# Automated Context Dissemination for Autonomic Collaborative Networks through Semantic Subscription Filter Generation

Steven Latré[a], Jeroen Famaey[a], John Strassner[b], Filip De Turck[a]

[a]*Ghent University - Department of Information Technology - IBBT, Belgium*
*steven.latre@intec.ugent.be*
[b]*Software R&D Laboratory, Huawei USA*

**Abstract**

The current manual management of services and applications in today's telecommunication networks is becoming increasingly complicated. In the Future Internet, management is assumed to be automated by introducing an autonomic layer of distributed management elements. These distributed management elements need to collaborate with each other to ensure end-to-end quality guarantees. In this article, we focus on the context dissemination between such collaborative management elements. Context dissemination is the exchange of all relevant management data and knowledge between the elements. Collaborating elements typically generate large amounts of context and it is important to filter this continuous stream. We propose a context dissemination approach that automates the context exchange between elements. The approach enables the automated generation of semantic subscription filters. Subscription filters allow an element to define where, how, and when context needs to be requested from other entities. Moreover, the proposed approach allows making the subscription filter generation dependent on the context. We present algorithms that intelligently filter the knowledge that is stored in the ontology. The results show that the generation of subscription filters can be done in the order of tens of milliseconds.

*Keywords:* Context dissemination, Autonomic collaborative networks, Future Internet, Semantic Approach, Service Management, OWL, RDF

## 1. Introduction

In recent years, the Internet has evolved from a best-effort packet forwarding network towards a service-oriented delivery framework that supports rich and complex services and applications. To support their management, delivery guarantees must be provided in terms of Quality of Service and Quality of Experience (QoE). Therefore, it is becoming too costly and complex to continue managing these services and applications manually. In the Future Internet, a more automated management approach is required that allows self-governing the network by introducing an intelligent autonomic layer on top of today's network [1]. This autonomic layer features a decision making process that supports an automated management of the network's resources. Given the scale and form of the current Internet, it is not possible to maintain a single decision making entity. Instead, an autonomic network management substrate consists of specialized distributed decision making components, called autonomic elements (AEs).

To ensure end-to-end management of the Future Internet, the different AEs need to collaborate with each other. This collaboration is crucial in achieving a well-performing autonomic management framework: it guarantees that AEs do not contradict each other's decisions. One of the primary challenges in designing a collaborative autonomic network is the efficient dissemination of data. In order to collaborate with each other, AEs require management data from other AEs such as (i) monitoring reports falling outside their authority, (ii) decisions taken by other AEs and (iii) knowledge that was inferred by other AEs. All this management data can be seen as context. More specifically, throughout this article, we call all management data that is relevant to the decision making process of an AE, context. More specifically. we use the following definition of context from DEN-ng [2]: *"The Context of an Entity is a collection of measured and inferred knowledge that describe the state and environment in which an Entity exists or has existed"*. In particular, our definition emphasizes two types of knowledge: facts (that can be measured) and inferred data, which results from machine learning and reasoning processes applied to past and current context. It also includes context history, so that current decisions based on context may benefit from past decisions, as well as observations of how the environment has changed.

Many autonomic architectures use the publish-subscribe paradigm to communicate with other AEs and enable the collaboration between them. The publish-subscribe paradigm allows consumers of context (i.e., AEs) to express their interest in context by subscribing to that context at the producer (i.e., another AE) [3]. This subscription is done through filters, which define what type of context producers need to send to the consumers. Traditionally, these subscription filters are statically defined by the consumer interested in the context. In this article, we focus on the automated generation of subscription filters between collaborating AEs in the autonomic networking paradigm. We focus specifically on the context dissemination process inside an administrative domain and declare the context dissemination

between administrative domains as out of scope. In this context, an administrative domain is a subnetwork that is managed by a single network operator. Inside an administrative domain, the main challenge of context dissemination is its scalability. Large amounts of context are generated by different AEs and the type of context that needs to exchanged can fluctuate rapidly. Instead, between administrative domains, other challenges are notable such as the negotiation of the context dissemination: this is part of future work. We propose a context dissemination process that allows coping with the fluctuation of context requirements by generating the subscription filters automatically. The process takes into account the requirements and goals of a specific AE in terms of context, as well as changes in the state of the environment.

The remainder of this article is structured as follows: an illustrative use case of context dissemination for multimedia delivery in an access network is described in Section 2. Section 3 discusses similar work in the field of context modeling in network management. The context dissemination process is described in full in Section 4 and Section 5, which focus on the design of the semantic model and the algorithmic contributions, respectively. Finally, Section 6 presents detailed evaluation results of the performance of the proposed approach.

## 2. Use case: autonomic multimedia service management in access networks

To provide a better understanding of the context dissemination process, we describe it for an illustrative use case. This use case is centered around the autonomic management of multimedia services in access networks. We assume that a service provider is offering a number of multimedia services to the end users. These multimedia services consist of the typical video-based services (e.g., digital TV broadcasting, Video on Demand) that are part of today's triple play offers in IPTV environments. As illustrated in Figure 1, the content originates at the video head-end in the access network. The service provider can use its own access network, consisting of caches, access routers and access nodes to stream the videos to its customers, residing in the home network. The streaming is done to set-top boxes (STBs), which are devices located in the home network that are capable of playing the video on a television screen. Despite being located in the home network, the service manager is typically the owner of a STB and therefore has access to it for management purposes.

To manage the multimedia services, a set of collaborating AEs are logically deployed on top of the access network. Each AE governs a specific type of device. The AEs collaborate with each other by (i) exchanging context with each other and (ii) requesting management actions from one AE to another, which can then propagate this request to the device(s) it manages. For example, every STB has its own AE that can monitor the delivered quality of the multimedia streams and offers an interface to remote AEs to make configuration changes (e.g., a request to send more context or a request to alter a configuration parameter at the video client).

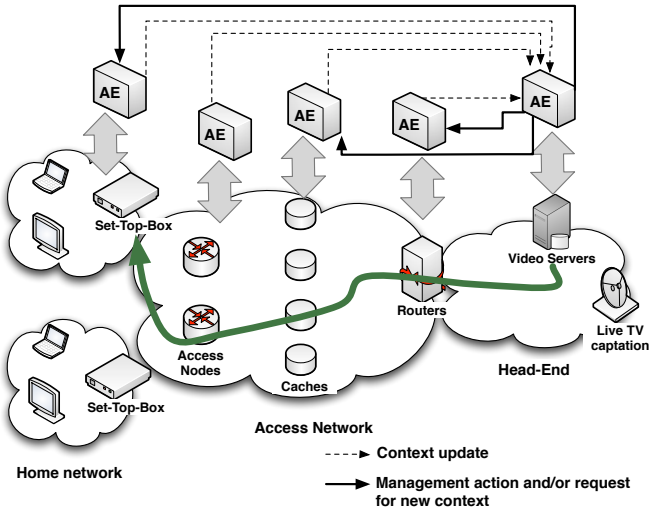



Figure 1: Overview of the illustrative use case, representing a multimedia delivery scenario in an access network. Different autonomic elements (AEs) exchange context with each other to maintain the highest possible video quality.

