is a binary predicate and the objects that satisfies it are the binoms that have one of them a parent of the other. The following is a definition of the semantics of FOL as we just described:

Definition 2.3:
An *interpretation* $\mathfrak{I}$ in FOL is the ordred pair $\langle \mathcal{D}, \mathcal{I} \rangle$,
where $\mathcal{D}$ is a non-empty set of objects, called the *domain* of $\mathfrak{I}$,
and $\mathcal{I}$ is the *interpretation mapping*, such that
if c_i is a constant then $\mathcal{I}(c_i) \in \mathcal{D}$
if P_j^n is an n-ary predicate, then $\mathcal{I}\left(P_j^n\right)$ is an n-ary relation over $\mathcal{D}$.

$$\mathcal{I}\left(P_j^n\right) \subseteq \mathcal{D} \times \cdots \times \mathcal{D}$$

Now that we defined an interpretation, we can define the meaning of more complex sentences. We introduce the *denotation* term which simply refers to the object referred by a certain term. So to find objects denoted by *parentOf(HerculePoirot)* given an interpretation $\mathfrak{I}$, we use $\mathcal{I}$ to identify the predicate *parentOf* and apply it on the object *HerculePoirot* in $\mathcal{D}$ to get some other object in $\mathcal{D}$ that would be the denotation of *parentOf(HerculePoirot)*. If we have a term that uses a variable, then we need to have a mapping from that variable to some object in $\mathcal{D}$, called *variable assignment* over $\mathcal{D}$. If x is a variable, the element denoted by x using the variable assignment μ would be $\mu[x]$. The following definition formalizes the denotation concept:

Definition 2.4:
Given an interpretation $\mathfrak{I}$ and a variable assignement μ, the denotation $\|t\|_{\mathfrak{I},}$ of the term t is:

1. $\|x\|_{\mathfrak{I},} = \mu[x]$ if x is a variable.
2. $\|P^n\|_{\mathfrak{I},} = \mathcal{P}^n$ if P^n is an n-arry predicate over $t_1 \cdots t_n$ terms, where

   $\mathcal{P}^n$ is the interpretation mapping of P^n, $\mathcal{I}(P^n) = \mathcal{P}(u_1 \cdots u_n)$ and $u_i = \mu[t_i]$

Now we have enough tools to check wether a formula is true or false given an interpretation. Formally we say that formula ϕ is satisfied in $\mathfrak{I}$, written $\mathfrak{I}, \vDash$ or ϕ is not satisfied in $\mathfrak{I}$, written $\mathfrak{I}, \mu \nvDash \phi$. We can omit μ from the expression when the formula does not have variables or when it is clear from the context which variable assignment we use. The following rules shows when a sentence is satisfied given an interpretation:

Definition 2.5:
$\mathfrak{I}, \vDash$:

1. if ϕ is an n-ary predicate, then $\mathfrak{I} \vDash$ iff $\langle \mu[t_1] \cdots \mu[t_n] \rangle \in \mathcal{I}(\phi)$
2. if ϕ is a negation, $\neg\psi$, then $\mathfrak{I} \vDash$ iff $\mathfrak{I} \nvDash$
3. if ϕ is a conjunction, $\psi \wedge \chi$, then $\mathfrak{I} \vDash$ iff $\mathfrak{I} \vDash$ and $\mathfrak{I} \vDash$
4. if ϕ is a disjunction, $\psi \vee \chi$, then $\mathfrak{I} \vDash$ iff $\mathfrak{I} \vDash$ or $\mathfrak{I} \vDash$ or both
5. if ϕ is a generalisation, $(\forall x)\,\psi$, then $\mathfrak{I} \vDash$

   iff $\mathfrak{I}' \vDash [x/a]$ for all interpretations $\mathfrak{I}'$ which are similar to $\mathfrak{I}$ with the exception of at most the individual a. Or, in other terms:

   iff $\mathfrak{I}, ' \vDash$ for every $'$ that differs from μ on at most x
6. if ϕ is in the form $(\exists x)\,\psi$, then $\mathfrak{I} \vDash$

   iff $\mathfrak{I}' \vDash [x/a]$ where there exists $\mathfrak{I}'$, an interpretation similar to $\mathfrak{I}$ with the exception of at most the individual a. Or, in other terms:

   iff $\mathfrak{I}, ' \vDash$ for some $'$ that differs from μ on at most x

2.3.4.3 Pragmatics

We introduce in this section the concepts of *logical consequence* and *entailment,* necessary to achieve the truth preserving inference. In the previous section we defined when a sentence is valid given a certain interpretation. However, we are more interested in the truth preserving inferences by abstracting from any given interpretation. In other terms, a sentence is valid if it is valid whatever interpretation we may use. We write $\vDash$.

$$\vDash \text{ iff } \forall \mathfrak{I}, \mathfrak{I} \vDash$$

A valid logical consequence is a generalization of the above. As an example, let α, β and γ be sentences such that $\forall (\alpha, \beta)$ and $\gamma = \neg (\beta \wedge \neg \alpha)$. For every interpretation where α is true we can see that γ is true regardless of the non-logical symbols that α or β may have[2]. We say that γ is a *logical consequence* of α or that α *entails* γ. More formally,

> Definition 2.6:
> Let S be a set of sentences, ϕ any sentence and $\mathfrak{I}$ an interpretation.
> We say that γ is a *logical consequence* of S or that S *entails* γ,
> iff $\forall \mathfrak{I}$, $\forall \psi \in S$, if $\mathfrak{I} \vDash$ then $\mathfrak{I} \vDash$.

We say that S is the *premises* and ϕ that is the *conclusion*. Another way of formulating the logical consequence is that there is no interpretation $\mathfrak{I}$ such that $\mathfrak{I} \vDash S \cup \{\neg\phi\}$, the set $S \cup \{\neg\phi\}$ is unsatisfiable. The most important conclusion to have here is that using logical consequence or entailment, it is impossible for the premises to be true and the conclusion to be false. In other terms, logical consequence is the mathematical formalization of the notion of a correct inference. The importance of this formalization is formidable as we have now a system with the proper tools to reason in a logical way, very close to what a human would do! The meaning of the non-logical symbols like the term *HerculePoirot* is non relevant. We just proved that whatever interpretation we may have (*HerculePoirot* may be a book character or the neighbor's dog), if we logically entails that *HerculePoirot* is smart, then *it* is smart because drawing logical conclusion using logical consequence is *safe*.

2.3.4.4 Proof theory

The proof theory is a set of inference rules used to derive valid conclusions. In [90] it is defines as "... an abstract specification of an inference machine

[2] $\gamma = \neg (\beta \wedge \neg true) = \beta \vee true = true$. Regardless of the value of β.

that determines valid consequences of a set of sentences". We say that a sentence ϕ is proven from the set of sentences S if, using the proof theory inference rules we can conclude ϕ from the sentences in S and we write $S \vdash \phi$. Proof theories can be characterized depending on their properties such as soundness, completeness and decidability.

Soundness

A system is sound if and only if its inference rules derive only expressions that are correct with respect to its semantics. Formally, if a system is sound then:

$$\text{if } S \vdash \phi \text{ then } S \models \phi$$

This is the most desirable property of the proof theory for obvious reasons: deriving incorrect results defies the purpose of using a reasoning system.

Completeness

A system is sound if its inference rules can derive all what could be true. Completeness is the converse of soundness: if ϕ is true given any interpretation then ϕ is a theorem of S. Formally, if a system is complete then:

$$\text{if } S \models \phi \text{ then } S \vdash \phi$$

Both soundness and completeness are necessary for a proof theory to be truth preserving.

Decidability

Given a proof theory, a set of sentences S and a sentence ϕ, we say that a system is decidable if proving that $S \models \phi$ or that $S \not\models \phi$ can be done in finite time. When proving $S \models \phi$, some systems have an inference procedure that can answer "yes" in finite time if $S \models \phi$ is true, but have no inference procedure that can answer "no" in finite time if $S \models \phi$ is false. We say that those systems are *semi-decidable*.

2.4 Analysis of methods of knowledge representation

In the following sections we discuss the advantages and disadvantages of each method and present a justification for the choice that we did as

knowledge representation formalism for this work. A deeper analysis of the representation formalism can be found in [92, 90, 57, 97, 15]. In our work we use a semantic representation of knowledge, mapped into a logical formalism. Specifically, we use the description logic fragment of the first-order logic.

2.4.1 Analysis of production rules systems

2.4.1.1 Advantages

The first advantage of rules is their modularity. Their intrinsic nature make them contained pieces of knowledge independent of each other and of the whole system where they reside. There is also a distinction between the *permanent* knowledge that is stored in the knowledge base and the *temporary* knowledge in the working memory, that represent the needed pieces of knowledge that the inference engine use/generate for its current problem [107]. The inference engine implementation is also independent of the rules in the knowledge base, making it possible to use the same rules with different inference engines as well as different control mechanism (backward or forward chaining). The closeness of the rules to the human language makes it easy for domain experts to capture their knowledge into rules without the need to spend much effort in learning a formal language. Another advantage is their restricted syntax, as what is allowed in the head or the body of the rule is quite limited. The body contains simple object-value-attribute triples, whereas the head contains one or few simple operations [90, 27]. The first advantage of the restricted syntax is the ease by which we could test for the *consistency* of the knowledge base by writing relatively simple consistency checking operations. The knowledge base is consistent when no contradictory conclusions can be drawn from it. If rules had a complex syntax, this checking would have been more complex to put in place.

2.4.1.2 Disadvantages

The main disadvantages with the production rules systems have to do with efficiency. The matching step in the inference engine cycle is computationally not efficient and can hinder the proper running of applications. Another inefficient process is the conflict resolution step. We have a conflict when several rules are a match and it is unclear which rule to fire first? Shall the result of the first fired rules affect the following rules? Shall we execute all the rules in a specific sequence, or have a mechanism to

select one or some of them? The inefficiencies for conflict resolution result from the process of selecting the set of rules that match and that constitute the conflict set, and from running a proper conflict resolution scheme. On large systems, one can see that those operations would computationally be very expensive. One of the most used solutions to optimize the conflict resolution process is the use of a tree-structured sorting network of the rule conditions called the Rete algorithm [38]. Although the restricted syntax of rules has its clear advantages, it has also some problems, especially with representing some types of knowledge such as incomplete knowledge.

2.4.2 Analysis of semantic networks

2.4.2.1 Advantages

Semantic networks in specific and structured objects in general have several advantages. Many of these are at the epistemological level and have to do with the way the knowledge is organized in those formalism. Semantic networks offer a natural way to capture domain knowledge and emulates how experts think about their domain, i.e. in terms of concepts, classes and relationships, which is a clear advantages over the fact-based approach. The visual organization of knowledge allows a quick grasp of the whole system and permits to identify at first glance relationships that may not appear clearly using other representation formalisms. Another practical advantage is that in semantic networks, all the predicates about a certain concept will be linked to that concept, contrast this to the fact-based approach where we would have a number of facts (not necessary at the same location) having that concept somewhere in them. Moreover, theorem-provers are more efficient with the network organization of the predicates than with the fact based one [92]. Another advantage relates to the inference procedure of semantic networks. Although the inference procedure of logic based systems is more powerful, the inference procedure of semantic networks has its advantages, especially efficiency, due to the simple, less expressive formalism of semantic networks and to the inheritance property that semantic concepts have [90]. A last advantage that is worth mentioning here is that, in semantic networks, the inference procedure deals very well with default reasoning and default value. The default value is a value that is true for an object until proven otherwise. If there was such value it would have been linked to the object due to the fact that properties are related to concepts in semantic networks. When the inference procedure does not find such value linked, it can assume then the default value for the object.

2.4.2.2 Disadvantages

The biggest issue with semantic networks, frames or other structured objects is that their semantics are not well defined, that if they exist at all, and the graphical representation of concepts and links leaves often some questions unanswered. We use the infamous example of "is-a" to illustrate the issue. The expression:

$$P \underrightarrow{is\ a}\ Q$$

is ambiguous, especially concerning the meaning of "is-a". Now if we rewrite the same expression using a logical formalization:

$$\forall x P(x) \rightarrow Q(x)$$

$$Q(p)$$

the meaning is not only clear to the author, but also to the reader of the expressions. For this reason mainly, logic is used as knowledge representation language for structured object.

2.4.3 Analysis of logic as knowledge representation language

2.4.3.1 Advantages

From the previous sections it was clear that the first advantage of logic is that it has a well defined model theory (semantics). We know exactly what $\forall x, P(x) \rightarrow Q(x)$ means and we know how the inference engine would deal with that expression. Another important advantage of having well-defined semantics is that it allows to verify the soundness of the inference procedure, the only guarantee that all inferred conclusions derived from valid knowledge are valid conclusions. From earlier discussions, we also noted that logic is very expressive and we also noted the difficulty of the representation formalism to represent missing or incomplete knowledge. In [74] the author argues that representing incomplete knowledge can typically only be done using formal logic. Moreover the expressiveness of logic allows to represent *temporal* logic, that an object had a certain property in the past and not anymore for instance, or to represent *believes* or *epistemic* logic, that somebody believes that it is raining for instance. A last advantage that has more to do with inference than with representation is the proof theory, that all knowledge that is true in all models of a theory can be proven: completeness.

2.4.3.2 Disadvantages

Problems with logic as representation formalism can be seen at three levels, namely at the implementational, logical and epistemological level. At the implementational logic, due to the expressiveness of logic, the efficiency of logic as a representation formalism can be quite expensive especially when little care is taken during implementations of the control regimes [74, 90]. The second issue, at the logical level, is that some logics are undecidable. FOL for instance is semi-decidable. Asking an inference procedure wether $\phi \vdash S$, will *always* conclude in finite time with true if the answer is yes (decidability), however there is no guarantee that the procedure will answer false in finite time if the answer is no (undecidability). This property makes FOL a semi-decidable formalism with its applications. It is again due to the expressiveness of logics (FOL in this case). However, if decidability is important, the expressiveness can be reduced to achieve just that. *Description logic* (DL) for instance is a decidable fragment of FOL that we will be discussing later on. At the last level, the epistemological level, logic has some issues representing specific types of knowledge and reasoning such as procedural knowledge, default reasoning and abductive reasoning.

2.5 Ontology and Ontology engineering methodology

2.5.1 Ontology

2.5.1.1 Definition

Ontology has its roots in philosophy, specifically in the metaphysics branch of philosophy. It deals with the study of the nature of being and existence, their basic constituents, and the relationships among those constituents. Fundamental questions that Ontology tries to answer are "what is a physical object?", "what are the different modes it can have?", and "what categories can things be sorted into?". Important concepts in Ontology are particulars and universals and their relationships and the relationship between essence and existence as well as between extrinsic and intrinsic properties.

In computer science we speak of an ontology instead of Ontology as in philosophy. In [47], Gruber gave what became the most quoted definition of an ontology: "An ontology is an explicit specification of a conceptual-

ization". This definition was later on extended and explained in [96]:

> *An ontology is a formal, explicit specification of a shared conceptualization. Conceptualization refers to an abstract model of some phenomenon in the world by having identified the relevant concepts of that phenomenon. Explicit means that the type of concepts used, and the constraints on their use are explicitly defined. Formal refers to the fact that the ontology should be machine-readable. Shared reflects the notion that an ontology captures consensual knowledge, that is, it is not private of some individual, but accepted by a group.*

So an ontology is a formal description of a domain using concepts and relationships between those concepts. The example given in section 2.3.1 explaining figure 2.1 is an ontology, with "Filidae" and "Muridae" being examples of concepts and "is a" and "eats" being examples of relationships.

2.5.1.2 Forms and formalisms of ontologies

Depending on the application and the knowledge domain, ontologies can be modeled using different techniques and be represented using different formalisms. In any case, it is important to have a clear definition of the knowledge modeling components (concepts and links for instance) that are going to be used as building blocks of the ontology, the knowledge representation formalism used to formally represent those building blocks (logic, production rules, etc. See section 2.3) and the specific knowledge representation formalism language to be used to implement the ontologies. Depending on the knowledge formalism used, ontologies can be *highly informal* if we use natural language to represent them, however, if we use a restricted and structured version of the natural language then the ontologies are *semi-informal.* If we use a formal language, the ontologies are then *semi-formal*, and finally, ontologies are called rigorously formal of the representation used have formal semantics and truth-preserving proof theory.

Ontologies were mainly built using AI modeling techniques based on frames and FOL [42], however, with the advance of the semantic web, we see more and more the use of description logics as basis for the knowledge representation techniques used to build ontologies with the emergence of new description logic languages used for the ontologies implementation. Such languages are discussed in section 2.5.4. What follows is a discussion of the modeling techniques most used to build ontologies.

Frames and first-order logic

The first model using frames and FOL was proposed in [47], the author proposed five modeling components: classes, relations, functions, formal axioms and instances. We will be using the example given in section 2.3.1 explaining figure 2.1 to illustrate and give examples of the components.

classes: are a representation of an abstract concept. "Filidae" and "Muridae" are examples of a class then, but not "Penny" as it is not an abstract concept. Classes are generally organized into taxonomies. If we remove "Penny" from figure 2.1 as well as the link "eats", the new graph (not anymore a network) is indeed a taxonomy or at least a significant (as in domain significance) sub-taxonomy of the tree of life.

relations: are associations between the classes of the ontology. In figure 2.1, "is a" and "eats" are examples of such relations. They are both binary relations which is the general case. However there could be instances where we could have relations of higher order.

functions: are a special case of relations where the n-th element is unique for the n-1 preceding elements. The "Child(?person, ?mother)" relationship is an example where mother would be unique for person.

formal axioms: are sentences used to model tautologies or sentences that are always true. An example would be a formal sentence describing the *fact* that cats cannot fly.

instances: are used to represent individuals in an ontology. "Penny" is a clear example of such an individual that is an instance of the concept "Cat".

Description logic

In description logic, we use three kinds of modeling components to build ontologies: concepts, roles and individuals.

concepts: represent classes of objects and are equivalent to the classes concept in FOL.

roles: are relationships between concepts and are equivalent to FOL's relationships and functions. In general, there are no derived roles in DL, i.e. roles defined in terms of other roles as they lead to several reasoning issues.

individuals: are instances of concepts and they are equivalent to FOL's instances. Individuals in DL are stored in the assertional knowledge base (ABox) whereas the concepts are stored in the terminological knowledge base (TBox).

Other modeling techniques

The techniques that we presented up to now are, of course, not the only methods used to build and represent ontologies. Another used method is modeling ontologies using database modeling techniques.

2.5.2 Ontology engineering

Ontology Engineering is the set of activities related to the ontology development process, the ontology life cycle, and the methodologies, tools and languages for building ontologies [42]. With ontology engineering we try to model a settled view of reality represented as ontologies. However, this contradicts the idea that the world depends on the person that looks at it and on their viewpoint, therefore something that is always the same regardless from where it is perceived is nonsensical [84]. Ontologies have, therefore, to accommodate unity with variety as noted in [42], although it is similar to the concepts of *Golden Mean* [51] in the Greek philosophy, the two concepts are different in essence. It is more assimilated to discovering unity in the variety of nature and the variety of our experience.

2.5.3 Design principles

Design principles are criterias for guiding and evaluating ontology designs [48]. This section will outline the best practices in designing ontologies. Unfortunately there is still no standard design patterns for designing ontologies [80]. However many ontology-related literature presented such principles. We will try to summarize the most relevant in here.

Gruber, in his seminal paper [48], presented five principles for designing ontologies, and these are clarity, coherence, extendibility, minimal encoding bias and minimal ontological commitment. These five principles,

described as philosophical principles in [80], constitute a good and sufficient starting point.

Clarity

Clarity is an important principle, described by Gruber [48] as follows:

> *An ontology should effectively communicate the intended meaning of defined terms. Definitions should be objective. While the motivation for defining a concept might arise from social situations or computational requirements, the definition should be independent of social or computational context. Formalism is a means to this end. When a definition can be stated in logical axioms, it should be. Where possible, a complete definition (a predicate defined by necessary and sufficient conditions) is preferred over a partial definition (defined by only necessary or sufficient conditions). All definitions should be documented with natural language.*

The clarity objective is to remove the disambiguation from the ontology by formalizing the definitions and stating the necessary and sufficient conditions in the statements definitions. Definitions written in natural language often carry ambiguities, formalizing the ontology definition using a formal language allows to have a clearer and a consistent definition of concepts and their relationships.

Coherence

Coherence is an important principle that, to the difference of the other principles, is more concerned with inference than with knowledge representation per se. Coherence states that the ontology needs to be coherent. Specifically, the knowledge that is explicitly stated in the ontology and the knowledge that is inferred have to be non-contradictory. This is enforeced using formal axioms that would prohibit/invalidate any inferred/imported knowledge that would contradict with the existing knowledge. Gruber presented coherence in the following terms:

> *An ontology should be coherent: that is, it should sanction inferences that are consistent with the definitions. At the least, the defining axioms should be logically consistent. Coherence should also apply to the concepts that are defined informally, such as those described in natural language documentation and examples. If a sentence that can be inferred from the axioms contradicts a definition or example given informally, then the ontology is incoherent.*

Extendibility

Extendibility is an essential principle that states that ontologies should be designed in a way that supports extending them with concepts and vocabularies that may not be known at the time when the ontology was designed. Not only those principles have to be taken into consideration while designing the ontologies, but there should be a conceptual foundation allowing this objective. In Gruber words:

> *... one should be able to define new terms for special uses based on the existing vocabulary, in a way that does not require the revision of the existing definitions.*

Minimal encoding bias

While discussing the minimal encoding bias, Gruber stated that:

> *The conceptualization should be specified at the knowledge level without depending on a particular symbol-level encoding.*

Encoding bias is defined as the dependability of ontological statements that are made with regard to the symbolic level rather than the knowledge level. In other terms, when the ontological definitions are closer to an implementation detail than to the logical abstraction where they should stick. This often happens with quantitative qualifications such as when using formats of unit of measurement. An example would be describing the weight of something as *Number*. To satisfy the minimal encoding bias, we should describe the weight in *WeightQuantity*, where *WeightQuantity* is a concept that has a *WeightUnit* and a *Quantity* defined. Such generalizations although necessary and useful are not always possible to achieve, and sometimes, it is better, mainly for simplification purposes, to have some encoding bias. To get back to our example, most of the web ontology standards support the XML Schema datatypes [80], so we may use the "standard" definition of integer or float used by the XML Schema instead of the forementioned *Quantity*. This is why our objective is only to minimize the encoding bias instead of eliminating it. Ontology is a representation, and to guarantee the conversion of the ontology to other representation formats, the minimal encoding bias is an important principle to follow.

Minimal ontological commitment

The last principle is the minimal ontological commitment that states that the ontology detail level should not go deeper than what is necessary to

share that knowledge. Assumptions about the modeled world should also be kept to a minimum allowing others to extend or specialize other instances of the ontology with more freedom. An example would be specifying that some event has a t composed of a year, a month and a day without specifying if the date format would be ISO or US based. Gruber's explanation is:

> *Since ontological commitment is based on consistent use of vocabulary, ontological commitment can be minimized by specifying the weakest theory (allowing the most models) and defining only those terms that are essential to the communication of knowledge consistent with that theory.*

2.5.4 Ontology languages

There is a number of ontology languages that differ by their knowledge representation paradigm, their expressiveness or their decidability among other things. We will first discuss the taxonomies of ontology languages that we present here before introducing the most used and known ontology languages.

2.5.4.1 Ontology languages Taxonomies

The first categorization of ontology languages is by the knowledge representation paradigm. It can either be based on description logic, logic programming or no specific formalism. There are however works that try to combine description logic ontologies with logic programming-based rules for interoperability and to gain from the advantages of both worlds. An example is the Semantic Web Rule Language (SWRL) [54]. However, such amalgam leads to an undecidable formalism. A solution would be to restrict the description logic ontologies to what is called DL-safe rules [77] in such a way to preserve decidability. Another categorization for the ontology languages is their expressiveness. More expressive languages permit expressing more complex knowledge and allow more possibilities for inference. The cost is a higher computational complexity. Several languages offer different subsets with different degrees of expressiveness to serve a larger spectrum of needs.

2.5.4.2 Languages

The Resource Description Framework are a family of specifications by the W3C intended to describe and model resources. It has three funda-

mental concepts: resources, properties and statements. Resources represent the objects that are described and are identified by a unique URI. Properties describe relationships between resources and are themselves a type of resources with URIs. Statements are object-attribute-value triples used to assert the properties of resources where object represent a resource, the attribute a property and value either a resource or a literal. RDF has different serialization formats with the most used being the XML encoding and the N3 [12] encoding. RDF has a schema definition language called RDF Schema that can be used to create vocabularies used with RDF. RDF(S) is semantically considered between DL and LP in what is called description logic programs (DLP) [46] that is the intersection of DL and LP.

OWL is a family of languages that are built on top of RDF(S) as semantic web languages [88]. OWL has three different languages that differ in their expressiveness and hence, reasoning adequacy. These are OWL-Lite, OWL-DL and OWL-Full. OWL-Lite and OWL-DL are based on description logic and are decidable, whereas OWL-Full is much more expressive as to be compatible with RDF(S) and is undecidable. OWL can be serialized using RDF and/or XML.

WSML is another, more recent, family of semantic web languages [29]. It is composed of five languages that tries to represent the knowledge representation formalisms of first-order logic, description logic and logic programming. WSML is developed in the context of the Web Service Modeling Ontology (WSMO) [28] project as a formal semantic language. WSMO is not a pure ontology language a it offers several construct aimed at annotating Web Services and it comes with some domain ontologies that describe Web Services.

2.6 Summary

In this chapter we presented the knowledge representation problem and the different approaches to knowledge representation. For the purpose of this work we use a semantic representation of knowledge, mapped into a logical formalism. Specifically, we use the description logic fragment of the first-order logic. We use the OWL ontology language to model resources. OWL has DL and decidable subset called OWL-DL. Although other OWL sub-languages can be used if simplicity or expressiveness is

the main characteristic needed. We also presented a set of best-practices to be used to create ontologies. This step was significant in our work, as we had to define the design principles to be used in building our own ontologies.

Chapter 3

Service Orientation and Semantic Approach

3.1 Overview

We are going to present in this chapter the foundation of our work as well as present some design decisions. At the end of the chapter we will discuss some related work. We argue for the importance of having a service-oriented foundation and how resources can then be modeled as services. We will talk about the advantages incurred from augmenting the resources with semantic description and how this augmentation can be manifested. We had two important requirements for our solution, and these are the principles of simplicity and automation. Simplicity in modeling and deploying resources and automation in generating service representation of resources and in managing the resources. The discussion in this chapter is materialized in chapters 4 and 5.

3.2 Wisdom hierarchy

In designing our management model, we were inspired partially from the Wisdom Hierarchy, known also as the Knowledge Hierarchy or with the DIKW acronym. DIKW stands for Data, Information, Knowledge and Wisdom. This hierarchy presents and describes the relationship between these four concepts. Basically, that wisdom can be represented in terms of knowledge, knowledge in terms of information and information in terms of data. This concept is certainly not new and was mentioned in works from several fields such as in Information Science or in Knowledge Management. It was also represented in literature in early works such as T.S.

Eliot poem from 1934:

> Where is the wisdom we have lost in knowledge?
> Where is the knowledge we have lost in information?

This hierarchy was also used in IT context, although in a different way, by Jeffery in [58]. This hierarchy is not fixed and can have other elements added to it, such as "Understanding" or "Structured Data" (figure 3.1). Although, the definition of the hierarchy elements is universal, it can be tailored and extended depending on the domain or context.

Data, being at the first level, denotes raw facts taken as they are. From a computing perspective, data would be the bits and bytes for instance. In our model, data symbolizes the raw IT resources (here resource is to be taken its in large sense) with the particularity that resource "format" or formally, the data syntax, are multiple and do not conform to a grammar. *Structured Data* is then data that conforms to a specific structure described using a formal syntax. It is possible then to understand (for humans) or process (for computers) the data if its structure is known before hand. As an example, once the string `http://www.example.com:8080` is known to be a URL (that has a specific structure), it is easier to understand that that string has three parts delimited by "`://`" and "`:`". However the individual meaning of each of those elements cannot be derived solely from the string structure. Semantics are used to add meaning to things such that we would understand that the mentioned string is composed from a protocol, a hostname and a port number. By augmenting the structured data with semantics we obtain *information*. Information is the second level, it represents data processed in a way to be understood by a target audience. Thus moving from object reference to object sense (Sinn und Bedeutung as described by the philosopher Frege [40]). The third level, *Knowledge*, is hardly distinguishable from information at the technical level. It is, however, considered as a view or as an organization of information for a specific context or a specific purpose. Chapter 2 presented in details about Knowledge and *Knowledge Representation*. Applying knowledge to our string example we could *know* that it indeed addresses a service running on port `8080` on the host `www.example.com` using the HTTP protocol. *Wisdom* is the last level on this hierarchy, although some researchers argue for a later level denoted enlightenment, however, this view is not wide-spread. Wisdom is adding value to the information and knowledge acquired with a certain level of "personal" touch from the actor. To follow on our example, we could argue that we should not use HTTP because it is not secure and use the HTTPS protocol instead if it is available.

