INTERNATIONAL JOURNAL OF NETWORK MANAGEMENT
Int. J. Network Mgmt 2011; **00**:1–26
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/nem

Semantic Context Dissemination and Service Matchmaking in Future Network Management

J. Famaey[1]*, S. Latré[1], J. Strassner[2] and F. De Turck[1]

[1]*Department of Information Technology, Ghent University – IBBT, Ghent, Belgium*
[2]*Division of IT Convergence Engineering, Pohang University of Science and Technology, Pohang, Korea*

SUMMARY

The ever increasing size, complexity and heterogeneity of telecommunications networks necessitates the introduction of autonomic elements that assist providers in managing and configuring the network's resources. To tackle this increased complexity, it is expected that many specialized autonomic elements will take part in the management process. It becomes necessary for them to collaborate and communicate in order to achieve high-level, human-specified, management goals. Therefore, the need for a scalable mechanism to facilitate the interactions between autonomic elements has arisen. This article presents a communications bus, augmented with semantics through the use of ontologies and semantic reasoning, which governs the communication and collaboration between autonomic elements. It supports filtering of context based on meaning. Additionally, it facilitates matchmaking of autonomic element goals with management services using semantic definitions of their inputs, outputs, preconditions and effects. Furthermore, the delay introduced by semantic reasoning was evaluated through an implemented prototype and was shown to be limited to only a few milliseconds. Copyright © 2011 John Wiley & Sons, Ltd.

Received ...

KEY WORDS: 5. Management Approaches / 5.3. Autonomic and self management, 6. Technologies / 6.6 Data, information, and semantic modelling

1. INTRODUCTION

The size, complexity and heterogeneity of telecommunications networks has been increasing rapidly in recent years. It is quickly becoming too difficult and costly for human operators to manage such networks manually. As such, a need has arisen for autonomic components that automatically manage and configure the network's resources. These components are guided by the high-level goals set forth by human operators [1]. It is expected that many specialized autonomic elements (AEs) will take part in managing the network to overcome this increased complexity [2]. They all perform specific functions and adhere to a set of different business and/or technical goals, but will need to collaborate and communicate in order to achieve the more complex human-specified management goals. Therefore, it becomes necessary to govern the communication and collaboration between AEs. Recently, the idea of a distributed communications bus, which facilitates the interaction between AEs, has been proposed to solve this problem. For example, the FOCALE architecture [2] includes the enterprise content bus, which is an extension of the enterprise service bus paradigm [3]. This bus should support semantics in order to facilitate a shared understanding of context, goals and actions between AEs.

*Correspondence to: Jeroen Famaey, Ghent University – IBBT, Gaston Crommenlaan 8/201, B-9050 Gent, Belgium, jeroen.famaey@intec.ugent.be

Additionally, the Future Internet is moving away from silo-based management approaches to inter-domain federated network management [4]. Such federations support the end-to-end management of loosely coupled value networks, spanning several autonomic domains [5]. We argue that in such a case, there is an even greater need for correct interpretation of information in the communication between AEs, as they will need to collaborate across domains. This further proves the necessity of semantically enriched communication.

The communication between AEs is characterized by several types of interaction. In this article, we focus on two: dissemination of context and service matchmaking. The management components achieve their goals by monitoring the current state of their managed environment and taking appropriate corrective actions to ensure that the system ends up in the desired overall state. This allows them to optimize performance and detect and rectify problems. In order to maintain an up-to-date view on the state of the managed resources, potentially very large amounts of information need to be disseminated from the network's resources to AEs and between AEs themselves [6]. The context of an entity is defined as the collection of measured and inferred knowledge that describes the state and environment in which that entity exists or has existed [7]. In this article, we propose a novel method that allows management components to semantically define the types of context in which they are interested, by way of a set of filter rules. Such a mechanism is necessary in order to reduce the amount of exchanged context to only that which is relevant and thus ensure scalability of the system. Our proposed method attaches semantics to this context, therefore supporting the definition of filter rules based on the actual meaning of information instead of static predefined keywords and/or string patterns. Additionally, these semantics can be used to automatically generate filter rules by way of semantic reasoning [8].

Although AEs can detect problems in the network's state through the dissemination of context, they may not always be able to solve the detected problems themselves. Therefore, to further support loosely-coupled collaboration between AEs, a matchmaking mechanism for the dynamic discovery of management services is needed. Matchmaking is defined as the process of discovering a set of services or functions that fulfil a given set of requirements. In this article, we propose a semantic service matchmaking algorithm that finds AEs offering the requested management functions based on the subsumption relationships of inputs, outputs, preconditions and effects (IOPEs) of the service descriptions. A concept D subsumes a concept B if B is a specialization of D (or alternatively, if D is a generalization of B). By including preconditions and effects in the matchmaking process, management components can better estimate the effects of specific functions on the environmental state. Additionally, as the IOPEs are semantically enhanced, a reasoner can be used to infer semantic relatedness of the requested and offered service definitions.

In order to facilitate these interactions, we propose the Semantic Communications Bus (SCB), which allows AEs to define filter rules that specify the types of context in which they are interested. Additionally, it offers a service matchmaking component, which supports matchmaking of management services with specific inputs, outputs, preconditions and effects. An example scenario, illustrating the SCB's role in the interactions between AEs, is shown in Figure 1. It depicts a federated cloud computing scenario, with two datacenters connected through the Internet. The SCB is depicted as a logical substrate spanning the different management domains. Although we make no assumptions concerning the SCB's deployment, in an operational environment its functionality will most likely be distributed among the AEs for scalability reasons. In the presented example scenario, AE 1 and AE 2 monitor the state of the first datacenter's resources. They extract relevant context and publish it onto the bus. AE 3, which previously subscribed to specific types of context information, receives a subset of this published context via the bus. AE 3 uses the gathered context to detect problems that occur within the datacenter (e.g., resource starvation of a specific hosted service). When a problem is detected, AE 3 determines a plan of action to rectify the situation. Such a plan contains a set of resource reconfigurations that need to be performed inside, and possibly outside, the management domain. In order to successfully execute these reconfigurations, the AE needs to discover and initiate functionality offered by other AEs. This is where the SCB's service matchmaking algorithms come into play. In the example scenario, the reconfigurations need to be performed in another domain, namely datacenter 2. Through the

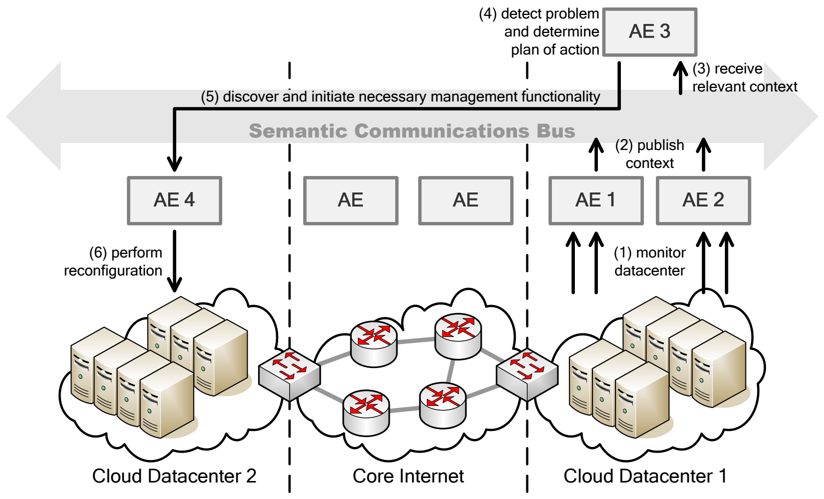

(4) detect problem and determine plan of action
AE 3
(3) receive relevant context
(5) discover and initiate necessary management functionality
Semantic Communications Bus
(2) publish context
AE 4
AE
AE
AE 1
AE 2
(6) perform reconfiguration
(1) monitor datacenter
Cloud Datacenter 2
Core Internet
Cloud Datacenter 1

Figure 1. An overview of the interactions between the network's resources, management components and human operators

SCB matchmaker, AE 3 discovers that the necessary functionality is offered by AE 4, which could, for example, be the migration of the previously mentioned overloaded hosted service to a server within datacenter 2. The described scenario clearly shows that for AEs to successfully perform their tasks, they need to be able to send and receive relevant context and be able to discover and initiate management services offered by other AEs.

The SCB's semantics are supported through a set of ontologies. These ontologies semantically represent context information, filter rules and service descriptions. Through a combination of semantic models and reasoning, context can be semantically filtered and service descriptions can be mapped to one another. The contributions of this article are threefold. First, three approaches to representing filter rules are proposed. The first builds upon our previous work [9] and uses pure OWL 2[†] (Web Ontology Language) constructs. It can be used in conjunction with a pure OWL reasoner. Additionally, two novel approaches using SWRL[‡] (Semantic Web Rule Language) and Jena rules [10] as filters are proposed. Although the use of SWRL rules requires a more advanced reasoner, it offers increased expressiveness compared to pure OWL. Jena's expressiveness is comparable to that of SWRL, but does not have the added benefit of OWL inferencing. As such, Jena is expected to execute much faster. Second, we propose a novel matchmaking algorithm capable of determining the semantic compatibility between IOPE descriptions constructed of SWRL atoms. Additionally, we formally define these relationships in this article. Third, we determine the impact on efficiency of introducing semantics in the communications bus. We thoroughly evaluate performance and scalability, in terms of execution time, of the proposed algorithms and approaches and explore the trade-off between performance and expressive capabilities.

The remainder of this article is structured as follows. Section 2 discusses existing work on semantic publish/subscribe systems and semantic service matchmaking. Additionally, we thoroughly describe the functional differences with the approaches introduced in this article. Subsequently, Section 3 presents the SCB. A cloud computing use case is detailed in Section 4. The examples and evaluations presented throughout the article are based on this use case. Sections 5 and 6 respectively elaborate upon the internal details of the context dissemination and service matchmaking components. The implementation details and evaluation results of the designed prototype are discussed in Section 7. Finally, Section 8 concludes the paper.

[†]OWL 2 Web Ontology Language Document Overview: http://www.w3.org/TR/owl2-overview/
[‡]A Semantic Web Rule Language Combining OWL and RuleML: http://www.w3.org/Submission/SWRL/

2. RELATED WORK

The context dissemination process pushes messages, containing context information, towards a set of interested subscribers. The solution proposed in this paper is therefore related to existing work in the field of semantic publish/subscribe systems. Automated service matchmaking and discovery has long been a topic of interest in the context of the semantic web, and more specifically semantic web services. This section discusses the state of the art in these two fields and highlights the novel aspects of our approach.

2.1. *Semantic Publish/Subscribe Systems*

A publish/subscribe system is a messaging paradigm in which a set of publishers loosely communicates with a set of subscribers. Publishers do not send their messages to specific receivers. Instead, subscribers express interest in specific types of information, and published messages that correspond to their interests are delivered to them. In recent years, publish/subscribe systems have evolved from static topic-based to dynamic content-driven systems. In topic-based systems, messages are published to specific topics, usually selected from a static and predefined hierarchy of keywords. In contrast, content-driven publish/subscribe systems allow subscribers to express interest in messages based on their actual content. Additionally, by augmenting the content with semantics, subscriptions can be created that take into account the actual meaning of the content.