In this use case, we focus on the context dissemination process of the video head-end AE. We assume that the video head-end AE has a number of management tasks available to manage the multimedia services. For example, it can reduce the streamed video quality if congestion occurs in the network. Alternatively, it can protect the video against packet loss due to lossy links by adding additional redundancy. In order to determine the optimal configuration of these management tasks, it requires knowledge about the status of the managed network. However, at the same time, it is useless to continuously request all this knowledge if the current management configuration is streaming the video at a satisfiable quality. Hence, there are varying context requirements, depending on the state of the network environment.

In terms of context, the video head-end AE initially only requires the overall quality of the streamed video. As such, it configures a subscription filter at the STB head-end to send periodic updates of the perceived video quality. It is only useful to request additional knowledge when these periodic updates signal a drop in video quality.

Suppose that part of the network becomes congested. This will result in packet loss in the network and ultimately in a drop in video quality. The video head-end AE should detect this and now requires a new set of subscription filters. This is because the decision making components in the AE now require additional knowledge to be able to track down the reason for the video quality drop. The context dissemination process, which will be described in Section 4 detects these new context requirements and will generate a new set of subscription filters. More specifically, these subscription filters will now also request context from the other nodes in the access network such as QoS parameters to determine the root cause of the problem.

Note that the generated subscription filters are semantic constructs. This means that, in contrast to a purely syntactic match-

ing, they can be semantically interpreted by the context producer to determine whether the context needs to be exchanged with the context consumer. Specifically related to the use case this means that, instead of generating a subscription filter that specifically defines the name and value of the packet loss on a particular router, a semantic filter can define that it wants "All QoS parameters related to flow X", where X is the flow that suffers from a drop in quality. The context producer can then semantically infer whether the QoS parameters they have available are related to flow X (e.g., because a router routes traffic from flow X or because a cache stores part of the video). A more in-depth discussion of these semantic subscription filters is presented in Section 5.2.2.

Based on this novel knowledge, the problem can be detected and solved (e.g., by reducing the streamed quality of some video to free additional resources) and the video quality is restored. Once the video quality is restored, an additional set of subscription filters can again be generated: the QoS parameters from other AEs in the access network become obsolete again and the set of subscription filters can be reverted to the original set. This dynamic generation of subscription filters ensures that the context dissemination process is scalable and that no useless context is exchanged. As will be detailed in Section 4, the subscription filters are automatically generated by (i) translating the contextual requirements of the decision making components residing in an AE, (ii) inferring the current contextual requirements based on the current environment state and (iii) summarizing the available information, residing in the model to improve the performance of the translation step.

## 3. Related work

In this section, we describe related work in the area of context retrieval and dissemination, its use of semantics and its application to network management challenges.

### *3.1. Context retrieval and dissemination*

In information systems, there is an increased attention towards context retrieval and dissemination research. While many solutions focus on effective presentation of the context (e.g., Bouras *et al.* [4]) another important challenge in context-related research is the dissemination of the context. Disseminating context in a scalable and effective way is key in designing a well performant information system. A survey of the current challenges in designing context retrieval systems is presented by Tamine-Lechani *et al.* [5]. As the benefits of an effective context dissemination are broad and generic, it has been applied to many problem domains in information systems. For example, Bikakis *et al.* [6] present motivating scenarios for designing a context exchange and retrieval system for ambient intelligence environments. Bakhouya *et al.* and Abdou *et al.* discuss how context dissemination can benefit the communication in Vehicular Ad Hoc Networks [7, 8]. Similarly, Abdou *et al.* proposes a novel communication strategy. Specifically to the field of network management, broadcasting the context is typically not possible due to scalability limits. In the Future Internet, contextual data such as configuration management data is typically stored distributed in high performant databases (e.g., the CMDBf solution from the DMTF [9]). The dissemination of aggregated monitoring information and context is necessary in order to guarantee the continuous satisfaction of network-wide goals. The publish-subscribe paradigm [3] is well suited to offer these functionalities. It allows interested parties to subscribe to specific types of events. When an event that matches the subscription is published, it is routed accordingly.

### *3.2. Semantic context dissemination*

To ensure inter-domain understanding and interoperability, the exchanged context should be semantically annotated. This is also argued by Jurisica *et al.* [10]: they survey the use of ontologies for knowledge management in information systems and conclude that ontologies are needed to attach meaning to knowledge management systems and solve syntactic incompatibilities through semantic integration [11]. Roantree *et al.* [12] show the strength of applying semantics to context dissemination systems. The use of semantics and the publish-subscribe model has been successfully combined to many network-based communication information structures in the past. Renault *et al.* apply it to an Information-Centric Networking (ICN) [13] architecture [14]. Their work focuses on the semantic annotation of the content itself and its impact on the ICN paradigm, while we focus on the context dissemination process. Petrovic *et al.* proposed a subscription (query) language suitable for filtering large numbers of RSS (Really Simple Syndication) documents [15]. In [16], a semantic approach is described. It extends the traditional attribute-value pair-based approach with capabilities to process syntactically different, but semantically-equivalent, information, by using an ontology. Our work is different from both of these approaches, in that the ontology used in [16, 15] is limited to RDF hyponym/hypernym relationships, whereas our approach can use different linguistic and functional relationships. In addition, we use OWL in one of our reasoning algorithms, as opposed to RDF, which provides greater flexibility and representation of semantics. Finally, our approach focuses on the automatic generation of subscription filters, while Petrovic *et. al* focus on subscription filter matching.

In [17], the DARPA Agent Markup Language (DAML) and the Ontology Inference Layer (OIL) were used to provide semantic publish-subscribe capabilities. A DAML+OIL reasoner was implemented for checking instance inferences between each subscriber class description and publisher instance description to see if they match. A drawback of this approach is that the DAML+OIL ontologies must be agreed beforehand by the subscribers and publishers. Our approach requires no such restriction, and uses more powerful inferencing. A similar approach is used by Wang et al. [18]. However, in this work, messages are represented in DAML+OIL, instead of message topics. Skovronski & Chiu propose a semantic publish-subscribe system that uses SPARQL queries as subscriptions [19]. However, this method scales poorly with an increasing number of publishers. When 16 publishers are registered with the publish-subscribe system, processing a single message took around 16

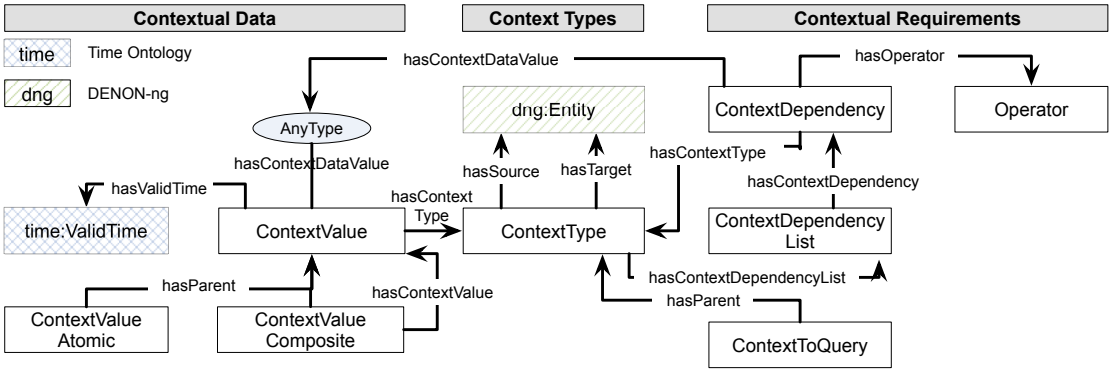



Figure 2: Overview of the ontological context model, highlighting the main concepts and the interaction with related ontologies such as DENON-ng and the Time ontology.

seconds. In this paper, we were able to improve this scalability significantly.