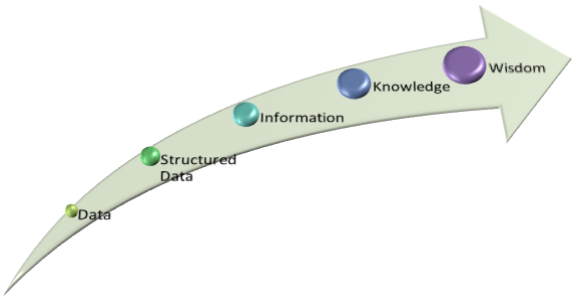

Figure 3.1: From Data to Wisdom

The DIKW represents layered prerequisites to attain wisdom. But what is wisdom in the context of IT and management of resources? And how is that relevant to this work? Wisdom is defined as the knowledge of the truth coupled with the proper judgment to act based on that truth. Projecting this definition to Information Technology management, a wise management system is an omnipotent system in the sense that it knows the system state (truth) that it is capable of reacting to change in the system with proper (i.e. compliant with the human administrator objectives) actions.

Using the DIKW, we built a layered conceptual model where data is the "raw" resources, syntax is a conceptual resource model, "linguistics" semantics are the IT-appropriated semantics, information is the structured resources that are semantically augmented and knowledge is the combination of information plus rules and predefined processes.

Another important issue is yet to be defined: what would be our conceptual model, or how would we organize the resource in an optimal way? In other terms we need to define the *ecosystem* of the resources or the System Organization.

3.3 System organization

Although we rely on intelligent agents for automation and dealing with system state exceptions, our system needs a certain level of conformity and defined expectation, especially while dealing with system components. Confronting the intelligent agents with an unknown system representation where resources may take unknown forms, use unknown communication protocols or present unknown control interfaces is a real chal-

lenge that is out of the scope of this work. We believe that such system configuration is too complex to be tackled by autonomous systems and we think that it is extremely difficult to come up with a system capable of navigating through such a configuration. We assume that all the system component have a certain degree of conformity making it (i.e. the system) a homogeneous and integrated system. This conformity is expressed at the level of component description and at the possible ways to interact with those components. This level of predictability is necessary whenever we deal with an automated or semi-automated management model, where several management tasks are offloaded to software agents.

Having this conformity of resources at the description level coincide with the service encapsulation that the service-orientation model follows. Where the service represents a single, contained and self-standing set of functionalities. In this model, everything is a service that shares similar communication protocols with all other services, allowing other software agents, services or not, to handle them in a predictable way. This important characteristic of service-orientation is an appealing argument that motivated the use of that model as our conceptual model. The service-orientation model is quite vague in its definitions leaving the room open for interpretation and assumptions related to many design principles. Although, it offers a lot of flexibility, this vagueness leads often to incompatible service-oriented architectures defying the whole purpose of the model. This is why we wanted to stick, as much as possible, to an abstraction level that would, at the same time, adhere to the guiding principles for a service-oriented system, while being extensible and compatible with the defacto standards used for creating service oriented systems and the design decisions that lead to them. This, in fact, justifies some design decisions such as the choice of WS*-[1] stack over REST [35] even if the latter is easier and has a more performant implementation than the first, because the first is indeed the defacto standard used in implementing service-oriented systems.

Even though modeling everything as a service would ease handling system components by software agents, these latter would still miss the semantics of the functionalities they are dealing with. In other terms, the software agents would have access to the "technical" description of the services and would be able to initiate and endure communication with the services, however, they will not be able to know what the service is actually doing, or to distinguish (or find, as a matter of redundancy support for instance) between different services supposedly doing the same thing.

[1] WS-* is used to refer to the collection of specifications related to the Web Services.

Another layer of information is needed to allow software agents to understand what they are dealing with in a manner close to what a human would do to apprehend the *usefulness* of a service. This layer is called the *knowledge layer*, and is described in more details in a following section. But, why would the software agents need an access to the knowledge layer? In fact, it is not that difficult to automatize a system, in a majority of cases it comes down to writing a set of scripts. The real value is coming from enabling software agents to become more than automated agents and be adaptive and even more: to behave in an autonomic manner.

The value of an autonomic system, defined as a self-governing system is not to replace the role of the human administrator from the landscape of system management, but to help in taking care of those tasks that can be anticipated and follow a proper protocol. The autonomic behavior is regulated using a set of policies that shape its actions and behavior. It has some requirements such as the use of small, manageable components that are well defined in a formal manner as to be comprehensible to the automation system. This definition should be independent of the implementation of those components for portability and integration with different management systems.

3.3.1 The case for a homogeneous and integrated system

Seeing the rapid change and development in technology, growing IT pools would be ultimately extended with systems different from the existing ones. The final result would be a system having elements with a large number of structural differences and variations. The variations that are most relevant for us are the way the elements communicate (communication protocols, data structures, calls interfaces... etc.) and the environment where they are supposed to live in (architectural variations, deployment variations... etc.). The more variations we have, the more difficult and complex the management would be. It is not possible to restraint the incoming elements to adhere to certain structural constraints in order to be used in the IT pool, because the functionalities needed may not have been developed using these constraints (and from-scratch design and development may not make sense), or simply because the new technology is better. Other reasons may exist, but we need to agree that the status-quo is that IT systems are by definition heterogeneous and this heterogeneity is deemed to grow and not shrink, and that sometimes it is not an option to restraint the IT system to a certain structural model. A solution would be to create a generic umbrella that encapsulates resources in a transpar-

ent way for the existing system infrastructure and allowing some proper services to handle them in a standard way. This can be done by creating a resource model, flexible enough to encompass a large span of resource types, yet concise as not be too generic and thus adding to the system complexity. The resources are described using a well formed language in such a way to present the resource characteristics, capabilities, interaction possibilities and management potential. Having a formal representation of the resources allows us the transformation of the representation of the resource to different formats depending on the needs or the context. Using the service-orientation model is appealing, however, SOA has some shortcomings that need to be dealt with beforehand.

3.3.2 Service-orientation (divide and conquer)

In the face of IT complexity, service-orientation is a well suited strategy to lower the complexity and simplify the management of IT by reorganizing the basic component of IT systems into small chunks with unified management interfaces. Resource's properties and capabilities (management capabilities or otherwise) are described using meta-data that is both accessible to applications and Software developers. A service representation in the form of a software component is derived from this description and used to interact with the resource through the advertised capabilities. Having this environment where resources are clearly partitioned and consistently represented is an efficient way with which we can lower the level of IT complexity and remove the complexity related to the heterogeneity of IT systems. The services as defined by the SOA specification [67] require, among other things, a service contract, which is a document that specifies an "agreement" on the details of the services as well as the communication mechanisms used. This document is written in a formal description language composed of two alphabets of symbols, an operational alphabet and a data alphabet as well as a set of structuring primitives and logic connectives [105]. Those elements are translated into an interface that represents the single interaction point with the service. Using this interface, we can access the internal state of the service and change it if need is to be. This document, referred to as the *description* of the service, is a public formulation of the characterization of the service [105] that allows developers to publish the characteristics of the service as well as its communication mechanisms without publishing any details on the implementation of the service. Services can hence be materialized for development testing by just having the service description alone. The description document acts as an

agreement or guarantees of how the service would really looks like even before it is implemented.

Although this document specifies how to access the service and hence the resource, it doesn't describe what the service is for. Semantic technologies can be used for this task by adding a meaning to the service description. In this work we present a model for describing resources semantically and generating a service representation from the description.

3.3.3 The need for the knowledge level

Describing the resource characteristics, capabilities, interaction possibilities and management potential in a formal language is not enough for other software components to figure out the exact purpose of the resource and the meaning of its capabilities. A language provides only syntactic level description that necessitates the intervention of humans to appreciate the value or the purpose of the resource. Hence the resources need to be augmented by a semantic description that would capture the forementioned meaning. A serious problem in today's IT management is the lack of information related to the managed resources [22]. This lack of information can be sensed at different levels which add to the complexity of the problem.

3.3.3.1 Effects of the lack of information

In what follows, we devised the areas at which such lack of information can hinder the IT management process.

At the level of IT system: There is a lack of information on what is installed and where it is installed. Often the purpose of the system itself is ambiguous if not unknown. Sometimes whole parts of the system that are unused are up and running and nobody can make a clear decision of the necessity of those components or if any other part of the system *may* depend on those components.

At the level of the resources relationships: Often the information of the relationships of resources to one another is missing. Information about why and how a set of resources is related is generally not documented, making failures analysis as well as problem source determination an extremely difficult tasks. Another problem that is also related to the first point is that with the lack of such information, predicting the impact of a

change in a system is nearly impossible, with cascading effects and lattent effects being the worse types of problems that may happen.

At the level of the resource itself: There is a lack of information about the resource itself, its purpose, capabilities and requirements. If this information is ever stored, it is done separately of the resource in external repositories that can get inconsistent if there is no system capable of reflecting into it the status of the resources in real-time. If such information is available, maintenance would be done in an easier manner as data about the resource would be available, which would allow making available the resource's needed ecosystem or changing it with an equivalent resource.

3.3.3.2 Affected tasks due to lack of information

All those problems affect considerably system management making it a tedious and an unpredictable process. If we take the ITIL classification for instance, lack of information affects all the processes used in managing IT. We take as illustrative examples the following processes:

Incident management. Incident management is the IT task that is concerned with restoring normal service operation as quickly as possible in the event of an *incident* and minimizing the effects on normal operations. An incident is defined as any event not part of the normal operations of a service that may lead to a halt of the service or a degradation of the quality of the service. Incidents are known problems or errors with a potentially identified root cause (root causes are dealt with in the problem management task). Different activities within incident management are affected by the lack of information, notably:

- Incident detection and recording
- Classification and initial support
- Investigation and diagnosis
- Resolution and recovery

Problem management. Problem management is the IT task related to the prevention of problems by identifying errors and determining the root cause of those errors. It is tightly related to incident management as it uses the incidents reports to identify the errors. Many activities are also affected by the lack of information, such as:

- Problem identification and recording
- Problem analysis
- Problem classification
- Error control

3.4 Semantically augmented resources

Describing resources semantically is crucial to understand what they are, what they do and how to interact with them. This semantic augmentation amounts to adding meta-data to resources as descriptive layer, thus publishing information about the resources in a standardized, machine-readable way. If we want management systems in our model to interact with the resources, we need more than the capacity to how to read information about the resource (syntax parsing), namely what does the information mean (semantics).

3.4.1 Benefiting from semantic augmentation at different layers

The use of semantics in our system model is perceived at three different layers: a *descriptive* layer, an *integrative* layer, and an *autonomic* layer.

Descriptive

Resources are described semantically to expose their properties, capabilities, and management potential. The semantic description is used as a starting point to generate resource representations to be used by specialized components for interaction purposes.

Integrative

Semantic data provides the meaning of resource characteristics and thus specialized components can use newly encountered resources if the semantic description of their capabilities matches the components needs for instance.

Autonomic

Semantic data can be used by autonomic agents for resource management by inferring the course of action based on the semantic resource description. This description coupled with semantic policies permits to achieve the autonomic level for management

3.4.2 Design issues

We rely on knowledge representation techniques to describe this semantic augmentation. We especially rely on the reasoning techniques over a set of knowledge and the the formal languages developed to capture this knowledge. There are different important issues that we need to tackle before creating our framework, they amount to different architectural and modeling decisions and those are the augmentation model (see section 3.5), the description language, and the description model (see section 3.7).

3.5 Capturing the semantic data

An important question that rises when talking about the knowledge level is how to associate the semantic description with the service representing the resource. We present in this section a discussion and taxonomies for the semantic augmentation of services. A presentation of the different research in this area is presented in the related work section 3.8. We define three taxonomies for augmenting services with semantic data: by abstraction level, by the semantics used to capture the service definition and by the the service definition source for implementing the augmentation.

3.5.1 Service definition abstraction level

The first taxonomy is the service definition abstraction level, meaning, at what abstraction level the definition of the semantic representation should occur. The representation can be *transparent*, meaning that it is not visible to the service nor to its ecosystem making the best solution if backward compatibility is a requirement, because a transparent semantic representation would allow the service to be deployed and run on traditional systems that do not inherently support semantics.

In this model, the description has to be represented externally and independently on the service and the service description. The semantic representation can also be *articulate*, i.e. visible and possibly required by the

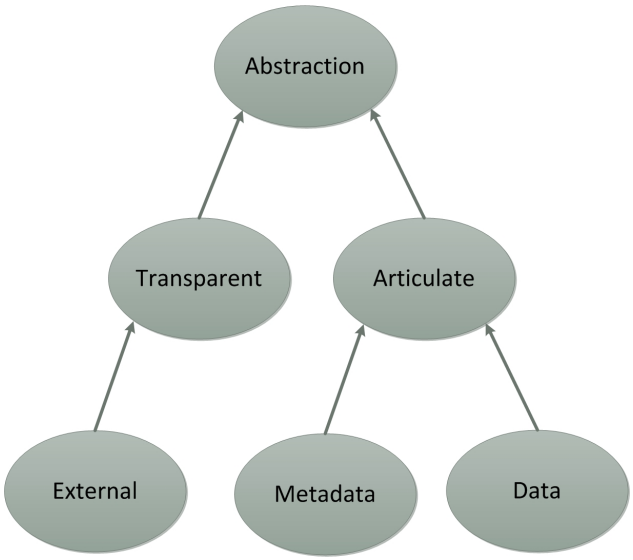

Figure 3.2: Abstraction Taxonomy

service and its ecosystem, making the deployment and running of the service dependent on the presence of the semantic description. This description in the latter case can be implemented at the level of *metadata* or at the level of the *data*. At the level of metadata the semantic description is represented en par with the service description document either in a combined manner using extensibility elements of the service description document or external to the description document and this latter having references inside of it to the semantic description document. At the data level, the semantic description is implemented at the level of the service implementation through an API that would expose the service concepts. Figure 3.2 shows the first taxonomy that we just discussed.

3.5.2 Service description development process

A second classification relates to the service description and what information does it contain. The service description can either be *contract* based, *semantic* based or both. In contract based description, the description document is written in a classical service description language such as WSDL, whereas in semantic based description, the description is written in a logic based language. The description can also be written using both approaches, either inline, where semantic data is added inline to the description document such as in WSDL-S [2], or completely in parallel, where both descriptions could be transparent to each other.

3.5.3 Service definition original source

The third classification relates to the implementation of the augmentation. As we discussed earlier, the resource can have a service description outlined in the service contract as well as a representation as a semantic description. If we suppose that the information contained in the service contract definition is the same as the information contained in the semantic description, then it is possible to recreate either document from the other. In case one representation is not complete, then this conversion can only be one way. The resource description reference is about which representation(s) describe completely the resource, and hence, which one is essential and what are the conversion directions that we may have.

Implementations can be semantic-based, where the complete description of the resource is defined in the semantic representation. The service contract can then be generated from this semantic representation and it may or may not contain the complete description of the resource as detailed in the semantic representation. As an example, consider a semantic description containing information about the owners of a resource. This information, because it is optional for the service contract, could be omitted from the contract and we would still have a functional service contract but an incomplete representation of the resource. Implementation can also be contract-based, where the contract hold all the necessary information to access the resource and the semantic representation is generated from this contract and it contains just a subset of the information like the capabilities and properties of the resource. The third way would be that both representations are equivalent and one document can be generated from the other.

3.5.4 Used approach

In our framework we use a transparent semantic representation where the description is represented externally and in parallel to the service description. The service definition source is captured in the semantic document. Using this configuration, our approach has the following advantages:

Single, compact reference source

All the resource description is stored in a single[2], compact file, that the framework use to derive the service contract and thus the service stub (or

[2]Although the description can span multiple files and use importing mechanisms, it is considered at the end as a single source file.

service implementation) and the client proxy among other things. Those artifacts and files can be re-generated at will from the semantic description, minimizing greatly the size and amount of documents describing the resource and making the transfer of the description of the resource over the network quicker. Having also a single point of reference for the specification enables consistency for the resource descriptions.

Multiple representations

Using a semantic description allows to derive more than just the service description that the service contract is supposed to achieve. The resource semantic description can be used to generate as much descriptions as semantic technology allows. Straight-forward examples would be generating a graphical representation of the resource and domain specific documentation such as security requirements.

Stricter formalism

Using logic allows to have higher expressiveness and better precision than using XML and WSDL to describe the resource. We discussed the general advantages of using logic in the previous chapter.

Backward compatibility

Having an external representation transparent to the WSDL representation allows to have backward compatibility with traditional service containers as the WSDL would be fully compatible with their requirements. This is a very important and desirable feature because our solution promotes integration with legacy systems for better transition to a fully manageable system.

3.6 Managing resources

One important principle behind the creation of Web services was for application integration, including legacy applications that were written using heterogeneous technologies and platforms. This was also the case for management applications that not only had to deal with heterogeneous resources, were themselves not inter-operable.

3.6.1 Managing resources as services

Using Web services principles, it is possible for management applications to use common messaging protocols between themselves and the manageable resources, thus becoming vendor-neutral and platform-independent solutions [16]. In the WS-* landscape, there are two important and competing sets of specifications that describe management messaging protocols on top of the Web service stack, namely WSDM [16, 109, 17, 18] and WS-Man [73]. There is an ongoing effort to merge the two specifications [4] as conformity and standardization was a key objective in the design of both specifications. Both specifications have a resource centric view of management, where the common messaging protocol is used to communicate directly with the resource through a standardized interface. Contrast this with the traditional model where management applications were often contacting agents on behalf of the resources. This standard interface is the interface of the Web service that represents the resources. In other words, resources can be accessed only through the End Point Reference of the Web service.

3.6.2 Representing resources as services

There is nothing special about representing a resource using a Web service, if that resource is already offering some API to access its capabilities and properties. It would be a matter of writing an interface to this API, accessible through a well-defined Web service. However there are some issues related to the intrinsic nature of Web services and resources. The most prominent difference is the fact that Web services are stateless, meaning that they do not keep data (a state) between different invocations. This contradicts the view of the physical resource that keeps a state during its lifetime. This stateless characteristic of Web services is not a limitation as it was a design choice aiming at the Web services being light-weight Software components. There are mechanisms that permit to emulate a statefull behavior for Web services, with the most traditional being session cookies or using persistent storage of state like a database or using WS-Session.

WSRF [6] proposes an elegant and integrated solution to the stateless issue of Web services by using descriptive document called *ResourceProperties* and introducing the concept of *WS-Resource*. The WSRF provides a general solution using Web services to an originally specific problem: describing and representing Grid resources that are statefull in nature. Another relevant feature of WSRF is that it brings a solution for management of a resource lifetime, faults and properties. Rendering resources as WS-

Resource and decomposing software components into services is the first step towards a SOA-enabled management architecture with all the advantages that it can bring such as ease of management, adaptability and automation.

3.7 Modeling resources

We cannot describe what we don't know; therefore, it is important to have a clear definition of the basic constituent of our system, which is a resource. We define a resource as:

> *a logical representation of an IT entity that is defined by its state, properties and capabilities and that can be further abstracted by representations allowing different kinds of interactions.*

IT entity is to be taken in its large definition; it encompasses physical resources (e.g. servers or printers), software and services and can be extended to a certain degree to represent the human actor. The properties represent the attributes and characteristics of the resource and can be subdivided into immutable properties that are fixed during the resource lifetime, and mutable properties that describe the resource state at any given moment, hence the definition of state as being a snapshot of the resource mutable properties. Resource capabilities represent the capacity of the resource to undergo a change or to affect its environment and are accessible through defined interfaces.

Creating a resource model allows the definition and conceptualization of an IT resource and its components. This degree of formalism is needed when we intend for these resources to be used in an autonomic environment.

3.7.1 Resource concepts

Resource concepts are the different constituents that define or relate essentially to a *resource* as defined previously. Due to the nature of the IT resources, the concepts are not related to resource instances but to resource classes. We would have for instance a *class* of printers sharing the same concepts but differentiated by the state or values of the properties of those concepts, both identifiers or non-identifiers concepts. So two printers can be from the same class but may have a different number of papers in the

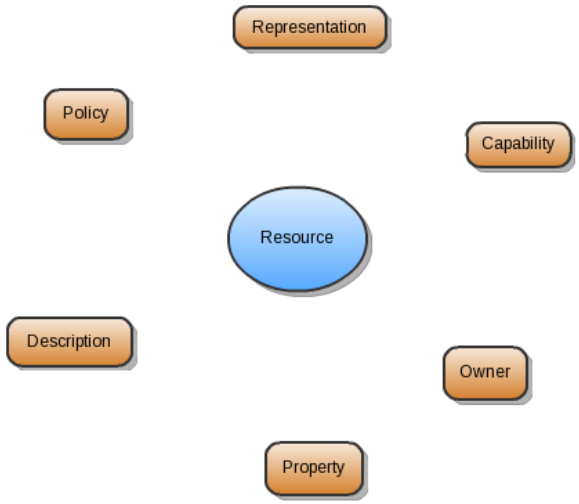

Figure 3.3: Resource concepts

tray and would have different serial numbers or MAC addresses for instances. The same example could be stated on computer programs or software components. Therefore, unless specified otherwise, when we talk about a *resource*, we imply a class of resources instead of a resource instance.

The resource model is composed of different concepts that are shown in figure 3.3. Those concepts are described in more detail in the following subsections.

3.7.1.1 Property

A property is one of three concepts that define a resource. It is an essential attribute of a resource and can be a resource by itself. Properties should be *typed*, limiting the range and the format of the values that they can hold as well as the operations on those values. Having a typed property will facilitate the detection of warnings, errors and exceptions (a printer's number of papers in a tray approaching zero vs. a negative number of papers). Resource properties can be *required* or *optional*, and *mutable* or *immutable*. The immutable properties are those that are fixed during the lifetime of the resource, and mutable properties are variable properties that can hold different values. There must be at least one required and immutable property that should be used to uniquely identify a resource instance.

3.7.1.2 State

The resource state is the second concept that defines a resource and is exclusively related to the resource instance, i.e. resource classes have no state. The resource state is a snapshot of the resource mutable properties. It is a view describing the resource in regards to its attributes. In the printer example that we mentioned earlier, the number of papers is a mutable property that constitutes, along with other mutable properties, the state of a specific printer identified by at least one immutable property.

3.7.1.3 Capability

Capability is the last concept that defines a resource. Resource capabilities represent the capacity of the resource to undergo a change, initiate a change in another resource, or affect its environment. A capability should be sound with a defined result. The requirements of the capability (modifiers, parameters,...) as well as the capability result should be well-defined in an *interface*. Using again the printer example, printing would be the most desirable capability of a printer with a well-defined result: printed documents (or a well-defined error code).

3.7.1.4 Representation

A resource representation is a presentation and/or abstraction of a resource using a formal method for the purpose of allowing different types of interaction with the resource or allowing apprehension of the resource at different levels. A representation can be complete or partial. A resource can have multiple representations. A service contract for instance, is a partial representation of the resource for the purpose of defining the service interface and communication mechanism.

3.7.1.5 Description

Description is the information associated with the resource in order to use that resource and/or document the resource. The description contains a component that is for human what representation is for machines, a tangible presentation and/or abstraction of a resource using a formal, semi-formal or even an informal, subjective description. The description also contains components describing the access methodology to the resource (a resource identifier's description is part off the resource description).

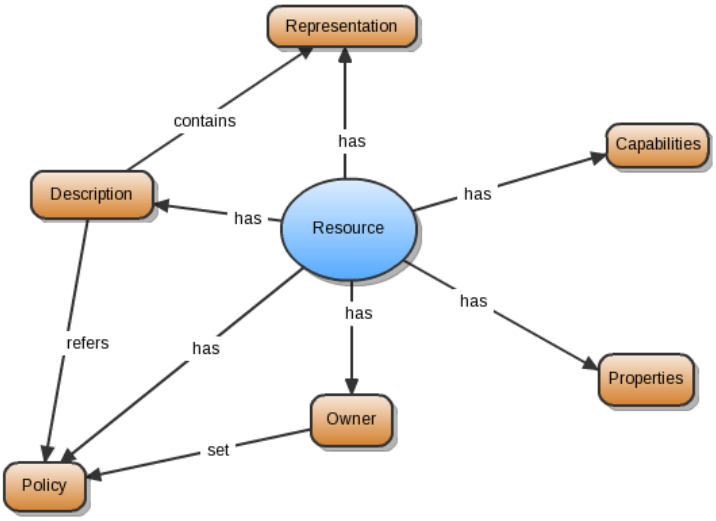

Figure 3.4: Resource concepts relationships

3.7.1.6 Policy

The policy is a set of constraints associated with the resource. They are put in place by the resource owner to specify conditions related to the use of the resource, such as security requirements, privacy, service level agreements and so on.

3.7.1.7 Owner

The owner of the resource is the person/entity that owns or has the physical/legal responsibility of the resource. The resource inherits the security credentials of its owner and this latter can set further restrictions/permission in the policy file.

3.7.2 Relationship between resources, concepts and service representation

Figure 3.4 shows a diagram with the relationships among resource concepts. We note that the policy is set by the owner and that the description of the resource is a general concept that refers to the policy of the resource and may contain different representations of the resource.

We aim at providing a framework to transform the resource into a service. The service, as defined by [67] has also some concepts that define

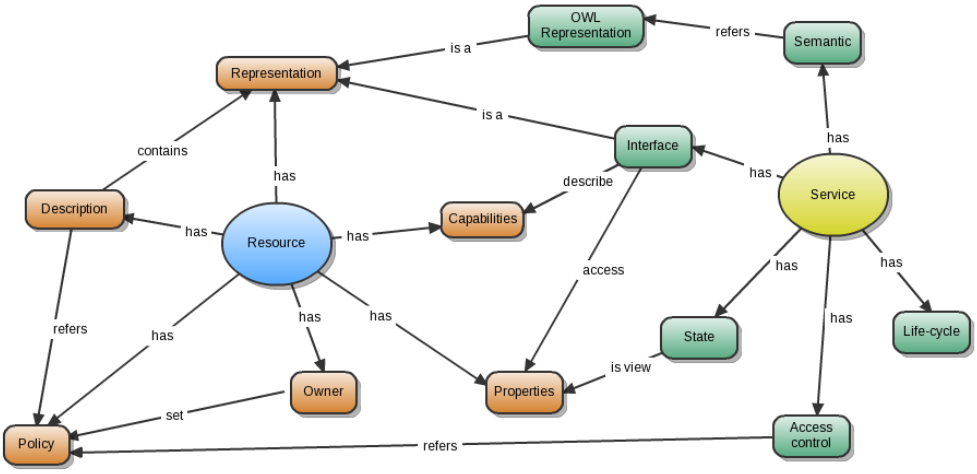

Figure 3.5: Resource and service concepts relationships

it. Most notable for our framework are the state, interface, life-cycle and access control. We add to those concepts the semantic description of the service. Figure 3.5 shows the connections between the resource concepts and the service concepts.

3.7.2.1 Service interface

The service interface is a representation of the resource. It describes the resource capabilities and provides access methods to the resource properties. It should be the sole access point to interact with the resource. Although the interface is a representation of the resource, it is not a complete representation i.e. a lossy representation as we cannot reproduce the full definition of the resource from its service representation.

3.7.2.2 Service semantic

Service semantics describes the service representation of the resource and is based on the OWL representation of the resource that is intended to be complete. In other terms, it is possible to construct either the service or the resource description from just the OWL representation.

3.7.3 Resource ontology

The resource ontology has two important objectives. First, to capture all the knowledge about a manageable resource as described in the previous sections, and second, to be used as a required template that resource instantiators have to rely upon when creating resource descriptions. The reason behind the requirement that the resource descriptions should be based on this template is for standardization and for coherence purposes, in the sense that components that are supposed to interact with those resources can expect a minimum set of elements, that are basic, yet sufficient to interact with the resources. We followed the principles outlined in section 2.5 for creating this ontology. We wanted the resource ontology to be as abstract as possible to support extensions easily. However, if they deem it necessary, system administrator can still use another ontology with the condition of supplying an implementation of the semantic parsing and vocabulary via Java classes. We should also note here that the following ontology is not a one-to-one translation of the above mentioned figures but contains the relevant concepts with addition to other necessary concepts for the framework. We would add new classes to the ontology as we come across new functionalities in the following sections.

3.7.3.1 Ontology classes, properties and individuals

Figure 3.6 shows an inheritance tree view of the different classes used in the implementation of our framework. What follows are descriptions of each of the classes:

AbstractResource

This class, as its name suggests, represents an abstract resource. All the resource classes and instances should be a subclass of this class and thus inherits from this class. The abstract resource has properties, capabilities, description, owner and representation as figure 3.3 shows.