Several types of semantic publish/subscribe systems have been proposed in literature. These works propose systems based on RDF (Resource Description Framework) graph-matching [11, 12, 13], ontological inferencing [14, 15] and attribute-value pair matching [16, 17]. RDF graph-matching algorithms represent messages as sets of RDF triples, which can be modelled as directed labelled graphs. Subscriptions take the form of graph patterns, which are matched to the published messages by the graph-matching algorithm. In contrast, we propose an OWL-based approach, which allows new, non-asserted knowledge to be inferred during the message publishing process. The semantic publish/subscribe systems based on OWL inferencing are more closely related to the approach presented in this article. Subscriptions are represented as ontological classes, while messages are defined as class instances. An ontological reasoner is used to determine if a message instance satisfies the constraints of a subscription class. Although the ontological inferencing approach is also used by the SCB, it, in contrast to existing work, also supports SWRL and Jena rules as subscription filters. Such semantic rules greatly increase expressiveness through a set of built-ins (e.g., mathematical and comparison operations). Finally, the systems based on attribute-value pairs significantly limit the format that messages and thus information is allowed to take, which is not the case in our approach. The remainder of this section describes the introduced publish/subscribe systems in more detail.

Early work on semantic publish/subscribe systems was performed by Petrovic et al. [18, 11]. They argued that the traditional topic-based systems are incapable of matching semantically related concepts when determining the interest of subscribers in a specific message. To solve these problems they proposed G-ToPSS (Graph-based Toronto Publish/Subscribe System) [11], which uses an RDF graph-matching algorithm for relating subscriptions to messages. Subscriptions are represented as a set of 5-tuples, augmenting the subject and object of the RDF triple with optional constraints. These constraints could be boolean constraints for literal values (e.g., $=$ or $\geq$) and *is–a* constraints for RDF individuals. In our work, we aim to provide more powerful inferencing capabilities by using OWL-based inferencing engines, instead of a custom graph matching algorithm. Additionally, by supporting SWRL and Jena built-ins, we greatly increase expressiveness by introducing a wider array of constraints, such as mathematical operations and date comparison operators. Similar to the approach used by Petrovic et al., Wang et al. proposed a semantic publish/subscribe system, which uses RDF graph-matching [19, 12]. As such, the expressiveness of both approaches is also similar. However, Wang et al. also support regular expressions as constraints, in addition to the boolean operators supported by G-ToPSS. However, it shares G-ToPSS' limitations in expressive and inferencing power. More recently, the authors further applied their approach specifically to the RSS (Really Simple Syndication) document dissemination use-case [13]. In this recent work, OWL is used to represent the concept taxonomy, as opposed to DAML+OIL (DARPA Agent Markup Language and Ontology Inference Layer) as used in their previous work.

The solution proposed by Li et al. more closely resembles the approach proposed in this article [14]. Subscriptions are represented as DAML+OIL classes, while messages are defined as class instances. In contrast, we use the more modern OWL, instead of DAML+OIL, to represent subscriptions. Additionally, we propose the use of SWRL and Jena rules as subscription filters, which greatly increases expressiveness. More recently, Skovronski and Chio adopted a similar method [15]. Messages are also represented by instances in the ontology. However, subscriptions take the form of SPARQL (SPARQL Protocol and RDF Query Language) queries. A SPARQL query consists of a set of RDF triple patterns, which are matched to the RDF triples in the ontology. Consequently, its inferencing and expressive capabilities share the limitations of other existing approaches [11, 12, 14].

Siena is a scalable event notification middleware [16]. Messages are represented as a set of typed attributes, comprised of a name, type and value. The supported types are limited to String, Long, Integer, Double and Boolean. Subscription filters support constraints on the values of attributes, including mathematical and string comparison operators. Keeney et al. proposed two extensions to the Siena system [17]. One extension provides ontological concepts as an additional message attribute type; this enables subsumption and equivalence relationships, along with type queries and arbitrary ontological subscription filters, to be applied. The second extension provides a bag type to be used that allows bag equivalence and filtering. Both of these extensions can be viewed as extending the semantic matching capabilities of Siena. In particular, the first extension looks at the semantics of the data and associated metadata contained in the message in addition to the contents of the message. This approach uses a set of subsumption operators (i.e., more specific (hyponyms), less specific (hypernyms), and equivalent concepts) as well as the ability to match on any ontological property, and then reasons on how subscriptions are related to published data. Our work is different in that we use a richer notion of semantic relatedness, and we are not limited to attribute-value pairs.

In summary, our goal is to improve existing work by greatly increasing inferencing power and subscription expressiveness. This can be achieved by using powerful OWL-based inferencing engines, instead of custom RDF graph-matching algorithms usually employed in existing semantic publish/subscribe systems, since OWL has more powerful features (e.g., a standard vocabulary, the ability to distinguish between classes and individuals enumeration and property restrictions) that enable it to perform more powerful inferencing than RDF. Additionally, we propose the use of SWRL and Jena rules for defining subscriptions, as they support a wide range of built-in operators which greatly increase expressiveness. These improvements should be achieved without sacrificing performance.

2.2. *Semantic Service Matchmaking*

Service matchmaking has long played an important role in the interaction between web services. Its goal is to find service descriptions that satisfy a set of functional criteria. Traditionally, keyword-based methods, like UDDI, have been proposed. However, they lead to low matching precision due to their lack of semantics [20]. More recently, web service descriptions have been augmented with semantics, which makes them machine-understandable and -readable. Most existing work on service matchmaking algorithms focusses on determining compatibility between inputs and outputs [21]. The matching of preconditions and effects, which is inherently more complex, has received relatively little attention [22].

OWLS-MX is a hybrid semantic web service matchmaker for OWL-S services [23, 24]. OWL-S is an ontological model for semantically describing services. It supports several logical matching relationships, such as exact, plug-in, subsume and subsumed-by. OWLS-MX also supports non-logic-based matching. However, we believe the latter cannot be used in an autonomic network management scenario, as compatibility on a logical level cannot be guaranteed. Additionally, the work focusses only on input and output matching. However, for AEs it is necessary to estimate the influence of services on their managed environment. Therefore, an approach that incorporates precondition and effect matching is required.

Recently, several semantic matchmaking algorithms that do take into account preconditions and effects have been proposed [20, 22, 25]. Shen et al. use description logics to describe service IOPEs [20]. A DL-reasoner can then be used to determine service compatibility. On one hand, using description logics allows them to formally prove the correctness of their algorithm. However, its expressiveness is limited compared to the SWRL-based approach introduced in this article. A SWRL-based approach was also proposed by Bener et al. [22]. They assign a score to every match, based on the type of subsumption relationship, semantic distance and synonym equivalence (using WordNet). The final score represents the semantic similarity between the service descriptions. However, a high score does not necessarily mean logical and functional compatibility. In contrast, our proposed algorithm returns the set of known logical and functional compatible service descriptions, which is more suitable for use by autonomic agents. Finally, Sbodio et al. use SPARQL for representing preconditions and effects [25]. However, their work focusses more on determining if the preconditions hold in the current state of the environment and relating the effects with the goals of the agent that is planning to execute the service. As such, this work complements ours.

In summary, most existing semantic service matchmaking algorithms focus on input and output matching only. In recent years, some algorithms have been proposed for the more complex precondition and effect matchmaking problem. However, we have identified several shortcomings in existing approaches (as described above), such as limited expressive power or the inability to sufficiently capture functional compatibility. Additionally, none of these matchmaking algorithms are capable of relating specific inputs and outputs to specific preconditions and effects. Supporting such relations significantly increases the accuracy of service descriptions and, consequently, of the matchmaking process. The algorithm proposed in this article supports such relations by way of SWRL variables.

3. SEMANTIC COMMUNICATIONS BUS ARCHITECTURE

Future networks will be managed by a large set of automated management components, which we refer to as autonomic elements (AEs). In order to achieve the network's management goals, they interact in loosely-coupled collaborations with each other, the network's resources and human operators. Although the latter is still responsible for governing the network through the specification of high-level business goals, the lower-level maintenance tasks and configurations are taken over by the AEs. This section describes the semantic communications bus (SCB) that orchestrates the interactions between AEs. The implementation details of the prototype are discussed in Section 7.

A detailed overview of the SCB's architecture is shown in Figure 2. It plays a central role in the interactions between AEs and network resources. It orchestrates the dissemination of context and the service matchmaking process. Its semantic capabilities stem from the *core ontologies* and *semantic reasoners* embedded within it. The ontologies provide a model for semantically representing the managed environment. There are no requirements about the specific concepts that should be present in these models. However, in order to achieve understanding between communicating entities, they should use (a subset of) the same ontologies. Additionally, the interaction complexity is limited by that of the models. As such, as more complex concepts are added to the ontologies, it becomes possible to model more complex interactions. Throughout the rest of this article, DENON-ng is used as a basis for the core ontologies [26]. DENON-ng is an ontological model based on the DEN-ng information model [27]. It can be used to represent the physical and logical state of the network and its resources, as well as the business goals and internal workings of the governing organisations. Section 4 describes how these core ontologies can be further extended to a specific problem domain. The semantic reasoners operate on top of the core ontologies. They can be used by the SCB's other components to perform semantic inferencing.

In a federated network management scenario, AEs are expected to communicate across the boundaries of management domains. This introduces additional challenges such as the need for a common communication model for interpreting and understanding exchanged information. In the proposed SCB architecture, this would mean that, the relevant subset of, the core ontologies should

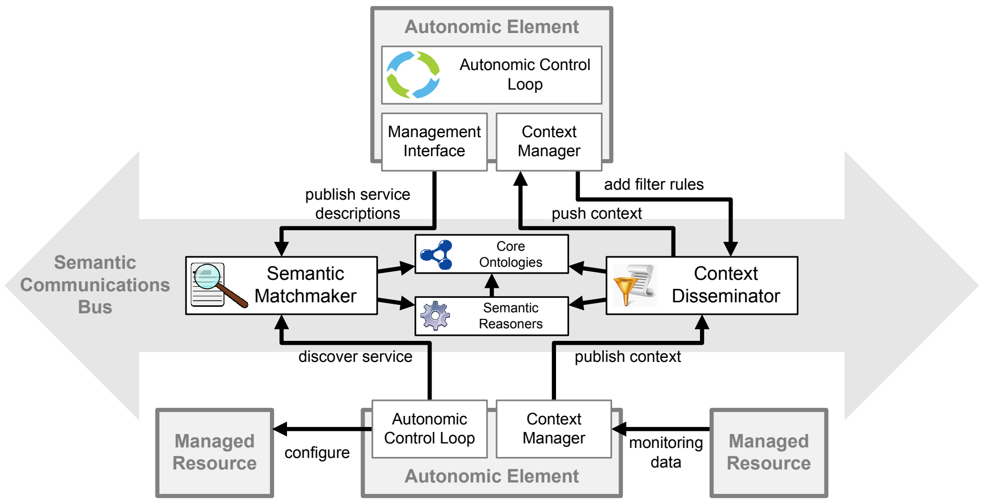


Figure 2. The SCB plays a central role in the interactions between AEs and network resources; its core ontologies are used in the matchmaking and context dissemination processes

be shared across the communicating management domains. Such complex interoperability problems are outside the scope of this article, but have been thoroughly described in literature [28].

The figure shows the SCB as a single entity. However, in large-scale deployments the number of AEs might potentially become very large. The SCB could then be distributed among those AEs, increasing scalability by letting them all take part in the semantic reasoning processes. For example, structuring AEs hierarchically has been shown to significantly improve scalability and reduce the amount of context that needs to be exchanged [6]. In such a scenario, the bulk of context exchange takes place between parent and child AEs. Every AE thus only needs to register its filter rules with the SCB component of its children, significantly reducing the number of filter rules and amount of context that needs to be processed by every SCB semantic reasoner instance. Nevertheless, the remainder of this article focusses on the semantic reasoning algorithms of the SCB and no assumptions are thus made regarding its structure and deployment.