### *3.3. Semantic context dissemination for network management*

The publish-subscribe paradigm has also been successfully applied to the dissemination of semantic information in large-scale network management environments under the banner of *knowledge based networking* (KBN) [20]. It is an extension of content based networking (CBN) [21], which involves the forwarding of events across a network based on subscription filters based on the (meta-)data of the event's contents. KBN extends this and states that the semantics of messages play an important part in the matching of publications to subscriptions. To this end, the Sienna publish-subscribe system, originally devised for CBN, was extended with more expressive semantics for the specification of subscription filters [22] to satisfy the KBN vision. Carzaniga *et al.* [21] additionally propose a set of efficient and scalable routing strategies to forward messages from publishers to interested subscribers. The KBN extension [22], proposed by Keeney *et al.*, adds limited support for semantic messages and filters. The JITIK framework presented by Brena *et al.* [23] takes a similar approach. Ontologies are distributed across distributed agents: each agent has a common ontology with local additions to support its specific management tasks. Similar to our approach, ontologies are used to interpret the context that is exchanged semantically.

This article builds further on previous work as described in [24], where an initial version of the subscription filter generation algorithm was proposed. While we use the idea of the initial subscription filter generation algorithm, this article presents a modified and more advanced generation algorithm specifically and broader context dissemination process in general. Hence, this article contains several new contributions compared to the work in [24]. First, the work in [24] assumed that the contextual requirements of the decision making components of an AE were already available in the ontology model. In this work, we present an algorithm that allows inferring these contextual requirements automatically. Second, in this article a more complex and ontology model is used, based on the DEN-ng information model [2], which improves the accuracy of the performance results. Third, while the work in [24] only featured an ontology-based reasoning algorithm, we now also propose a more scalable algorithm based on RDF. Fourth and finally, this article also features several summarization methods to reduce the size of the model.

## 4. Semantic Context Model

To support the automated generation of subscription filters, we use a semantic context model to describe, in formal terms, the contextual requirements of the management components and the applications that they are governing. In this section, we discuss the details of this semantic model. Figure 2 provides an overview of the main concepts in the context model. The basic notion of a context type is modeled in our model using the `ContextType` concept. A `ContextType` represents different types of context that can be requested from other components. Typical examples of context types are the video quality of a service or the experienced packet loss in a router.

The `ContextType` concept also has some descriptive relationships such as the `hasName` property that provides a name for this context type (e.g., PACKET_LOSS) and other optional properties (e.g., the `hasSource` and `hasTarget` properties that provide information where the context originated from and what entity it describes, respectively). Through these optional properties context can be described as broadly or narrowly as required by a specific task. For example, to describe all packet loss related context, it suffices to define a packet loss concept without a `hasSource` or `hasTarget` property.

The `hasSource` and `hasTarget` properties are linked with the `Entity` class of the DENON-ng model. DENON-ng [25] is an ontology-based subset of the DEN-ng information model. DEN-ng is an information model that can be used to model a telecommunication network. DEN-ng is 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 organizations. The `Entity` class of DEN-ng is the root class to which all concepts belong. As such, we are able to describe accurately to which entities in a telecommunication network the context belongs.

Contextual data is modeled through the `ContextValue` concept: we can define the type of context belonging to the data

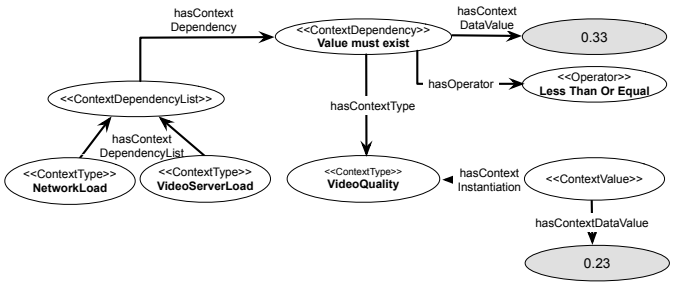



Figure 3: An example of an instantiation of the context model for a multimedia delivery access network scenario.

(i.e., with its relationship with the `ContextType` property) and define the time range during which it was measured. In our approach, time is modeled using the standard Time [26] ontology. Since the context value can be a single value or a complete series of data, we use the composite pattern to model both concepts as instances of ContextValueAtomic and ContextValueComposite.

The exchange of a context type can be restricted by introducing dependencies between context types. We define a context dependency as follows: a context type X depends on another context type Y if data from X depends on values from Y. To model a context dependency, a `ContextType` has a `hasContextDependencyList` property that links it with one or more sets of context dependencies (modeled through a `ContextDependency`). A context type can be modeled with multiple context dependency lists, meaning that it depends on multiple sets of context types.

To illustrate the use of a context dependency, Figure 3 shows an example of how data can be stored in the context model. In this case, we use three context types: the maximum reported video server load, the maximum reported network load and the average video quality of a network AE. We define that the load types should only be requested if there is an indication of an actual problem (i.e., a drop in video quality). This problem indication is modeled through a context dependency that introduces a less than or equal comparison between the video quality score and a constant equal to 0.33. Hence, the load context types will be requested only if the value of the video quality is lower than 0.33. As can be seen, in this example, this is indeed the case, as there is a context value instance with value 0.23. Note that not all relations of the context model are shown here for the sake of presentation clarity.

The instantiation of the context model, as illustrated in Figure 3, is a non-trivial task. However, in many cases (e.g., those where the management algorithms are implemented using expert systems such as a rule-based system), this is instantiation can be automated by an algorithm that automatically generates the context dependencies from an expert system. Such an algorithm is proposed in Section 5.1. If more advanced generation use cases need to be supported, a domain expert can manually define additional instantiations, assisted by a generation algorithm as proposed in Section 5.1.

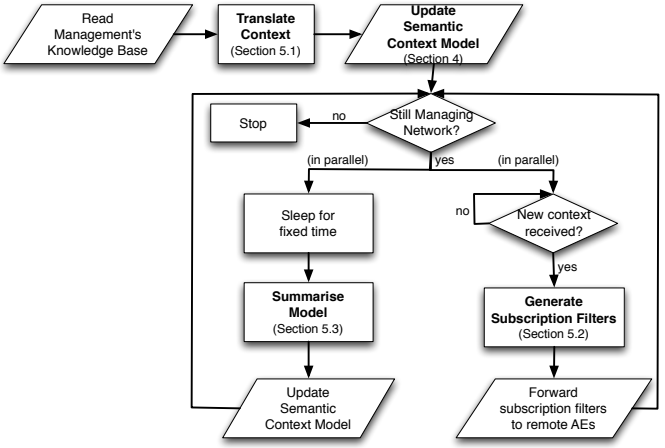



Figure 4: Overview of the context dissemination process.