ResourceId

This class represents a unique identifier for the resource.

ManagedResource

This class represents a resource with management capabilities. It is a subclass of *AbstractResource*.

Property

Every resource property is of type *Property*. A property has a name and a type.

Capability

This class represents a resource capability. It has a name, a type and an implementation.

ManagementCapability

The capabilities that are used to manage the resource should be of type *ManagementCapability*. This class is a subclass of *Capability*.

BuiltInManagementCapability

Many management capabilities are shared among different resources and across classes of resources. Some of those are part of the framework and can be used when needed. Their implementation is generally also available and added when building the service.

Parameter

Resource capabilities can have parameters represented by the *Parameter* class. Every parameter has a name and a type.

Description

Represents the resource description.

AbstractPolicy

Superclass for a policy class that follows a specific protocol.

Owner

Represents the owner of the resource.

Ontology classes are not enough to describe a model. A set of classes' properties is also necessary to describe the characteristics of the classes and their relationship among each other. Figure 3.7 shows the different

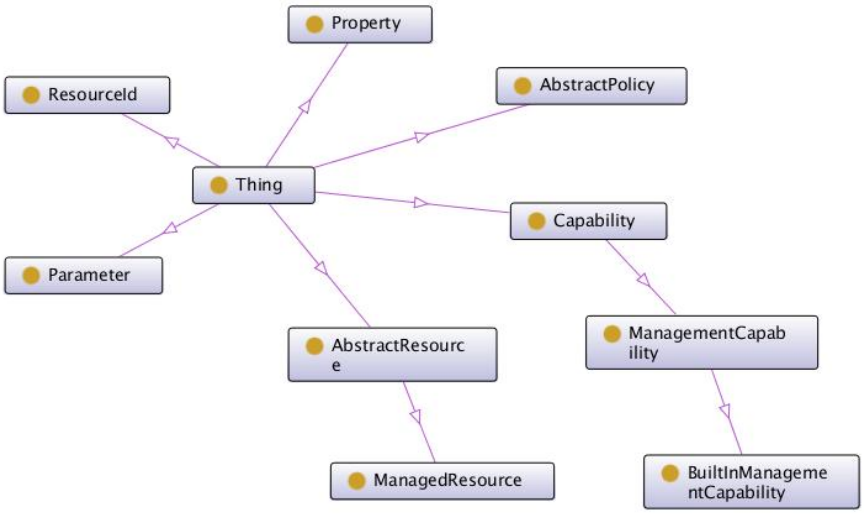

Figure 3.6: AbstractResource Ontology

object properties used in our implementation. There exist also other data properties associated with some classes and not shown in the figure such as the capability data type or its implementation class. The built-in capabilities can also be defined at this level, and for that, an individual needs to be instantiated from the *BuiltInManagementCapability* class and, if need there is, one or more parameter individuals that are needed by the capability. Figure 3.8 shows an example of a built-in capability *SimpleMetaDataExchange* along with its needed parameter *GetMetadataMsg*.

3.8 Related work

3.8.1 Semantic Web Services

There are numerous works done in the field of semantic Web Services, such as WSDL-S [2] that tries to extend WSDL by adding some semantic annotation to the file as XML tags that can reference entities in models outside the WSDL document. WSDL-S builds on the establishment of WSDL as a service description language for ease of migration. WSDL-S is intended to be a minimalist approach that tries to extend the preexisting Web Services with semantic annotation, which is quite different from the other methods that try to create a more complete, sometimes complex framework. Another effort is the Web Service Modeling Ontol-

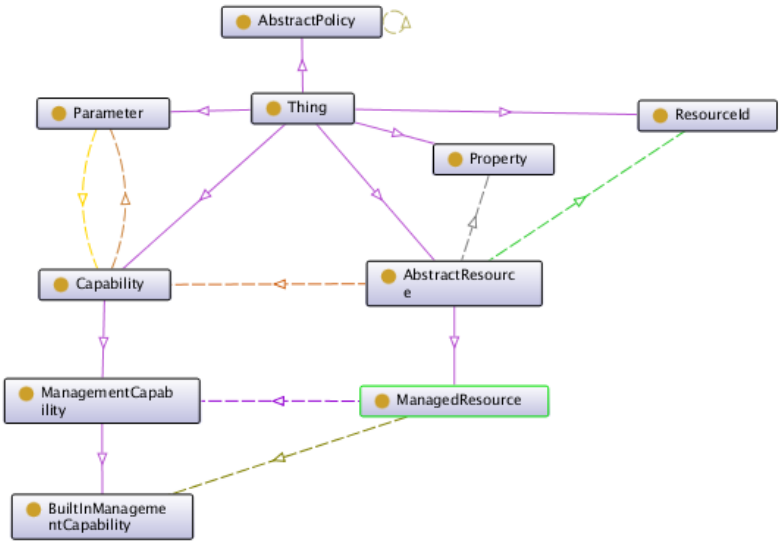

Figure 3.7: Resource classes with properties

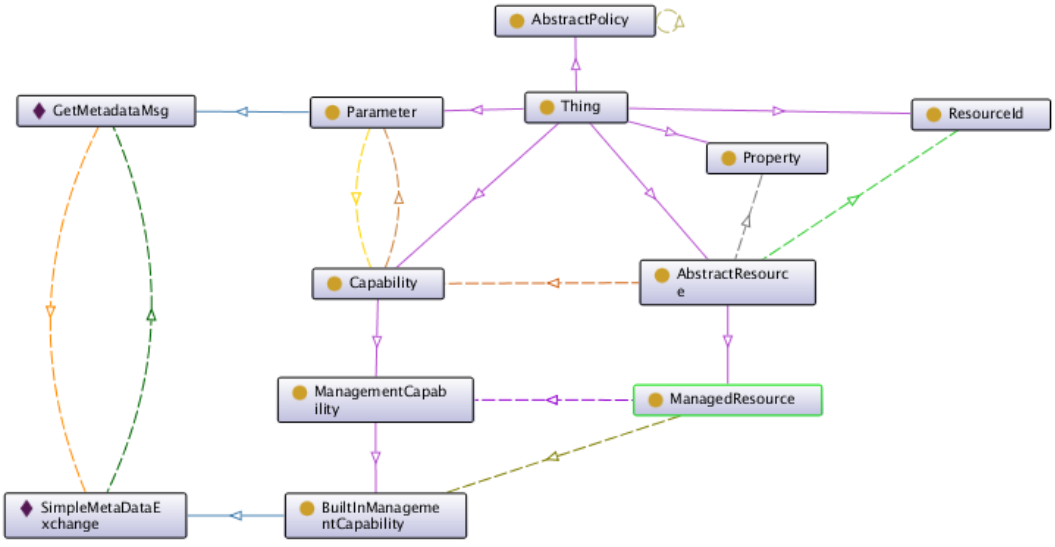

Figure 3.8: Example of a built-in capability

ogy (WSMO) that tries to describe all aspects of semantic web services with the goal of automating the discovery, selection, composition, execution and other tasks related to the integration of web services. WSMO is based on the Web Service Modeling Framework (WSMF) [34] and has three working groups: WSMO [28], Web Service Modeling Language (WSML) [29] that represents a family of languages used as representation format for WSMO and Web Service Execution Environment (WSMX) [75] that acts as a reference implementation of WSMO. WSMO is composed of four elements: ontologies that define the common representation of information, web services that represent the services, goals that describes aspects of the requests to web services and finally mediators that act like connector between the different layers of WSMO. The major drawbacks of WSMO are that it is a quite complex framework that uses proprietary technologies at almost every level. It does not use WSDL, but instead a new representation formalism, it ignores UDDI for its own solution, and uses a family of languages that are not XML conform and are meant to be replacements to the already established languages such as OWL and RDF. Another work is OWL-S [71], an effort to define an ontology for semantic markup of Web Services. OWL-S was meant to be a replacement to the WSDL, however this effort was not successful. Other works on semantic web services worth mentioning here are IRS-II [78], Meteor-S [104] and SWSF [7, 8, 9].

3.8.2 Rules and policy languages

There are quite a number of rules and policy languages developed either in academia or industry. Of interest, we mention Ponder [25, 66, 31], a hierarchical policy language that tries to support a wide range of policy types by providing an extension mechanism based on its hierarchical design. Ponder supports authorization, obligation, delegation and management policies. Another interesting policy language is Rei [60], a Prolog-based language that uses predicates to formalize rules, hence its support for reasoning. It uses OWL-Lite with the possibility of combining it with SWRL[54]. Rei also provides a simple protocol for negotiating between the policy system and the managed system. One shortcoming of Rei is its lack of events or triggered rules. The third related work is WS-Policy [102, 103], a specification meant to add the support to web services the ability to advertise their policy in XML, but also for the service consumers to enforce security requirements. IT has some related specification such as WS-PolicyAttachment [101] that is used to add the policies to WSDL and

UDDI or WS-SecurityPolicy [79] that is used to specify security assertions for other security-related WS-* stack specifications. Another interesting domain-specific work is the Rule-Based Service Level Agreements (RB-SLA) [87, 86], a declarative rule based approach for SLA representation and service level management of services. Other works also of interest are the eXtensible Access Control Modeling Language (XACML) [32], an OASIS standard used specifically for security and privacy policy and KAoS [100] a description-logic approach to policy representation.

3.8.3 Autonomic Computing

The autonomic computing vision had a high impact across the IT spectrum with especially companies building autonomic systems and incorporating them in their product lines. IBM started with two prototypes to validates their architectural ideas [108]. The prototypes [30, 23] explore the use of autonomic systems for data center management and resource allocation with applications representing the autonomic elements. IBM has then included autonomic computing capabilities in more than 50 of its products [41] including some of its most important ones: Tivoli, WebSphere and DB2. Other interesting work from IBM is the Policy Management for Autonomic Computing (PMAC) which, when integrated with applications can add support for self-management by evaluating policies and commanding the application. From Intel, we note the Autonomic Platform Research (APR) that aims at building an autonomic platform composed of hardware sensors as well as ambient sensors to be installed in the data centers, and an autonomic and policy manager. Oracle had also introduced since the version 10g of its database what is called the Autonomic Workload Repository (AWR) that is used to automate many aspects of the database in an autonomic manner. In the research community we note the work in [106] that deals with resource allocation in an autonomic data center with the objective of maximizing a specific utility function. In [21], the authors present a work for resource management in a hosting center with an emphasis on energy efficiency by moving application based on negotiated SLAs. Another "competing" approach to the autonomic computing but with the same objectives is presented in [5].

3.9 Summary

We are using service-orientation principles for our management model to be able to organize the managed elements into small, homogeneous com-

ponents with a defined access interface and communication protocol. We augment these components with a semantic description to a add a knowledge layer that allows handling them in a more autonomic manner and to tackle the issue of the lack of information related to the managed resources. Our solution follows the principles of simplicity and automation. Simplicity in modeling and deploying resources and automation in generating service representation of resources and in managing the resources. We also presented a resource model to be used as a basis for modeling resources using a semantic description and generating service representations.

Chapter 4

Managed Resource Framework

4.1 Overview

As mentioned in section 3.7.1, resources may have different representations for different purposes. This chapter discusses the *Managed Resource Framework (MRF)*, a framework for automatically generating ready-to-deploy service representations of resources from their semantic representations. The objective is to have a computer aided process by which resources can be rapidly *instantiated,* deployed and managed in a relatively quick and transparent manner for the user.

4.2 From semantic representation to service representation

The framework assumes the existence of a semantic representation of a resource written in OWL that extends an *AbstractManagedResource* as described in chapter 3 and outputs a deployable service representation called *Managed Resource Archive (MRA)*. The only "human" intervention during this process would be necessary if there were custom capabilities defined in the semantic representation and that lack an implementation (see figure 4.1).

Using the Managed Resource Framework, it is possible to generate resource artifacts that would eventually constitute the Managed Resource Archive. The MRF assumes a target-based mechanism by which, only one or several constituents of the MRA can be generated as needed instead of the monolithic MRA, depending on the specified target.

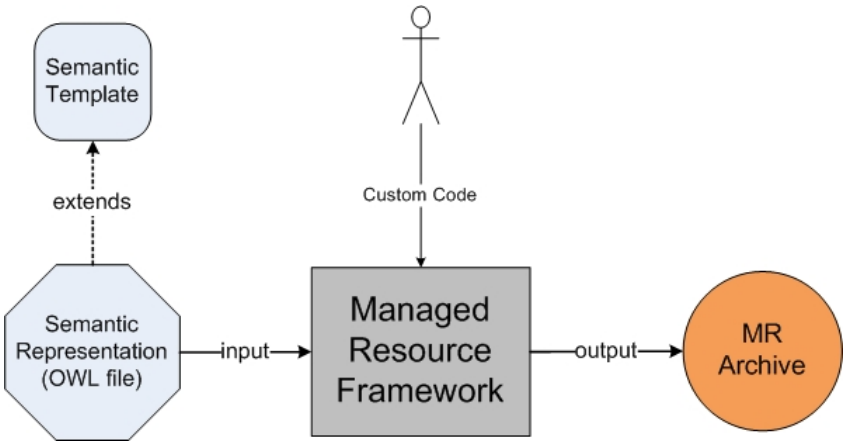

Figure 4.1: Managed Resource Framework (MRF) input/output

4.3 Automatic programming

The MRA files and their constituents are generated using *Automatic Programming* techniques. Automatic programming, also called *Generative Programming*, is the mechanism by which source code and other files and artifacts are automatically generated by a computer following a set of specifications. Automatic programming is mainly used to increase the developer productivity and ease the development of systems; it is also used for code reuse or component-based programming. There are two types of automatic programming, *inductive* and *deductive* [69]. Inductive automatic programming generates programs from a set of instances and other programs, whereas deductive automatic programming generates programs from a high-level description. Both methods have the objective of *program synthesis*, which is the derivation of a program to meet a given specification. The specification generally describes the relationship between the input and the output without necessarily hinting on methods to use to achieve the output from the input. The MRF uses a simple deductive-based methodology to generate the program for the Managed Resource (i.e. service representation) as well as other support programs (proxy, libs... etc) that are not intended to be runnable and other resource files and artifacts.

4.4 Managed Resource Archive

The Managed Resource Archive is a deployable service representation of the resource that is generated by the Managed Resource Framework. The

MRA is a Web application formed by a bundle of servlets, classes and other resources and intended to run on Managed Resource Containers or on Java Servlet Containers with, however, a loss of capabilities.

Once deployed, every MRA has a unique URI represented by its location in the container. An example would be `http://www.example.com/site1/res_A42`. Requests to the resource would have to start with that URL as a prefix. Every request is then forwarded to the `ServletContext` representing the resource. The `ServletContext` is an interface that defines a servlet's view of the Web application. It is up to the container provider to provide an implementation to the `ServletContext` and to ensure the one to one correspondence between every resource and a single `ServletContext`. A Web application may consist of servlets, utility Java Classes, static documents, client-side Java applets, beans, and classes and descriptive meta information. In this context, the MRA is composed of the following elements that are described in details in the following section:

- Service Representation: Servlet
- Service Stub
- Proxy Library: Java Classes
- Proxy Source: Java source files
- HTML Documentation
- Service Contract: WSDL file(s)
- Semantic Representation: OWL file(s)

The process by which a resource is directly accessed by some software agent is described bellow:

1. A resource capability is invoked
2. The resource client proxy constructs the HTTP request
3. The HTTP request is sent to the service container
4. From the request URL and the services configurations, the container identifies the concerned resource, builds a request and response objects and invokes the appropriate capability using those objects.
5. The response object is filled with the capability result and sent back to the client using the HTTP protocol.

6. At the client side, the Java object representing the result is created from the response and returned as the result of the call.

4.4.1 Constituents

4.4.1.1 Service Representation

The central component of the MRA is the service representation which is compliant with Java Servlet specification version 2.5 [76]. Servlets are platform-independent Java classes that are compiled to platform-neutral byte code and are intended to run inside servlets engines. The servlets engines are Web containers that are generally attached to, or are extensions to Web servers. Servlets are used to generate dynamic content, and can be loaded/unloaded at run-time into the Web containers. Servlets interact with external applications via a request/response paradigm implemented by the servlet container [76].

The service representation can be thought of as a lossy encoding of the semantic representation of the resource in the Java language. It has an implementation of all the resource capabilities or a call to an external library that holds all or parts of the capabilities implementation. The service implementation provides also methods to access or modify the properties of the resource.

4.4.1.2 Service Stub

The service stub is the source code of the service representation (i.e. service implementation) that is also included in the archive[1]. The implementation of the custom capabilities of the resource can thus be modified and the service recompiled and redeployed.

4.4.1.3 Proxy Library

The proxy library constitutes all the necessary libraries and APIs needed to access the resource, its properties and its capabilities. The proxy library is available at the client side and is linked to the application accessing the service. The communication details, marshaling and unmarshaling of the messages is hence transparent to the developer. Calling remotely a certain resource capability can be done by including the resource proxy library

[1] In the implementation of the MRF framework, a flag can be turned on if the users are not interested in including the service source code in the archive.

and calling the capability as if it is a function to a locally accessible class. The following code snapshot demonstrates how this is actually done:

Listing 4.1: An example of using the proxy library

```
import com.example.site1.res_A42.proxy...
class ResourceConsumerExample {
    MyResource a = new MyResource();
    a.getCapabilityXYZ();
    //...
}
```

4.4.1.4 Proxy Source

The proxy source contains all the source code for the proxy. This can be used and adapted by the developers to fit their particular needs, such as changing a communication library that is based on a license would not fit the application they are developing.

4.4.1.5 HTML Documentation

The HTML documentation is intended solely to the users and developers and contains information about the resource, its properties and its capabilities as well as on how to access it. It contains also the Java documentation of the service representation and the proxy. It offers a user friendly interface to download selectively the different resource artifacts. The HTML documentation is intended as a starting point for administrators and developers with a human understandable description of the resource as well as code examples on how to interact with the resource. All of those elements are of course automatically generated and do not need any action from the user generating the MRAs.

4.4.1.6 Service Contract

The service contract is a central element necessary to generating most of the other elements such as the service representation and the service stub. It is an XML file written using the Web Service Description Language (WSDL) and is generated from the semantic representation. The service contract follows the W3C specification for WSDL 1.1 [24]. Once the service is deployed, the service contract is published and can be accessed by third parties if they are interested in developing their own clients that are

different from the one provided with the archive or if they want to use another language than Java that is used for the clients.

4.4.1.7 Semantic Representation

The semantic representation is also included and published in the archive. It can be accessed by third-party applications interested in accessing the semantic representation of the resource for any reason they see fit.

4.4.2 Directory Structure

As specified in [76], the MRA follows a specific directory structure to be compatible with servlet containers. The MRA archive's root is the document root for the files that are part of the Web application. All the files that should not be in the public document tree of the application should sit under the `WEB-INF/` directory. Anything under `WEB-INF/` cannot be served by the Web container, however, it is accessible to the servlet code through method calls and can even be programatically exposed. The contents of the MRA archive is the following:

- The `/WEB-INF` directory as described above.
  - The `/WEB-INF/web.xml` deployment descriptor.
  - The `/WEB-INF/classes/` directory.
  - The `/WEB-INF/lib/` directory.
- The `/META-INF` directory
- The HTML documentation that is served if the service is pointed to using an HTML browser.

The deployment descriptor is a required configuration file written in XML and that defines among other things, the initialization parameters of the servlet, the class mappings, welcome files and error pages. The following listing shows a typical deployment descriptor file as generated by the MRF:

Listing 4.2: An example of a generated deployment descriptor

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app id="WebApp_ID" version="2.4"
3              xmlns="http://java.sun.com/xml/ns/j2ee"
```

```
4               xmlns:xsi="http://www.w3.org/2001/XMLSchema-
                    instance"
5               xsi:schemaLocation="http://java.sun.com/xml/
                    ns/j2ee http://java.sun.com/xml/ns/j2ee/
                    web-app_2_4.xsd">
6           <display-name>Managed Resource Servlet</display-
                name>
7           <context-param>
8                   <param-name>MRFversion</param-name>
9                   <param-value>0.9</param-value>
10          </context-param>
11          <servlet>
12                  <display-name>Managed Resource Servlet</
                        display-name>
13                  <servlet-name>ManagedResourceServlet</
                        servlet-name>
14                  <servlet-class>org.apache.muse.core.
                        platform.mini.MiniServlet</servlet-
                        class>
15          </servlet>
16          <servlet-mapping>
17                  <servlet-name>ManagedRersourceServlet</
                        servlet-name>
18                  <url-pattern>/*</url-pattern>
19          </servlet-mapping>
20          <welcome-file-list>
21                  <welcome-file>index.html</welcome-file>
22          </welcome-file-list>
23          <error-page>
24                  <error-code>404</error-code>
25                  <location>/404.html</location>
26          </error-page>
27  </web-app>
```

The `classes` directory contains the compiled service implementation along with helper and utility classes. All those classes are generated by the MRF and in a typical use case, no additional class is necessary as additional functionality should go in the `lib` directory. The `lib` directory contains the libraries used by the service classes in JAR format. At build time, the MRF takes care of putting all the required libraries in that directory. The user can provide an additional list of libraries to be included also.

The `/META-INF` directory contains information used by the Java archive tools and is not considered as web archive (WAR) content and thus will not be served in response to a request and 404 error code will be returned instead.

The HTML documentation is generated automatically by the MRF and provides useful information for the developers intending to use the resources. It also provides relative, but constant links to the different artifacts and libraries that are required to consume the resource. Giving the URI of the resource, automatic agents can find those artifacts and libraries at expected locations. We should note here that the MRF can also automatically generate management capabilities that would serve those artifacts programatically to the software agents requesting them, however, it is not possible to contact the resources if the agents do not have the resource client library or at least its service contract. Accessing them through known URLs using the HTTP protocol is thus the way to be able to access the other artifacts and capabilities of the managed resource. The HTML documentation provide three main sections, first, the *Resource Description* section, that provides a detailed description of the resource, its properties and its capabilities. For every capability it documents the URI, operation name, implementation class, return type and parameters. It also distinguishes between the user defined capabilities and the imported capabilities that are standardized, resources-wide capabilities that the MRF also provides. Second, the *Client Access* section documents the resource contract, the client proxy and an automatically generated code sample that shows how the developer can access the resource programatically. The last section is the *Downloads*, that provides permanent links to download the resource artifacts and libraries.

The MRA archive format follows the Java archive specification as defined in the JAR File specification [98]. The MRA archive is a ZIP file with the additional manifest file located in `META-INF/MANIFEST.MF` that defines, through property-value pairs, several configuration parameters and information related to the package.

4.4.3 Traditional service containers and MRA-capable containers

The servlet container is an application that provides the network services such that other applications can access deployed services through MIME-based requests and MIME-based responses mechanisms and manages their life-cycle [76]. The servlet container must implement the Java Servlet spec-

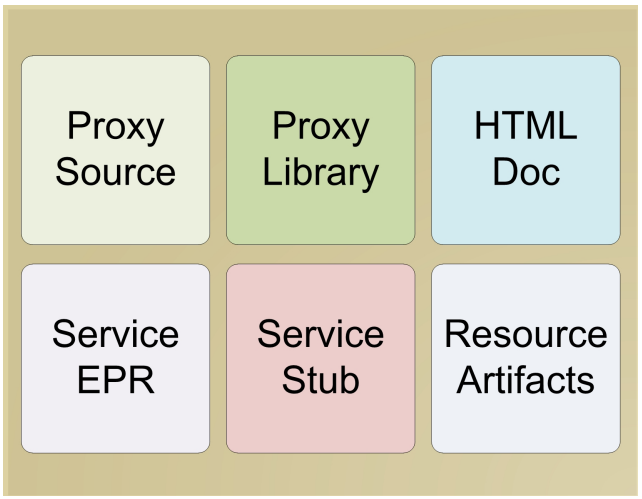

Figure 4.2: Managed Resource Archive constituents

ification. Because the MRA format follows the JAR specification, it is considered by the traditional service containers as a valid WAR that they are able to deploy and expose, however, the additional features incorporated into the MRA are not exposed. For these additional features to be exposed, another layer is added on top of the traditional service containers that would do this task. Although this layer is container dependent, an interface was specified for the layers to implement. This allows the management software agents querying the additional layers to be independent of the service container. Currently, only the implementation for the Apache Foundation's Tomcat Servlet Container [2] is provided.

Figure 4.2 shows the different constituents of the MRA. Only the EPR is used by the traditional servlet containers. This allows the MRA to be backward compatible with the traditional servlet containers and at the same time being able to expose its additional features. The layer is itself a managed resource and encapsulates the service container. Using its interface, it is possible to programatically deploy/undeploy MRAs, migrate MRAs to other containers, inquire about their status and get the listing of deployed MRAs.

[2] http://tomcat.apache.org

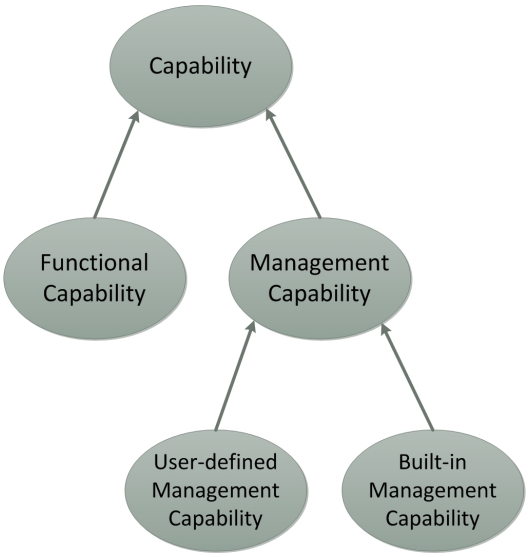

Figure 4.3: Capabilities taxonomies

4.5 Resource capabilities definition

In the previous chapter, we defined a capability as the capacity of the resource to undergo a change, initiate a change in another resource, or affect its environment. In organizing the capabilities we arranged them following the taxonomy shown in figure 4.3.

4.5.1 Functional capabilities

The functional capabilities are the capabilities that define the behavior of the resource. They can affect the resource itself, other resources or its environment. This type of capabilities, being resource-specific, is generally defined by the resource owner and has no default implementation. However, once an implementation is provided, other resources from the same resource class can share these implementations for their capabilities.

4.5.2 Management capabilities

The management capabilities are capabilities that have a management purpose. They have either no effect on the resource (e.g. `getResourceStatus`) or only affect the resource itself (e.g. `startUp`). They can be specific to a class of resources, span multiple classes of resources or be part of a stan-

dard set of management capabilities that all resources should/can implement.

The management capabilities can either be user-defined just like the functional capabilities, or built-in. The standardized management capabilities are built-in and can be associated to the resource at design time. The MRF will then, generate all the necessary implementations and link it with the resource. Section 4.7 describes the built-in capabilities in more details. The management capabilities that can span multiple classes can either be user defined or built-in. In the latter case, requirements are enforced at the level of the semantic description to ensure that the default implementation of the management capability can be applied to the resource class (e.g. `startUp` capability shared between different servlet containers and web containers that are built using the same engine).

4.5.3 Providing an implementation for the capabilities

During the design phase of a resource, every capability description requires a valid name for the capability, a list of valid parameters, a valid result type and a valid implementation class name including the package.

When using such definitions to generate a resource's MRA, the MRF searches in the provided class path for the class that implements that capability. If the class is found, it will be included in the MRA linked libraries that will be linked when building the archive. If the class is missing, the MRF will generate a server stub for the capabilities and the developer has to provide an implementation for the capabilities. Here is an example stub as generated by MRF:

Listing 4.3: An example of a generated capability Class

```
1  package com.ibm.de.serom.pool.sampleresource.impl;
2
3  import org.apache.muse.core.AbstractCapability;
4
5  public class StartImpl extends AbstractCapability
       implements IStartImpl
6  {
7      public boolean startCapability(int after) throws
           Exception    {
8          //TODO implement startCapability
9          throw new RuntimeException("Unimplemented Method:
               startCapability");
10     }
11 }
```

You should note here that, although the capabilities have no implementation, the above code is valid and can be compiled and a valid archive can be built using it. However the capabilities will only throw the exception `RuntimeException` if they are invoked, pointing to the missing implementation. Which is an effective way to quickly test if the resource "connectivity" is working by generating MRA, deploying it and calling the capabilities without implementation. If the fore-mentioned exception is thrown then the resource is behaving as expected, and any error happening during the latter stages of development would be due to the user-provided implementation of the capability.

The only requirement for providing the implementation of the capability is to inherit a generated interface that defines the proper functions to implement. The following code is the generated interface relative to the above capability implementation:

Listing 4.4: An example of a generated capability Interface