In addition to the core ontologies, two other components play an important role in the interactions between AEs and network resources; the *context disseminator* and the *service matchmaker*. The context disseminator is responsible for routing published context information to the AEs that have expressed interest in it. An AE may express interest in a specific type of context by creating a corresponding filter rule and registering it with the context disseminator. When an AE publishes context information, for example gathered through a monitoring probe on a managed resource, an ontological reasoner is used to match this information to the registered filters. Section 5 explains how such filter rules can be expressed and how they are matched with published context. The service matchmaker consists of two components. The *service model repository* stores service descriptions of management functions offered by AEs or network resources. The *matchmaking algorithm* maps these offered service descriptions to descriptions of requested management services. It is thus responsible for the service matchmaking process. The definition of service descriptions and the internal workings of the matchmaking algorithm are further discussed in Section 6.

4. USE CASE: CLOUD INFRASTRUCTURE MANAGEMENT

In this section, a cloud infrastructure management use case is described. The examples given throughout the following sections will be based on this use case. Additionally, the evaluation scenario used in Section 7 is based on it. First, the use case is described in more detail. Subsequently, the ontological classes of the core ontologies, relevant to this use case, are presented.

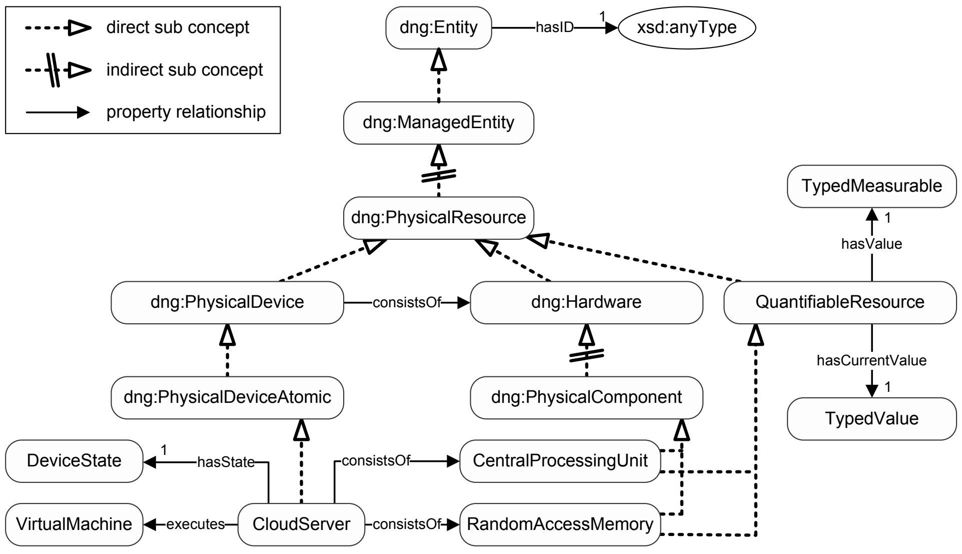


Figure 3. An overview of classes representing the physical resources of the cloud management use case; they are connected to the logical resources through the *VirtualMachine* class

4.1. *Scenario Description*

In the considered scenario, a set of AEs are responsible for managing a cloud computing infrastructure. The infrastructure consists of a set of servers, offered to service providers as virtualized resources. As such, every server contains a set of elastic virtual machines (VMs). Through Service Level Agreements (SLAs), service providers indicate the resource requirements of their services. The cloud provider makes sure the necessary amount of resources are provisioned within the bounds of the SLA. Cloud computing enables Service Providers and Enterprises to reduce both capital (CAPEX) and operational expenditure (OPEX) through hard- and software consolidation and automating business processes, respectively. In addition, the dangers of under- and over-provisioning are replaced by on-demand resource allocation that adjusts to varying resource and service consumption patterns.

The AEs are responsible for provisioning resources to the VMs and managing the state of the physical servers themselves. Therefore, they monitor resource availability on the servers, and resource consumption by the services. They make sure that all VMs receive the necessary resources within the bounds of the SLAs. These SLAs are represented as a set of policies.

4.2. *Ontological Concepts*

As previously stated, there is no set of rules that determines what information should be present in the core ontologies of the SCB. In this section we describe a minimal set of classes that can be used to semantically describe the context and management functions of the cloud infrastructure management use case. For clarity, the classes are shown in three different figures. Figure 3 presents the physical resources, such as servers and hardware components. The logical resources, including cloud services, requests and virtual machines are depicted in Figure 4. Finally, Figure 5 shows how measurements, such as CPU values, are represented. The core ontology is based on the DENON-ng ontology. Classes originating from DENON-ng are prefixed with "*dng:*". Additionally, the prefix "*temporal:*" is used for classes from the SWRL temporal ontology [29], which we use for representing time.

The physical resources shown in Figure 3 are all subclasses of *dng:Entity*. An entity is any physical or logical object that is important to the managed environment in some way. The *dng:PhysicalResource* class is an indirect subclass of *dng:Entity* and represents a managed piece of hardware. The different types of hardware components are represented by its subclasses. For

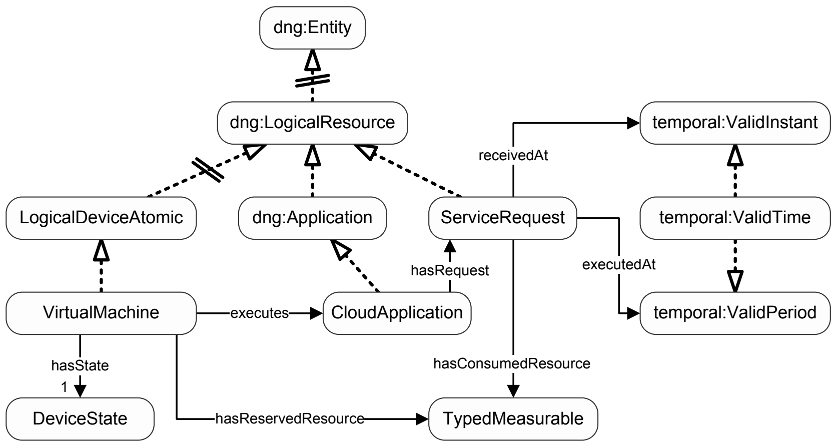


Figure 4. An overview of classes representing the logical resources of the cloud management use case

example, *dng:PhysicalDevice* is the aggregation of the hardware components that make up a device. It enables management of the device as a whole. As such, it has a *consistsOf* relationship with *dng:Hardware* as its range. This last class represents the physical components that make up the device. Its *dng:PhysicalComponent* subclass serves as a super type for hardware that resides inside equipment and cannot be used as a stand-alone object.

In addition to these DENON-ng classes, we have defined several physical classes used to model the Cloud Computing use-case. The *CloudServer* class represents a physical server in the cloud datacenter. It has a *DeviceState*, which is one of TurnedOn, TurnedOff, Hibernating, Suspended or Unknown. It can be used to determine if a server is currently ready to receive service requests or not. Additionally, it hosts a set of *VirtualMachine*s. A cloud server consists of many different hardware components. However, our model focusses on two of these that will be used to monitor the resource consumption of the servers. It is easily extensible to include additional entities as necessary. The *CentralProcessingUnit* class represents a CPU core, while *RandomAccessMemory* denotes a memory module. Both these physical components depict physical resources that can be consumed by the software running on the servers. Therefore, they are also subclasses of *QuantifiableResource*. This is a resource that has a set of values associated with it. Such a set of values is represented as a *Measurable*. Additionally, the last measured value is explicitly defined through the *hasCurrentValue* relationship. The *Measurable* and *Value* classes are further explained in Figure 5.

The physical and logical resources are linked through the *VirtualMachine* classes. This and other logical resources are depicted in Figure 4. Every cloud server is capable of hosting a set of virtual machines. They represent elastic virtual servers that can be reserved and used by the customers of the cloud infrastructure for hosting their services. Just as a physical server, it has a *DeviceState*. Additionally, it has a set of reserved resources associated with it, which are represented by the *TypedMeasurable* class. Finally, services may be executed from the virtual machine, represented by the *CloudApplication* class. The *ServiceRequest* class models the client requests sent to the different cloud applications. Every request has an associated receive time and execution interval. Additionally requests consume resources, which are depicted by the *TypedMeasurable* class.

Management components often make decisions based on the current, past or expected future load of specific resources. As such, a mechanism is needed for modelling the load of hardware components, such as CPU and memory modules. Our proposed model, as shown in Figure 5, is based on the DENON-ng *dng:Measurable* and *dng:MeasurableValue* classes. A measurable categorizes a type of value that can be measured. The actual values themselves are represented by the *dng:MeasurableValue* class. A measurable thus depicts the type and groups together several measurable values. In addition to these standard DENON-ng types, we have defined our own measurable subclasses. The *TypedMeasurable* represents a measurable that is attached to a quantifiable resource. Additionally, it has a unit of measurement (e.g., Hertz, Megabyte),

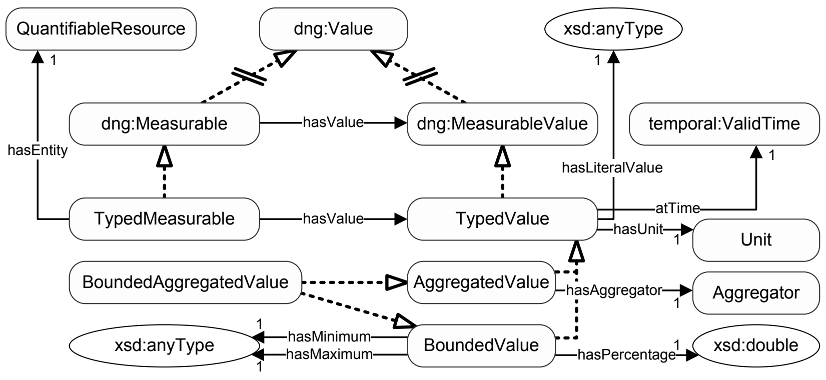


Figure 5. An overview of the classes representing measured values

which can be accessed through the *TypedValue* class. The typed value itself also has a literal attached to it, which represents the actual value. Finally, there is also a time instant or period at which the measurement was taken attached to the typed values. The *TypedValue* class has several subclasses further differentiating between different types of measured values. An *AggregatedValue* depicts a measurement that was aggregated over a certain time interval, such as an average or a maximum. A *BoundedValue* has a minimum, maximum and percentage attached to it. Measurements in a network management scenario often are of this type. For example, a CPU load value is constrained by a minimum of 0 and a maximum denoting the CPU's total capacity. Finally, the *BoundedAggregatedValue* class combines the characteristics of *BoundedValue* and *AggregatedValue*. Note that in line with these value types, *BoundedMeasurable*, *AggregatedMeasurable* and *BoundedAggregatedMeasurable* also exist. However, these have been omitted from Figure 5 for clarity.

5. SEMANTIC CONTEXT DISSEMINATION

An important aspect of the interaction between management components is the exchange and dissemination of context information. The AEs monitor their set of managed resources and use the gathered information to create a view on the state of the managed environment. This context information is thus forwarded both from resources to AEs and between AEs themselves. In this section, we present three alternative approaches to modelling filter rules. Additionally, this section describes how these filter rules are matched with published context.