## 5. Automated generation of semantic subscription filters

An overview of the context dissemination process is shown in Figure 4. As shown, the context dissemination process consists of three steps: first, the context requirements of decision making components are translated and stored in the context model. Second, a generation algorithm generates the subscription filters, either through through OWL/SWRL or RDF/SPARQL reasoning. Finally, the knowledge in the model is summarized to keep the size of the model controlled. Once generated, the subscription filters are updated at the remote AEs. We discuss these three steps in more detail in the remainder of this section.

### *5.1. Context translation*

The goal of this step is to derive the contextual requirements from the decision making components of the local AE and store them in a way that allows semantic interpretation by the subsequent steps. These contextual requirements, modeled as a `ContextDependency` concept in the context model, depend on the specific technology used by the decision making components. For example, the way context is required by a rule-based system can differ from the contextual requirements of an artificial intelligence-based approach. On the other hand, many management algorithms use the same technologies and by only defining a handful of translation algorithms, the contextual requirements of the decision making components can be automatically derived. In this section, we describe a context translation algorithm for a rule-based system. Other translation algorithms can follow the same principle for deriving the requirements. The translation algorithm is illustrated in Algorithm 1. The output of the algorithm is a context requirement set, which is modeled as a set of axioms that are stored in the context model.

For the translation, we assume that all rules are rewritten in Conjunctive Normal Form (i.e., as a conjunction of disjunctions). The contextual requirements of a rule-based system are in essence determined by the conditions of the rules in the rule set. If we take all context types described in the conditions and add them as context types, we have a basic set of contextual

requirements. This is illustrated in Algorithm 1, which queries context types - as part of a context operand in the rule's condition - and adds them to the context model (lines 5 - 9).

**Algorithm 1** The translation of the contextual requirements of a rule-based system to the context model.

```
1:  translateRuleBasedSystem(graph, rulebase) ≜
2:      let contextset = φ
3:      ∀rule ∈ rulebase :
4:          ∀andclause ∈ rule.conditions :
5:              ∀orclause ∈ andclause :
6:                  ∀atomicclause ∈ orclause :
7:                      ∀co ∈ getAllContextOperands(atomicclause) :
8:                          cset = cset ⊔ createContextType(c,co.ContextType)
9:                          cset = cset ⊔ getDependencies(c,atomicclause,graph)
10:     return contextset
11:
12: getDependencies (c, clause, graph) ≜
13:     let dependencylist = φ
14:     let dependencyVertices = getDependencyFromClause(graph, clause)
15:     ∀andclause ∈ dependencyVertices :
16:         let deps = createContextRequirementList(l)
17:         ∀orclause ∈ andclause :
18:             ∀atomicclause ∈ orconditions :
19:                 ∀co ∈ getAllContextOperands(atomicclause) :
20:                     found = true
21:                     deps = ⊔ createContextRequirement(req,co.ContextType)
22:                     deps = ⊔ hasContextRequirement(l,req)
23:         if found ≡ true
24:         then
25:             dependencylist = dependencylist ⊔ deps
26:         endif
27:     return dependencylist
```

We can further refine this set of contextual requirements by introducing dependencies between different context types (described in the `getDependencies` function). To do this, we first examine the rule base and build a condition dependency graph. A condition dependency graph is a directed graph that defines which conditions can only be true if other conditions are true. In a condition dependency graph, the vertices are conditions and an edge from vertex A to vertex B means that condition A is needed to trigger condition B. For example, consider the following 3 rules:

$$A \wedge B \Rightarrow C \wedge D \tag{1}$$

$$X \wedge Y \Rightarrow F \tag{2}$$

$$C \wedge F \Rightarrow G \tag{3}$$

This results in the following condition dependency graph:

$$A \wedge B \rightarrow C \wedge F \tag{4}$$

$$X \wedge Y \rightarrow C \wedge F \tag{5}$$

The construction of this dependency graph is straightforward. Since the execution of rule 3 (with condition $C \wedge F$) can only be fired if rule 1 and rule 2 are fired previously, there is a dependency between the conditions of rule 1 and rule 3 on one hand and rule 2 and rule 3 on the other hand (lines 17 - 22).

Based on this dependency graph, dependencies between context types are constructed by building a `ContextDependencyList` in the context model (lines 23 - 26). First, a list of conditions is computed that define the dependencies of the given atomic clause (i.e., if the atomic clause is part of a vertex in the dependency graph). Second, the context dependency list is constructed by adding every context type that is described in the atomic clause to the list. The output of the getDependencies function is one or more context dependency lists.

The output of the context translation step is thus a set of context dependencies that can be stored in the context model. Following the instantiation, illustrated in Figure 3, a set of rules of the form:

```
IF VideoQuality < 0.3
       THEN Raise_Alarm("BadQuality")
IF BadQuality ∧ NetworkLoad>90%
       THEN Raise_Alarm("NetworkCongestion")
IF BadQuality ∧ ServerLoad>90%
       THEN Raise_Alarm("ServerCongestion")
```

will lead to the following context dependency graph:

```
VideoQuality < 0.33 → BadQuality ∧ NetworkLoad>90%
VideoQuality < 0.33 → BadQuality ∧ ServerLoad>90%
```

and subsequently to the storage of the set of axioms that are illustrated in Figure 3 in the context model.

### *5.2. Subscription filter generation algorithm*

In this section, we discuss how the knowledge of contextual requirements of the decision making components can be exploited to automatically generate the subscription filters. The structure of the context model lends itself to efficiently generate subscription filters since the `ContextType` concept and its associated properties already model candidate subscription filters. The goal of the context reasoning step is to select a specific set of context types that need to be requested. Afterwards, the translation of these types to subscription filters is straightforward. The subscription filter generation algorithm consists of two steps: in a first step, the context types that need to be requested from remote AEs are identified. In a second step, the identified context types are translated to subscription filters.

#### *5.2.1. Step 1: Identification of context types*

The goal of this step is to identify the relevant context types. As illustrated in Figure 4 it is steered by policies, defined by the network provider. These policies define how the generation algorithm should interpret the contextual requirements of the decision making components. Additionally, they allow defining restrictions on the generation of the subscription filters. In this section, we propose two approaches to define these policies and consequently identify the context types. The first approach uses a combination of RDF [1] and SPARQL [2]. The Ontology Web Language (OWL) [3] can be seen as an extension of RDF. As such, it is possible to treat an OWL ontology as a RDF document. SPARQL is the standard query language for

[1] Resource Description Framework (RDF) - http://www.w3.org/RDF/

[2] SPARQL Query Language for RDF - http://www.w3.org/TR/rdf-sparql-query/

[3] OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax - http://www.w3.org/TR/owl2-syntax/

RDF documents. It mainly focuses on a syntactic tuple matching, which improves its scalability. In the second approach, we use SWRL [4], which is a rule language for OWL ontologies, for inferring knowledge about the modelled data. Compared to RDF, SWRL features a higher expressivity (as it is able to do interpret complex semantical relationships such as transitivity and symmetry) at the cost of a lower scalability. These approaches were chosen for several reasons. First, RDF/SPARQL and OWL/SWRL are two of the most widely used approaches for inferring knowledge from ontologies in the semantic web. Second, they are in various levels of standardization by the W3C, and are therefore widely supported by several semantic tools. Third, as SWRL and SPARQL have different levels of expressivity and scalability, it is interesting to compare their performance. For a more in depth discussion of these semantic web technologies, we refer to [27].