```
1 package com.ibm.de.serom.pool.sampleresource.impl;
2
3 public interface IStartImpl
4 {
5     String PREFIX = "tns";
6     String NAMESPACE_URI = "http://de.ibm.com/serom/pool/
          sampleresource/start";
7     public boolean startCapability(int after) throws
          Exception;
8 }
```

A good practice is to always create the capabilities implementation in their proper packages and classes allowing for code reuse when generating other resources from the same resource class and providing this implementation (that would be contained in a single JAR) to the MRF in its class path so it would be included in the new MRAs without any further intervention from the user.

4.6 Under The Hood

The MRF implementation of the management mechanisms is done following the Web Services Distributed Management (WSDM) set of standards [109, 17, 18]. WSDM defines protocols to allow services to expose their manageability interface in a standard way such that any consumer

that is WSDM-compliant is able to access the service manageability functions. WSDM specification relies on several WS-* standards for its operations, namely WS-MetadataExchange [26], WS-ResourceFramework [6], WS-ResourceProperties [45], WS-ResourceLifetime [95], WS-ServiceGroup [68] and WS-Notification [44]. The manageability of a service is exposed through a Web service and is accessed through that service end-point reference (EPR), called a manageability endpoint. Any software application that accesses the manageability endpoint is called manageability consumer. The manageability consumer can interact with the manageability endpoint, and hence, the resource, in three distinct ways:

- The manageability consumer can retrieve management information from the managed resource through calls to its management capabilities.
- The manageability consumer can affect the state of the managed resource by changing its state through calls to its management capabilities.
- The manageability consumer can receive notifications from the managed resource if the consumer had subscribed to receive events from the managed resource.

The methods stated above show that the relationship between the managed resource and the manageability consumer can be a pull or a push based communication mechanism depending on the nature of the resource and the consumer and the rate by which the resource can produce events. Producing events by the resource is, however, optional. WSDM does not, in general, define the content of the messages exchanged between the managed resource and the consumer, but only the communication protocol and the format of the exchanged data. Using MRF, the user can also specify some WSDM-defined management capabilities to be added to the resource definition. The MRF takes then care of generating the proper configuration files and capabilities implementation. The following section describes the WSDM-defined capabilities as well as other capabilities inherited from the other supporting WS-* standards.

4.7 Built-in capabilities

This section lists the different built-in capabilities in the MRF grouped by the WS-* standards.

WS-DistributedManagement (WSDM)

Advertisement

This capability notifies consumers of the creation and destruction events of the managed resource.

Configuration

This capability provides the possibility to modify some configuration parameters of the resource to change its behavior or status.

CorrelatableProperties

This capability provides a list of configurable parameters that can be used to compare resources and identify similar resources.

Description

This capability provides description and version properties of the managed resource.

Identity

The identity capability return the *ResourceId*, a unique, final and non-mutable identifier of the resource.

ManageabilityCharacteristics

This capability is used to get a list of all the supported capabilities of the managed resource, the built-in as well as the custom capabilities.

Metrics

This capability lists the metrics used in measuring the resource performance or its operations.

OperationalStatus

This capability returns the status of the managed resource as one of the following values: *Available*, *PartiallyAvailable*, *Unavailable* or *Unknown*.

State

This capability allows the managed resource to expose its state to the consumers as well as it transitional state, or previous state.

WS-Notification (WSN)

Offers the possibility for consumers to register to a class of events that the resource advertises.

WS-ResourceLifetime (WSRL)

Provides an interface to manage the lifetime of a managed resource.

WS-MetadataExchange (WSMEX)

GetMetadata

This capability is used to retrieve metadata information about a resource EPR.

4.8 Writing a semantic description

Any descent graphical OWL editor can be used to write semantic descriptions. We are going to describe here the most important steps needed to write a sample resource. The first thing is to include the *AbstractResource* ontology and any other ontology that the resource may inherit from a specific resource class (see listing 4.5).

Listing 4.5: Importing Abstract Resource ontology

```
<owl:Ontology rdf:about="">
  <owl:imports rdf:resource="http://lrr.in.tum.de/serom/
     mrf/AbstractResource.owl"/>
</owl:Ontology>
```

The resource is described in the `<mrf:ManagedResource>` tag, and it should contain all the concepts that describe the resource. Listing 4.6 shows the structure of the `<mrf:ManagedResource>` tag with its most important constituents, namely the resource id, the resource properties and the resource capabilities.

Listing 4.6: Structure of a managed resource with its most important elements

```
<mrf:ManagedResource rdf:ID="...">
  <mrf:hasResourceId> ... </mrf:ResourceId>
  <mrf:hasProperty> ... </mrf:hasProperty>
  <mrf:hasBuiltInManagementCapability rdf:resource="..."/
     >
  <mrf:hasManagementCapability> ... </
     mrf:hasManagementCapability>
  <mrf:hasCapability> ... </mrf:hasCapability>
</mrf:ManagedResource>
```

The resource id is composed of one `ResourceId` concept instance, that is itself composed of a string that acts as an id[3] and a reference to the resource owner as shown in the following listing:

```
<mrf:hasResourceId>
  <mrf:ResourceId rdf:ID="srId">
    <mrf:rid rdf:datatype="http://www.w3.org/2001/
       XMLSchema#string"
    > ... </mrf:rid>
    <mrf:owner rdf:datatype="http://www.w3.org/2001/
       XMLSchema#string"
    > ... </mrf:owner>
  </mrf:ResourceId>
</mrf:hasResourceId>
```

The resource can have multiple properties. Every property need to have a name and a type as shown in the following listing:

```
<mrf:hasProperty>
  <mrf:Property rdf:ID="Type">
    <mrf:name rdf:datatype="http://www.w3.org/2001/
       XMLSchema#string"
    >type</mrf:name>
    <mrf:type rdf:datatype="http://www.w3.org/2001/
       XMLSchema#string"
    >string</mrf:type>
  </mrf:Property>
</mrf:hasProperty>
```

[3]The datatype of the id is defined in *AbstractResource* as a string. This can be changed to whatever seems appropriate for the application

The resource capabilities can all be defined following the same patterns. However, the built-in capabilities will more probably reference some external resource that details the implementation information for that capability as shown below:

```
<mrf:hasBuiltInManagementCapability rdf:resource="http://
   lrr.in.tum.de/serom/mrf/AbstractResource.owl#
   SimpleMetaDataExchange"/>
```

These built-in capabilities, as their name suggests, should have a preexisting definition and a preexisting implementation accessible to the Managed Resource Framework.

The other capabilities can also be defined as a reference to an external resource or they can have their definition inlined. A capability must have a name, a return type if it returns a result and optional parameters for the capability request. The parameter definition can also be inline and has to define a parameter name and type. The following listing shows how this can be done:

```
<mrf:hasCapability>
  <mrf:Capability rdf:ID="...">
    <mrf:implClass  rdf:datatype="http://www.w3.org/2001/
       XMLSchema#string"
    > ... </mrf:implClass>
    <mrf:name rdf:datatype="http://www.w3.org/2001/
       XMLSchema#string"
    > ... </mrf:name>
    <mrf:type rdf:datatype="http://www.w3.org/2001/
       XMLSchema#string"
    > ... </mrf:type>
    <mrf:hasParameter>
      <mrf:Parameter rdf:ID="...">
        <mrf:isParameterOf rdf:resource="..."/>
        <mrf:type rdf:datatype="http://www.w3.org/2001/
           XMLSchema#string"
        > ... </mrf:type>
        <mrf:name rdf:datatype="http://www.w3.org/2001/
           XMLSchema#string"
        > ... </mrf:name>
      </mrf:Parameter>
    </mrf:hasParameter>
  </mrf:Capability>
</mrf:hasCapability>
```

The final document should have the minimum set of sections as described above and should look like listing 4.7.

Listing 4.7: Structure of a semantic definition file

```
<rdf:RDF
  <!--namespaces definitions -->
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:xsp="http://www.owl-ontologies.com/2005/08/07/xsp
     .owl#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  ....

  <!-- imports section -->
  <owl:Ontology rdf:about="">
    <owl:imports rdf:resource="http://lrr.in.tum.de/serom
       /mrf/AbstractResource.owl"/>
    ...
  </owl:Ontology>

  <!-- resource description section -->
  <mrf:ManagedResource rdf:ID="...">

    <!-- resource description section -->
    <mrf:hasResourceId> ... </mrf:ResourceId>

    <!-- resource properties section -->
    <mrf:hasProperty> ... </mrf:hasProperty>

    <!-- resource built-in capabilities section -->
    <mrf:hasBuiltInManagementCapability rdf:resource="...
       "/>

    <!-- resource management capabilities section -->
    <mrf:hasManagementCapability> ... </
       mrf:hasManagementCapability>

    <!-- resource capabilities section -->
    <mrf:hasCapability> ... </mrf:hasCapability>

  </mrf:ManagedResource>

</rdf:RDF>
```

4.9 Service contract generation

The Service Contract Generator creates a service contract following the WSDL specification from the semantic description on the resource. The resource semantic description file should have a structure following listing 4.7. Such representation is used to generate the service contract in WSDL. Listing 4.10 shows a sample WSDL file that was generated using the MRF. The WSDL file has five sections that we are going to outline here. Listings 4.8 and 4.9 are taken from the semantic definition file used to generate the example shown for the WSDL file.

Listing 4.8: Example of a resource built-in capability definition

```
1 <mrf:hasBuiltInManagementCapability rdf:resource="http://
     lrr.in.tum.de/serom/mrf/AbstractResource.owl#
     SimpleMetaDataExchange"/>
```

Listing 4.9: Example of a capability definition

```
1  <mrf:hasCapability>
2    <mrf:Capability rdf:ID="Status">
3      <mrf:implClass  rdf:datatype="http://www.w3.org/2001/
          XMLSchema#string"
4      >com.ibm.de.serom.pool.sampleresource.impl.StatusImpl
          </mrf:implClass>
5      <mrf:name rdf:datatype="http://www.w3.org/2001/
          XMLSchema#string"
6      >Status</mrf:name>
7      <mrf:type rdf:datatype="http://www.w3.org/2001/
          XMLSchema#string"
8      >string</mrf:type>
9      <mrf:hasParameter>
10       <mrf:Parameter rdf:ID="DetailLevel">
11         <mrf:isParameterOf rdf:resource="#Status"/>
12         <mrf:type rdf:datatype="http://www.w3.org/2001/
              XMLSchema#string"
13         >integer</mrf:type>
14         <mrf:name rdf:datatype="http://www.w3.org/2001/
              XMLSchema#string"
15         >detailLevel</mrf:name>
16       </mrf:Parameter>
17     </mrf:hasParameter>
18   </mrf:Capability>
19 </mrf:hasCapability>
```

Types. This section is used to describe the data types used by the capabilities and its parameters. Lines 19-67 in listing 4.10 are the types used by the sample resource. XML Schema is used to define the data types values. These are the different parameters and return values related to the resource capabilities that are extracted from the semantic description and used to generate the relevant data element. The current implementation of the MRF only supports the native XML datatypes as defined by the XML XSD Schema [99, 14]. However, the semantic parser used to parse the semantic description and the artifacts generator can be easily extended to support additional, more complex datatypes.

Messages. Messages (lines 68-94 in listing 4.10) represent the service operations datatypes. Messages are composed of one or more message parts that reference the types defined in the previous section. The messages are the service requests parameters and the service responses values.

Port Types. The port types (lines 96-124 in listing 4.10) are the set of operations of the service and correspond to the resource capabilities. The operations are the most important section of the WSDL file as they describe the service through its functionalities and the way to interact with it as well as the different messages involved in the operations.

Binding. The binding (lines 125-179 in listing 4.10) defines the protocol used for every port (SOAP in this case) as well as the message format.

Service. The Service (lines 181-185 in listing 4.10) references all the previous section and associate them with the service. The service section also defines the address under which the service will be deployed and its name. This last bit of information is not taken from the semantic description, but is instead provided at configuration time to a component called the MRF Resource Config component as every resource should have a unique URL to which resource consumers can point to access the resource. The address also specifies the physical location of the resource, or at least the servlet container location that will be serving the resource.

Listing 4.10: A generated WSDL file for a sample resource

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <wsdl:definitions
3     targetNamespace="http://de.ibm.com/serom/pool/
        sampleresource"
```

```
4       xmlns:tns="http://de.ibm.com/serom/pool/
           sampleresource"
5
6       xmlns="http://schemas.xmlsoap.org/wsdl/"
7       xmlns:wsa="http://www.w3.org/2005/08/addressing"
8       xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
9       xmlns:wsdl-soap="http://schemas.xmlsoap.org/wsdl/soap
           /"
10      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
11      xmlns:wsx="http://schemas.xmlsoap.org/ws/2004/09/mex"
12
13      xmlns:status="http://de.ibm.com/serom/pool/
           sampleresource/status"
14      xmlns:stop="http://de.ibm.com/serom/pool/
           sampleresource/stop"
15      xmlns:start="http://de.ibm.com/serom/pool/
           sampleresource/start"
16
17      name="SampleResource">
18
19      <wsdl:types>
20          <xsd:schema
21              elementFormDefault="qualified"
22              targetNamespace="http://www.w3.org/2005/08/
                   addressing">
23              <xsd:include schemaLocation="WS-Addressing
                   -2005_08.xsd"/>
24          </xsd:schema>
25          <xsd:schema
26              elementFormDefault="qualified"
27              targetNamespace="http://schemas.xmlsoap.org/
                   ws/2004/09/mex">
28              <xsd:include schemaLocation="WS-
                   MetadataExchange-2004_09.xsd"/>
29          </xsd:schema>
30          <xsd:schema
31              elementFormDefault="qualified"
32              targetNamespace="http://de.ibm.com/serom/pool
                   /sampleresource/status">
33              <xsd:element name="StatusCapability">
34                  <xsd:complexType>
35                      <xsd:sequence>
36                          <xsd:element name="detailLevel"
```

```
                            type="xsd:integer"/>
37                      </xsd:sequence>
38                  </xsd:complexType>
39              </xsd:element>
40              <xsd:element name="StatusCapabilityResponse"
                    type="xsd:string" />
41          </xsd:schema>
42          <xsd:schema
43              elementFormDefault="qualified"
44              targetNamespace="http://de.ibm.com/serom/pool
                    /sampleresource/stop">
45              <xsd:element name="StopCapability">
46                  <xsd:complexType>
47                      <xsd:sequence>
48                          <xsd:element name="after" type="
                                xsd:integer"/>
49                      </xsd:sequence>
50                  </xsd:complexType>
51              </xsd:element>
52              <xsd:element name="StopCapabilityResponse"
                    type="xsd:boolean" />
53          </xsd:schema>
54          <xsd:schema
55              elementFormDefault="qualified"
56              targetNamespace="http://de.ibm.com/serom/pool
                    /sampleresource/start">
57              <xsd:element name="StartCapability">
58                  <xsd:complexType>
59                      <xsd:sequence>
60                          <xsd:element name="after" type="
                                xsd:integer"/>
61                      </xsd:sequence>
62                  </xsd:complexType>
63              </xsd:element>
64              <xsd:element name="StartCapabilityResponse"
                    type="xsd:boolean" />
65          </xsd:schema>
66
67      </wsdl:types>
68      <wsdl:message name="StatusCapabilityRequest">
69          <wsdl:part name="StatusCapabilityRequest" element
                ="status:StatusCapability" />
70      </wsdl:message>
```

```
71      <wsdl:message name="StatusCapabilityResponse">
72          <wsdl:part name="StatusCapabilityResponse"
               element="status:StatusCapabilityResponse" />
73      </wsdl:message>
74      <wsdl:message name="StopCapabilityRequest">
75          <wsdl:part name="StopCapabilityRequest" element="
               stop:StopCapability" />
76      </wsdl:message>
77      <wsdl:message name="StopCapabilityResponse">
78          <wsdl:part name="StopCapabilityResponse" element=
               "stop:StopCapabilityResponse" />
79      </wsdl:message>
80      <wsdl:message name="StartCapabilityRequest">
81          <wsdl:part name="StartCapabilityRequest" element=
               "start:StartCapability" />
82      </wsdl:message>
83      <wsdl:message name="StartCapabilityResponse">
84          <wsdl:part name="StartCapabilityResponse" element
               ="start:StartCapabilityResponse" />
85      </wsdl:message>
86
87
88
89      <wsdl:message name="GetMetadataMsg">
90          <wsdl:part name="GetMetadataMsg" element="
               wsx:GetMetadata" />
91      </wsdl:message>
92      <wsdl:message name="GetMetadataResponseMsg">
93          <wsdl:part name="GetMetadataResponseMsg" element=
               "wsx:Metadata" />
94      </wsdl:message>
95
96      <wsdl:portType name="SampleResourcePortType">
97          <wsdl:operation name="StatusCapability">
98              <wsdl:input wsa:Action="http://de.ibm.com/
                   serom/pool/sampleresource/status/
                   StatusCapability"
99                          name="StatusCapabilityRequest"
                               message="
                               tns:StatusCapabilityRequest" /
                               >
100             <wsdl:output wsa:Action="http://de.ibm.com/
                   serom/pool/sampleresource/status/
```

```
                StatusCapabilityResponse"
101                       name="StatusCapabilityResponse"
                             message="
                             tns:StatusCapabilityResponse"
                              />
102         </wsdl:operation>
103         <wsdl:operation name="StopCapability">
104             <wsdl:input wsa:Action="http://de.ibm.com/
                   serom/pool/sampleresource/stop/
                   StopCapability"
105                       name="StopCapabilityRequest"
                             message="
                             tns:StopCapabilityRequest" />
106             <wsdl:output wsa:Action="http://de.ibm.com/
                   serom/pool/sampleresource/stop/
                   StopCapabilityResponse"
107                       name="StopCapabilityResponse"
                             message="
                             tns:StopCapabilityResponse" /
                             >
108         </wsdl:operation>
109         <wsdl:operation name="StartCapability">
110             <wsdl:input wsa:Action="http://de.ibm.com/
                   serom/pool/sampleresource/start/
                   StartCapability"
111                       name="StartCapabilityRequest"
                             message="
                             tns:StartCapabilityRequest" />
112             <wsdl:output wsa:Action="http://de.ibm.com/
                   serom/pool/sampleresource/start/
                   StartCapabilityResponse"
113                       name="StartCapabilityResponse"
                             message="
                             tns:StartCapabilityResponse"
                             />
114         </wsdl:operation>
115
116
117         <wsdl:operation name="GetMetadata">
118             <wsdl:input wsa:Action="http://schemas.
                   xmlsoap.org/ws/2004/09/mex/GetMetadata"
119                       name="GetMetadataMsg" message="
                             tns:GetMetadataMsg"/>
```

```
120             <wsdl:output wsa:Action="http://schemas.
                   xmlsoap.org/ws/2004/09/mex/
                   GetMetadataResponse"
121                             name="GetMetadataResponseMsg"
                                   message="
                                   tns:GetMetadataResponseMsg"/>
122         </wsdl:operation>
123
124     </wsdl:portType>
125     <wsdl:binding name="SampleResourceBinding" type="
           tns:SampleResourcePortType">
126         <wsdl-soap:binding style="document" transport="
               http://schemas.xmlsoap.org/soap/http" />
127         <wsdl:operation name="StatusCapability">
128             <wsdl-soap:operation soapAction="
                   StatusCapability" />
129             <wsdl:input>
130                 <wsdl-soap:body
131                     use="encoded"
132                     encodingStyle="http://schemas.xmlsoap
                           .org/soap/encoding/" />
133             </wsdl:input>
134             <wsdl:output>
135                 <wsdl-soap:body
136                     use="encoded"
137                     encodingStyle="http://schemas.xmlsoap
                           .org/soap/encoding/" />
138             </wsdl:output>
139         </wsdl:operation>
140         <wsdl:operation name="StopCapability">
141             <wsdl-soap:operation soapAction="
                   StopCapability" />
142             <wsdl:input>
143                 <wsdl-soap:body
144                     use="encoded"
145                     encodingStyle="http://schemas.xmlsoap
                           .org/soap/encoding/" />
146             </wsdl:input>
147             <wsdl:output>
148                 <wsdl-soap:body
149                     use="encoded"
150                     encodingStyle="http://schemas.xmlsoap
                           .org/soap/encoding/" />
```

```
151             </wsdl:output>
152         </wsdl:operation>
153         <wsdl:operation name="StartCapability">
154             <wsdl-soap:operation soapAction="
                   StartCapability" />
155             <wsdl:input>
156                 <wsdl-soap:body
157                     use="encoded"
158                     encodingStyle="http://schemas.xmlsoap
                           .org/soap/encoding/" />
159             </wsdl:input>
160             <wsdl:output>
161                 <wsdl-soap:body
162                     use="encoded"
163                     encodingStyle="http://schemas.xmlsoap
                           .org/soap/encoding/" />
164             </wsdl:output>
165         </wsdl:operation>
166     <wsdl:operation name="GetMetadata">
167             <wsdl-soap:operation soapAction="GetMetadata"
                    />
168             <wsdl:input>
169                 <wsdl-soap:body
170                     use="encoded"
171                     encodingStyle="http://schemas.xmlsoap
                           .org/soap/encoding/" />
172             </wsdl:input>
173             <wsdl:output>
174                 <wsdl-soap:body
175                     use="encoded"
176                     encodingStyle="http://schemas.xmlsoap
                           .org/soap/encoding/" />
177             </wsdl:output>
178         </wsdl:operation>
179     </wsdl:binding>
180
181     <wsdl:service name="SampleResourceService">
182         <wsdl:port name="SampleResourcePort" binding="
               tns:SampleResourceBinding">
183                 <wsdl-soap:address location="http://
                       localhost:8080/sampleresource/services
                       /SampleResource"/>
184         </wsdl:port>
```

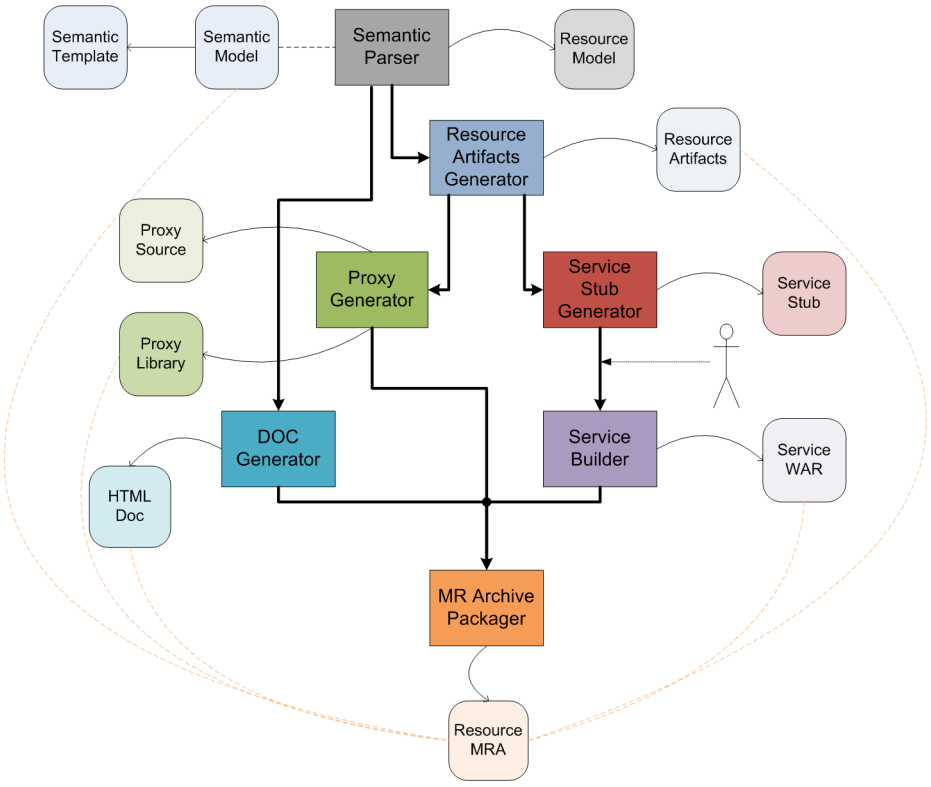

Figure 4.4: General view of the MRF components

```
185     </wsdl:service>
186 </wsdl:definitions>
```

4.10 Architectural views

4.10.1 General view

The Managed Resource Framework has a plug-in based architecture where every component can get input from other components and outputs artifacts that can be fed to other components. Figure 4.4 shows a simplified view of the MRF workflow, where components are represented as rectangles and their outputs as round-angles rectangles. The whole process can be automatized with the human intervention happening optionally between the Service Stub Generator and the Service Builder to implement custom capabilities . MRF manages the MRA creation process as a project. IT follows the concept of a target or a goal for specifying the needed arti-

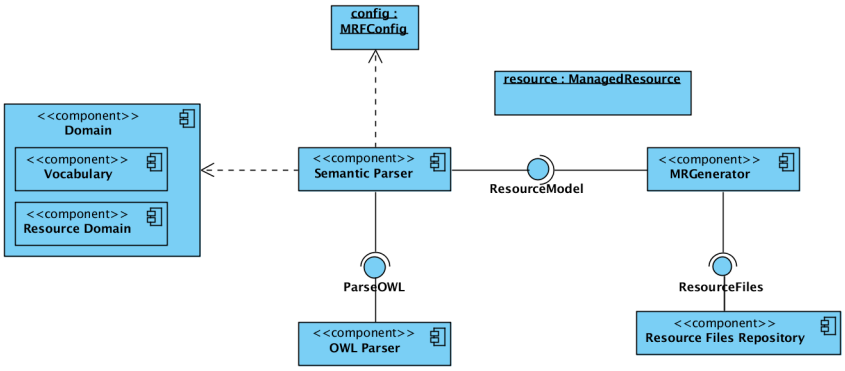

Figure 4.5: Semantic Parser component diagram

facts that have to be generated from a unique source of information, which is the resource semantic description. The fact that MRF architecture is plug-in based, allows the possibility to include new components with new build targets. If there is, for instance, an interest in producing resource representations as Flex [4] with AMF endpoints, a component that can generate the necessary files from the resource model could be integrated into MRF, and Flex objects could be generated by specifying the correct build target.

In the following section we describe the logical view of MRF by explaining each of the several components of the framework and presenting some the activity diagram that can show the workflow of building services.

4.10.2 Component view

4.10.2.1 Resource configuration

The resource configuration component is used for the initialization of the MRF and contains the different parameters that the different MRF plug-ins may use. Examples of these parameters are the different paths for the templates used, the class paths, the different namespaces and the deployment host for instance.

4.10.2.2 Semantic parser

The Semantic Parser component parses the resource description file and creates a resource model that provides fucnctionalities to query the model

[4] http://www.adobe.com/products/flex/

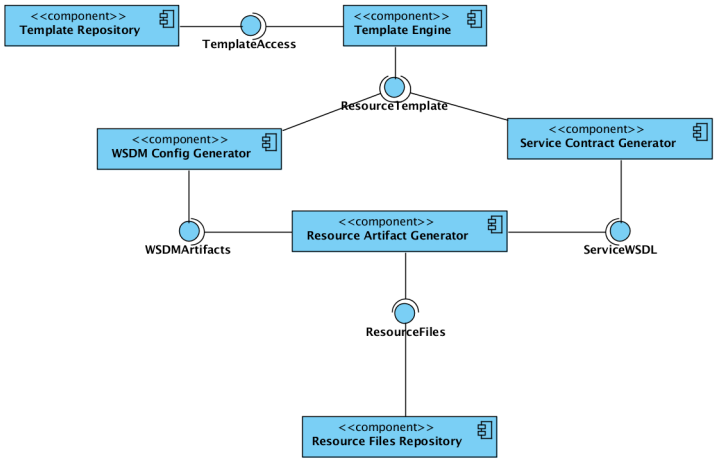

Figure 4.6: Resource Artifact Generator component diagram

for different information on the resource and its capabilities and properties. The parser is fed a configuration file that contains details such as additional semantic repositories that the semantic description may need, the path or the repository to where the artifacts would be generated, deployment location of the resource and the build targets or the types of artifacts to be generated. The parser relates to the Domain component that contains the Resource Domain and the Vocabulary. Figure 4.5 represents the Semantic Parser UML component diagram.

Domain. The Domain is a component that contains two other components that are the vocabulary that holds a description and a mapping to all the semantic properties and classes that are used to describe the resources, and a resource domain that represent the model of a managed resource as defined by the *AbstractResource* semantic model.

OWL Parser. The OWL Parser is, as its name suggests, an OWL parser used to parse the semantic description of the resource. It is used to create the resource model that contains the classes, properties, literals and all the concepts that define a resource. The resource model is the main object used throughout the MRF to handle the resource description.

4.10.2.3 Resource artifacts generator

The Resource Artifacts Generator component generates the necessary artifacts to create the resource service representation. The artifacts generated

depend on the target chosen. What follows is a listing of the required artifacts for generating an MRA.