The SCB allows management components to register interest in specific topics, by way of filter rules. The semantic reasoners are an integral part of internal mechanism used by the SCB to match filter rules with the context that is published onto the bus. If a published unit of context matches at least one of the filter rules of an AE, that context is forwarded to the AE. The use of such a publish/subscribe system greatly reduces the amount of context information that is forwarded to AEs, preventing them from becoming swamped with useless information. Additionally, by introducing semantics into the filter rules, management components can indicate interest based on the meaning of data that makes up the context received by an AE, allowing them to reason about what information would be relevant in face of the tasks they perform. Figure 6 further clarifies the context dissemination process. Context publishers can freely forward data to the SCB. The SCB uses the semantic reasoners to determine which subscribers are interested in the context contained within the message. The reasoners use the core ontologies to check if the message satisfies at least one filter rule defined by the subscriber. If it does, the message is forwarded.

In order to be able to semantically reason on messages and filter rules, the core ontologies need to be augmented with a *Message* class. As DENON-ng already contains the *dng:Message* class, we use this as a basis for our model. A message represents a *dng:Event* with the addition of a payload. In turn an event is a type of *dng:Entity*. Additionally, a message has exactly one source and zero

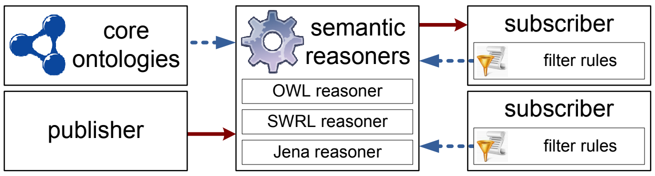


Figure 6. A detailed overview of the context dissemination process and the involved actors

or more targets, which are both of type *dng:Entity*. When an AE publishes context information, it creates a message, which is an instance of the *dng:Message* class, and adds the context it wishes to publish as the message payload.

The three presented approaches differ in the way filter rules are defined. The first uses pure OWL constructs, while the second and third use SWRL and Jena rules, respectively. The approaches are presented in more detail throughout the rest of this section.

5.1. OWL Filter Rules

The OWL-based algorithm exploits the instance reasoning capabilities of the ontological reasoner. As stated, published context is represented as an instance of the *dng:Message* ontology class. Filter rules, on the other hand, are represented by way of subclasses of the *dng:Message* class. As such, context can be matched to a filter rule by asking the OWL reasoner if the message instance belongs to the *dng:Message* subclass defined by the filter rule.

More specifically, filter rules are defined as an equivalent class of a conjunction of OWL class expressions. A class defined in such a way is called a *defined class*. The conjunction can be seen as a set of *necessary and sufficient* conditions to which instances must adhere in order to belong to the class. This means that the reasoner is capable of inferring that an instance belongs to a specific *defined class*, even if this information was not asserted. This is not possible with a *primitive class*, which is defined using subclass axioms instead of an equivalent class axiom. Additionally, the SCB asserts that the filter rule class is a subclass of *dng:Message*. The reasoner can then be used to check if the filter rule class is satisfiable. If it is unsatisfiable, it means that it is not a valid subclass of *dng:Message* and the filter rule is removed from the ontology. If it is satisfiable, then the subscription operation finishes successfully.

In order to further clarify the use of OWL filter rules, we give some examples based on the use-case . All OWL examples given throughout this section use the OWL Manchester syntax[§]. A simple rule stating that an AE is interested in all messages containing information about servers can be defined as follows:

dng:Message **and** hasPayload **some** CloudServer

As previously stated, the filter rule is a conjunction of class expressions. The first expression is the obligatory subclass assertion, stating that the newly defined filter rule class is a subclass of *dng:Message*. The second expression states that the filter rule should only trigger if the message has at least one payload of type *CloudServer*.

Using OWL 2 specific constructs, such as datatype restrictions, more complex and expressive filter rules can easily be defined. For example, an AE that is responsible for resource allocation of virtual machines could be interested in all types of quantifiable resources with a load higher than 95%, in order to be able to react before those resources become overloaded. This filter rule can be defined as follows:

dng:Message **and** hasPayload **some** (QuantifiableResource
and hasCurrentValue **some** (BoundedValue
and hasPercentage **some** double[$>$ 95.0]))

[§]http://www.w3.org/TR/owl2-manchester-syntax/

Again, the first expression states that filter rule class should be a subclass of *dng:Message*. The second class expression again targets the message payload. However, in contrast to the first example, a more detailed payload description is given. The payload should abide to several restrictions. First, it must be of type *QuantifiableResource*. Second, its current value must be of type *BoundedValue*. Third, the current percentage of this value must be of type double and higher than 95%.

5.2. SWRL Filter Rules

In the previous section, it was shown how filter rules can be specified in the form of OWL class descriptions. The advantage of such an approach is that an OWL-only reasoner can be used to infer if a filter rule matches a message instance. However, in order to further increase expressiveness, this section introduces an alternative approach, using SWRL rules as filters. The advantage of this approach is that it combines OWL inferencing with the increased expressiveness of SWRL. Obviously, in order to apply this approach, a SWRL-capable reasoner is required.

A SWRL rule consists of an antecedent and a consequent, each consisting of zero or more atoms. Multiple atoms are always treated as a conjunction. Semantically, if all the conditions of the antecedent hold, then the conditions specified in the consequent must also hold. Conceptually, our approach works as follows. The filter rule itself makes up the antecedent of the SWRL rule. The atoms in the antecedent thus specify terms that a message must match in order to be forwarded. On the other hand, the consequent is automatically created by the SCB. Every subscriber that registers with the SCB is allotted a unique asserted subclass of *dng:Message*. The consequent adds the message instance to this uniquely defined class. Determining if a message should be forwarded to a specific subscriber thus comes down to triggering the SWRL rules and checking if that message is an individual of the unique class associated with the subscriber. For example, a complete SWRL rule associated with a simple filter rule that accepts all messages that contain information about a server, would look as follows:

$$\text{dng:Message}(?\text{m}) \wedge \text{CloudServer}(?\text{s}) \wedge \text{hasPayload}(?\text{m}, ?\text{s}) \Rightarrow \text{SubscriberMessage}(?\text{m})$$

The antecedent consists of three atoms. The first is, in line with the OWL filter rules, obligatory. It identifies the variable that refers to the message instance. The second atom defines a placeholder variable for a *CloudServer*. Finally, the third atom states that the message *?m* should have a payload of type server. The consequent asserts that, if the atoms in the antecedent hold, the message *?m* belongs to the class *SubscriberMessage*. The class specified in the consequent should be the unique class associated with the subscriber that registered the rule. Note that this consequent is automatically generated by the SCB. The subscriber thus merely needs to define the atoms of the antecedent.

The antecedent atoms of the filter rule that matches all messages that contain information about a quantifiable resource with a load higher than 95% can be defined as follows:

$$\begin{gathered}\text{dng:Message}(?\text{m}) \wedge \text{QuantifiableResource}(?\text{r}) \wedge \text{BoundedValue}(?\text{v}) \wedge \\ \text{hasPayload}(?\text{m}, ?\text{r}) \wedge \text{hasCurrentValue}(?\text{r}, ?\text{v}) \wedge \text{hasPercentage}(?\text{v}, ?\text{p}) \wedge \\ \text{swrlb:greaterThan}(?\text{p}, 95.0)\end{gathered}$$

This more complex example consists of 7 atoms. The first is again the obligatory message class axiom. The second and third atom identify the variables associated with the quantifiable resource and the bounded value associated with this resource. The fourth atom states that the resource *?r* should be a payload of message *?m*. The fifth atom links *?r* to its value *?v*. Finally, the last two atoms identify the percentage associated with the value as the variable *?p* and state that this percentage should be higher than 95%.

The last atom defined in the previous example is called a SWRL built-in atom. The expressive power of SWRL is greatly increased by these built-ins. Built-ins have been defined for comparison (as in the example above), mathematical operations (e.g., summation, division, power), strings (e.g., substring, concatenation) and dates. The operations concerning dates are facilitated through the SWRL temporal ontology [29].

5.3. Jena Filter Rules

The OWL and SWRL-based approaches are capable of inferring complex connections between concepts, using an OWL reasoner. However, this is computationally hard [30], leading to degraded performance. Therefore, we propose a third approach, based on Jena rules. Conceptually, it is similar to the SWRL-based approach. Nevertheless, it does not perform OWL-based inferencing, but applies the rules directly to the underlying RDF graph. This is expected to greatly increase performance, at the cost of decreased semantic inferencing capabilities.

Jena rules are applied as context filters in the same way as SWRL rules. The rule body, which corresponds to the SWRL antecedent, contains the actual filter. The head, which corresponds to the SWRL consequent, consists of a single atom that adds the message to the subscriber's message subclass. The example that admits all messages that contain as a payload an entity of type *CloudServer* is defined using Jena as follows:

(?m rdf:type dng:Message) (?s rdf:type CloudServer) (?m hasPayload ?s)

The first two atoms state that variables *?m* and *?s* belong to the classes *dng:Message* and *CloudServer*. The *rdf:type* property thus corresponds to the SWRL class atom. The third atom connects the variables and states that the message should have a payload of type *CloudServer*. The more complex example that admits all messages about resources with a current load of 95% or higher, is defined as follows:

(?m rdf:type dng:Message) (?r rdf:type QuantifiableResource)
(?v rdf:type BoundedValue) (?m hasPayload ?r) (?r hasCurrentValue ?v)
(?v hasPercentage ?p) greaterThan(?p, 95)

As shown in the last atom, Jena also supports a set of built-ins, such as comparison and mathematical operators.

6. SEMANTIC SERVICE MATCHMAKING

AEs and managed resources may offer management services to other AEs across the network. However, a scalable matchmaking mechanism is needed for finding suitable candidate services to complete specific tasks. This section describes the service matchmaking algorithm, introduced in Section 3, in further detail. The algorithm is responsible for matching offered and requested service descriptions. Service descriptions consist of a set of inputs, outputs, preconditions and effects (IOPEs). The inputs represent the information that the service requires in order to perform its function. The outputs, on the other hand, represent the information that results from its execution. The preconditions and effects respectively represent the state that the environment should be in before the service execution starts, and the state it will be in after it finishes. The IOPEs are defined using SWRL atoms. SWRL was chosen instead of Jena, as it is compatible with existing service models, such as OWL-S. Inputs and outputs take the form of SWRL class or data range atoms, while preconditions and effects can be any type of SWRL atom. The algorithm uses the subsumption relationship between these SWRL atoms to determine potential matches between requested and offered service descriptions. Concept A is said to subsume concept B if B is a specialization of A, or in other words, if B is a subclass of A. Additionally, SWRL supports the use of variables. This makes it possible to link inputs and outputs to preconditions and effects, which would not be possible when using pure OWL. As the IOPEs are modelled using SWRL atoms, our algorithm is compatible with any semantic service description model that supports SWRL IOPEs, such as OWL-S¶.