*RDF/SPARQL approach.* The RDF/SPARQL approach identifies the relevant context types through a series of intelligent SPARQL queries that exploit the knowledge in the context model. Each query represents a specific policy and thus defines how the subscription filter generation process should occur. For example, to use the context dependencies defined in the previous context translation step, a generic SPARQL query can be specified that takes into account the context dependencies for all context types as follows:

```
SELECT ?type
WHERE {
        ?type rdf:type ContextType
        ?type hasContextDependencyList ?list
        ?list hasContextDependency ?dep
        ?dep hasOperator ?op
        ?dep hasContextDataValue ?th
        ?dep hasContextValue ?cval
        ?cval hasDataContextValue ?val
        NOT_EXISTS(!eval(?th,?op,?val))
}
```

The above SPARQL query identifies all context types of which all their context dependencies evaluate to true as context types to request and thus subscription filters. The `eval` function evaluates if the context dependency evaluates to true by making a straightforward comparison of the the operator `?op` with the most recent context value `?val` and the threshold defined in the context dependency `?th`.

*OWL/SWRL approach.* While the RDF/SPARQL approach already allows interpreting the contextual requirements in some way (e.g., to automatically generate subscription filters based on context dependencies) the overall expressivity of the approach is limited to that of RDF. As such, for more advanced scenarios, where expressivity such as the definition of cardinality restrictions, or the transitivity of properties is required, the OWL/SWRL approach is more suited.

[4] SWRL: A Semantic Web Rule Language Combining OWL and RuleML - http://www.w3.org/Submission/SWRL/

In the OWL/SWRL approach, an additional concept - called `ContextToQuery` - is defined in the context model. The subscription filter generation process is controlled based on the definition of this `ContextToQuery` concept. Only the context types that actually comply with the definition of `ContextToQuery` are retained as subscription filters. The context type identification works thus as follows: each time the generation process is started, an ontology-based reasoner performs an ontological subsumption to check which instances of the `ContextType` concept are also instances of the `ContextToQuery` concept. Hence, the ontological reasoner selects the appropriate context types from all available context types (modeled as instances of the `ContextType` concept) based on the given definition of `ContextToQuery`.

A network or service provider can provide its own definition of `ContextToQuery` to introduce additional policies that control the context dissemination. By adjusting the `ContextToQuery` definition, the context dissemination can thus be governed. For example, an operator might state that context that is linked with a service that has received at least two management alarms recently, should always be requested. To do this, it suffices to give the following additional definition to `ContextToQuery`:

```
ContextToQuery
and hasSource some (TrafficFlow
    and hasContext min 2 (ContextValue
      and hasContextType some Alarm))
```

In this OWL definition, a qualified cardinality restriction is used that constrains the number of context values that are at least needed particular context type (an `Alarm`). Similar expressive policies can be defined through SWRL rules as well. For example, to require the retrieval of all context types that are somehow linked with a traffic flow with a bad video quality, the following SWRL rule can be used.

```
Entity(?e) and hasSource(?c,?e)
  and BadVideoQuality(?c)
  and isConnectedTo(?e,?e2)
  and hasSource(?c2,?e2)
==> ContextToQuery(?c2)
```

The above SWRL rule defines that all context ?c2, which is part of an entity that is connected to an entity that has a bad video quality context value should be requested. In this case, we defined the `isConnectedTo` property as a symmetric and transitive property that links entities with each other. Hence, the check whether one entity is connected to an entity will automatically be propagated through inference of the semantic reasoner. Furthermore, the above SWRL rule relies on the concept `BadVideoQuality`. This concept can easily be defined in OWL to define video quality context, of which the value is lower than a threshold (e.g., 0.33).

```
ContextValue
and hasContextType some VideoQuality
and hasContextDataValue[<=0.33] some float
```

These policies cannot be defined in RDF and SPARQL as restrictions on the cardinality and the definition of transitivity and symmetry of properties are not supported.

Note that, although these policies need to be defined manually by the network provider, the on-line subscription filter generation process is still completely automated. This is because the specification of policies occurs off-line, while the subscription filter generation occurs on-line. Furthermore, the number of policies that are needed to govern the subscription filter process is significantly less than the complete set of possible subscription filters. This is because a single policy can easily trigger the generation of plentiful of subscription filters.

#### *5.2.2. Step 2: Translation to semantic subscription filters*

Once the appropriate context types are selected as part of the generation algorithms, the translation of these context types to actual subscription filters is straightforward. This is illustrated in Algorithm 2, which shows the translation algorithm for the OWL/SWRL approach. As shown, the filter generation process simply queries all `ContextToQuery` instances and translates them into a set of subscription filters, bundled by their source. The translation algorithm of the RDF/SPARQL approach is similar but depends on the result of the queries. Once constructed, the list of subscription filters is sent to every corresponding context producer, so that the required context can be pushed when needed.

**Algorithm 2** The algorithm for automatically generating the subscription filters based on the context model.

**getSubscriptionFilters**$(ont) \triangleq$
  $let\ \ nodes = getNodes(ont)$
  $let\ \ filtermap = \phi$
  $\forall n \in nodes:$
    $let\ \ filters = getSubscriptionFiltersByNode(ont, n)$
    $filtermap = rulemap \sqcup (n \rightarrow filters)$
  **return** $filtermap$

**getSubscriptionFiltersByNode** $(ont, node) \triangleq$
  $let\ \ filters = \phi$
  $let\ \ types = getContextToQuery(ont)$
  $\forall i \in types$
    **if** $hasSource(i, node)$
    **then**
      $filters = filters \cup getFilterRestriction(i)$
  **return** $filters$

A subscription filter is modeled as a string that defines the context type that needs to be retrieved (specified through the `getFilterRestriction`). In its simplest form, this string is just a reference to the context type itself. For example, to refer to all possible packet loss values on all nodes in the network, the filter restriction simply specifies "`PacketLoss`". This refers to a subclass of Context in DEN-ng with the same name. However, by adding additional restrictions, the context type can be specified in more detail. The default behavior of a subscription filter is to refer to the node at which this context type can be found as well. For example, a packet loss context type that is located on a specific node can be referred to as follows:

```
PacketLoss and hasSource some (Entity and
    hasIPAddress "157.193.43.50")
```

The above subscription filter rule is in itself also an ontological definition. Hence, the context producer can match context to this subscription filter by means of an ontological reasoning process. The context dissemination process thus supports semantic subscription filters. These subscription filters can be made even more complex. The expressiveness of OWL allows defining restrictions on the values of the context itself. For example, to state that we are only interested in packet loss context updates that report values of 5% and higher, the following OWL definition can be used:

```
PacketLoss
and hasSource some (Entity
  and hasIPAddress "157.193.43.50")
and hasContextValue some (BoundedValue
  and hasPercentage some double[> 0.05]))
```