- Deployment Descriptor: The deployment descriptor is defined following an XML schema document and contains the elements for configuring the web application and for its proper deployment in a servlet container. The deployment descriptor contains information on the servlet context initiation parameters, session configuration, servlet declaration, servlet mappings, error pages and some other elements.
- Service Contract: The service contract is built around a model for service description using the Web Service Description Language (WSDL)
- Management Capability Artifacts: These are different files generated that depend on the implementation of the WSDM specification.

Template Engine. The template engine is a component used to store and manage the different templates that can be used to create or instantiate new resources. New resource classes that inherit from the top level ontology can be added to the template repository and used later on.

WSDM Config Generator. The WSDM Config generator is based on Muse[5], the Apache implementation for WSDM. It requires its own deployment descriptor file called `muse.xml` that is used during the initialization step of the resource. The classes generated by the MRF are referenced in `muse.xml`.

Service Contract Generator. The Service Contract Generator creates a service contract following the WSDL specification from the semantic description of the resource as described in section 4.9.

4.10.2.4 Online documentation generator

The online documentation generator uses the resource semantic description and the other generated artifacts to generate an online documentation in HTML. The documentation contains descriptions of the resource and its properties and capabilities as well as the Java documentation and information on how to programatically access the resource along with code snippets.

[5]http://ws.apache.org/muse

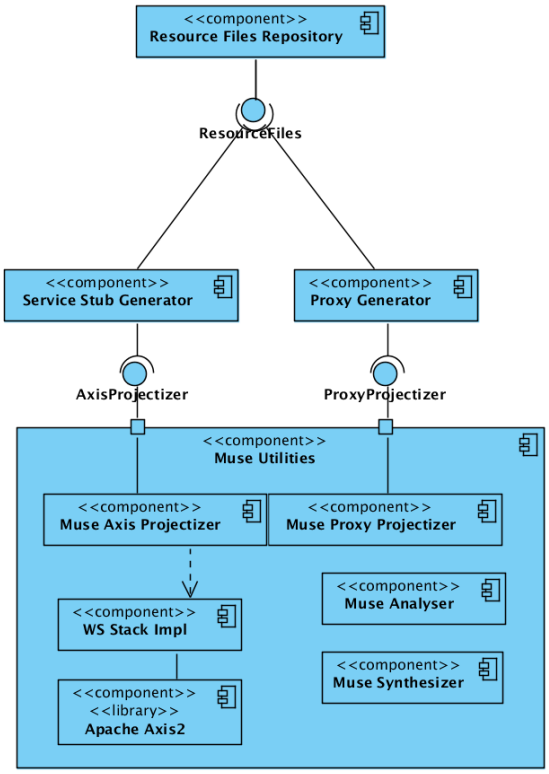

Figure 4.7: Source Code Generator component diagram

4.10.2.5 Source code generator

Figure 4.7 shows the component diagram for the source code generator. The source code generator can generate the source code for the service representation of the resource as well as the client proxy to connect to the resource. There are two components that take care of those tasks, the Service Stub Generator and the Proxy Generator. The MRF extends the excellent open source implementation from the Apache foundation of the WSDM and the W3C SOAP specifications in the implementation of both components.

4.10.2.6 Builders and packager

The MRF has two builders, the Java builder used to build the generated source code and the WAR builder that generates the MRA archive and the proxy library.

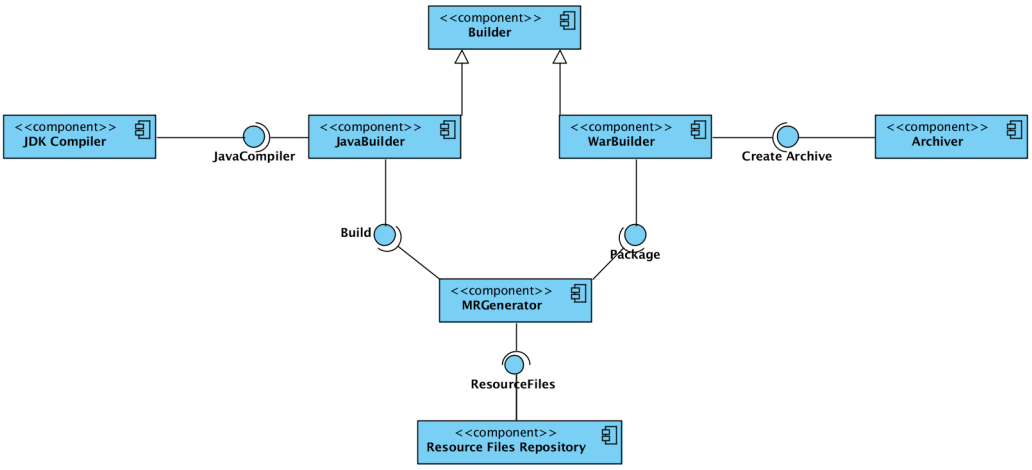

Figure 4.8: Builders component diagram

Java Builder. The Java builder is used to compile the java source files, either for the service stub or for the client proxy. The Java builder can also package the compiled classes into a JAR archive, especially for the client proxy that can be then integrated into the MRA. A build file (listing 4.11) is also generated to be used by the developers in case they need to customize the build process.

Listing 4.11: Proxy builder xml file

```
1  <project name="Build Proxy" default="proxy">
2
3      <target name="init">
4          <basename property="NAME" file="${basedir}"/>
5
6          <property name="JAVA_SRC_DIR" value="src"/>
7          <property name="JAVA_DEST_DIR" value="bin"/>
8          <property name="LIB_DIR" value="lib"/>
9
10         <property name="JAR_FILE" value="${NAME}.jar"/>
11
12         <path id="class.path">
13             <fileset dir="${LIB_DIR}">
14                 <include name="**/*.jar"/>
15             </fileset>
16         </path>
17     </target>
18
```

```
19      <target name="clean">
20          <delete dir="${JAVA_DEST_DIR}"/>
21          <delete file="${JAR_FILE}"/>
22      </target>
23
24      <target name="java" depends="init">
25          <mkdir dir="${JAVA_DEST_DIR}"/>
26          <javac srcdir="${JAVA_SRC_DIR}" destdir="${
                JAVA_DEST_DIR}" classpathref="class.path"/>
27          <jar destfile="${JAR_FILE}">
28              <fileset dir="${JAVA_DEST_DIR}">
29                  <include name="**/*.class"/>
30              </fileset>
31          </jar>
32      </target>
33
34      <target name="proxy" depends="clean, java"/>
35
36  </project>
```

WAR Builder. The WAR builder packages the different artifacts generated by the MRF into the MRA archive. A builder and archiver file is also generated to allow the customization of the process (listing 4.12).

Listing 4.12: Builder xml file

```
1   <?xml version="1.0"?>
2   <project name="project" default="war">
3       <target name="init">
4               <basename property="NAME" file="${basedir}"/>
5               <property environment="env"/>
6               <property name="MUSE_HOME" value="${env.
                    MUSE_HOME}"/>
7               <property name="BUILD_DIR" value="build"/>
8               <property name="JAVA_SRC_DIR" value="
                    JavaSource"/>
9               <property name="WAR_FILE" value="${BUILD_DIR
                    }/${NAME}.war"/>
10              <property name="WEB_INF_DIR" value="${
                    BUILD_DIR}/WEB-INF"/>
11              <property name="LIB_DIR" value="${WEB_INF_DIR
                    }/lib"/>
```

```
12              <property name="CLASSES_DIR" value="${
                    WEB_INF_DIR}/classes"/>
13              <path id="muse.class.path">
14                      <fileset dir="${MUSE_HOME}/modules/ws
                            -fx-api/">
15                          <include name="*.jar"/>
16                      </fileset>
17                      <fileset dir="${MUSE_HOME}/modules/ws
                            -fx-impl/">
18                          <include name="*.jar"/>
19                      </fileset>
20                      <fileset dir="${MUSE_HOME}/modules/
                            core/">
21                          <include name="*.jar"/>
22                      </fileset>
23                      <fileset dir="${MUSE_HOME}/modules/
                            mini/">
24                          <include name="*.jar"/>
25                      </fileset>
26                      <fileset dir="${MUSE_HOME}/lib/common
                            ">
27                          <include name="*.jar"/>
28              </fileset>
29              </path>
30      </target>
31          <target name="layout" depends="init, clean">
32              <copy file="${basedir}/web.xml" todir="${
                    WEB_INF_DIR}" flatten="true"/>
33              <copy todir="${CLASSES_DIR}">
34                  <fileset dir="${basedir}">
35                      <include name="muse.xml"/>
36                          <include name="wsdl/**"/>
37                          <include name="router-entries/**"
                                />
38                      </fileset>
39              </copy>
40          <copy todir="${LIB_DIR}" flatten="true">
41              <fileset dir="${MUSE_HOME}/modules">
42                  <include name="ws-fx-api/*.jar"/>
43                  <include name="ws-fx-impl/*.jar"/>
44                  <include name="core/*.jar"/>
45                  <include name="mini/*.jar"/>
46              </fileset>
```

```
47              <fileset dir="${MUSE_HOME}/lib/common">
48                  <include name="*.jar"/>
49              </fileset>
50              </copy>
51          </target>
52      <target name="java" depends="layout">
53          <javac srcdir="${JAVA_SRC_DIR}" destdir="${
               CLASSES_DIR}" classpathref="muse.class.path"/>
54      </target>
55      <target name="war" depends="java">
56          <jar destfile="${WAR_FILE}">
57                      <fileset dir="${BUILD_DIR}">
58                          <include name="WEB-INF/**"/>
59                      </fileset>
60              </jar>
61      </target>
62
63      <target name="proxy" depends="java">
64          <copy todir="${BUILD_DIR}" flatten="true">
65              <fileset dir="clientProxy">
66                  <include name="*.jar"/>
67              </fileset>
68          </copy>
69          <jar destfile="${WAR_FILE}">
70              <fileset dir="${BUILD_DIR}">
71                  <include name="WEB-INF/**"/>
72                  <include name="clientProxy.jar"/>
73              </fileset>
74          </jar>
75      </target>
76
77      <target name="clean" depends="init">
78              <delete dir="${BUILD_DIR}"/>
79      </target>
80
81      <target name="run" depends="init">
82          <java classname="${main}">
83              <classpath>
84                  <pathelement location="${CLASSES_DIR}"/>
85                  <fileset dir="${LIB_DIR}">
86                      <include name="*.jar"/>
87                  </fileset>
88              </classpath>
```

```
89          </java>
90      </target>
91  </project>
```

4.10.3 Other diagrams

Figure 4.9 shows the MRF activity diagram and figure 4.10 shows the MRF sequence diagram. The activity diagram shows the flow of control within the MRF. It is a graphical representation of the possible workflows that happen inside the MRF. The rounded rectangles represent the different activities, the small squares represents either an output pin or an input pin, the black bar represent in this diagram a split of concurrent activities and finally the black circle represents the final state.

The sequence diagrams shows how the different components interact with each other in a timely fashion. The dotted vertical lines represent the lifeline of the component on the top. The rectangles on top of the lifelines are called activation boxes and represent an active component initiating or reacting to a message. The messages are represented as arrows and responses as dotted arrows.

4.11 Summary

We presented in this chapter the Managed Resource Framework (MRF), a framework to automatically generate ready-to-deploy services from the semantic description of resources. The resources can be semantically described by extending a basic resource model called *AbstractManagedResource* resulting in a rich document containing, among other things, the properties and capabilities of the resource as well as a description of the management layer for that resource. The objective is to have a computer aided process by which resources can be rapidly *instantiated*, deployed and managed in a relatively quick and transparent manner for the user.

The framework can generate an archive called the Managed Resource Archive (MRA), that is an extension to the service archive that contains in addition to the service impementation, the resource semantic description, the proxy libraries, the service contract and online documentation, to name the most important.

The MRF uses a target-based mechanism, meaning that only parts of the MRA can be generated depending on the target used during the execution of the MRF process.

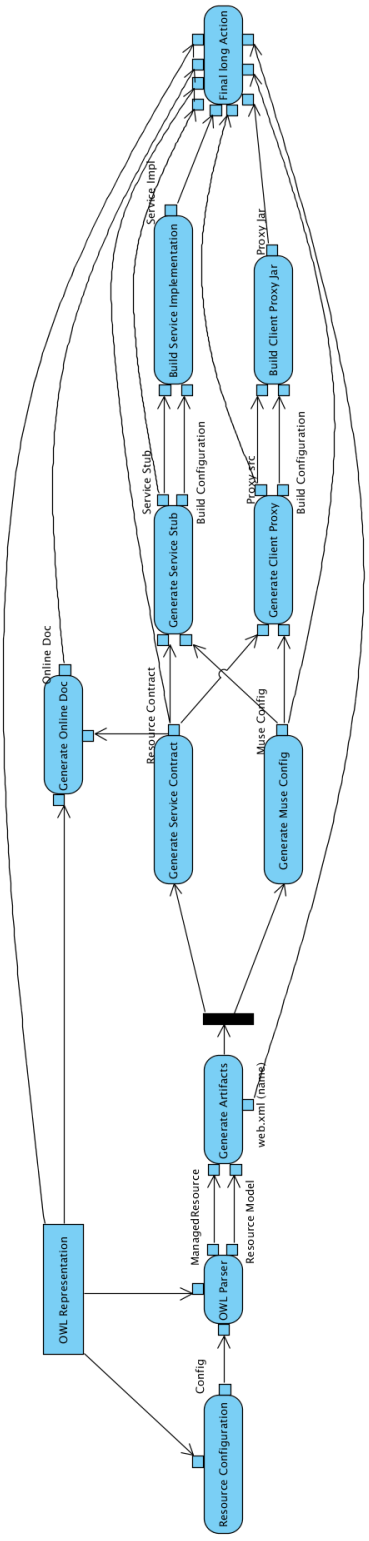

Figure 4.9: MRF activity diagram

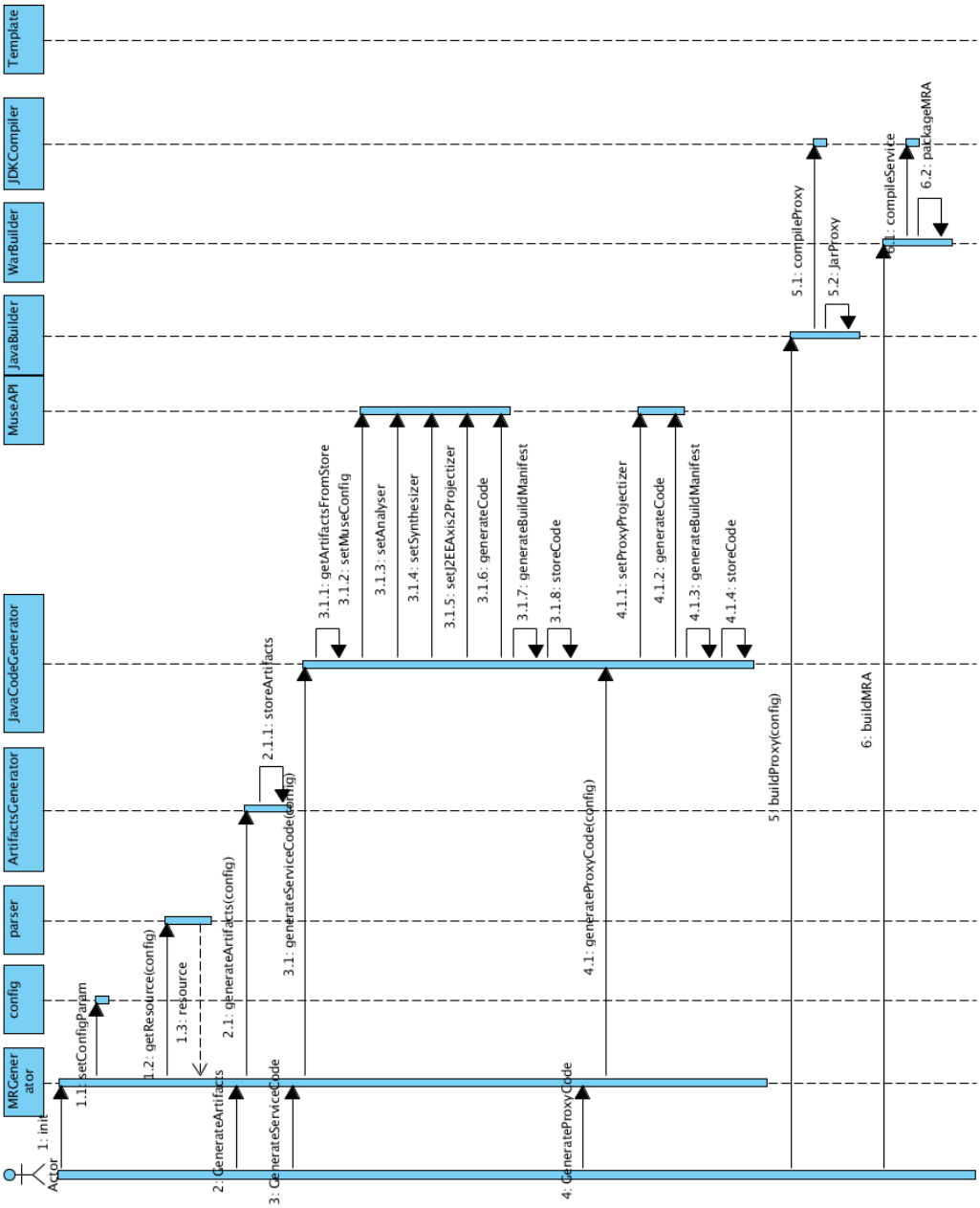

Figure 4.10: MRF sequence diagram

Chapter 5

Semantic Resource Management Infrastructure

5.1 Overview

The Semantic Resource Management Infrastructure (SRMI) is an infrastructure of services to allow access and management of resources uniformly, transparently and in a standard way. It allows to manage a set of IT resources in an autonomous way following a SOA conceptual model while relying on a semantic approach. The SRMI allows the separation of management domains and supports a pluggable inference engines architecture.

5.2 Autonomic management

In the SEROM infrastructure (the SRMI implemention), there are services used to access the different IT resources. These accesses are abstracted from the rest of the system thanks to the MRF and the different MRAs, that expose standard interfaces for accessing and managing the resources. The autonomic management is based on the MAPE acronym that stands for the four steps used in managing resources in an autonomic manner, and these are: Monitor, Analyze, Plan and Execute. The management part, represented in the center has these four main functions corresponding to the MAPE steps. Monitoring is about the collection of usage and state data of the managed resources. Analysis is based on the collected information and extracts relevant issues that need to be tackled. Planning is about creating plans of action to implement the specified changes done in the previous function. Finally, execution is about committing those changes.

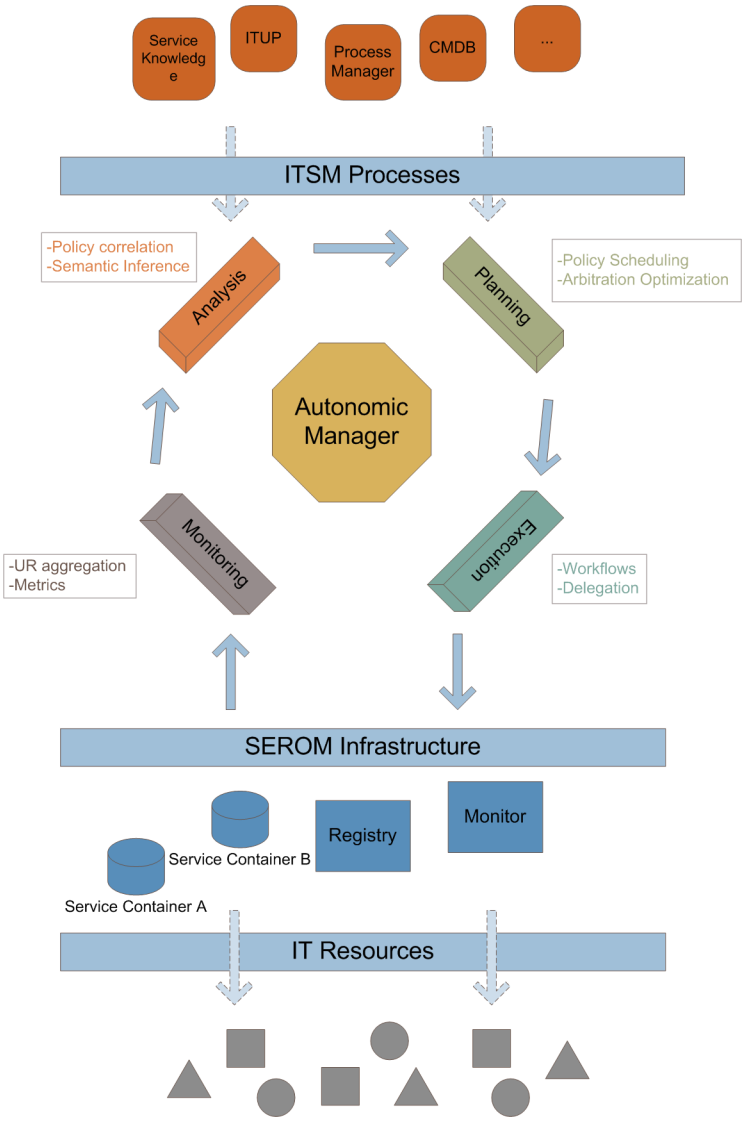

Figure 5.1: MAPE loop

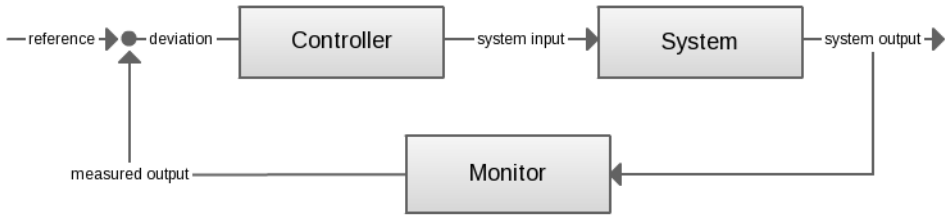

Figure 5.2: Self-management control loop

The whole process is controlled based on some guidelines that can originate from different sources, such as policy usages or IT management best practices, that represent the higher level in figure 5.1. After the execution step is performed, monitoring the new state of the system is done (actually the monitoring is performed on a continuous basis) to check if the state of the system corresponds to the desired state. If not, a new cycle is initiated. Hence, MAPE constitute an intelligent control loop (figure 5.2).

The objective of the controller in the closed loop is to bring the IT system into some desired state by acting on the system resources following some administrative guidelines. This closed loop is further developed and incorporated into the global system in figure 5.1. The reference denotes the desired state and the deviation: the difference between the actual state and the desired state.

5.2.1 Minimal requirement for the self-managed system

Our self-managed system needs to exhibit at least the following four characteristics to be able to act in an autonomic manner:

Awareness is the ability to sense information about the environment either by querying the resources or by receiving timely notification about resource state changes.

Ability to analyze allows making sense of the information gathered. This analysis is based on some specified goals defined by the human manager. The analysis function is an intelligent function that uses logic to achieve its results.

Ability to plan allows the system to come up with an action plan to implement the results reached in the previous function. Planning can be as simple as sequential task execution or a complex graph with dependencies and conflict resolution mechanisms.

Ability to affect its state or its environment state allows the system to affect itself or its environment based on the plan of the previous function. The changes can be done directly or delegated to other components that take care of this task.

The self-managed system needs two external interfaces to achieve its goal: sensor interface and effector interface. The sensor interface is used to get external information of the different resources either directly or through some monitoring interface. The effector interface is used to commit the changes to the resource.

5.2.2 Conceptual model

The self-managed system is a goal-driven system. It derives its goals from policies and rules that dictate its behavior and needs to be able to control itself and its environment based on these rules. There are two types of knowledge involved in this process. A state knowledge about the self-managed system and its environment and a behavioral knowledge that specifies the goals of the self-managed system and generally how to reach them. The first knowledge is gathered using the sensing ability of the system based on its sensor interface, and the second knowledge is internally stored in some logical representation. Logic plays an important role in the self-managed system as it is logic that is used to derive the actions of the system. Finally the actions are executed using the effector ability of the self-managed system.

5.3 MAPE implementation in SRMI

In SRMI, the Automation Engine is the component that represent the analysis, planing and execution steps in the MAPE loop. Monitoring is done by another component called the Monitor. The Automation Engine is composed of a set of Domain-Specific Inference Engines (DSIE) and an Execution Engine. The DSIE takes care of the analysis and planning steps while the Execution Engine deals with the execution step. Section 5.6.2.4 describes the Automation Engine and section 5.6.2.7 describes the Monitor. The following section (5.4) describe in greater details the DSIE.

5.4 Domain-specific inference engine (DSIE)

The Domain-Specific Inference Engine (DSIE) is a specialized inference engine tightly related to a set of resource classes and/or management model(s)(see section 5.4.1). The DSIE is highly customizable and can be instantiated whenever there is a need. It is built on top of Jena [72][20], an open source java framework for building Semantic Web applications and includes a rule-based inference engine. Jena was designed such that it is possible to plugin additional inference engines to derive additional assertions depending on the new reasoner axioms and rules. We took advantage of this feature to build our custom inference engines on top of Jena. We designed the DSIE to be a modular inference engine built as a stack of different inference engines, mainly for engine reusability and ease of development for the management applications developers.

5.4.1 Domain-based management

In SRMI, the management is domain-based, meaning that the management logic depends on a specific definition of the context of the resources and the definition of the resources. A definition of a management domain is the triplet: context specification, resource-specific definitions and behavioral logic.

The context specification represents the management model, i.e. a formal representation of the requirements needed in a management context. If we take as an example the Billing context, where we would be interested in producing usage bills for resources, then the formal definition of a bill and its components (e.g. resource id, usage period, cost per period unit, over-head cost, total cost... etc) and the billing process (e.g. a billing request, checking the resources are actually billable, the terms of the billing... etc) and all the necessary elements for a proper billing context would be our context specification.

The resource-specific definitions are the resource formal definitions that are necessary and relevant for the management context. In our previous example of billing, the properties `billable`, `oneOffCost`, `costPerSecond` would be examples of billing context-specific definitions of the resources. Thanks to the semantic model used to describe resources, a resource can "gain" these definitions by making it a sub-type of a *BillableResource* class, that we would define as a sub-type of a *ManagedResource* with the forementioned properties.

The behavioral logic is the logic to be used in the management context and would be represented in the form of rules. In our previous example of billing, a `generate_bill` rule can constitute the logic used in generating bills. It can have many sub-rules that would have logic such as if the resources is used for more than a day then apply a flat-rate, or if the customer is of a certain class then apply a discount.

Management contexts can be composed of several other management contexts. Examples would be Reporting. If we define reporting as composed of monitoring and billing, then the reporting management domain will be the definitions of the monitoring and billing tasks along with the definitions of the specificities of the requirements needed from the resources to be used in those two operations, as well as all the behavioral logic that formally determines the actions to perform accounting and billing. The motivation behind the domain-based management is to be able to create pluggable management tasks to be incorporated into the system whenever there is a need, without affecting the proper functioning of the existing management tasks. A system would be using Incident Management context for instance and could have Reporting added without affecting the first task. This is possible by creating independent domain-specific inference engines (DSIE) that hold behavioral logic specific to the management context. This behavioral logic is called the domain logic. The domain logic references some constructs specific to the management domain. Those constructs are defined in the schema of the management domain and in the resource-specific semantic description. And because a semantic resource can inherit its definitions from multiple semantic models (management domains schemata), it is possible for the resource to be part of different DSIEs and hence different management contexts. However, DSIEs can command actions that could affect the resource, and hence, indirectly affect other DSIEs, therefore mechanisms need to be implemented to deal with such scenarios.

5.4.2 Domain logic

As stated in the previous section, the domain logic is the behavioral logic related to a management domain. It is defined as a collection of basic rules that should capture all the logic that defines the management context. The rules have premises and conclusions and they are used to infer new facts or initiate some actions. The premises and the conclusions are composed of terms called body terms for the premises and head terms for the conclusion. The domain logic is modeled as a `RuleSet` object composed of `Rule`

objects that are used by the DSIE reasoner at run-time. The `Rule` object is used as the internal representation of a rule in Jena. Any rule syntax can be used as long as interpreters are provided that can construct the `Rule` objects. The textual representation of the RDF Schema that uses the triple representation of RDF descriptions as well as the N3 notation [12] and Turtle [10] are natively supported as rule syntax. The basic syntax for the textual rules is presented below [91]:

Listing 5.1: JenaRules Syntax

```
Rule       :=   bare-rule .
           or   [ bare-rule ]
           or   [ ruleName : bare-rule ]

bare-rule  :=   term, ... term -> hterm, ... hterm
           or   term, ... term <- term, ... term

hterm      :=   term
           or   [ bare-rule ]

term       :=   (node, node, node)
           or   (node, node, functor)
           or   builtin(node, ... node)

functor    :=   functorName(node, ... node)

node       :=   uri-ref
           or   prefix:localname
           or   <uri-ref>
           or   ?varname
           or   'a literal'
           or   'lex'^^typeURI
           or   number
```

The terms can be rules, triple patterns or functors. The triple patterns are composed of three nodes, and every node can be a variable, a literal, some URI or an embedded functor. Examples of triple patterns are `(ns1:Resource1 rdf:type serom:ManagedResource)` that states that Resource1 is of type *ManagedResource* or `(?ResourceA e2e:startsAfter ?ResourceB)` that states that ResourceA has to start after ResourceB, the `startsAfter` property is as defined in the context with the namespace `e2e`. The functor has a name and a list of arguments that can be nodes of any type but the functor type. There is a number of built-in functors, however

it is possible for the user to create custom functors associated with their semantic implementation, and hence act as the rule "action" if the functor is present in the head of the rule. If the functor is present in the body of the rule, it represents a predicate.