As depicted in Figure 7, the matchmaking algorithm operates in five steps when checking the compatibility between an offered and requested service description. First, it checks the validity

¶http://www.w3.org/Submission/OWL-S/

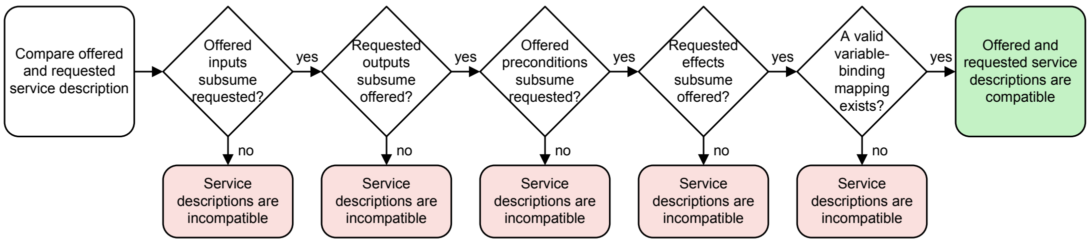

Compare offered and requested service description
Offered inputs subsume requested?
yes
no
Requested outputs subsume offered?
Offered preconditions subsume requested?
Requested effects subsume offered?
A valid variable-binding mapping exists?
Offered and requested service descriptions are compatible
Service descriptions are incompatible

Figure 7. A flowchart depicting the five steps of the semantic matchmaking algorithm

of the subsumption relationship between the offered and requested input atoms. This process is subsequently repeated for the output, precondition and effect atoms. Finally, the algorithm checks if the variable bindings of the offered service correctly map to those of the requested service. If during the execution any of the five steps fail, the algorithm knows that the service descriptions are incompatible, and it can skip to the next offered service description. The remainder of this section describes these five steps in more detail. However, as the first four steps of the algorithm rely on the definition of subsumption between SWRL atoms, we first formally define the subsumption relationship between the different SWRL atoms.

6.1. *Subsumption Relationships*

The SWRL recommendation does not formally define the subsumption relationship between SWRL atoms. Therefore, we first present a formal definition of this relationship. SWRL defines seven atom types: class, data range, object property, data property, sameAs, differentFrom and built-ins. Note that an atom can only subsume atoms of the same type.

A class atom $C(x)$ subsumes a class atom $D(y)$ if D is an equivalent class or subclass of C, and if x and y are matching arguments. An argument can be a SWRL variable, OWL individual or OWL data value. Two arguments x and y match if either they are both SWRL variables, they refer to the same OWL individual, or they refer to the same OWL data value. The equivalence and subclass relationships between C and D can be checked through applying an ontological reasoner on the core ontologies.

The subsumption relationship between object and data property atoms is similar to that between class atoms. An object property atom $P(x_1, x_2)$ subsumes another object property atom $Q(y_1, y_2)$, if Q is an equivalent property or subproperty of P and if arguments x_1 and y_1 match and x_2 and y_2 match. Alternatively, if P and Q are symmetric, then x_1 may match y_2 and x_2 may match y_1. The subsumption relationship between data property atoms is defined in the same way, with the minor difference that data properties cannot be symmetric.

The sameAs atoms cannot subsume one another, as it is not possible to define a subconcept relationship between them. However, two sameAs atoms can be considered equivalent, which means that subsumption holds in both directions. Two sameAs atoms *sameAs*(x_1,x_2) and *sameAs*(y_1,y_2) are considered equivalent if x_1 matches y_1 and x_2 matches y_2 or x_1 matches y_2 and x_2 matches y_1. The equivalence of *differentFrom* atoms is defined in the same way.

The subsumption between data range atoms is somewhat more complex. OWL supports several types of data ranges; datatypes, intersections, unions, complements, data oneOf and datatype restrictions. A data complement restriction $\neg C(x)$ subsumes a data complement restriction $\neg D(y)$ if $D(y)$ subsumes $C(x)$. A datatype atom $C(x)$ subsumes a datatype atom $D(y)$ if D is an equivalent or sub-datatype of C and x and y match. To determine sub-datatypes, we used the XSD datatype hierarchy specified in the W3C recommendation “XML Schema: Datatypes”[‖]. A data one of atom $C(x)$ subsumes a data one of atom $D(x)$ if C is a superset of D (i.e., C contains all the values in D) and x and y match. Furthermore, datatype restrictions, intersections and unions are not supported by our algorithm. Note that the semantics of datatype restrictions can be achieved using

[‖]http://www.w3.org/TR/xmlschema-2/

the built-in atoms for comparison. Therefore, not supporting datatype restrictions does not reduce expressiveness.

Finally, SWRL defines a wide range of built-in atoms. In this article, we define subsumption relationships only for the comparison built-ins. Subsumption relationships for the other SWRL built-ins can be defined in a similar fashion. A *swrlb:equal(x_1,x_2)* built-in atom subsumes another *swrlb:equal(y_1,y_2)* built-in atom if x_1 matches y_1 and x_2 matches y_2 or x_1 matches y_2 and x_2 matches y_1. The subsumption relationship of the *swrlb:notEqual* built-in is defined identically. The subsumption of the other comparison built-ins is more complex. More specifically, *swrlb:greaterThan*, *swrlb:greaterThanOrEqual*, *swrlb:lessThan* and *swrlb:lessThanOrEqual* can all subsume each other. As *swrlb:lessThan(x_1,x_2)* and *swrlb:lessThanOrEqual(x_1,x_2)* can be translated into *swrlb:greaterThan(x_2,x_1)* and *swrlb:greaterThanOrEqual(x_2,x_1)*, they can be ignored without loss of expressiveness. Intuitively, one of these atoms subsumes another if the range of possible values specified by the first contains the range specified by the second atom. Formally, an atom *swrlb:greaterThan(x_1,x_2)* subsumes an atom *swrlb:greaterThan(y_1,y_2)* if one of the following conditions holds for the arguments:

- All four arguments are SWRL variables
- x_1 and y_1 are SWRL variables, x_2 and y_2 are OWL data values and $x_2 \leq y_2$
- x_2 and y_2 are SWRL variables, x_1 and y_1 are OWL data values and $x_1 \geq y_1$

In all other cases, the subsumption does not hold. A *swrlb:greaterThanOrEqual* built-in atom subsumes a *swrlb:greaterThan* or *swrlb:greaterThanOrEqual* atom under the same conditions. Finally, to check if a *swrlb:greaterThan* atom subsumes a *swrlb:greaterThanOrEqual* atom, the conditions are slightly different. More specifically, in the second and third condition, $x_2 \leq y_2$ and $x_1 \geq y_1$ respectively become $x_2 < y_2$ and $x_1 > y_1$.

Let us further clarify by way of an example. The built-in atom *swrlb:greaterThanOrEqual(x,5)* specifies that $x \in [5, \infty[$, while *swrlb:greaterThan(y,6)* specifies that $y \in]6, \infty[$. Intuitively it is apparent that the first atom subsumes the second, as the second value range is entirely contained within the first. Additionally, the atoms adhere to the second condition above, which validates the subsumption relationship.

IOPEs usually consists of a set of SWRL atoms. Therefore, this section is concluded with a definition of the subsumption relationship between SWRL atom sets. Additionally, we propose an algorithm to check this relationship. Assume we want to check if a set of SWRL atoms $A = \{a_1, a_2, ..., a_n\}$ subsumes a set of SWRL atoms $B = \{b_1, b_2, ..., b_m\}$. The set A subsumes the set B if a mapping exists between the atoms in A and B, where every atom in A is mapped to exactly one atom in B and every atom in B is linked to at most one atom in A. An atom a_i of A can be mapped to an atom b_j of B if a_i subsumes b_j. The algorithm finds all valid mappings, because in the final step of the service matchmaking process they are needed to check variable binding matches (cfr. Section 6.4). The algorithm does this as follows. First, it constructs a bipartite graph, with all atoms of A as left vertices and those of B as right vertices. An edge is added to the graph between every atom a_i of A and b_j of B, if a_i subsumes b_j. If a left vertex has a cardinality of 0, no valid mappings exist and the algorithm finishes unsuccessfully. Otherwise, the algorithm constructs all valid mappings. In the context of bipartite graphs, finding such valid mappings is called the *maximum bipartite matching problem*. Existing algorithms, such as the one proposed in [31], can be leveraged to solve this problem.

6.2. *Inputs & Outputs*

The inputs and outputs defined in a service description represent the arguments and return values of the associated management function, respectively. As such, both the inputs and outputs are defined as SWRL class and data range atoms, which take the form $A(x)$. Here, A is an OWL class or data range expression, and x is a SWRL variable, an OWL individual or an OWL data value. If a SWRL variable is used in an input or output, the same variable can be reused in a precondition or effect in order to semantically link them together. For example, a management function could exist that activates a hibernating server. This function would have a server as input, a precondition stating that a server should be in hibernation and an effect stating that a server should be turned on. By reusing the same variable in the input, precondition and effect, they would be semantically linked and the reasoner would know all three refer to the same server instance.

The goal of the matchmaking algorithm is to find offered service descriptions of which the inputs and outputs are compatible with those of the requested service description. The inputs and outputs have slightly different semantics attached to them in the requested and offered service descriptions. In a requested service description, the inputs represent the information that the requester is willing to provide, while the outputs represent the minimum set of information that the requester wants to receive. On the other hand, the offered inputs describe the minimum set of information that the executor requires in order to successfully execute the management function, while the offered outputs represent the exact information that will be returned. As such, an offered set of inputs I_o is compatible with a requested set I_r if I_o subsumes I_r. This can be checked using the subsumption definition and algorithm for SWRL atom sets introduced in Section 6.1. The reason the offered inputs need to subsume the requested ones is explained as follows. If a service is requested that takes as input a specific concept (e.g., server) and one is offered that takes as input a more broad concept (e.g., device), then the offered service can potentially be used to perform the requested task, as every instance of the more specific concept is also an instance of the broader concept. For example, a broad management function that is capable of turning on any type of device, can be used by an AE that wants to turn on a server. Additionally, this means that the set I_r may contain additional inputs not defined in I_o. However, all inputs in I_o must have a match in I_r.

For outputs, the subsumption relationship is inversed. In other words, an offered set of outputs O_o is compatible with a requested set of outputs O_r if O_r subsumes O_o. Intuitively, this is explained as follows. If an AE requests a function that returns output about a concept, it might also be interested in a function that returns output about one of its subconcepts. Additionally, this means that the offered service may return more outputs than requested.

The matchmaking algorithm compares the compatibility of inputs and outputs between every offered service description and the requested service description. As previously stated, checking the subsumption between offered and requested inputs or outputs results in a set of maximum bipartite matchings. If the returned set of matchings of the inputs or outputs is empty, the offered and requested services are incompatible and the matchmaker no longer needs to check preconditions, effects and variable bindings for this offered service.

6.3. Preconditions & Effects

The preconditions of a service description describe the conditions that must be valid before the management function can be executed, while the effects represent the conditions that will hold after the execution finishes. Similarly to inputs and outputs, the matchmaking algorithm checks the compatibility between offered and requested preconditions and effects. Again there is a slight semantic difference between preconditions and effects in the requested and offered service descriptions. A requested set of preconditions actually defines the current state of the environment, or at least the state the environment will be in when the requester is planning to execute the management function. A set of requested effects represent the state the requester expects the environment to be in after the execution finishes. As such the requested effects are obligations the executor must adhere to. On the other hand, the set of offered preconditions define the state of the environment that must be valid before the management function can be executed, while the executor ensures that the set of offered effects will be valid after execution. Much like inputs, preconditions are compatible if the offered preconditions subsume the requested ones, which also means that the requested service description may contain more preconditions than the offered one. In contrast, the offered and requested effects are compatible if the requested subsume the offered effects. Obviously, checking the subsumption relationship for precondition and effect sets is more complex than for inputs and outputs, as they may contain any type of SWRL atom, as opposed to only class and data range atoms.