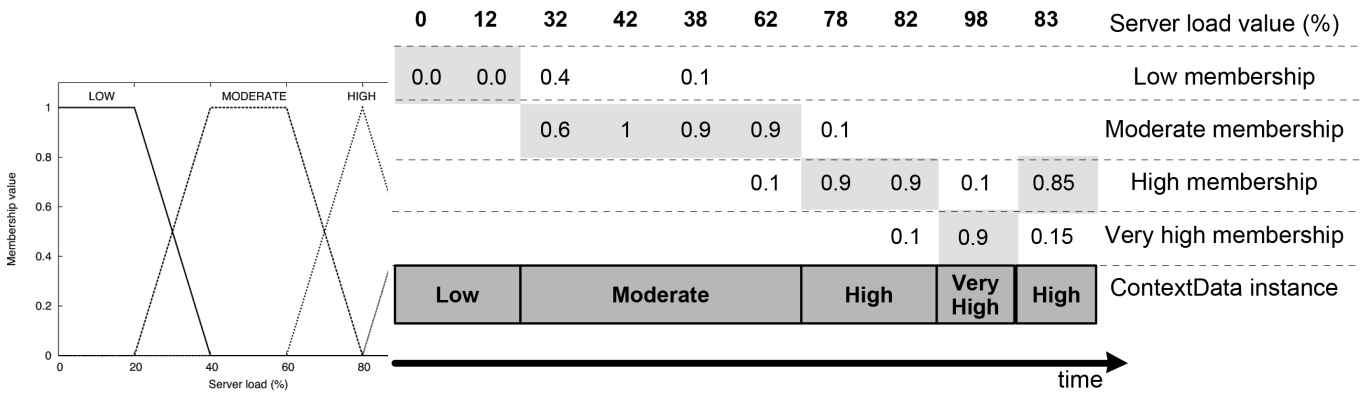
In previous work, we have proposed a context matching algorithm that allows interpreting such OWL definitions (or other semantic constructs such as SWRL or Jena rules) to derive whether to forward context to the consumers. The algorithmic details of the context matching algorithm is thus out of the scope of this article, and we refer to [28] for more information.

### *5.3. Model Summarization*

The context dissemination process described until now stores all context in the context model. This allows all context to be queried from the instances that are part of the `ContextToQuery` class (if the OWL/SWRL generation algorithm is used) or through the SPARQL queries (if the RDF/SPARQL generation algorithm is used). However, it also introduces a significant performance penalty, especially in the OWL/SWRL case. The time needed for performing reasoning in ontologies is known to scale exponentially with the number of instances in the ontology [29]. In order to control the performance of the context reasoning, we try to limit the amount of contextual data in the model without losing the expressiveness required by the automated subscription filter generation algorithms. We refer to this process as model summarization. The model summarization process consists of two distinct algorithms: a fuzzification algorithm and a history-based pruning algorithm. We explain both in the remainder of this section.

*Context fuzzification.* The different AEs typically generate a continuous stream of contextual data, which can be summarized for two reasons. First, the context dissemination process often does not need detailed values of every context type to base its decision on. Second, changes in the context are especially of importance, as they can potentially lead to a change in the generation of subscription filters. If the AEs forward context that provide no major updates in values, there is no use in creating a new `ContextValue` instance.

To introduce such a summarization behavior we propose a fuzzification process that translates each contextual data into a fuzzy variable (i.e. a linguistic label) and groups consecutive contextual data values with the same fuzzy variable to the same `ContextValue` instance in the context model but with a larger timeframe. Figure 5 illustrates the fuzzification process




(a) Used fuzzy variables

(b) Fuzzification process for exemplary server load values

Figure 5: Overview of the fuzzification process for illustrative server load values. The fuzzification results in a significant reduction of the number of instances being stored in the ontological context model.

for an illustrative flow of context of the video server load context type. For the server load, we defined 4 fuzzy variables, each with their corresponding membership functions: low, moderate, high and very high (Figure 5a). Figure 5b shows how each new value of context triggers the calculation of the 4 membership functions (membership values of 0 have been omitted for simplicity), a labeling to a fuzzy variable and a modification to the context model, either by changing an existing `ContextValue` instance or by creating a new one.

*History-based pruning.* The context fuzzification will significantly decrease the number of instances of `ContextValue` that are created in the context model. However, without removing data from the context model, the ontology will keep growing, leading to a significant overhead. To tackle this issue, we propose a simple data pruning algorithm that only stores the context values that are not older than $T$ seconds. The higher this time window $T$ is, the more history can be taken into account in the context dissemination process. We characterize the impact of this time window $T$ value on the reasoning performance in Section 6.

## 6. Performance evaluation

### *6.1. Experimental setup*

A prototype of the context dissemination process was implemented to characterize the performance in terms of time required to generate the subscription filters. The context model was constructed using the OWL API [30]; the Pellet library [31] was used as an OWL2 reasoner and the Jena [5] library was used to support the SPARQL queries. The Pellet reasoner was configured both in normal and incremental reasoning mode. Furthermore, we investigated the different subscription filter generation algorithms and characterized the impact of the model summarization step. This led to seven different configurations of the context dissemination process, which are summarized in Table 1. As, in the OWL/SWRL generation algorithm, the policies can be either specified solely through OWL constructs or solely through SWRL rules, these two configurations were decoupled from each other. Note that incremental reasoning and model summarization cannot be turned on together, as the knowledge base is not monotonic when summarization is applied. Each experiment was repeated 100 times and carried out on a Dual Core 2Ghz AMD machine with 3.5 GB of RAM memory. We present average values; the corresponding standard deviation were always smaller than 5% of the mean.

Table 1: Overview of the investigated context dissemination configurations. Each configuration differs in the generation algorithm that was used, the configuration of the reasoner and the use of summarization.

| Name | Generation Algorithm | Reasoner | Summarization |
|---|---|---|---|
| SPARQL | RDF/SPARQL | SPARQL querier | no |
| SumSPARQL | RDF/SPARQL | SPARQL querier | yes |
| OWL | OWL | Pellet | no |
| IncrOWL | OWL | Incremental Pellet | no |
| SumOWL | OWL | Pellet | yes |
| SWRL | SWRL | Pellet | no |
| SumSWRL | SWRL | Pellet | yes |

To characterize the performance of the context dissemination process, we modeled an access network multimedia delivery scenario as described in Section 2. We focus on the video server AE's context dissemination process. Table 2 provides an overview of the different context types that are available in this network model. As shown, each context type, except the video quality type, has both an aggregated view (i.e., an average over

[5] Apache Jena - http://jena.apache.org

Table 2: Overview of the available context types and their granularity. The aggregated context types denote the maximum value reported by the corresponding local context types that are controlled by the AE.

| Name | Aggregated | Local |
|---|---|---|
| Video quality | N/A | 1 per Flow |
| Server CPU load | 1 per Server AE | 1 per Server |
| Memory consumption | 1 per Server AE | 1 per Server |
| Video quality score | 1 per Network AE | 1 per Flow |
| Network load | 1 per Network AE | 1 per Router |
| Packet loss | 1 per Network AE | 1 per Router |
| Jitter | 1 per Network AE | 1 per Router |
| Delay | 1 per Network AE | 1 per Router |

an AE) and a local view.