The symbol '->' is used for forward chaining rules and the symbol '<-' is used for the backward chaining rules. Hybrid rules can be used, they would be composed of one or more backward chaining rules in the head of a forward chaining rule.

5.4.3 Schema and model instances

The schema represents the semantic model needed for the management context. It is composed of a resource-specific model and the domain model. A billable resource for instance would be augmented by a semantic model defining the cost and terms of usages. A *BillableResource* model can be defined and the resource can either be an instance of the *BillableResource* as well as the *ManagedResource* or the *BillableResource* can itself be an instance of the *ManagedResource,* depending on the design decisions.

The model instances are the instantiations of the schema. Typically there is one instance of the management model and one instance for every resource used in this management context.

5.4.4 Generic Reasoner

The generic reasoner is the base reasoner used in Jena that other reasoners can be built on top. It is a general purpose rule-based reasoner that supports inference over RDF graphs. It provides forward chaining, backward chaining and a hybrid reasoning model. The generic reasoner has an implementation of the RETE algorithm [38] that acts as a forward chaining engine and a tabled datalog engine that acts as the backward chaining engine.

5.4.5 Custom Reasoner = Generic Reasoner + Domain Logic

The generic reasoner still needs some rules to set its behavior. Associating the generic reasoner with the domain logic allows having what we call a custom reasoner, able to answer queries on some specific model. The generic reasoner can be bound to one or several domain logic sets, allowing to add functionalities to the reasoner when needed. As long as the domain logic sets are on different namespaces, adding them to a generic

reasoner will not lead to a clash between the rules, however, if the domain logic sets act on the same namespaces, additional care needs to be used to insure that the rules do not contradict themselves or that they don't lead to situations like deadlocks or starvation.

Another interesting feature of the generic reasoner is that it is possible to define new procedural primitives called functors that have their semantic implementation defined as Java functions. This allows a better integration of the rules in the Java environment and allows the rules to have as a conclusion (or body) a call to a Java function thus allowing a concrete and easy way to implement the rules actions especially when the action deals with a change to the resource as the resource has often a Java interface to interact with it.

5.4.6 Specialized Reasoner = Custom Reasoner + Schema

Having a custom reasoner is not enough because the rules would reference classes of elements or concepts that are dependent on the management context. We need to associate the custom reasoner with some RDF graphs and models that define these concepts. This is what we define here as schema: the necessary semantic models referenced from the domain logic. The schema is called the ABox (for assertion box) and represents also the facts in the knowledge base.

5.4.7 DSIE = Specialized Reasoner + Model Instances

A knowledge base is composed of the ABox and the TBox. The ABox is the schema, whereas the TBox is the model instances. The TBox is the terminological component of the knowledge base. The model instances represent all the instances, or realizations of the concepts defined in the management context schema or in the superset of concepts defining the manageable resources. The managed resource representations are part of the instance models. Once a specialized reasoner is associated with some model instances we have what we call a domain-specific inference engine (DSIE) that can be queried and can act on a set of resources. Specialized reasoners can be duplicated without consequences, however DSIEs are unique as multiple copies of the same DSIE would reach the same conclusions.

A DSIE is defined by a specialized reasoner and some model instances as we mentioned. However, some instances maybe shared among different DSIEs. The controller (section 5.6.2.6) and scoping mechanisms are used to attenuate the effect of DSIEs' rules clashing, nevertheless, conflict

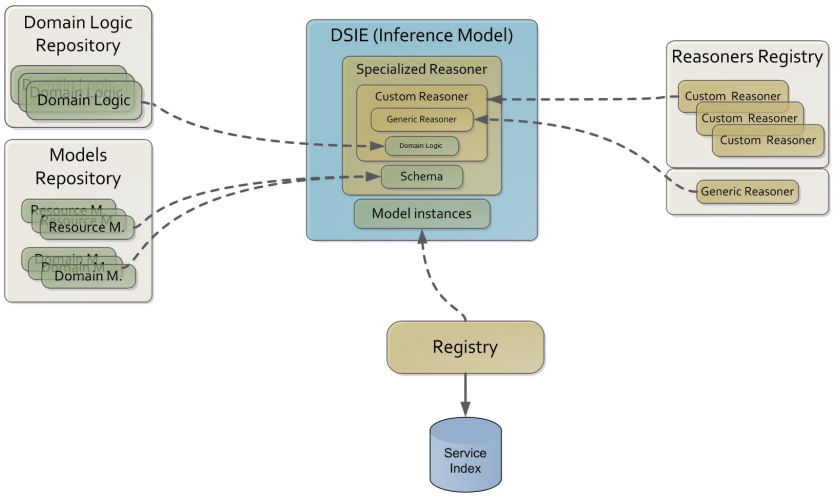

Figure 5.3: Constructing a DSIE

resolution mechanism are also necessary as contradictory rules can be part of the same DSIE. Figure 5.3 shows the components involved in the construction of a DSIE.

5.4.8 Composing DSIEs from non-generic reasoners

Figure 5.4 shows the basic stacking of a reasoner. However, any of those reasoner can be used as basis for another DSIE stacking. For instance we could have a custom reasoner that we could add to additional domain logic and a schema to create a new specialized reasoner. Or take a specialized reasoner and add to it some domain logic to create another specialized reasoner. This allows us to create some common reasoners to be instantiated whenever they are needed and stack on top of them some new logic or schemata to create our DSIEs. Candidates are OWL or RDF reasoners such as the ones present in Jena or in other external reasoner such as Pellet [93]. Another candidate would be the specialized reasoner with the basic *ManagedResource* model that every reasoner should extend.

5.5 Organization of the DSIEs and the Knowledge Base

Two organizational models could be used for the DSIEs and the knowledge base (KB). A centralized organization and a distributed organization.

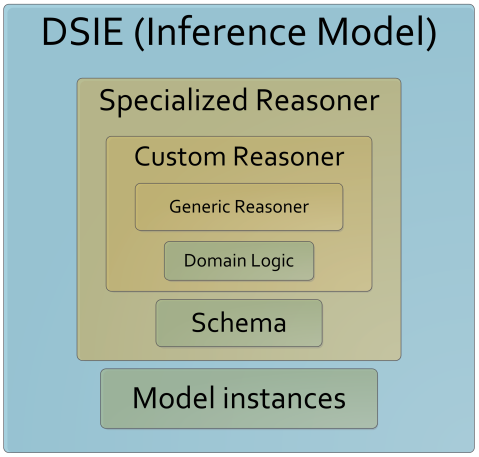

Figure 5.4: Inference engines stacking

In a centralized organization, we have a single KB that is accessed by all the inference engines whereas a distributed organization decomposes the KB to several smaller KBs assigned to each and every inference engine.

Even if the distributed organization seems to be the more natural choice from the point of view of the inference engines, it has several drawbacks, such as the decomposition rules, i.e. what part of the KB shall be assigned to which inference engine? And what if some knowledge pertains to more than one inference engine? What if an inference engine creates knowledge that nullifies another piece of knowledge somewhere else? How could we propagate new knowledge? Issues like synchronization, consistency and prioritization makes using a distributed KB organization a very challenging issue.

In the centralized setting, a single KB is shared by all the inference engines. This solution being easier to realize, still comes with some challenging issues. Every inference engine acts on a representation of the KB that holds knowledge of interest to that engine while omitting knowledge that does not compass the span of the engine. The span of an inference engine is defined as all and only the knowledge related to the IE scope and management domain. This KB representation is a local copy of the management domain associating it with and only with that engine. This association is transparent to the engines, meaning that it only sees a single local KB that it has full control on. This method, while easing the management of KB access as well as not requiring special procedures for the engine to access the KB, leads to another category of problems. Namely, KB coherence, local KB selection procedure, KB access synchronization,

access prioritization, infinite chain reaction, atomicity of transactions, consistency, isolation (access lock), or durability (once a rule is run there shall be no rollback) just to name a few.

In SRMI, we take a hybrid approach to tackle those problems. There is one central component, called the *Controller*, that has a global view on the knowledge base and feeds local copies for every DSIE. More details are described in the related section of the controller (see section 5.6.2.6).

5.6 Architectural views

5.6.1 General view

Figure 5.5 gives an overview of the architecture of the SRMI with the Controller playing a central role in organizing the flux of data between all the major components of the system. In the component view section we will describe every single major component in more details as well as its relationship with the other components.

5.6.2 Components view

5.6.2.1 Service Deployer

The *Service Deployer* is a managed resource itself that is used to deploy other managed resources programatically. It can either deploy resources to a special managed resource container: *MRContainer*, or it can deploy resources to classic servlet containers. To deploy on a servlet container, an adapter needs to exist, that would translate from the deployment commands of the service deployer to the servlet container. This adapter is specific to every servlet container and needs to extend the *ServletContainerAdapter* and override some functions, namely:

- `deployServlet`
- `undeployServlet`
- `deleteServlet`
- `listDeployedServlets`
- `startServlet`
- `stopServlet`
- `getServletContainerStatus`

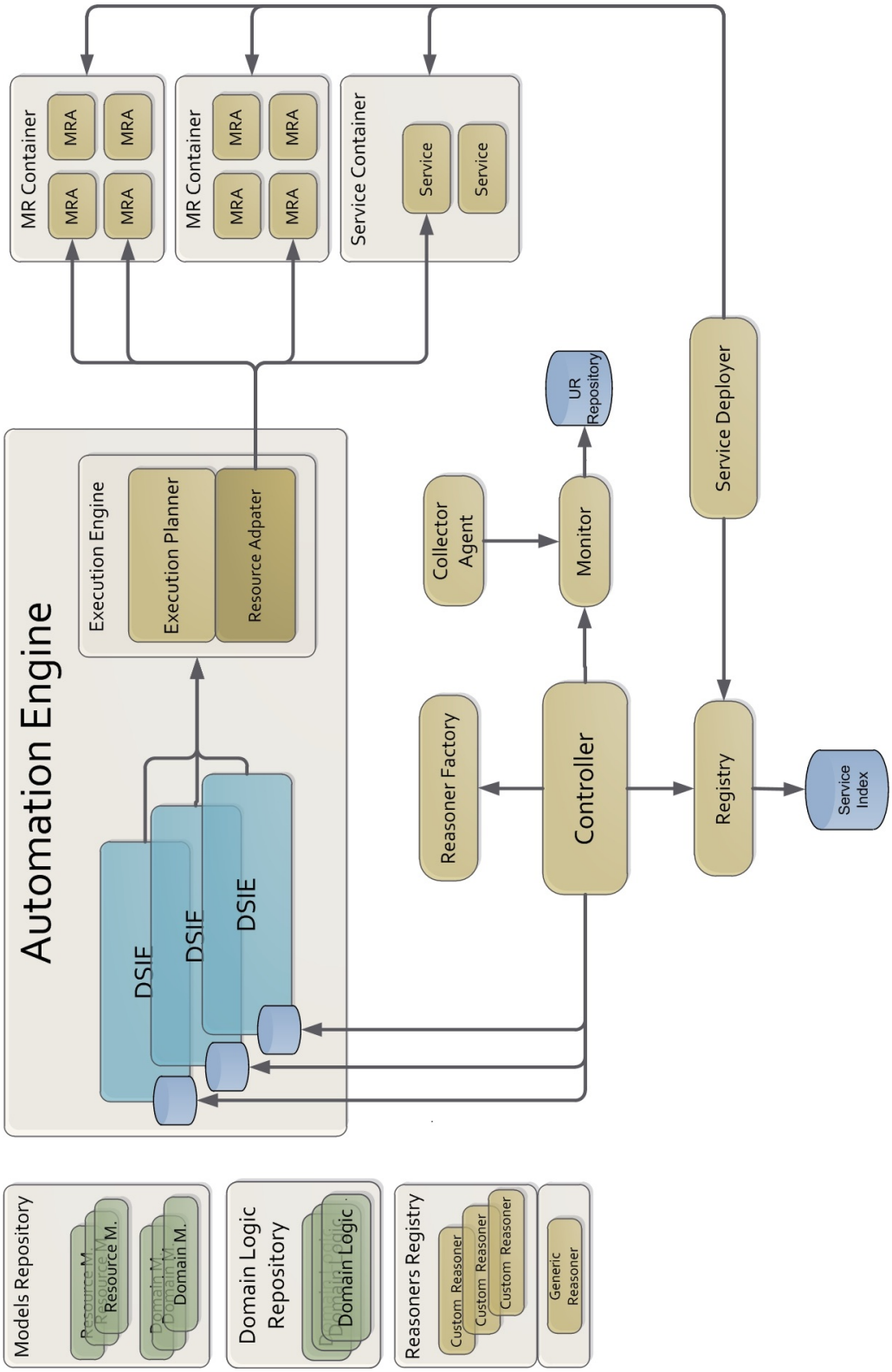

Figure 5.5: SRMI architecture overview

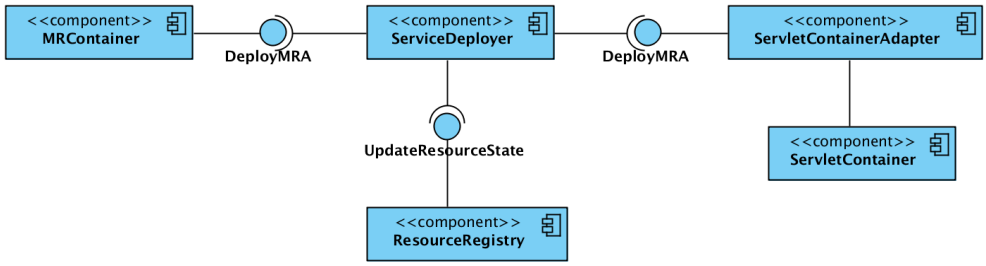

Figure 5.6: Component diagram of the Service Deployer component and the related components

5.6.2.2 MRContainer

The MRContainer is a servlet container with apecific support for MRA and it can act also as a consumer interface for the standard management capabilities. The MRContainer is implemented as a wrapper to a Java servlet engine and supports the functions available from the *ServletContainerAdapter* from the previous section and it adds to that the following functions:

- `getSemanticResourceDescription`
- `getResourceContract`
- `getCapabilitiesList`
- `isCapabilitySupported`
- `getClientProxy`
- `getResourceArtifacts`

Figure 5.6 shows the component diagram of the service deployer, MRContainer, and the related components.

5.6.2.3 Resource Registry

The resource registry, as its name suggests, is a registry holding information about the resources. Specifically it is a table holding the following information for every resource.

- Resource id
- Resource URI

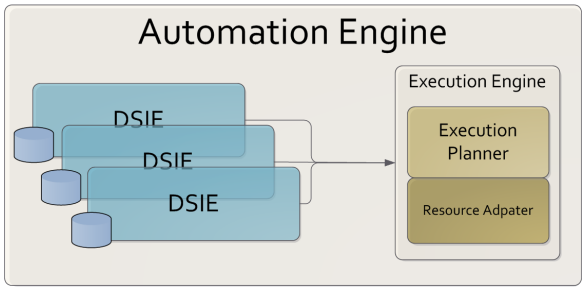

Figure 5.7: Automation Engine

- Resource deployment status (deployed/undeployed)
- Resource container id
- A list of the resource states (as a semantic graph) with time stamps.

The main purpose is to be able to locate any resource and to get the state of the resource. To limit the number of states kept for every resource, the number of states can be configured per resource either by number (e.g. keep the last 10 states) or by time (e.g. keep the states of the last two hours).

5.6.2.4 Automation Engine

The automation engine (see figure 5.7) is composed of a number of DSIEs and an execution engine. The DSIEs are completely independent of each other, in fact, they don't "see" each other and their actions do not presume the existence of other engines. The DSIEs contained in one automation engine are said to be *logically grouped* as opposed to being *physically grouped*. A set of DSIEs that are logically grouped are associated with the same automation engine and share the same execution engine. The physical grouping is a low-level organization related to the DSIEs as services themselves. As everything in SRMI is a service, the DSIEs are services that need to be deployed into a service container. A set of DSIEs are physically grouped if they are deployed in the same container.

There are several motivations for logical grouping. Logical grouping of DSIEs allows organizing the engines in an easier manner. Every group can have a rank that reflects its priority level (used to define workflows and engines execution precedence rules). DSIEs can be logically grouped because they belong to the same process or management task. Grouping DSIEs by process for instance would make it easier to enable or disable

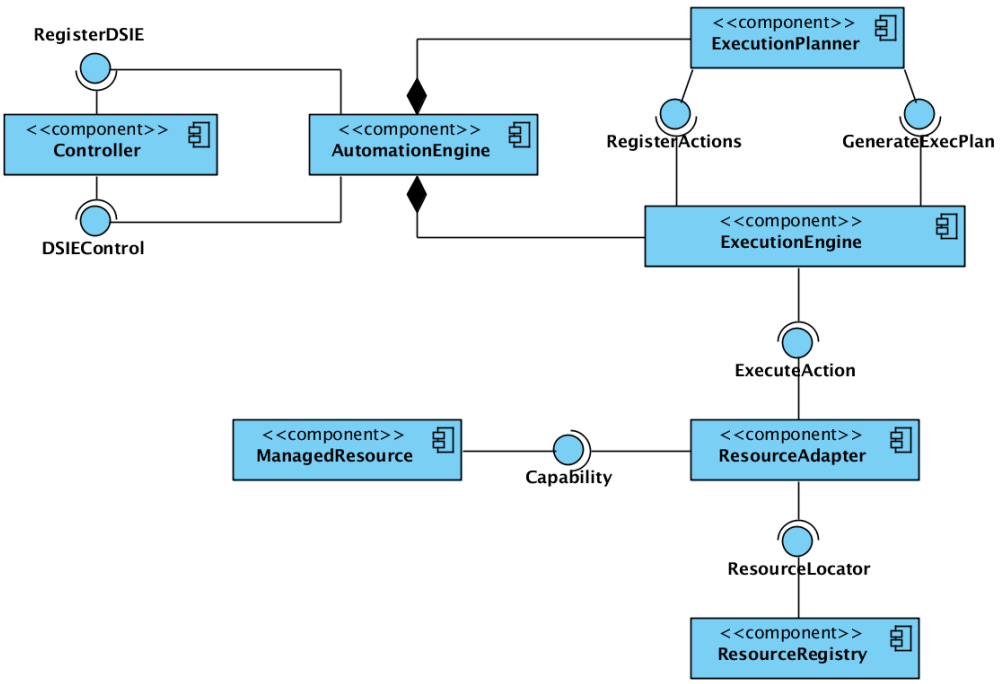

Figure 5.8: Component diagram of the Automation Engine and related components

that specific process. Like having for instance a "controlling" group containing "monitoring", "accounting", "reporting" and "billing" DSIEs and that can be enabled or disabled as a group and not as individual DSIEs.

The execution engine is itself composed of the execution planner and the resource adapter. The execution planner is tasked with resolving any conflicts that may rise from the different rules results. Once this is done and a final list of actions is generated, the execution engine uses the resource adapter to get the needed resource handles to execute the necessary operations.

5.6.2.5 Reasoner Factory

Figure 5.9 shows the component diagram of the Reasoner Factory and related components. The reasoner factory is used to create reasoners from scratch or from other reasoners. The reasoner registry is a store that holds references to different reasoners. Developers can create their own reasoners and then register them with the registry. This allows an easy access to the reasoners and allows to easily locate some specific type of reasoner

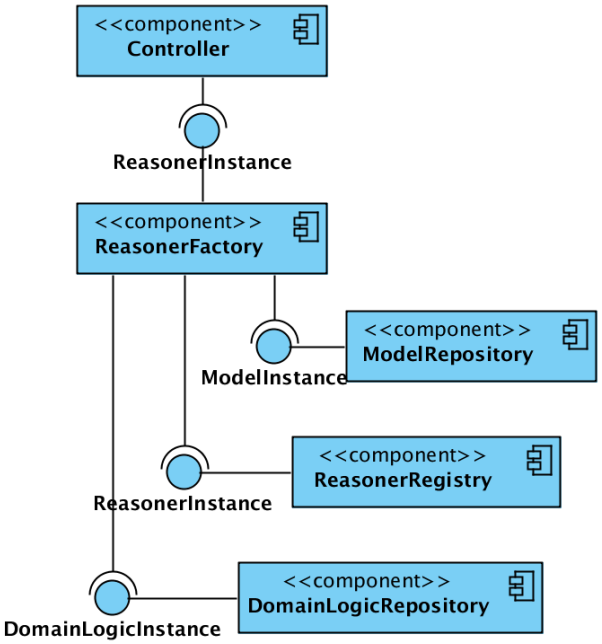

Figure 5.9: Component diagram of the Reasoner Factory and related components

and not have to create a new one from scratch. There is also the domain logic registry that holds different rules to be reused and shared between different instances of reasoners.

5.6.2.6 Controller

Every resource has a unique identifier and it belongs to several namespaces. These namespaces are inherited from the parent concepts that this resource is derived from. For instance, the *ManagedResource* concept has the namespace:
`ns_mr=''http://lrr.in.tum.de/serom/mrf/ManagedResource''`
and *GreenResource* concept has the namespace:
`ns_gr=''http://lrr.in.tum.de/serom/mrf/GreenResource''`
If a resource is a *ManagedResource* and a *GreenResource* then it belongs to both namespaces `ns_mr` and `ns_gr`. In fact, not only the resources have namespaces but also the management domains and every concept in SRMI.

During the initialization phase of a DSIE, it is associated to all the namespaces that it should act upon. A green IT DSIE for instance would register `ns_gr`, but not `ns_bl`, the billing namespace. This mechanism

gives big hints to the controller to know which events are of interest to which DSIE(s), lowering considerably the size of the copies to the DSIEs local database, and minimizing the interaction between the DSIEs. This mechanism is, however, not perfect and leaves some instances where two DSIEs would act on the same resource at the same moment, it is then up to the execution planner to resolve such issues. Algorithm 5.1 presents the steps used in the Controller module to orchestrate the SRMI where step 1 refers to algorithm 5.2.

Algorithm 5.1 Controller Algorithm

```
1  Create a Management Context
2  Get state updates
3  Filter events based on model instance(s) affected
4  Mark DSIE local KB for update
5  Update local knowledge bases
6  A rule firing round is triggered
7  Mark affected resources with an incoherent state flag
8  Submit actions to the planner
9  Request actions conflicts resolution
10 Execution manager gets handle from the resource adapter
       and sequentially execute
11 Update resource state
12 Mark state as coherent
```

Algorithm 5.2 Management Context creation by the Controller

```
1 Instantiates ManagementContext object
2 Load suitable reasoners from the registry
3 Instantiate DSIEs
4 Associate with every reasoner domain logic, domain
     models and resource models
5 Register the list of resources in an internal resource
     index
6 Register in the ScopeTable the namespaces that every
     DSIE is concerned with
7 Create AutomationEngine object(s) and associate every
     DSIE with an automation engine
```

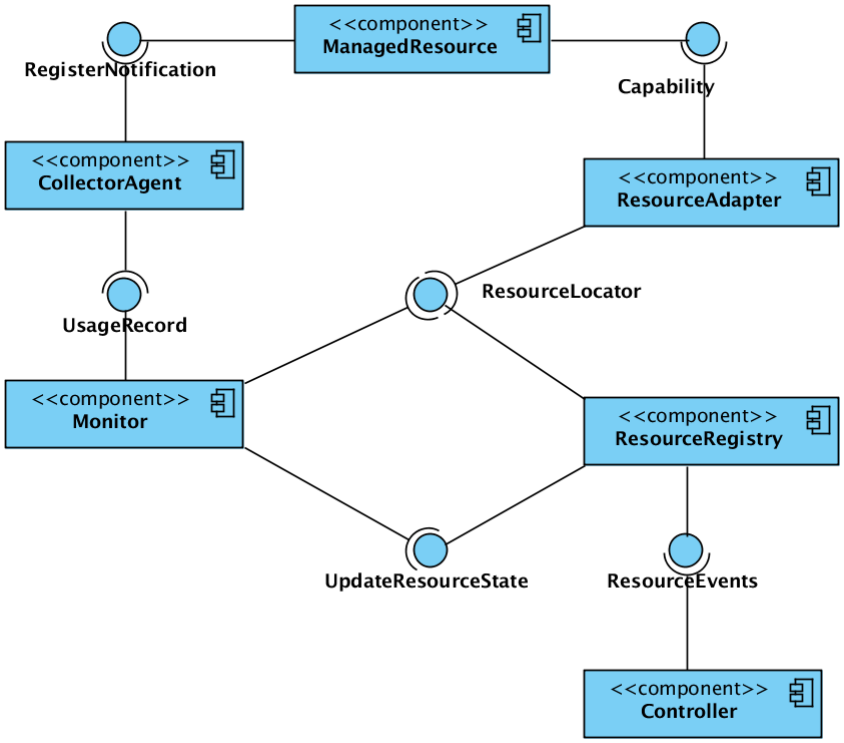

Figure 5.10: Component diagram of the Monitoring Service

5.6.2.7 Monitor

We use resource-specific monitoring in a way that for every type of resources a monitoring collector agent takes care of collecting resource updates. The best case scenario is when the resource offers the possibility to register to its events using the notification system. That way the collectors can register to the events and get state updates of the resources as they happen. This method is referred to as a *push* mechanism where the resources push updates by opposition to the *pull* mechanism where the agents have to pull the updates themselves. This can be done by periodically invoking the `State` capability of the resource and updating its model. The disadvantage of this method is that the resource state updates are not instant, and shortening the interval time between every pull can waste resources and lower the performance.

5.7 Summary

We presented in this chapter the Semantic Resource Management Infrastructure (SRMI) a set of services that use the properties of the resources having an MRA generated using the Managed Resource Framework to manage these resources in a autonomic manner based on some set of rules.

We introduced some concepts for the SRMI such as the management domains or the Domain-Specific Inference Engines (DSIE). The management domains is specific definition of the context of the resources usages. It allows to logically cluster resources, rules or management actions depending on their purpose. The DSIE are modular inference engines that support the addition (or removal) of domain logic and resource schema to accommodate their management objectives in such a way to control in a fine-grained manner the properties of the management system. This allows to add management capabilities (e.g. accounting and billing) to the system whenever they are needed.

Chapter 6

SEROM Prototype

SEROM (SERvice Oriented Management) is the prototype that was developed in the frame of this work. The MRF was fully implemented whereas the SMRI had partial implementation sufficient as a proof of concept, although some parts of it, like the accounting system were fully implemented as a stand-alone system.

6.1 Managed Resource Framework

We developed a full implementation of the Managed Resource Framework to materialize some concepts presented in this work. The development environment was constituted of the Java programming language for the richness of its libraries and APIs and its portability, eclipse[1] as IDE and Maven[2] as project management tool.

MRF is being developed as a modular, plug-in based software to insure the future extendability of the framework. It comes in two versions a command line tool (figure 6.1) and an API to be programatically invoked. MRF is target-based, meaning that a target needs to be specified while invoking the framework and these targets can be: `model`, `wsdl`, `doc`, `source`, `proxy`, `build` or `all`. `model` can only be invoked programatically, all the other targets can also be invoked from the command line tool. The `model` target generates an internal model of the resource, this model is needed by all the developed modules. `wsdl` generates the WSDL document from the model, `doc` the online documentation, `source` the server stub and the proxy skeleton source code, `proxy` the proxy client library, `build` builds the WAR archive and finally `all` generates all of the above and packages the

[1] http://www.eclipse.org
[2] http://maven.apache.org

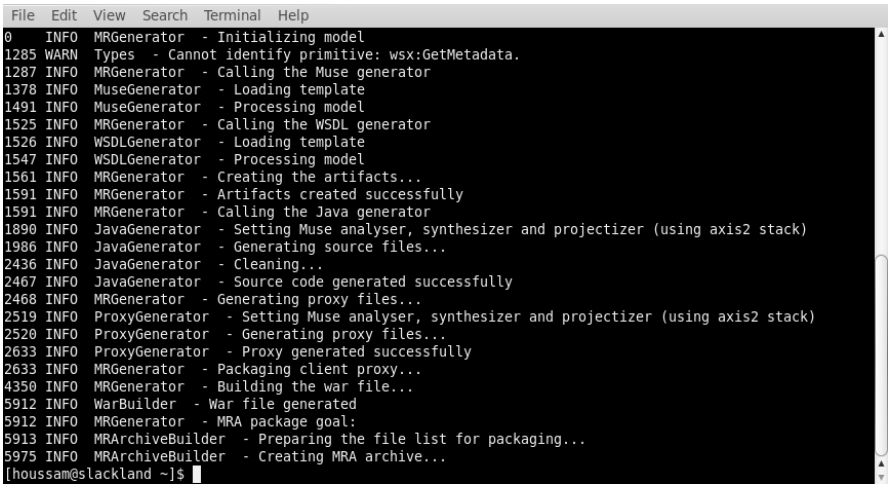

Figure 6.1: Command line run of the MRF with the target `package` (same as `all`)

archive into a deployable artifact. Every new module can provide new targets but it has to specify its required inputs from the existing targets. MRF uses a templating system to generate the different documents. Appropriate templates are loaded at run-time and processed using the in memory model of the resource. The WSDM implementation used by MRF is Apache Muse.