In line with inputs and outputs, the matchmaking process between preconditions and effects also yields two sets of maximum bipartite matchings. Again, if either set is empty, the offered and requested services are incompatible. Otherwise, the matchmaker checks compatibility between the variable bindings of both service descriptions.

6.4. Variable Binding Matches

If the matchmaking algorithms finds at least one maximum bipartite matching for the inputs, outputs, preconditions and effects between the offered and requested service descriptions, it will check if there exists a 4-tuple of matchings (one of inputs, outputs, preconditions and effects each), that has a compatible variable binding. If this is the case, the offered and requested service descriptions are considered compatible.

As stated, the variable binding matchmaking algorithm takes as input four maximum bipartite matchings; one of inputs, outputs, preconditions and effects. First, the algorithm creates a variable mapping of the offered service, which maps all SWRL variables on the set of SWRL atoms that contain this variable and are present in the IOPEs of the offered service description. For every SWRL variable x of the offered service description, this results in the set $A_x = \{a_1, a_2, ..., a_n\}$. Using the four selected maximum bipartite matchings, the SWRL atom b_i of the requested service that maps to every a_i of A_x can be determined. Note that some a_i from the outputs and effects may not have an associated b_i; in this case, they can be ignored. Subsequently, the algorithm checks if all atoms b_i contain the same variable y as an argument on the correct position. For most SWRL atom types the correct position is the position of x in the argument list of the corresponding atom a_i. However, some atom types, such as the built-in atom *swrlb:equal*, represent a symmetric relationship. As such, if an atom a_i, and correspondingly b_i, represents a symmetric relationship, the variable y in b_i does not necessarily need to be on the same position as x in a_i. If all b_i corresponding to the atoms in A_x contain the same variable y on the correct position, then the variable bindings between the requested and offered service match for variable x. If this is the case for all variables that occur in the offered service description for the four selected maximum bipartite matchings, then this 4-tuple of matchings is considered valid and the offered and requested service descriptions are fully compatible. Otherwise, the algorithm selects a different combination of matchings and retries. This process is repeated until a valid 4-tuple is found, or all possible combinations have been depleted. If no valid 4-tuple exists, then the descriptions are incompatible.

6.5. Illustrative Examples

In order to further clarify the described matchmaking process, this section is concluded with some example service descriptions. Additionally, an example is given that clarifies the matchmaking process between offered and requested services.

In a cloud computing environment, the amount of reserved and consumed resources varies greatly over time. In order to reduce energy consumption, idle servers might be put in hibernation mode when the load on the infrastructure is low. As such, an AE might exist that offers management functions to turn on or off devices. Another AE, which executes a management algorithm that decides when to put servers in or out of hibernation, requires this function to perform its tasks. It could, for example, request a management service to activate specific hibernating servers. An example semantic description of the offered function to turn on devices and the requested function to activate servers is shown in Table I.

The service matchmaking algorithm checks if the offered and requested descriptions match as follows. First, the algorithm constructs the bipartite graph mapping the inputs. As both descriptions contain only a single input SWRL class atom, the algorithm only needs to check if *dng:PhysicalDevice(?d)* subsumes *CloudServer(?s)*. The ontological model in Figure 3 shows that *CloudServer* is indeed an indirect subclass of *dng:PhysicalDevice*. Additionally, both $?d$ and $?s$ are SWRL variables, so the arguments of both atoms match. The two input atoms will thus be connected in the bipartite graph. For this very simple graph, with two vertices and one edge, the maximum bipartite matching equals the graph itself and it is thus stored by the algorithm. As neither service description contains any outputs, the second step of the algorithm is skipped. In the third step, the preconditions are matched. Although the offered service description contains no preconditions and the requested does, both precondition sets are still compatible. More specifically,

Table I. An example of an offered and requested service description to turn on devices and activate servers respectively

IOPEs	Offered	Requested
inputs	dng:PhysicalDevice(?d)	CloudServer(?s)
outputs		
preconditions		hasState(?s, Hibernating)
effects	hasState(?d, TurnedOn)	hasState(?s, TurnedOn)

Table II. A service description of a management function that allows AEs to allocate physical resources to a virtual machine

IOPEs	SWRL atoms
inputs	VirtualMachine(?v), QuantifiableResource(?r), xsd:long(?i)
outputs	
preconditions	CloudServer(?s), executes(?s, ?v), hasState(?s, TurnedOn), consistsOf(?s, ?r), hasCurrentValue(?r, ?c), hasLiteralValue(?c, ?l), hasMaximum(?c, ?m), swrlb:substract(?d, ?m, ?l), swrlb:greaterThanOrEqual(?d, ?i)
effects	hasReservedResource(?v, ?t), hasEntity(?t, ?r), hasValue(?t, ?u), hasLiteralValue(?u, ?i)

the set of offered preconditions still subsumes the set of requested preconditions, as the definition we proposed allows atoms in the subsumed set to have no match in the subsuming set. Intuitively, unmatched preconditions in the requested service are not a problem, as even under these conditions the offered service can still be executed. Subsequently, the match between effects is determined. Both service descriptions are compatible if the requested atom *hasState(?s, TurnedOn)* subsumes the offered *hasState(?d, TurnedOn)*. This is trivially clear, as both object property atoms refer to the same object property. Additionally, there is a match between the arguments, as in both cases the first argument is a SWRL variable and the second argument refers to the same *DeviceState* individual *TurnedOn*. The fifth and final step of the algorithm constitutes the matching of variable bindings. As previously stated, a map is created that links all SWRL variables in the offered service to the atoms in which they occur. In this case, there is only variable $?d$, which is linked to both of the atoms occurring in the service description. Subsequently, the algorithm iterates over all 4-tuples of IOPE maximum bipartite matchings. As there are no defined outputs, and the offered service defines no preconditions, the matchings of outputs and preconditions are empty. The other two matchings both contain a single edge, connecting *dng:PhysicalDevice(?d)* to *CloudServer(?s)* and *hasState(?d, TurnedOn)* to *hasState(?s, TurnedOn)*. As such, the algorithm needs to check if the atoms *CloudServer(?s)* and *hasState(?s, TurnedOn)* contain the same SWRL variable argument on the position of $?d$ in the corresponding atoms of the offered service. As they both refer to the variable $?s$ on this position, the variable bindings are compatible. The matchmaking algorithm thus concludes that the offered service description meets the requested requirements.

In conclusion of this section, a more complex service description is given to further illustrate the expressive power of the proposed SWRL-based approach. An important management issue in cloud computing environments is the allocation of physical server resources to virtual machines. As such, we formalize a service description of a management function that allows AEs to allocate resources to virtual machines. The details are shown in Table II. The function takes three inputs, the virtual machine for which the resources should be reserved, the physical resource that should be reserved and the exact amount of resources that should be reserved. It has no outputs, but does change the state of the environment and therefore has several preconditions and effects. Informally, the preconditions makes sure that the virtual machine and the reserved resource are actually on the same cloud server (atoms 1 and 2), the server is turned on (atom 3) and the requested amount of resources is actually available (atoms 4-9). The latter is done by using several built-in atoms. The current $?l$ and maximum $?m$ used resources of $?r$ are defined and substracted from one another.

This results in the variable $?d$ containing the currently available resources of $?r$, which must be greater than or equal to $?i$. Finally, the effects state that after the execution of the function finishes, the requested amount of resources will be reserved for the specified virtual machine.

7. EVALUATION & RESULTS

Semantic reasoning and ontologies facilitate the understanding and interpretation of context information and management functions by autonomic elements. However, this comes at the cost of performance degradation, as ontological reasoning is widely known to scale poorly in terms of ontology size. In this section, we explore the effects of semantic reasoning on the SCB's performance, as a function of several parameters. As a performance metric, the total *reasoning time* is used, which corresponds to the total execution time of the algorithms and excludes any other delays such as network latency.

The remainder of this section considers performance of three aspects of the SCB in separate subsections. These subsections evaluate performance of creating filter rule subscriptions, context publication and the service matchmaking algorithm, respectively. However, first follows a brief description of the implemented prototype and the evaluation setup.

7.1. Implementation & Evaluation Setup

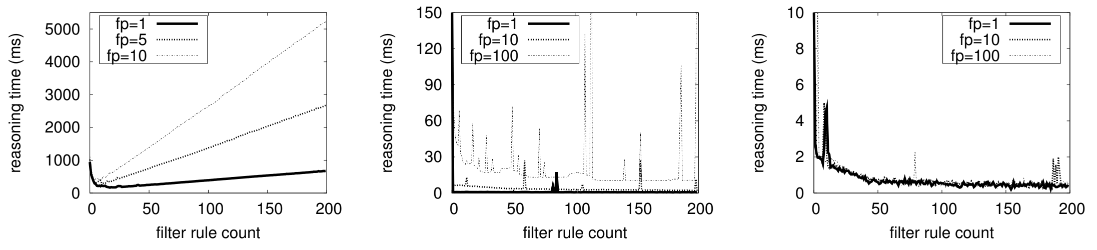
A prototype implementation of the context dissemination and service matchmaking components was created specifically for this evaluation. The prototype was built in Java, based on the Pellet OWL 2 reasoner version 2.1.1[**] and OWL-API version 3.0.0[††]. In addition to OWL 2 reasoning, Pellet supports DL-safe SWRL rules [32]. Jena rules are supported through Jena's own built-in rule reasoner, of which version 2.6.2[‡‡] was used. All evaluations were performed using a core ontology containing a subset of DENON-ng, the complete SWRL temporal ontology and our own cloud computing model (as described in Section 4). In total, the core ontology consists of 160 classes, 37 object properties, 21 data properties, 19 individuals and a total of 490 asserted axioms. The used DENON-ng subset contains 135 classes in total, including all higher level classes of the dng:Entity subtree and the complete dng:Resource substree. For simplicity, some class relationships, not needed for the evaluated use-case, were omitted, including those of the policy model, identity and behavioural aspects. Finally, the tests were performed on a server with two dual-core AMD Opteron 2 Ghz processors and 4 GB memory, running Debian 5.0 and Linux kernel 2.6.30.

The evaluation setup is based on the use case described in Section 4. For the evaluation of the context dissemination component, filter rules are used that admit messages that contain information about *QuantifiableResource*s with a current load higher than a randomly generated percentage. This filter is thus similar to the example used throughout Section 5, which admits messages about *QuantifiableResource*s with a current load higher than 95%. The messages used in the evaluation similarly contain context information about one or more *QuantifiableResource*s with a randomly generated current load. The complexity of filter rules and messages is expressed as the number of payloads. A single payload contains information about one *QuantifiableResource*. In the evaluation of the semantic matchmaking algorithm, service descriptions similar to the one depicted in Table II are used. It describes a management function that allows AEs to reserve resources for a specific *VirtualMachine*.