We assume that every context type provides an updated value of its contextual status every second. We modeled an access network of 10,000 nodes. Based on this network size, in total, 150,000 possible context types are continuously generated. This thus leads to a huge amount of context that is being requested if the context communication between the collaborating AEs is not carefully controlled. By using the context dissemination process, only a subset of this context will actually be requested.

### *6.2. Detailed evaluation results*

We investigated the impact of the different configuration sets on the total time required to generate subscription filters. We first compare the different generation algorithms with each other without applying any summarization on the model (Section 6.2.1). The impact of performing summarization is characterized next (Section 6.2.2) and also investigate the influence of the configuration of the summarization step (Section 6.2.3). Furthermore, to investigate the scalability of the approach we characterized the reasoning time for an increasing amount of context types involved, triggered by an increase of the number of AEs in the collaborative context dissemination process (Section 6.2.4).

#### *6.2.1. Reasoning time without model summarization*

In this experiment, we characterize the performance of the configuration sets without model summarization. We emulated the scenario described in Section 2, with a network topology containing 16 collaborating AEs. Figure 6 shows the reasoning time as a function over time as more and more context is being stored into the context model. As shown, there are significant differences between the four configuration sets. For the OWL/SWRL-based approaches (i.e., OWL, SWRL and IncrOWL), the reasoning time quickly increases as time elapses and more context is being stored in the model. Especially the OWL-based approach has important scalability issues: after 5 minutes of operation, it takes 2.82 seconds to generate the appropriate subscription filters, while at startup only 400 milliseconds were needed. Furthermore, the reasoning time follows an

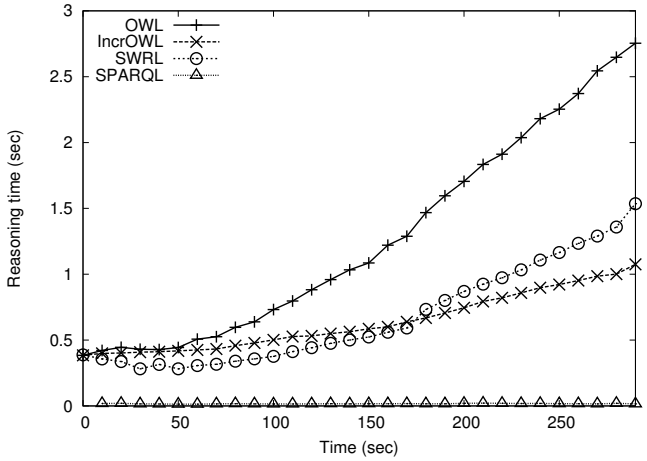



Figure 6: Influence of different configuration sets without model summarization on the reasoning time as a function of time.

exponential increase, which renders this approach infeasible for any real-time operation. This significant increase in the reasoning time of the OWL reasoning configuration set is due to the poor scalability of ontological reasoning as the number of instances in an ontology increases. After 5 minutes more than 15,000 context values have been added to the ontology, as 50 context types generate an updated context value every second. This results in an explosion of the size of the context model and consequently in an explosion of the reasoning time.

Specifying the policies of the context dissemination process in SWRL instead of OWL definitions can aid the performance. As shown, the SWRL configuration set achieves an overall lower reasoning time compared to the OWL configuration set. However, also the SWRL configuration set clearly shows an exponential increase as the context model increases. As the reasoning process of the generation algorithm is in essence monotonic (i.e., only new data is added, old data is not removed), an incremental reasoner can also be used. This is shown in Figure 6, where the IncrOWL configuration set corresponds with the incremental reasoner supported by Pellet. As shown, the IncrOWL clearly has a better performance than the OWL reasoning without incremental reasoning. While the perceived reasoning time still increases as the size of the model increases, the growth is more linear then exponential. This more linear evolution can also be seen when comparing with the SWRL configuration set. While SWRL initially outperforms IncrOWL, the steeper increase of SWRL makes IncrOWL better for larger context models. However, although the performance is already ameliorated considerably by using incremental reasoning, the overhead of OWL-based reasoning is still considerable. After 5 minutes of operation, generating the appropriate subscription filters takes approximately 1.21 seconds.

The fourth and final configuration set without model summarization, the RDF/SPARQL-based approach, has a significantly better performance than the OWL/SWRL-based approaches. This is also shown in Figure 6: as shown, generating the appro-

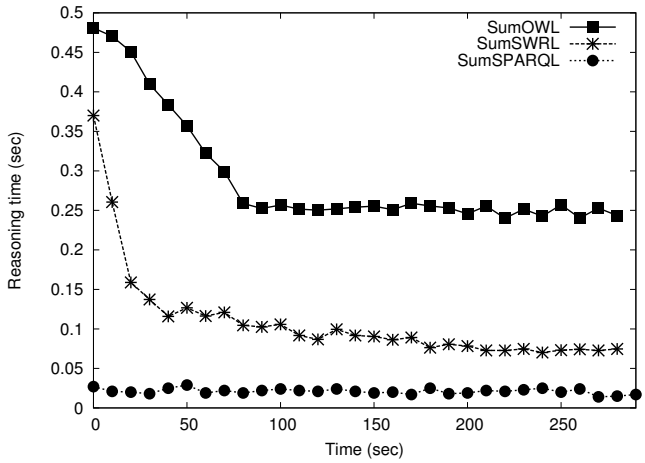



Figure 7: Influence of the Summarized OWL, Summarized SWRL and Summarized SPARQL reasoner configuration sets on the reasoning time as a function of the time.

priate subscription filters only takes approximately 19 msec. Moreover, the performance of the RDF/SPARQL approach is far less impacted by an increase in the context model's size. After 5 minutes of operation, the time required to generate the subscription filters is 19.28 msec, compared to 18.67 msec at the beginning of the experiment. Therefore, at the end of the experiment, the RDF/SPARQL approach was a factor 148 times faster than the OWL-based approach and 64 times faster than the IncrOWL-based approach. Obviously, this performance gain also comes with a cost of expressivity. As discussed in Section 5.2, the RDF/SPARQL approach is a simpler and less expressive generation algorithm than the OWL/SWRL-based configuration sets. As such, it is able to scale better with an increasing context model size but it does not support inferring new knowledge based on, amongst others, the transitivity and symmetry of properties.

*6.2.2. Influence of applying model summarization*

The impact of applying model summarization is shown in Figure 7. We characterize the time required to generate the subscription filters for the 3 configuration sets that use model summarization: SumOWL, SumSWRL and SumSPARQL. For the summarization algorithms, we defined 5 fuzzy variables per context type in the fuzzification algorithm, and chose to keep the last 30 seconds of context in the history-based pruning algorithm.

A number of observations can be made for these results. First, for the OWL/SWRL-based approaches, applying summarization clearly improves the performance of the context dissemination process, regardless if OWL or SWRL-based reasoning is used. Where the OWL reasoning without semantic summarization resulted in reasoning times of 2.82 seconds after 5 minutes, enabling the semantic summarization process keeps the obtained reasoning times around 240 milliseconds. A similar behavior can be observed when enabling semantic summarization for the SWRL-based reasoning. As the semantic summarization process limits the context that is being stored into the context model, we are able to decrease the required reasoning time considerably to reasoning times of only 74 milliseconds.