Several tests were successfully conducted to evaluate MRF, including the generation of some of the SRMI components, and we think that the MRF achieved the requirements of usability practicability and simplicity. As an example, consider the WSDM tutorial at IBM's Developer Works[3] that aims at building a WSDM interface for an HTTP server with only two basic capabilities: Start and Stop. In that tutorial the developer had to write at the conclusion of the tutorial a 2000 lines WSDL document that would be used as a contract for a manageable service for the HTTP server. We used the same requirements presented in that tutorial and using MRF with the target `wsdl` we achieved similar result with a description of around 20 lines of code that the developer had to write. Listing 6.1 is an extended version of that description with additional properties, capabilities and built-in capabilities.

Listing 6.1: A sample resource semantic description

```
1 <?xml version="1.0"?>
2 <rdf:RDF
```

[3] http://www.ibm.com/developerworks/tivoli/tutorials/ac-wsdmmuse/

```
3     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns
         #"
4     xmlns:protege="http://protege.stanford.edu/plugins/
         owl/protege#"
5     xmlns:xsp="http://www.owl-ontologies.com/2005/08/07/
         xsp.owl#"
6     xmlns:owl="http://www.w3.org/2002/07/owl#"
7     xmlns:pool="http://de.ibm.com/serom/resourcePool#"
8     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
9     xmlns:swrl="http://www.w3.org/2003/11/swrl#"
10    xmlns:mrf="http://lrr.in.tum.de/serom/mrf/
         AbstractResource.owl#"
11    xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
12    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
13  xml:base="http://de.ibm.com/serom/resourcePool">
14  <owl:Ontology rdf:about="">
15    <owl:imports rdf:resource="http://lrr.in.tum.de/serom
         /mrf/AbstractResource.owl"/>
16  </owl:Ontology>
17  <mrf:ManagedResource rdf:ID="sampleResource">
18    <mrf:hasProperty>
19      <mrf:Property rdf:ID="Type">
20        <mrf:name rdf:datatype="http://www.w3.org/2001/
             XMLSchema#string">type</mrf:name>
21        <mrf:type rdf:datatype="http://www.w3.org/2001/
             XMLSchema#string">string</mrf:type>
22      </mrf:Property>
23    </mrf:hasProperty>
24    <mrf:hasBuiltInManagementCapability rdf:resource="
         http://lrr.in.tum.de/serom/mrf/AbstractResource.
         owl#SimpleMetaDataExchange"/>
25    <mrf:hasResourceId>
26      <mrf:ResourceId rdf:ID="srId">
27        <mrf:rid rdf:datatype="http://www.w3.org/2001/
             XMLSchema#string">sampleResource</mrf:rid>
28        <mrf:owner rdf:datatype="http://www.w3.org/2001/
             XMLSchema#string">houssam</mrf:owner>
29      </mrf:ResourceId>
30    </mrf:hasResourceId>
31    <mrf:hasProperty>
32      <mrf:Property rdf:ID="Version">
33        <mrf:name rdf:datatype="http://www.w3.org/2001/
             XMLSchema#string">version</mrf:name>
```

```
34          <mrf:type rdf:datatype="http://www.w3.org/2001/
               XMLSchema#string">string</mrf:type>
35        </mrf:Property>
36      </mrf:hasProperty>
37      <mrf:tns rdf:datatype="http://www.w3.org/2001/
           XMLSchema#string">http://de.ibm.com/serom/pool/
           sampleresource</mrf:tns>
38      <mrf:name rdf:datatype="http://www.w3.org/2001/
           XMLSchema#string">SampleResource</mrf:name>
39      <mrf:hasManagementCapability>
40        <mrf:ManagementCapability rdf:ID="Stop">
41          <mrf:name rdf:datatype="http://www.w3.org/2001/
               XMLSchema#string">Stop</mrf:name>
42          <mrf:implClass rdf:datatype="http://www.w3.org
               /2001/XMLSchema#string">com.ibm.de.serom.pool.
               sampleresource.impl.StopImpl</mrf:implClass>
43          <mrf:hasParameter>
44            <mrf:Parameter rdf:ID="After">
45              <mrf:isParameterOf>
46                <mrf:ManagementCapability rdf:ID="Start">
47                  <mrf:hasParameter rdf:resource="#After"/>
48                  <mrf:name rdf:datatype="http://www.w3.org
                       /2001/XMLSchema#string">Start</
                       mrf:name>
49                  <mrf:type rdf:datatype="http://www.w3.org
                       /2001/XMLSchema#string">boolean</
                       mrf:type>
50                  <mrf:implClass rdf:datatype="http://www.
                       w3.org/2001/XMLSchema#string">com.ibm.
                       de.serom.pool.sampleresource.impl.
                       StartImpl</mrf:implClass>
51                </mrf:ManagementCapability>
52              </mrf:isParameterOf>
53              <mrf:isParameterOf rdf:resource="#Stop"/>
54              <mrf:name rdf:datatype="http://www.w3.org
                   /2001/XMLSchema#string">after</mrf:name>
55              <mrf:type rdf:datatype="http://www.w3.org
                   /2001/XMLSchema#string">integer</mrf:type>
56            </mrf:Parameter>
57          </mrf:hasParameter>
58          <mrf:type rdf:datatype="http://www.w3.org/2001/
               XMLSchema#string">boolean</mrf:type>
59        </mrf:ManagementCapability>
```

```
60      </mrf:hasManagementCapability>
61      <mrf:hasManagementCapability rdf:resource="#Start"/>
62      <mrf:hasCapability>
63        <mrf:Capability rdf:ID="Status">
64          <mrf:implClass rdf:datatype="http://www.w3.org
               /2001/XMLSchema#string">com.ibm.de.serom.pool.
               sampleresource.impl.StatusImpl</mrf:implClass>
65          <mrf:name rdf:datatype="http://www.w3.org/2001/
               XMLSchema#string">Status</mrf:name>
66          <mrf:type rdf:datatype="http://www.w3.org/2001/
               XMLSchema#string">string</mrf:type>
67          <mrf:hasParameter>
68            <mrf:Parameter rdf:ID="DetailLevel">
69              <mrf:isParameterOf rdf:resource="#Status"/>
70              <mrf:type rdf:datatype="http://www.w3.org
                   /2001/XMLSchema#string">integer</mrf:type>
71              <mrf:name rdf:datatype="http://www.w3.org
                   /2001/XMLSchema#string">detailLevel</
                   mrf:name>
72            </mrf:Parameter>
73          </mrf:hasParameter>
74        </mrf:Capability>
75      </mrf:hasCapability>
76    </mrf:ManagedResource>
77  </rdf:RDF>
```

Writing manageability interfaces, although being attractive, is not simple or practical and thus removing its value and interest in investing time and effort to model every resource with huge documents that are manually written. Not only MRF could achieve the same result with a hundredfold less code lines but MRF also constructed semantic models that can be used to generate several other artifacts including the client library and the service implementation. Moreover the semantic document can be re-used and extended for other resources.

MRF has the following modules developed:

Semantic Parser: generates an internal representation of the resource by parsing the semantic representation and creating a model. It provides the `model` target.

Resource Artifacts Generator: generates the WSDL document, the web.xml document and several other configuration files for the WSDM implemen-

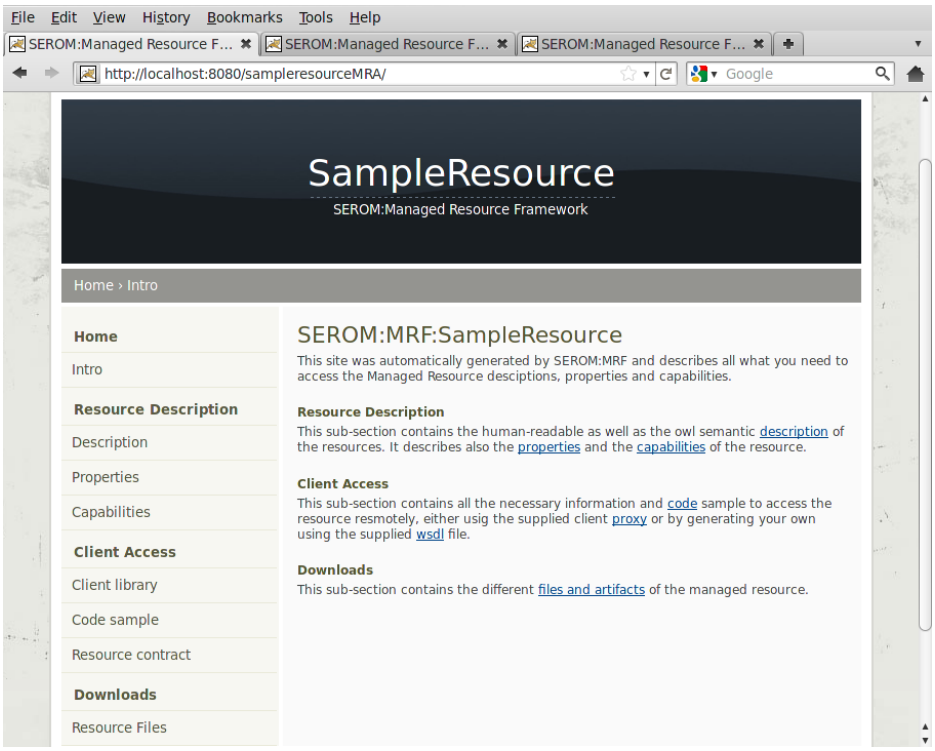

Figure 6.2: Automatically generated online documentation - Introduction

tation muse. It provides the target `wsdl` and internal target `artifacts`.

Online documentation: generates the online documentation of the semantic resource, which include detailed information for every property and capability as well as information on using the resource. This module provides the target `doc`. Figures 6.2, 6.3 and 6.4 show example screenshots of the generated online documentation.

Source Generator: generates the service stub as well as the proxy skeleton for client connection. It also generates two Ant[4] build scripts for every source that can be used to automatically compile the generated classes. The build script reference all the necessary WS-* connectivity libraries (which are also included in MRF) and includes them in the Jar. The code can be directly compiled and deployed and it will work, although invoking non-implemented capabilities will throw the exception `RuntimeException`. This module provides the target `source`.

[4]http://ant.apache.org/

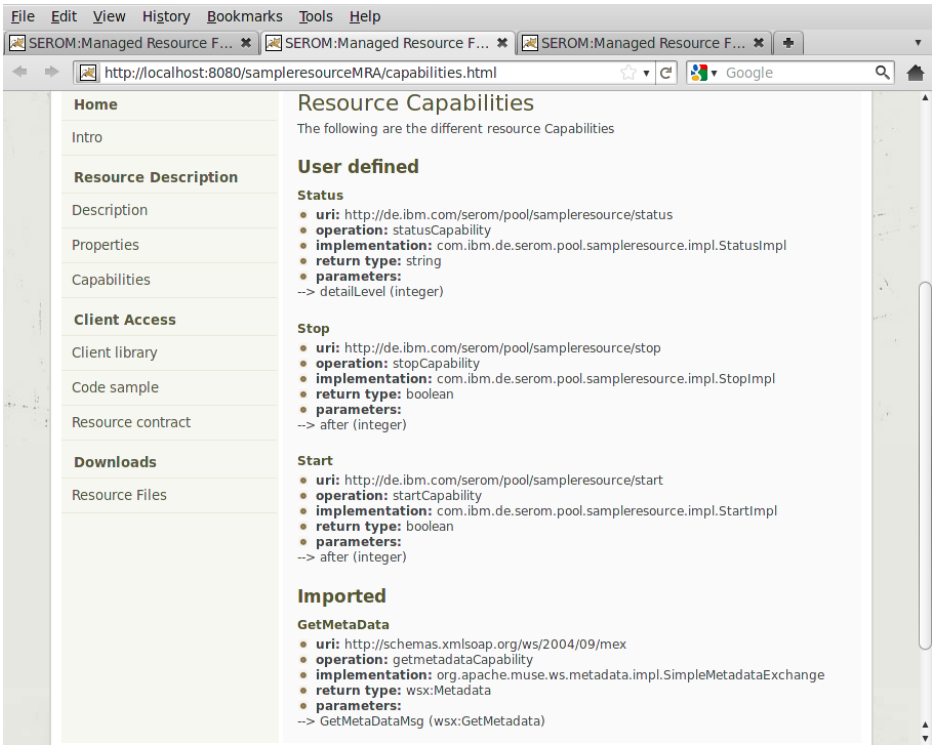

Figure 6.3: Automatically generated online documentation - Capabilities

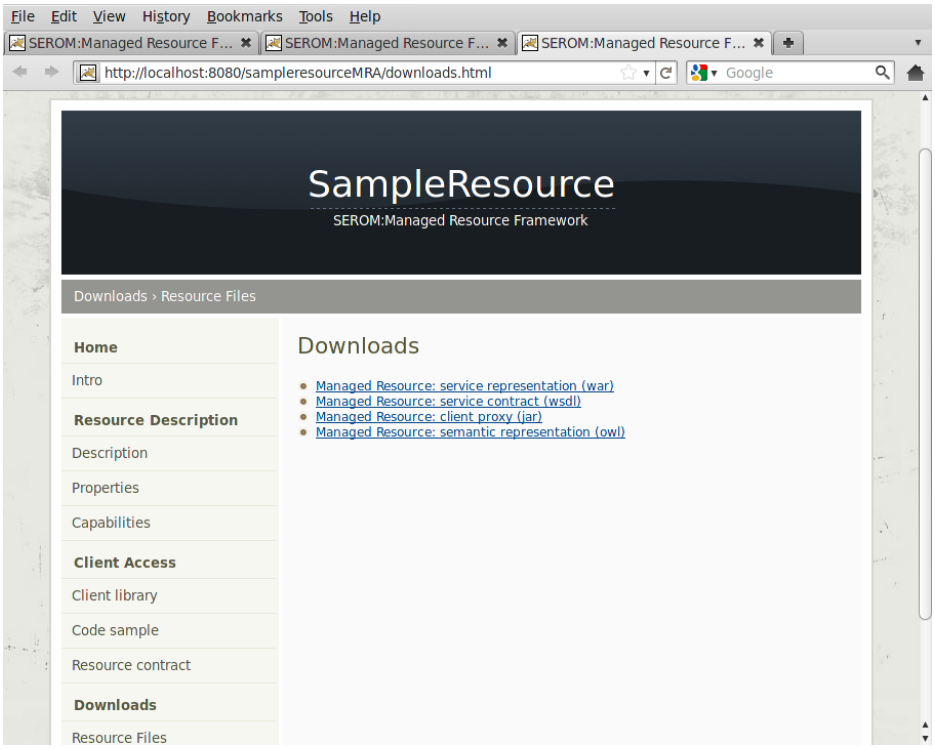

Figure 6.4: Automatically generated online documentation - Downloads

ProxyBuilder: generates the proxy client library that can be immediately incorporated into client projects and used to interface with the resources. The proxy library is self contained and include all the libraries necessary for its proper working. The building uses the JDK building interface which means that a proper JDK needs to be installed for this module to work. It provides the `proxy` target.

MRABuilder: generates the service instantiation. It is similar to the `ProxyBuilder` and uses the same parent module `Builder`. This module provide the `build` target.

Packager: generates the MRA archive. It creates the necessary folder structures and copies to it the necessary libraries and references XML document. This module provides the target `all`.

6.2 Semantic Resource Management Infrastructure

We used the same development environment to develop SRMI as the one used for MRF. The SRMI is intended as an infrastructure to use the generated MRAs in a management context. It is composed of several components that can be stand alone and act independently to different degrees. The most appropriate example is the accounting system that is a self-contained module that implements the WG-RUS specification. This module is more detailed in the following section. Other components were partially implemented to allow the testing of the use cases. Resources states updates were simulated and were scripted to follow the test cases. The most important module that was not implemented is the Execution Planner where the operations to execute should have been reordered and checked for conflicts, they are now executed sequentially as they are. The most important components of SMRI are:

Resource deployer: is a service used to deploy the resources in servlet containers or MRContainers. We provide a servlet container adapter to be extended to add the support of any servlet container. Currently we have the support of Apache Tomcat and all the containers that are based on it. The adapter allows to perform the following standards functions on the container:

- `deployServlet`
- `undeployServlet`
- `deleteServlet`
- `listDeployedServlets`
- `startServlet`
- `stopServlet`
- `getServletContainerStatus`

MRContainer: is an MRA-aware servlet container. It is built on top of a standard servlet container (Apache Tomcat) and adds the support of some operations that make handling MRAs programatically easier. These operations are the same as the container adapter plus the following:

- `getSemanticResourceDescription`
- `getResourceContract`
- `getCapabilitiesList`
- `isCapabilitySupported`
- `getClientProxy`
- `getResourceArtifacts`

Automation Engine: is composed of the set of reasoners and the execution engine. The reasoners are built on top Jena a framework for semantic applications that provides an extensible rule-based inference engine. It allowed us to build custom inference engines with specific rules and combine them to build more specialized reasoners. Jena reasoners can also be used with external reasoners, Pellet, an OWL-DL reasoner is also made available as a reasoner to be used. We use Jena rule language, a simple rule language (described in section 5.4.2) used for the domain logic and the behavioral rules. However, the Jena rule language does not support advanced rules, especially reaction rules, such as the ContractLog framework [86, 87]. Reaction rules are very important especially in a resource management context (see future work section 7.2)

Controller: is the SRMI orchestrator and is implemented as a monolithic component. Its sole purpose is to test the MRF and SRMI and should normally be changed to a more appropriate piece of software depending on the application as some of its parts may be unnecessary (e.g. events routing to appropriate reasoners) or too simple (e.g. internal resource registry).

6.3 FinGrid Accounting and Billing

In the frame of the FinGrid project, we built a general purpose, standard-compliant, Grid accounting system that tracks resource usage for tasks, like management, SLA enforcement or billing. We also built a rule-based billing system, seamlessly integrated with the accounting module. The billing system allows to generate bills from the resource usage according to a set of billing policies. All the components of FinGrid are composed within a SOA model.

The Financial Service Grid (FinGrid) is a project funded by the German Federal Ministry of Education and Research, to develop a Grid architecture to virtualize services and processes in the financial sector and to build banking Grid services based on an accounting and pricing infrastructure through the development of several prototypes. In this context, we pursue research on the necessary components for a financial Grid to better model an industrialization and pricing scheme. We draw the architecture and implemented the resulting accounting and billing services.

For our test purposes, we modeled resources using MRF and wrote basic billing rules and used the SRMI infrastructure to derive bills for the resource usage.

6.3.1 Accounting Service

The purpose of this component is to provide an interface for collecting (upload) and accessing (download) accounting information of the resources consumption and provision. Accounting is the collection of information and data on the usage of resources resulting in a report of the resource consumption and/or provision. The resulting report is generally used for capacity planning, trend analysis, auditing, billing and/or cost allocation. Gathered accounting information is in the form of an XML document complying with the Usage Record (UR) Recommendation [64] proposed by the Open Grid Forum (OGF).

6.3.1.1 Resource Usage Specification Overview

Record Usage Service (RUS) [83] is a stable OGF draft defining a basic infrastructure for accounting and auditing, it specifies the service interface to normalize operations on the accounting information of the resource usage as described by the Usage Record (UR) specification. The UR document can be maintained either centrally or in a distributed fashion. The RUS would permit the upload of usage records or the extraction of necessary information and possibly the aggregation of resource usage data. The RUS service interface definition is based on WS-I Basic Profile 1.0. We are using our implementation of the RUS for managing the Usage Records XML documents. Our RUS implementation also supports secured access based on the implementation of WS-Security.

6.3.1.2 Usage Records

Gathered accounting information will be in the form of an XML document complying with the Usage Record (UR) Standard defined by the OGF. The UR standard specifies a common format for representing resource consumption data. It contains basic accounting and usage information gathered at the local Grid sites. The standard defines an XML Schema for the UR that is conforming to the W3C XML Schema Specification. The Usage Record is an XML document that contains some required fields called base properties that are necessary for identifying user and the job as well as some items that are necessary for proper measurements. The usage metrics are divided into three categories: base properties, differentiated properties, and extensions. The base properties define the most common metrics necessary for proper accounting, such as user and job identification. The differentiated properties are task-related measurement metrics that every site can accommodate to its particular needs. The last category are the extensions, which are a set of metrics specific to the site and tasks that can be added to the UR specification. The set of required items to be accounted is, of course, site and situation dependent.

6.3.1.3 Metrics

Metrics are the measurable values gathered from the resources and represented in the UR document. For the purpose of our implementation, we used the standard metrics of the UR recommendation as well as a set of

custom metrics as extensions. The custom metrics offer capabilities that were necessary to our use cases but that are not provided by the recommendation. They relate to the categorization of clients and resources and to the representation of the cost for the resource usages.

6.3.1.4 Collector Agent

We use collectors to gather accounting information from the local resources. A Collector is resource environment-specific agent that collects accounting data generated at the level of the resources. This data is then submitted to the Accounting Service in the form of UR documents. There is no proper standard for usage logs at the level of the resource manager. The collectors' task is to extract the relevant data from the generated logs and transform it into compliant OGF UR-WG document.

6.3.1.5 UR Repository

The RUS standard does not specify how the UR should be physically stored and leaves this decision to the implementer. We opted for the most natural way to store XML documents which is a native XML database. Our choice settled on Apache Xindice[5] for persistent storage. Xindice is a Java implementation of a native XML database. The advantage of using a native XML database is that we don't have to worry about mapping our XML document to a specific data structure to store in a normal RDBMS for instance. We store documents as XML and we retrieve them as XML. Xindice supports XPath 1.0 for querying XML documents (very handful for aggregating results from different stored Usage Records) and XUpdate 1.0 for updating XML documents. Xindice comes also with an implementation of XML:DB API, so all operations on the XML database can be performed using Java code. To interact with and access the native XML database, we used an implementation of the XML:DB API for Java development. XML:DB is vendor neutral and supports a wide array of databases. Our implementation supports WS-Security as security layer for authentication and authorization.

[5] http://xml.apache.org/xindice/

6.3.1.6 Resource Usage Specification Details

Record format

The RUS specification has chosen the Usage Record specification as the resource usage accounting format. We are using the UR schema file to validate all the usage files before storing them in the database. If they are not valid an invalid record exception is thrown to the client.

Mandatory elements

The Usage Record specification describe almost all elements as being optional. However some Grid sites would require that some information should always be recorded. RUS offer the possibility to be configured to require the presence of some mandatory elements. Every record is checked against those elements. If any is missing a mandatory element is missing exception is thrown to the client. RUS specification do not specify how to implement this. At the moment, the element are stored in a configuration file that is loaded at the service initialization.

Record uniqueness

Every record stored has a unique id that can be generated by the RUS service or provided by the client. There is no specific format for the id. Currently, the id is provided by the client, however depending on the needs we can have ids generated by the service following a specific pattern. In case the client provides an existing id, a duplicate id exception is thrown.

Record history

RUS specification suggest that some type of record history should be maintained, currently we store the creation time and the user who created the record.

Operation result

All RUS operations return the `rus:operationResult` element that contains a boolean "Status" element that is true if no error were reported. It contains also an optional "Processed" element that contains the number of records successfully processed. And finally a sequence of faults that gives explicit details of the errors that occurred.

RUS operations

We implemented all the functionalities of RUS as described in the RUS-WG specification:

`RUS::insertUsageRecords` The `insertUsageRecords` port type enables users to populate usage files into usage repository.

`RUS::extractSpecUsageRecords` Enables the client to find out usage records using record identifier as keywords.

`RUS::extractUsageRecords` Enables the client to find out usage records relating to certain search criteria.

`RUS::extractRecordIds` Enables the client to find all usage records relating to a more complicated set of requirements, returning just the record identifiers and not the full usage records.

`RUS::modifySpecUsageRecords` This port type allows users to modify a set of usage records identified by record identity with XUpdate expression. The charge service, for example, could make use of this port type to insert charge information to a usage record as with usage information calculated.

`RUS::modifyUsageRecords` This operation will modify a set of Usage Records according to a XUpdate expression. The resource manager, for example, may use this port type to modify resource-specific property (e.g. `urf:MachineName`) values.

`RUS::replaceUsageRecords` The `replaceUsageRecords` replaces records held in the RUS.

`RUS::deleteSpecificUsageRecords` Enables the client to delete usage record(s) from a usage file.

`RUS::deleteUsageRecords` Enables the client to delete all usage records with the specified `RUSRecordIds`.

`RUS::listMandatoryUsageRecordElements` Enables the client to retrieve a list of the usage record elements required by this RUS implementation.

6.3.2 Billing Service

Billing is the process of generating bills from the resource usage data using generally a set of predened billing policies. The bill can be in real money or it can use more abstract notions depending on the site general policies. We should note here that the billing service does not preconize the use of a specific economic model. In fact it is independent from the economic model to be used. FinGrid Billing permits the denition, storage and manipulation of billing rules through a set of services.

6.3.2.1 Billing Rules

The billing system supports two types of billing schemes: duration-specific and one-off cost. The duration-specific type applies the billing rule based on the usage duration whereas the one-off cost type is concerned with the proper usage of the resources. The following is an example of a duration specific rule, it specifies 0.0002 unit (Euro, Pounds...), for every second of usage of the described resource and user type:

```
when EVENT = "VM Assignment", CLIENT_TYPE = "Platinum",
    RESOURCE_TYPE = "BLADE Type 4", RESOURCE_AGE < 240 * 60 * 60,
     SERVICE_LEVEL = "Platinum" then COST_PER_SECOND = 0.0002
```

6.3.2.2 Billing architecture

Our billing model separates the logic of the billing service from the data or the rules, and uses a *hot-deployment* scheme where users can manage the billing rules and deploy them without restarting the billing service. This separation make our services accessible to a wide audience of users. The kind of users that would generally specify the billing policies but would not be necessarily a developer.

6.3.2.3 Billing Portal

In the billing portal we can mark single or multiple usage records for billing, generate usage bills and export the bills as XML files. However, sometimes we would need a higher degree of control over which data to be billed. Our implementation is powerful enough to support aggregation of usage records over any combination of usage metrics elements as well as ranges of times. It also supports aggregation of usage data using complex XPath queries, giving the user a high degree of liberty in composing usage records. Records of virtual organizations or some specific parts of job can be then created using aggregation for a detailed billing document.

6.3.2.4 FinGrid Billing Service

The billing service provides port types to generate bills from usage data according to predefined billing rules. The records can then be marked as billed. We can also bill records created from aggregation results and save the aggregation for later reference. Our billing service implementation parses the usages records and extract the relevant usage metrics for the billing process. It feeds this data to the inference engine working memory and fires the evaluation process. The bills generation process is not automated and needs to be initiated. However, we do have a command line version of our billing service that can be used with cron for instance for automated and scheduled bills generation.

6.3.3 Use case

Goal

This test case demonstrates the autonomic service management capabilities of the system and the use of the MRF and SRMI.

Summary

Usage bills are produced from resources usage records according to a set of billing rules.

Actors

- The resource manager: Models resources.
- The billing rules manager: Creates billing rules.
- The billing process initiator: Initiate bills calculations.

Preconditions

- Resources are modeled using MRF.
- Resources are capable of producing usage record data (through a specific RUS management capability)
- Existing billing rules that matches some resource usage.

Triggers

The billing process is manually initiated whenever there is need for the bills to be produced.

Basic Course of Events

Once the billing process initiated, the usage record documents are parsed and the proper rules are applied. The rules call some Java function that produce bills in the form of reports.

Post conditions

Bills are produced and the usage records documents used for their production are marked as billed.

6.4 Green IT

Green IT is a fictional setting aimed at demonstrating the MRF and SRMI. It has the objective of controlling the overheating servers by lowering their load or moving their load to other similar server and turning them off. It is based on a project about energy-aware resource management in IBM Böblingen lab.