7.2. Filter Rule Subscription

The SCB's context dissemination component operates as a publish/subscribe system. It thus consists of two steps, subscription through filter rules and publication of messages. This section considers scalability in terms of reasoning time of the subscription step. In the experiment, 200 filter rules

[**]http://clarkparsia.com/pellet
[††]http://owlapi.sourceforge.net/
[‡‡]http://jena.sourceforge.net/



(a) OWL (b) SWRL (c) Jena

Figure 8. The evolution of total reasoning time as more filter rules are added to the SCB

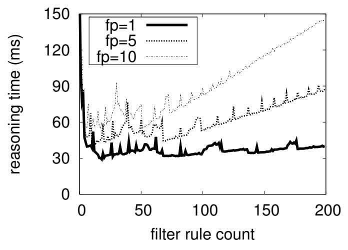


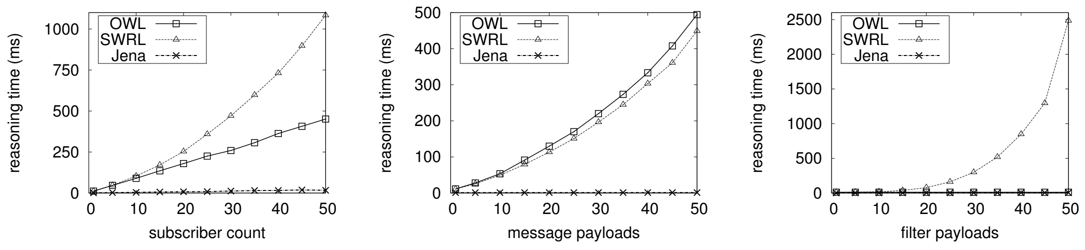
Figure 9. The evolution of total reasoning time as more OWL filter rules are added to the SCB with satisfiability and consistency checks turned off

were added sequentially to the SCB. The entire process was repeated 30 times. The results depicted in Figure 8 show the average reasoning time for adding each of these 200 filters, averaged over the 30 iterations. More specifically, every value x on the x-axis shows the time (in milliseconds) it took to add a filter rule to the SCB when $x - 1$ filter rules have already been added. The figure depicts reasoning time as a function of the number of filter rules for the three proposed filter rule approaches (i.e., OWL, SWRL and Jena). All graphs depict results for filters with an increasing number of payloads (*fp*). The more payloads a filter rule contains, the more complex it becomes.

As described in Section 5.1, the use of OWL filter rules allows their satisfiability and consistency to be checked by the OWL reasoner. On the other hand, this is not possible when using SWRL or Jena filter rules. The results depicted in Figure 8 demonstrate that performing such satisfiability and consistency checks severely impacts scalability. Figure 8a shows that, for OWL filters, the subscription time increases as the number of filters in the SCB increases. Additionally, the effect becomes worse as their complexity (i.e., *fp*) increases. On the other hand, as depicted in Figures 8b and 8c, SWRL and Jena show the desired scaling behaviour, as their subscription performance does not degenerate as more filter rules are added to the ontology. In terms of filter rule complexity, SWRL shows a slight degradation in performance, while Jena shows none. However, even for very complex rules ($fp = 100$), adding a subscription when the ontology already contains 200 filter rules takes, on average, only 10 and less than 1 ms for SWRL and Jena respectively. Note that all three approaches show a slight increase in reasoning time when adding the first few filters. This is caused by the dynamic class loading behaviour of Java and can thus be safely ignored. Additionally, the SWRL and Jena curves show random peaks, which are caused by measurement inaccuracies occurring due to measured times being in the range of only a few milliseconds.

For reference, Figure 9 shows the reasoning time for adding OWL filter rule subscriptions with satisfiability and consistency checks turned off. Surprisingly, the reasoning time still increases as more filter rules are added to the SCB. However, the reasoning time is reduced to over one thirtieth as compared to using OWL filter rules with satisfiability checks. Consequently, scalability is greatly increased.

In summary, SWRL and Jena filter rules have been shown to scale well, both in terms of subscription count and rule complexity. From the poor scaling behaviour of the OWL approach



(a) subscription filter count (b) message complexity (c) filter rule complexity

Figure 10. The average reasoning time for publishing a single message over the SCB, as a function of number of subscribers, message complexity and filter rule complexity.

it can be concluded that checking the satisfiability and consistency of filter rules severely impacts overall performance, making it unsuitable for large-scale dynamic systems. We have shown that turning off satisfiability and consistency checks greatly increases OWL's scalability. However, even then its scalability remains worse than that of SWRL and Jena.

7.3. *Context Publication*

An important aspect of the SCB is the publication of context messages. In contrast to filter rule subscriptions, which only change occasionally, the publication of context information happens frequently in highly dynamic environments, such as the management of future networks. As such, it is important that messages are matched with filter rules swiftly. This section further explores performance of the context publishing component, which performs the actual matching and dissemination of the messages. In the experiment, the effect on performance of three different parameters (i.e., subscription filter count, message complexity and filter rule complexity) was evaluated. For every evaluated combination of the three parameters, 500 messages were sequentially published unto the SCB. Filter rules of the same type as in the first experiment were used, but with a fixed load threshold of 50%. Additionally, every message payload contains information about a single *QuantifiableResource* with a randomly generated current load between 0 and 100%. The depicted reasoning time equals the time it takes to match the message with all subscription filters, averaged over the last 450 published messages. The first 50 messages are ignored, as the reasoning time for sending the first few messages is adversely influenced by Java's dynamic class loading behaviour. All experiments were repeated for OWL, SWRL and Jena filter rules. Figure 10 depicts the results.

The results shown in Figure 10a depict the reasoning time as a function of the number of subscription filters, for messages and filters with 1 payload each. The graph shows that SWRL and OWL filter rules scale poorly in terms of the number of subscription filters. In a scenario with 50 subscription filters, it takes over 1 second to publish a single message using SWRL filters and almost 500 milliseconds when using OWL. Such large delays are obviously unacceptable in a large scale dynamic network management scenario where context is constantly being exchanged between components. The fast degradation of these approaches is a consequence of the fact that the reasoner performs OWL inferencing when matching messages to filter rules, which is known to scale poorly. As such, a third approach, based on Jena rules was proposed. In contrast, it does not perform OWL inferencing, but merely applies the rules to the asserted RDF graph. This is clearly reflected in the results, as the Jena approach scales much better. Even for 50 subscription filters, publishing a messages takes only 16 ms on average, which means that over 60 semantic messages can be published per second.

Figure 10b shows the reasoning times as a function of the message complexity, for 1 subscription filter with 1 filter payload. The results are in accordance with previous observations and once again show that the message publishing process scales much better when using Jena rules. There is actually no noticeable degradation as message complexity increases. Specifically, for a message

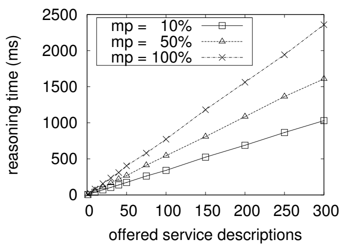


Figure 11. The evolution of total reasoning time as a function of the number of offered service descriptions, for different percentages of matching descriptions (*mp*)

with 1 payload, publishing takes, on average, 1.15 ms, while for a message with 50 it takes only 1.25 ms.

Finally, the graphs in Figure 10c show scalability in terms of filter rule complexity. SWRL once again shows poor scalability. However, OWL and Jena show no performance degradation. This shows that determining if a message instance belongs to a certain OWL filter rule is independent of the filter rule's complexity. On the other hand, SWRL rule complexity does greatly influence performance.

Several conclusions follow from these observations. First, SWRL performs worst in terms of subscription filter count and filter complexity, while OWL performs worse in terms of message complexity. Pure OWL reasoning thus scales worse as a function of increasing ABox size (i.e., number of individuals), while SWRL reasoning scales worse in terms of increasing TBox size (i.e., number of classes and SWRL rules). Second, it was shown that only Jena rules are currently suitable for usage in a large-scale scenario containing many publishers and subscribers. It has been shown to scale very well in terms of subscription filter count, message complexity and filter rule complexity.

7.4. Service Matchmaking

In contrast to the context dissemination component, the service matchmaker has less stringent timing constraints. Although it is still expected to react in a timely fashion, its delay can be in the order of seconds, rather than milliseconds. This section explores the effect of several parameters on the service matchmaker's performance in terms of execution time. The matchmaker's performance is influenced by two parameters, the number of service descriptions it offers and the percentage of these descriptions that actually match (i.e., are compatible with) the requested service. Obviously, the service matchmaker has to iterate over all offered service descriptions in order to find the matches. The number of offered service descriptions is thus expected to be directly proportional to the execution time of the matchmaker. However, as was explained in Section 6, the service matchmaker is often capable to detecting incompatibilities between service descriptions early on during the comparison. For example, if the offered service's input parameters do not match the requested ones, the matchmaker no longer needs to check outputs, preconditions and effects. As such, determining matchings takes longer if the services are actually compatible. Consequently, execution time depends on the percentage of offered service descriptions that match the requested service.

During each experiment run, the matching process was repeated 100 times for the same requested service description. The first 50 iterations are ignored, once again to negate the effects of Java's dynamic class loading. The depicted results are averaged over the last 50 iterations. The used service descriptions are based on the complex example given in Table II, which allows server resources to be reserved for virtual machines. Of the offered service descriptions that do not match the requested, 2 out 3 have an incompatibility in the inputs, while the other third has an incompatibility in the preconditions.

The experimental results are depicted in Figure 11. The graph shows the execution time of the matchmaking algorithm as a function of the total number of offered service descriptions, for different percentages of matching descriptions (*mp*). The graph clearly shows that there is a direct linear relation between reasoning time and both the number of offered service descriptions and the percentage of matches. In reality, the amount of service descriptions that actually match with the requested service is expected to be very low. As a wide range of differing services will be offered and requested. Consequently, the reasoning time will be significantly reduced. Even when 10% of the offered descriptions match with the requested service, the reasoning time is reduced by over half compared to when all service descriptions match. Additionally, these results show that on the test server, the matchmaker is capable of evaluating 300 possible matches, with a match rate of 10%, in under 1 second.

8. CONCLUSION

This article presents the Semantic Communications Bus (SCB), which facilitates the communication and interaction between autonomic management elements (AEs) and network resources. It supports the semantic dissemination of context and matchmaking of service descriptions. This article presented several novel contributions. First, we proposed three alternative methods for representing filter rules, with differing inferencing capabilities, expressiveness and performance. Additionally, we have shown how existing semantic reasoners can be employed to match these rules with context. Second, we proposed a method for modelling service inputs, outputs, preconditions and effects (IOPEs) by means of SWRL atoms. In contrast to existing work, our proposed matchmaking algorithm takes into account semantic links between different IOPE atoms.

In a federated network management scenario, context information, representing the state of the network and its resources, needs to be efficiently disseminated between AEs in order to detect and solve problems in a timely fashion. The proposed context dissemination approach uses ontologies and semantic reasoning, which support context filtering based on the actual meaning of information instead of static string patterns or predefined topics. Three different reasoning approaches were introduced, respectively based on OWL, SWRL and Jena. As these approaches have different inferencing capabilities, expressiveness and performance, we believe they all have their merits. The evaluation of our implemented prototype shows that Jena-based filter rules exhibit the best scaling behaviour in terms of number of filters and message complexity. Jena allows context to be matched to filter rules in a matter of milliseconds, while for OWL and SWRL rules it takes several hundreds of milliseconds in larger scenarios. On the other hand, OWL and SWRL filter rules support advanced inferencing, which is not supported when using Jena rules.

When managing large-scale networks, detected problems can often not be solved locally. Consequently, AEs need to cooperate in order to solve the network's management issues. The semantic matchmaking algorithm proposed in this article allows AEs to discover the management services, offered by other AEs, they require in order to complete their management tasks. Additionally, by taking into account the preconditions and effects of the management services, AEs can determine their consequences on the state of the managed environment. As the IOPEs of these services are semantically defined using SWRL atoms, the matchmaking algorithm can determine compatibility between offered and requested functionality based on the meaning and inferred semantic relatedness of ontological concepts. This greatly augments its accuracy compared to traditional keyword-based matchmaking approaches. The evaluated prototype implementation shows that the algorithm can determine semantic and functional compatibility between two service descriptions in a few milliseconds.