6.4.1 Background and objective

Energy efficiency was never an issue for the data centers, it was always "performance at any cost", the result is that since more than 15 years, the performance of supercomputers saw a 10.000-fold increase compared to a mere 300-fold increase for performance per watt [33]. Taking only into account the server part (computational part) and not the data storage and communication (storage and networking parts), electricity use doubled from 2000 to 2005, representing an aggregate annual growth rate of 16% per year worldwide. If we consider the cooling equipment of the servers, the total electricity consumption was 120 billion kWh in 2005 worldwide, totaling a utility bill of 7.2 billion USD (2006 dollars) [61]. With an emission of 0,35 kg of CO2 per kWh if produced from Gas burning plants or 1 kg if produced from Coal burning plants, the energy consumed by the data centers confront us with a considerable environmental issue especially that forecasts from [61] predicted that total electricity used by servers by 2010 would be 76% higher than it was in 2005.

Green IT, or Green Computing, represents the environmentally aware efforts for an efficient use of computing resources. It has three big components: reducing the use of hazardous materials, improving the recyclability of the computing materials and reducing the energy used by those resources. Green IT is not a new concept, however it saw an increasing interest lately, especially with the soaring prices for energy and an increased awareness of environmental issues. It has its origins in the "Energy Star" labeling program launched by the U.S. Environmental Protection Agency to promote energy efficiency in several equipment types. Several other labels and governmental regulations followed, such as the Swedish TCO program or the European directives 2002/95/EC, concerning the reduction of hazardous substances, and 2002/96/EC concerning the waste electrical and electronic equipment. There exist also a number of industrial consortia and organization promoting best practices and pushing for a "greener" IT.

In this context, we had a project with IBM Böblingen lab about Green IT, where the objective was to use heat sensors from the servers in a data center to harmonize the temperature in the data center by bringing the individual temperatures of the over-heating servers down to near the average temperature and this by moving the servers load around. In our fictive use case, we detect over-heating of a server and we move some running applications to another idle server.

6.4.2 Use case

Goal

This test case demonstrates the autonomic service management capabilities of the system and the use of the IT business process guidelines, namely the ITIL event management, incident management and problem management.

Summary

A server overheats. This is detected and the running job(s) are stopped/-moved to other servers.

Actors

- The resource manager: Models resources.
- The Green IT rules manager: Creates rules.

Preconditions

- Resources are modeled using MRF.
- There exist similar servers able to run the same jobs.
- Existing rules that matches some server state.

Triggers

During the normal record operations of resources usages, a trigger is launched when a record operation causes an operational rule to fire. The rule detects the heating of a server. For our testing purposes, we are simulating resources and manually inducing an overheat by modifying the server temperature value.

Basic course of events

Event: It starts with an event that would cause the halt of the normal functioning of a running job on a server.

Detection and record: The event is launched by the Green IT Monitor (GIM) in response to a match of an operational rule. Namely, the server acceptable temperature was exceeded. An incident record is opened and all necessary information related to the incident is registered.

Classification and matching: The incident is classified according to different criterion such as the priority (derived in this example from the job priority and the the operational rule priority), origin, scope, effects or type of resource/job/operation. An RDF graph that represents the classification is constructed for the purpose of matching this incident with previously occurring incidents or with predefined solutions to similar incident classification. In the case that there is no directly induced solution from the incident classification, a linked problem error is created and a problem management process activity is initiated.

Investigation and diagnosis: A solution (or course of action) is identified at this stage. The server should have its load lowered and hence the running job(s) should be stopped or better, moved. Following the job semantic description and its requirements semantic description, another server is identified that shall run the job or what's remaining of it. A course

of action is constructed in the form of a workflow and is escalated to the execution engine for execution.

Resolution and recovery: The execution engine executes the workflow. The job(s) are stopped and restarted on other servers. The actions are performed through the resources (jobs, services, physical resources) management endpoints. The incident is marked as resolved and closed.

Post conditions:

The KB is enhanced with new incident resolution records.

6.5 Summary

We implemented SEROM (SERvice Oriented Management) as a prototype for the concepts discussed in this work. We implemented MRF as a modular, plug-in based software to insure the future extendability of the framework. SRMI was developed as a set of independent services, some of which were themselves generated using the MRF. We used two use cases to test SEROM, a financial Grid Computing setting and a Green IT end-to-end automation setting where the behavior of both MRF and SRMI matched our expectations. The outlook of MRF and SRMI is promissing with many of their components being used in Universities and companies and with ongoing projects being developed based on SEROM (see section 7.1).

Chapter 7

Conclusion

7.1 Summary

Having pervasive computing induce a considerable amount of complexity that system architects need to deal with. Growing number of interconnected systems and services as well their increasing complexity makes the management of a heterogeneous pool of IT resource a real nightmare. It is clear that this complexity is the limiting factor for development, deployment and management of large-scale/complex systems. A solution would be to leverage the IT infrastructure by enabling it to manage itself. This is the aim of an Autonomic System, where the human administrator shifts from managing the system directly to defining the policies and rules as input to the system so it can achieve a state of self-management. In this work we aimed at laying the foundation, and providing a solution for an intelligent policy-based self-manged system. By intelligent we mean showing sound judgment and rationality, capable of providing an intelligent solution to system complexity. By policy-based we infer that the input to the system are a set of policies written by a human administrator. The result is a self-managed adaptive and autonomous system that is easily configurable and achieves the predefined human service objectives.

Our solution was based on two principles: simplicity and automation. We brought simplicity by providing a mechanism to model resource in a standard and uniform way using an expressive language for the purpose of generating software components that would provide a standard management interface to access and manage the resources represented by these components. The simplicity is apparent in modeling the resources, in the process by which the service artifacts were generated and the end-result: simple manageable components. For the automation aspect, we aimed at

automatizing everything from the component service generation to the resource management. The MRF automatically generates hundred of lines of code and builds integrated archives that are ready to deploy, and in the SRMI, the DSIEs are used to automatize management tasks based on predefined rules.

We implemented SEROM (SERvice Oriented Management) as a prototype for the concepts discussed in this work. Due to the its modularity, several of its constituents are independent and can be used as they are. As an example, parts of SMRI (specifically RUS and the accounting system) were deployed in projects at the University of Marburg, at Deutsch Bank UK and at the IBM Böblingen Laboratory in a Grid related project due to the fact that RUS was the only stable implementation of the OGF WG-RUS specification [83] at the time of its development. There is also an interest to use the SMRI in the context of a "green" data center at J.T. Watson research laboratory. Due to the versatility of the MRF, there is also an interest for it to be used as rapid prototyping system for system architectures for the purpose of realistic testing of deployment at the IBM Böblingen Lab, and a project is under planning to use the MRF for resource provisioning in the Cloud [49].

7.2 Future Work

7.2.1 MRF project management interface

One of the works related to the MRF but that is not mature enough is the eclipse based project management interface for MRF. This eclipse plug-in is used to manage the process from the resource definition to the resource monitoring passing by service generation and resource deployment. We rely on eclipse's rich client platform (RCP) to build the management interface. It is composed of three editors: Java, OWL and XML editors, two perspectives: Resource-centered project management and deployment and monitoring perspectives, as well as two builders: A Java builder and an MRF builder used to build the MRA. There will also be wizards to help the developers define resource, generate services and deploy resources.

7.2.2 Reaction rules

Rules definition and management is not a central focus of this work. However, we think that adding the support for reaction rules would add significant value due to their intrinsic nature and the kind of behaviors expected

in an IT resource management setting. Event-condition-action (ECA) rules are a natural candidate for that. Using ECA rules, we can automatically perform management actions when some event occurs and the conditions hold. The integration of ContractLog [87] with the SRMI is one the most attractive options to achieve that.

7.2.3 Cloud Computing

We are currently working on a way for prototyping architectures based on the generation of service representations of resources. This generated infrastructure can be used to rapidly build on-demand settings for application/scenario requirements in a Cloud Computing context where such requirements can be as diverse as the applications running on the Cloud. The resources used to build the infrastructure are semantically described to capture their properties and capabilities. We use the MRF to automatically generate service descriptions with an added manageability interface from these semantic description. These services are then ready for deployment in a Cloud context. A paper entitled "Rapid Prototyping of Architectures on the Cloud Using Semantic Resource Description" describing this aspect of our work is accepted for publication.

Bibliography

[1] *1st International Conference on Autonomic Computing (ICAC 2004), 17-19 May 2004, New York, NY, USA* (2004), IEEE Computer Society.

[2] AKKIRAJU, R., FARRELL, J., MILLER, J., NAGARAJAN, M., SCHMIDT, M., SHETH, A., AND VERMA, K. Web service semantics - wsdl-s. Tech. rep., World Wide Web Consortium, November 2005.

[3] AMAZON. Elastic compute cloud (amazon ec2). `http://aws.amazon.com/ec2/`, July 2010. [Online; accessed July-2010].

[4] ANTONY, J., ET AL. Wsdm/ws-man reconciliation. Tech. rep., IBM, August 2006.

[5] BABAOGLU, Ö., JELASITY, M., AND MONTRESOR, A. Grassroots approach to self-management in large-scale distributed systems. In *UPP* (2004), J.-P. Banâtre, P. Fradet, J.-L. Giavitto, and O. Michel, Eds., vol. 3566 of *Lecture Notes in Computer Science*, Springer, pp. 286–296.

[6] BANKS, T. Web services resource framework (wsrf) - primer v1.2. Tech. rep., OASIS, May 2006.

[7] BATTLE, S., BERNSTEIN, A., BOLEY, H., GROSOF, B., GRUNINGER, M., HULL, R., KIFER, M., MARTIN, D., MCILRAITH, S., MCGUINNESS, D., SU, J., AND TABET, S. Semantic web services framework (swsf) overview. Tech. rep., World Wide Web Consortium, September 2005.

[8] BATTLE, S., BERNSTEIN, A., BOLEY, H., GROSOF, B., GRUNINGER, M., HULL, R., KIFER, M., MARTIN, D., MCILRAITH, S., MCGUINNESS, D., SU, J., AND TABET, S. Semantic web services language (swsl). Tech. rep., World Wide Web Consortium, September 2005.

[9] BATTLE, S., BERNSTEIN, A., BOLEY, H., GROSOF, B., GRUNINGER, M., HULL, R., KIFER, M., MARTIN, D., MCILRAITH, S., MCGUINNESS, D., SU, J., AND TABET, S. Semantic web services ontology (swso). Tech. rep., World Wide Web Consortium, September 2005.

[10] BECKETT, D., AND BERNERS-LEE, T. Turtle - terse rdf triple language. `http://www.w3.org/TeamSubmission/turtle/`, January 2004. [Online; accessed July-2010].

[11] BERMAN, F., HEY, A. J. G., AND FOX, G. *Making the Global Infrastructure a Reality*. Advanced Information and Knowledge Processing. John Wiley and Sons Ltd, 2003.

[12] BERNERS-LEE, T. Notation 3. `http://www.w3.org/DesignIssues/Notation3`, March 2006. [Online; accessed July-2010].

[13] BERNERS-LEE, T., HENDLER, J., AND LASSILA, O. The semantic web. *Scientific American* (May 2001).

[14] BIRON, P. V., PERMANENTE, K., AND MALHOTRA, A. Xml schema part 2: Datatypes. Tech. rep., World Wide Web Consortium, October 2004.

[15] BRACHMAN, R. J., AND LEVESQUE, H. J. *Knowledge Representation and Reasoning*. Morgan Kaufman, 2004.

[16] BULLARD, V., MURRAY, B., AND WILSON, K. An introduction to wsdm. Tech. rep., OASIS Official Committee Specification, February 2006.

[17] BULLARD, V., AND VAMBENEPE, W. Web services distributed management: Management using web services (muws 1.1) part 1. Tech. rep., OASIS Web Services Distributed Management TC, August 2006.

[18] BULLARD, V., AND VAMBENEPE, W. Web services distributed management: Management using web services (muws 1.1) part 2. Tech. rep., OASIS Web Services Distributed Management TC, August 2006.

[19] CANNON, D., AND WHEELDON, D. *ITIL – Service Operation*. TSO, May 2007.

[20] CARROLL, J. J., DICKINSON, I., DOLLIN, C., REYNOLDS, D., SEABORNE, A., AND WILKINSON, K. Jena: implementing the semantic web recommendations. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters* (New York, NY, USA, 2004), WWW Alt. '04, ACM, pp. 74–83.

[21] CHASE, J. S., ANDERSON, D. C., THAKAR, P. N., VAHDAT, A., AND DOYLE, R. P. Managing energy and server resources in hosting centres. In *SOSP* (2001), pp. 103–116.

[22] CHESS, D. M., HANSON, J. E., PERSHING, J. A., AND WHITE, S. R. Prospects for simplifying itsm-based management through self-managing resources. *IBM Systems Journal 46*, 3 (2007), 599–608.

[23] CHESS, D. M., SEGAL, A., WHALLEY, I., AND WHITE, S. R. Unity: Experiences with a prototype autonomic computing system. In *ICAC* [1], pp. 140–147.

[24] CHRISTENSEN, E., CURBERA, F., MEREDITH, G., AND WEERAWARANA, S. Web services description language (wsdl) 1.1. `http://www.w3.org/TR/wsdl`, March 2001. [Online; accessed July-2010].

[25] DAMIANOU, N., DULAY, N., LUPU, E., AND SLOMAN, M. The ponder policy specification language. In *POLICY* (2001), M. Sloman, J. Lobo, and E. Lupu, Eds., vol. 1995 of *Lecture Notes in Computer Science*, Springer, pp. 18–38.

[26] DAVIS, D., MALHOTRA, A., WARR, K., AND CHOU, W. Web services metadata exchange (ws-metadataexchange). Tech. rep., World Wide Web Consortium, Avril 2011.

[27] DAVIS, R., AND KING, J. An overview of production systems. 300–332.

[28] DE BRUIJN, J., BUSSLER, C., DOMINGUE, J., FENSEL, D., HEPP, M., KIFER, M., KNIG-RIES, B., KOPECKY, J., LARA, R., OREN, E., POLLERES, A., SCICLUNA, J., AND STOLLBERG, M. Web service modeling ontology (wsmo). Tech. rep., WSMO, April 2005.

[29] DE BRUIJN, J., LAUSEN, H., KRUMMENACHER, R., POLLERES, A., PREDOIU, L., KIFER, M., AND FENSEL, D. The web service modeling language (wsml). Tech. rep., WSMO, October 2005.

[30] DEVARAKONDA, M. V., CHESS, D. M., WHALLEY, I., SEGAL, A., GOYAL, P., SACHEDINA, A., ROMANUFA, K., LASSETTRE, E., TETZLAFF, W. H., AND ARNOLD, B. Policy-based autonomic storage allocation. In *DSOM* (2003), M. Brunner and A. Keller, Eds., vol. 2867 of *Lecture Notes in Computer Science*, Springer, pp. 143–154.

[31] DULAY, N., DAMIANOU, N., LUPU, E., AND SLOMAN, M. A policy language for the management of distributed agents. In *AOSE* (2001), M. Wooldridge, G. Weiß, and P. Ciancarini, Eds., vol. 2222 of *Lecture Notes in Computer Science*, Springer, pp. 84–100.

[32] ED., T. M. extensible access control markup language (xacml) version 2.0. `http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf`, February 2005. [Online; accessed July-2010].

[33] FENG, W.-C. Making a case for efficient supercomputing. *Queue 1* (October 2003), 54–64.

[34] FENSEL, D., AND BUSSLER, C. The web service modeling framework wsmf. Tech. rep., Vrije Universiteit Amsterdam, 2002.

[35] FIELDING, R. T. Architectural styles and the design of network-based software architectures, 2000.

[36] FINDLER, N. V., Ed. *Associative Networks: The Representation and Use of Knowledge by Computers*. Academic Press, Inc., Orlando, FL, USA, 1979.

[37] FINDLER, N. V., Ed. *On the Epistemological Status of Semantic Networks* (Orlando, FL, USA, 1979), Academic Press, Inc.

[38] FORGY, C. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence 19* (1982), 17–37.

[39] FOSTER, I. What is the grid? a three point checklist. *GRIDToday* (July 2002).

[40] FREGE, G. Über sinn und bedeutung. *Zeitschrift für Philosophie und philosophische Kritik* (1892), 25–50.

[41] GENTZSCH, W., IWANO, K., JOHNSTON-WATT, D., MINHAS, M. A., AND YOUSIF, M. S. Self-adaptable autonomic computing systems: An industry view. In *DEXA Workshops* (2005), IEEE Computer Society, pp. 201–205.

[42] GOMEZ-PEREZ, A., CORCHO, O., AND FERNANDEZ-LOPEZ, M. *Ontological Engineering : with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web*. Advanced Information and Knowledge Processing. Springer, 2004.

[43] GOOGLE. Google app engine. `http://code.google.com/appengine/`, July 2010. [Online; accessed July-2010].

[44] GRAHAM, S., HULL, D., AND MURRAY, B. Web services base notification 1.3 (ws-basenotification). Tech. rep., OASIS, October 2006.

[45] GRAHAM, S., AND TREADWELL, J. Web services resource properties 1.2 (ws-resourceproperties). Tech. rep., OASIS, June 2004.

[46] GROSOF, B. N., HORROCKS, I., VOLZ, R., AND DECKER, S. Description logic programs: combining logic programs with description logic. In *WWW* (2003), pp. 48–57.

[47] GRUBER, T. R. A translation approach to portable ontology specifications. *In: Knowledge Acquisition, An International Journal of Knowledge Acquisition for KnowledgeBased Systems*, 5 (1993), 199–220.

[48] GUARINO, N., POLI, R., AND GRUBER, T. R. Toward principles for the design of ontologies used for knowledge sharing. In *In Formal Ontology in Conceptual Analysis and Knowledge Representation, Kluwer Academic Publishers, in press. Substantial revision of paper presented at the International Workshop on Formal Ontology* (1993).

[49] HAITOF, H. Rapid prototyping of architectures on the cloud using semantic resource description. Workshop on Cloud Computing: Project and Initiatives, Euro-par.

[50] HAITOF, H., WHELE, H.-D., AND GERNDT, M. Fingrid accounting and billing. In *Business Process, Services Computing and Intelligent Service Management* (Leipzig, 2009), vol. 147, GI, pp. 167–178.

[51] HAMILTON, E. *The Greek Way*. W. W. Norton & Co., 1993.

[52] HEVNER, A. R., MARCH, S. T., PARK, J., AND RAM, S. Design science in information systems research. *MIS Quarterly 28*, 1 (2004), 75–105.

[53] HORN, P. *Autonomic Computing: IBM Perspective on the State of Information Technology*. IBM, 2001.

[54] HORROCKS, I., PATEL-SCHNEIDER, P. F., BOLEY, H., TABET, S., GROSOF, B., AND DEAN, M. Swrl: A semantic web rule language combining owl and ruleml. `http://www.w3.org/Submission/SWRL/`, May 2004. [Online; accessed July-2010].

[55] INSTITUTE, I. G. *Cobit 4.1*. ISA, 2007.

[56] IQBAL, M., AND NIEVES, M. *ITIL – Service Strategy*. TSO, 2007.

[57] JACKSON, P. *Introduction to Expert Systems*. Addison-Wesley Publishing Company, 1986.

[58] JEFFERY, K. G. Knowledge, information and data. Tech. rep., Information Technology Department, CLRC, September 1999.

[59] KEPHART, J. O., AND CHESS, D. M. The vision of autonomic computing. *Computer magazine* (January 2003), 41–50.

[60] KOLARI, P., DING, L., GANJUGUNTE, S., JOSHI, A., FININ, T. W., AND KAGAL, L. Enhancing web privacy protection through declarative policies. In *POLICY* (2005), IEEE Computer Society, pp. 57–66.

[61] KOOMEY, J. G. Estimating total power consumption by servers in the u.s. and the world. Tech. rep., 2007.

[62] KOWALSKI, R., Ed. *Logic for Problem Solving*. Elsevier, 1979.

[63] LACY, S., AND MACFARLANE, I. *ITIL – Service Transition*. TSO, May 2007.

[64] LEPRO-METZ, R., JACKSON, S., MCGINNIS, L., AND MACH, R. *Usage record - Format Recommendation*. Open Grid Forum, 2007.

[65] LLOYD, V., RUDD, C., AND TAYLOR, S. *OCG Books ITIL - Service Design*. TSO (The Stationery Office), 2007.

[66] LUPU, E., SLOMAN, M., DULAY, N., AND DAMIANOU, N. Ponder: Realising enterprise viewpoint concepts. In *EDOC* (2000), IEEE Computer Society, pp. 66–75.

[67] MACKENZIE, C. M., LASKEY, K., MCCABE, F., BROWN, P., AND METZ, R. Oasis reference model for service oriented architecture v 1.0. Tech. rep., OASIS Official Committee Specification, August 2006.

[68] MAGUIRE, T., SNELLING, D., AND BANKS, T. Web services servicew group 1.2 (ws-servicegroup). Tech. rep., OASIS, April 2006.

[69] MANNA, Z., AND WALDINGER, R. J. Fundamentals of deductive program synthesis. *IEEE Trans. Software Eng. 18*, 8 (1992), 674–704.

[70] MARCH, S. T., AND SMITH, G. F. Design and natural science research on information technology. *Decis. Support Syst. 15* (December 1995), 251–266.

[71] MARTIN, D., BURSTEIN, M., HOBBS, J., LASSILA, O., MCDERMOTT, D., MCILRAITH, S., NARAYANAN, S., PAOLUCCI, M., PARSIA, B., PAYNE, T., SIRIN, E., SRINIVASAN, N., AND SYCARA, K. Owl-s: Semantic markup for web services. Tech. rep., White Paper, DARPA Agent Markup Language (DAML) Program.

[72] MCBRIDE, B. Jena: a semantic web toolkit. *Internet Computing, IEEE 6*, 6 (2002), 55–59.

[73] MCCOLLUM, R., MURRAY, B., AND REISTAD, B. Web services for management (ws-management) specification. Tech. rep., DMTF, 2008.

[74] MOORE, R. C. The role of logic in knowledge representation and commonsense reasoning. In *In AAAI* (1982), pp. 428–433.

[75] MORAN, M., AND ZAREMBA, M. Wsmx architecture. Tech. rep., WSMO, June 2004.

[76] MORDANI, R., ET AL. *Java Servlet 2.5 Specification*. SUN MICROSYSTEMS, INC., July 2007.

[77] MOTIK, B., SATTLER, U., AND STUDER, R. Query answering for owl-dl with rules. In *International Semantic Web Conference* (2004), S. A. McIlraith, D. Plexousakis, and F. van Harmelen, Eds., vol. 3298 of *Lecture Notes in Computer Science*, Springer, pp. 549–563.

[78] MOTTA, E., DOMINGUE, J., CABRAL, L., AND GASPARI, M. Irs-ii: A framework and infrastructure for semantic web services. Tech. rep., Knowledge Media Institute at the Open University, Milton Keynes, UK.

[79] NADALIN, A., GOODNER, M., GUDGIN, M., BARBIR, A., AND GRANQVIST, H. Ws-securitypolicy 1.3. `http:`

//docs.oasis-open.org/ws-sx/ws-securitypolicy/v1.3/os/ws-securitypolicy-1.3-spec-os.html, February 2009. [Online; accessed July-2010].

[80] NAGYPAL, G. Methodologies for ontology engineering. *Semantic Web Services: Concepts, Technologies, and Applications* (2007), 107–134.

[81] NEWELL, A. The knowledge level. *Artif. Intell. 18*, 1 (1982), 87–127.

[82] NILSSON, N. J. *Principles of Artificial Intelligence*. Morgan Kaufmann, 1982.

[83] OPEN GRID FORUM. *Resource Usage Service (RUS) based on WS-I Basic Profile 1.0*, August 2005.

[84] ORTEGA Y GASSET, J. *El tema de nuestro tiempo*, 2 ed. Colección Austral. Espasa-Calpe, 1939.

[85] PARASHAR, M., AND HARIRI, S. *Autonomic Computing: Concepts, Infrastructure and Applications*. CRC Press, 2007.

[86] PASCHKE, A. Rbsla a declarative rule-based service level agreement language based on ruleml. In *CIMCA/IAWTIC* (2005), IEEE Computer Society, pp. 308–314.

[87] PASCHKE, A., BICHLER, M., AND DIETRICH, J. Contractlog: An approach to rule based monitoring and execution of service level agreements. In *RuleML* (2005), A. Adi, S. Stoutenburg, and S. Tabet, Eds., vol. 3791 of *Lecture Notes in Computer Science*, Springer, pp. 209–217.

[88] PATEL-SCHNEIDER, P., HAYES, P., AND HORROCKS, I. Owl web ontology language; semantics and abstract syntax. Tech. rep., World Wide Web Consortium, November 2002.

[89] QUILLIAN, M. Semantic memory. *Semantic Information Processing* (1968), 227–270.

[90] REICHGELT, H. *Knowledge Representation: An AI Perspective*. Alex Publishing Corporation, 1991.

[91] REYNOLDS, D. Jena 2 inference support. http://jena.sourceforge.net/inference/index.html, March 2010. [Online; accessed July-2010].

[92] RINGLAND, G. A., AND DUCE, D. A. *Approaches to Knowledge Representation: An Introduction*. Knowledge-Based and Expert Systems. Research Studies Press LTD., 1988.

[93] SIRIN, E., PARSIA, B., GRAU, B. C., KALYANPUR, A., AND KATZ, Y. Pellet: A practical owl-dl reasoner. *J. Web Sem. 5*, 2 (2007), 51–53.

[94] SPALDING, G. *ITIL – Continual Service Improvement*. TSO, May 2007.

[95] SRINIVASAN, L., AND BANKS, T. Web services resource lifetime 1.2 (ws-resourcelifetime). Tech. rep., OASIS, April 2006.

[96] STUDER, R., BENJAMINS, V. R., AND FENSEL, D. Knowledge engineering: Principles and methods, 1998.

[97] STUDER, R., GRIMM, S., AND ABECKER, A. *Semantic Web Services: Concepts, Technologies, and Applications*. Springer, 2007.

[98] SUN/ORACLE. Jar file specification. `http://download.oracle.com/javase/6/docs/technotes/guides/jar/jar.html`, March 2010. [Online; accessed July-2010].

[99] THOMPSON, H. S., BEECH, D., MALONEY, M., AND MENDELSOHN, N. Xml schema part 1: Structures. Tech. rep., World Wide Web Consortium, October 2004.

[100] USZOK, A., BRADSHAW, J. M., JEFFERS, R., SURI, N., HAYES, P. J., BREEDY, M. R., BUNCH, L., JOHNSON, M., KULKARNI, S., AND LOTT, J. Kaos policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement. In *POLICY* (2003), IEEE Computer Society, pp. 93–.

[101] VEDAMUTHU, A. S., ORCHARD, D., HIRSCH, F., HONDO, M., YENDLURI, P., BOUBEZ, T., AND YALCINALP, U. Web services policy 1.5 - attachment. `http://www.w3.org/TR/ws-policy-attach/`, September 2007. [Online; accessed July-2010].

[102] VEDAMUTHU, A. S., ORCHARD, D., HIRSCH, F., HONDO, M., YENDLURI, P., BOUBEZ, T., AND YALCINALP, U. Web services policy 1.5 - framework. `http://www.w3.org/TR/ws-policy/`, September 2007. [Online; accessed July-2010].

[103] VEDAMUTHU, A. S., ORCHARD, D., HIRSCH, F., HONDO, M., YENDLURI, P., BOUBEZ, T., AND YALCINALP, U. Web services

policy 1.5 - primer. http://www.w3.org/TR/ws-policy-primer/, November 2007. [Online; accessed July-2010].

[104] VERMA, K., GOMADAM, K., SHETH, A. P., MILLER, J. A., AND WU, Z. The meteor-s approach for configuring and executing dynamic web processes. Tech. rep., LSDIS Lab, University of Georgia, June 2005.

[105] VETERE, G., AND LENZERINI, M. Models for semantic interoperability in service-oriented architectures. *IBM Systems Journal 44*, 4 (2005), 887–904.

[106] WALSH, W. E., TESAURO, G., KEPHART, J. O., AND DAS, R. Utility functions in autonomic systems. In *ICAC* [1], pp. 70–77.

[107] WATERMAN, D., AND ROTH, F. H., Eds. *Principles of pattern-directed inference systems* (1978), Academic Press.

[108] WHITE, S. R., HANSON, J. E., WHALLEY, I., CHESS, D. M., AND KEPHART, J. O. An architectural approach to autonomic computing. In *ICAC* [1], pp. 2–9.

[109] WILSON, K., AND SEDUKHIN, I. Web services distributed management: Management of web services (wsdm-mows) 1.1. Tech. rep., OASIS Web Services Distributed Management TC, August 2006.