ACKNOWLEDGEMENT

Jeroen Famaey is funded by the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT-Vlaanderen) under grant no. 73185; Steven Latré is funded by the Fund for Scientific Research Flanders (FWO-Vlaanderen).

REFERENCES

1. Jennings B, van der Meer S, Balasubramaniam S, Botvich D, Ó Foghlú M, Donnelly W, Strassner J. Towards autonomic management of communications networks. *IEEE Communications Magazine* 2007; **45**(10):112–121, doi:10.1109/MCOM.2007.4342833.
2. Strassner J, Kim SS, Won-Ki Hong J. The design of an autonomic communication element to manage future internet services. *Proceedings of the 12th Asia-Pacific Network Operations and Management Symposium*, 2009; 122–132, doi:10.1007/978-3-642-04492-2_13.
3. Christudas B. *Service-Oriented Java Business Integration: Enterprise Service Bus Integration Solutions for Java Developers*. Packt Publishing, 2008.
4. Serrano M, van der Meer S, Holum V, Murphy J, Strassner J. Federation, a matter of autonomic management in the Future Internet. *Proceedings of the 12th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2010; 845–849, doi:10.1109/NOMS.2010.5488357.
5. Jennings B, Brennan R, Donnelly W, Foley S, Lewis D, O'Sullivan D, Strassner J, van der Meer S. Challenges for federated, autonomic network management in the Future Internet. *Proceedings of the 1st IFIP/IEEE International Workshop on Management of the Future Internet (ManFI)*, 2009; 87–92, doi:10.1109/INMW.2009.5195942.
6. Famaey J, Latré S, Strassner J, De Turck F. A hierarchical approach to autonomic network management. *Proceedings of the 2nd IFIP/IEEE International Workshop on Management of the Future Internet (ManFI)*, 2010; 225–232, doi:10.1109/NOMSW.2010.5486571.
7. Strassner J, de Souza J, Raymer D, Samudrala S, Davy S, Barrett K. The design of a novel context-aware policy model to support machine-based learning and reasoning. *Cluster Computing* 2009; **12**(1):17–43, doi:10.1007/s10586-008-0069-4.
8. Latré S, van der Meer S, De Turck F, Strassner J, Won-Ki Hong J. Ontological generation of filter rules for context exchange in autonomic multimedia networks. *Proceedings of the 12th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2010; 575–582, doi:10.1109/NOMS.2010.5488448.
9. Famaey J, Latré S, Strassner J, De Turck F. An ontology-driven semantic bus for autonomic communication elements. *Proceedings of the 5th IEEE international conference on Modelling Autonomic Communication Environments (MACE)*, 2010; 37–50, doi:10.1007/978-3-642-16836-9_4.
10. Carroll JJ, Dickinson I, Dollin C, Reynolds D, Seaborne A, Wilkinson K. Jena: Implementing the semantic web recommendations. *Proceedings of the 13th International World Wide Web Conference*, 2004, doi:10.1145/1013367.1013381.
11. Petrovic M, Liu H, Jacobsen HA. G-ToPSS: Fast filtering of graph-based metadata. *Proceedings of the 14th international conference on World Wide Web (WWW)*, 2005; 539–547, doi:10.1145/1060745.1060824.
12. Wang J, Jin B, Li J. An ontology-based publish/subscribe system. *Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware (Middleware)*, 2004; 232–253, doi:0.1007/978-3-540-30229-2_13.
13. Ma J, Xu G, Wang J, Huang T. A semantic publish/subscribe system for selective dissemination of the rss documents. *Fifth International Conference Grid and Cooperative Computing (GCC)*, 2006; 432–439, doi:10.1109/GCC.2006.19.
14. Li H, Jiang G. Semantic message oriented middleware for publish/subscribe networks. *Sensors, and Command, Control, Communications, and Intelligence (C3I) Technologies for Homeland Security and Homeland Defense III*, vol. 5403, 2004; 124–133, doi:10.1117/12.548172.
15. Skovronski J, Chiu K. An ontology-based publish-subscribe framework. *International Conference on Information Integration and Web-based Applications Services*, 2006.
16. Carzaniga A, Rosenblum DS, Wolf AL. Design and evaluation of a wide-area event notification service. *Foundations of Intrusion Tolerant Systems*, 2003; 283–334, doi:10.1109/FITS.2003.1264940.
17. Keeney J, Roblek D, Jones D, Lewis D, O'Sullivan D. Extending siena to support more expressive and flexible subscriptions. *Proceedings of the Second International Conference on Distributed Event-Based Systems (DEBS)*, 2008; 35–46, doi:10.1145/1385989.1385995.
18. Petrovic M, Burcea I, Jacobsen HA. S-ToPSS: Semantic toronto publish/subscribe system. *Proceedings of the 29th international conference on Very large data bases (VLDB)*, 2003; 1101–1104.
19. Wang J, Jin B, Li J, Shao D. A semantic-aware publish/subscribe system with RDF patterns. *Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC)*, 2004; 141–146, doi:10.1109/CMPSAC.2004.1342818.
20. Shen G, Huang Z, Zhang Y, Zhu X, Yang J. A semantic model for matchmaking of web services based on description logics. *Fundamenta Informaticae* 2009; **96**(1):211–226, doi:10.3233/FI-2009-175.
21. Paolucci M, Kawamura T, Payne TR, Sycara KP. Semantic matching of web services capabilities. *Proceedings of the First International Semantic Web Conference (ISWC)*, 2002; 333–347, doi:10.1007/3-540-48005-6_26.
22. Bener AB, Ozadali V, Ilhan ES. Semantic matchmaker with precondition and effect matching using SWRL. *Expert Systems and Applications* 2009; **36**(5):9371–9377, doi:10.1016/j.eswa.2009.01.010.
23. Klusch M, Fries B, Sycara K. Automated semantic web service discovery with OWLS-MX. *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, 2006; 915–922, doi:10.1145/1160633.1160796.
24. Klusch M, Fries B, Sycara K. OWLS-MX: A hybrid semantic web service matchmaker. *Web Semantics: Science, Services and Agents on the World Wide Web* 2009; **7**(2):121–133, doi:10.1016/j.websem.2008.10.001.
25. Sbodio ML, Martin D, Moulin C. Discovering semantic web services using SPARQL and intelligent agents. *Web Semantics: Science, Services and Agents on the World Wide Web* 2010; **8**(4):310–328, doi:10.1016/j.websem.2010.05.002.
26. Serrano MJ, Serrat J, Strassner J, Ó Foghlú M. Management and context integration based on ontologies, behind the interoperability in autonomic communications. *Proceedings of the SIWN International Conference on Complex Open Distributed Systems (CODS)*, 2007.

27. Strassner J, Souza JN, van der Meer S, Davy S, Barrett K, Raymer D, Samudrala S. The design of a new policy model to support ontology-driven reasoning for autonomic networking. *Journal of Network and Systems Management* 2009; **17**(1):5–32, doi:10.1007/s10922-009-9119-3.
28. Wong A, Ray P, Parameswaran N, Strassner J. Ontology mapping for the interoperability problem in network management. *IEEE Journal on Selected Areas in Communications* 2005; **23**(10):2058–2068, doi:10.1109/JSAC.2005.854130.
29. O'Conner MJ, Das AK. A lightweight model for representing and reasoning with temporal information in biomedical ontologies. *Proceedings of the International Conference on Health Informatics (HEALTHINF)*, 2010.
30. Krötzsch M, Rudolph S, Hitzler P. On the complexity of horn description logics. *Proceedings of the 2nd Workshop on OWL: Experiences and Directions (OWLED)*, 2006.
31. Alt H, Blum N, Mehlhorn K, Paul M. Computing a maximum cardinality matching in a bipartite graph in time $o(n^{1.5}\sqrt{m/\log n})$. *Information Processing Letters* 1991; **37**(4):237–240, doi:10.1016/0020-0190(91)90195-N.
32. Sirin E, Parsia B, Grau BC, Kalyanpur A, Katz Y. Pellet: A practical OWL-DL reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web* 2007; **5**(2):51–53, doi:DOI:10.1016/j.websem.2007.03.004.

AUTHOR BIOGRAPHIES

Jeroen Famaey obtained a masters degree in computer science from Ghent University, Belgium, in June 2007. Since August 2007 he is affiliated as a Ph.D. student with the Department of Information Technology at Ghent University, where he is supported by a Ph.D. grant of the Flemish Institute for the Promotion and Innovation by Science and Technology (IWT-Vlaanderen). His main research interests include autonomic network management, semantic reasoning and multimedia content delivery in broadband access networks. He was also involved in the European FP7 ALPHA project and the EUREKA CELTIC RUBENS project. Currently, he is participating in the European FP7 STREP OCEAN project.

Steven Latré obtained a masters degree in computer science from Ghent University, Belgium, in June 2006. Since august 2006 he is affiliated as a Ph.D. student with the Department of Information Technology at Ghent University, where he is supported by a Ph.D. grant of the Fund for Scientific Research Flanders (FWO-Vlaanderen). His main research interests include the use of autonomic communications to optimize the Quality of Experience management of multimedia services in broadband access networks. He was also involved in the IST FP6 project MUSE, EUREKA CELTIC project RUBENS and is currently participating in the FP7 STREP ECODE project as well as several other, national, projects.

John Strassner is a Professor of Computer Science and Engineering at POSTECH, and leads its Autonomic Computing group. Previously, he was a Visiting Professor at Waterford Institute of Technology in Ireland. Before that, he was a Motorola Fellow and Vice President of Autonomic Research at Motorola Labs, where he was responsible for directing Motorola's efforts in autonomic computing and networking, policy management, and knowledge engineering. Previously, John was the Chief Strategy Officer for Intelliden and a former Cisco Fellow. John is a TMF Distinguished Fellow, and is the Chairman of the Autonomic Communications Forum, and the past chair of the TMF's NGOSS SID, metamodel and policy working groups, along with the past chair of several IETF and WWRF groups. He has authored two books (Directory Enabled Networks and Policy Based Network Management), written chapters for 5 other books, and has been co-editor of 5 journals dedicated to network and service management and autonomics. John is the recipient of the Daniel A. Stokesbury memorial award for excellence in network management, a recipient of the Albert Einstein award for autonomic networking, is a member of the Industry Advisory Board for University of California Davis, and has authored over 265 refereed journal papers and publications.

Filip De Turck leads the network and service management research group at the Department of Information Technology of Ghent University, Belgium and the IBBT (Interdisciplinary Institute of Broadband Technology, Flanders). He received his Ph.D. degree from Ghent University in 2002 and his M.Sc. in Electronic Engineering from Ghent University in 1997. He was a part-time professor from October 2004 till October 2006 and a full-time professor since October 2006 in the area of telecommunication and software engineering. He is author or co-author of more than 300 refereed papers published in international journals and conferences. His main research interests include scalable software architectures for telecommunication network and service management, performance evaluation and design of new telecommunication architectures and services. In this research area, he is involved in several research projects with industry and academia, both on a national and European scale (FP7 projects). He is a member of the network management research community by serving to Technical Program Committees of many network management conferences such as NOMS, IM, CNSM, APNOMS, and several workshops held in co-location.