Second, the SPARQL-based configuration set is less impacted by applying summarization on the model. Applying model summarization results in average reasoning times of approximately 16.94 msec, compared to 18.81 msec without summarization. This is only a marginal difference. Therefore, taking into account that summarization also removes some knowledge from the model, applying summarization is typically not useful for the SPARQL-based approach. Additionally, the SPARQL-based approach was already sufficiently fast and not impacted severely by an increase in the size of the context model, making the summarization of the model less required.

Third, enabling the summarization also stabilizes the obtained reasoning time for the OWL/SWRL-based approaches. Note that the reasoning times were already stabilized in the RDF/SPARQL approach without summarization. The beneficial effect of the summarization is clearly visible in Figure 7 where the obtained reasoning times do not increase for the SumOWL and SumSWRL configuration sets as the time increases. Moreover, the reasoning time slightly decreases in the first 30 seconds to 60 seconds because of the reduced initial start-up overhead of the reasoner. This is certainly an important observation. In order to be feasible in an on-line deployment, the performance of the generation process should remain stable as a function of the time. As shown in Figure 7, summarization is needed for the OWL/SWRL-based approaches. This shows that, for the OWL/ SWRL-based approaches, applying summarization on the model is key in making an on-line deployment feasible.

These results highlight an important aspect: when deploying the context dissemination process, the network and service provider has an important choice to make. Either the expressiveness of the context dissemination process is favored, focusing on describing complex relationships between the collaborating AEs. In this case, the OWL/SWRL-based generation algorithms are best suited but it is important to apply model summarization. This allows the on-line deployment of the context dissemination process and ensures that the subscription filters can be generated sufficiently fast. Another option is to favor the scalability of the context dissemination process. In this case, the RDF/SPARQL-based approach is more suited. Additionally, it has the advantage that it is still faster then the OWL/SWRL-based approaches with summarization and that it does not require any removal of knowledge from the context model.

*6.2.3. Configuration of the summarization algorithm*

The previous results showed the advantage of summarizing the context before it is stored in the context model. In this section, we investigate the impact of the parameters of the summarization algorithms on the reasoning time. More specifically, we characterize the influence of the number of fuzzy variables in the fuzzification algorithm and the time window T of the history based pruning algorithm that defines the amount of history that is stored in the context model. Both parameters in-

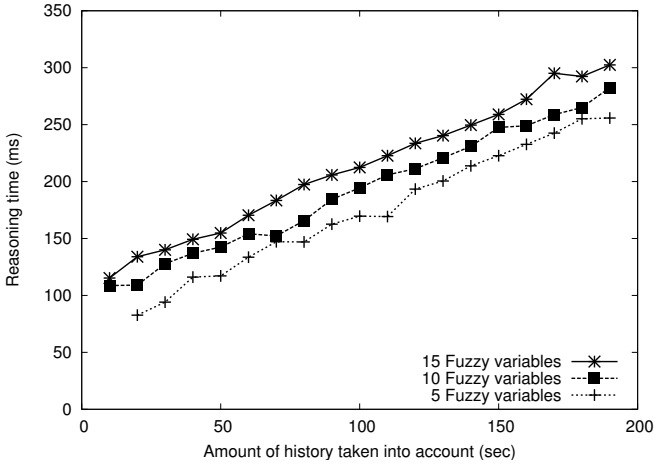



Figure 8: Influence of an increasing time window T in the history based pruning algorithm, defining the history that is stored in the context model, on the reasoning time. Different fuzzification configurations are investigated.

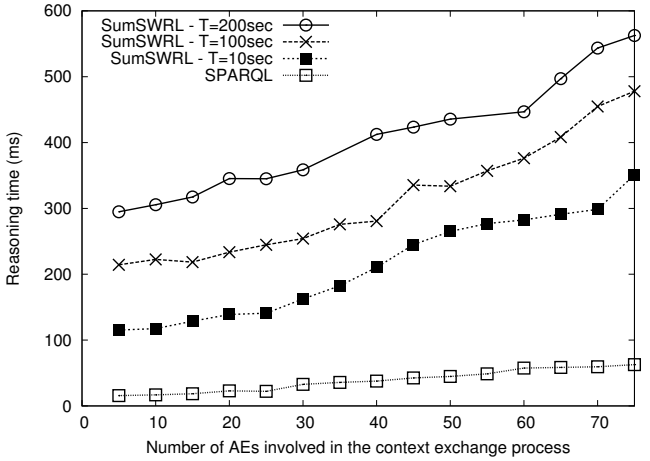



Figure 9: Impact of an increasing number of AEs on the reasoning time. A context dissemination process with 75 collaborating AEs, which represents a network topology of hundred of thousands of nodes, results in a reasoning time of 478 msec.

fluence the ontology's size, and also the reasoning time. We focus on the context dissemination process of the SWRL-based approach with summarization (SumSWRL) as the previous results showed that this is the configuration set that performs best, while still needing the summarization of the model.

Figure 8 shows the reasoning time as a function of this T parameter for various configurations of the fuzzification algorithm. These various configurations are differentiated by a different number of fuzzy variables that are defined. As can be expected, increasing the time window leads to an increase in overall reasoning time: as more history is being stored into the ontology and not removed by the pruning algorithm, the ontology increases, as does the reasoning time. However, the increase in reasoning time is still linear because of the fuzzification algorithm. As the fuzzification algorithm groups context with the same value, increasing the history that is stored for one particular context type does not always lead to an additional context value instance being stored into the context model. This allows us to avoid a typical exponential increase in scalability of the ontology's performance. Instead, increasing the amount of history that is taken into account results in a linear increase from 70-120 msec, if 5 seconds of history is taken into account, up to 250-300 msec, if 3 minutes of history is taken into account.

As shown, the configuration of the fuzzification algorithm has a small but clear influence on the reasoning time. As more fuzzy variables are defined, potentially more context value instances are added to the ontology, especially if the context features a strong oscillation in one or more data values. Consequently, the higher ontology size results in a higher reasoning time. However, the experienced increase in reasoning time is rather small: increasing the number of fuzzy variables from 5 to 15 only leads to a 40 msec increase in reasoning time. As the use of 15 fuzzy variables allows differentiating between 15 levels for a particular context value, this already offers a considerable amount of expressivity.

*6.2.4. Scalability evaluation results*

Although the context dissemination process of the investigated scenario already consisted of a managed network of moderate size (i.e., 16 collaborating AEs), we investigate the scalability of the context dissemination approach by increasing the number of collaborating AEs that are involved in the context dissemination process up to 75 AEs. As these AEs represent AEs that each manage a subset of the network, that many AEs involved in the context dissemination corresponds to a network of hundreds of thousands of nodes. For this experiment, 5 fuzzy variables were defined.

Figure 9 shows the obtained reasoning time for an increasing number of AEs involved in the context dissemination. Two types of reasoner configuration sets were investigated: the SumSWRL configuration (with 3 configurations of the history-based pruning's T parameter) and the SPARQL-based approach. The results show that the reasoning time increases for both configuration types as the dimension of the managed network increases as well. This is obvious: as more AEs are taken into account, the context dissemination process increases in complexity and therefore also in the time required to generate the subscription filters.

The SPARQL-based approach still obtains the best performance. Although a small increase in reasoning time is perceived, it is still able to generate the subscription filters fast. With 75 AEs collaborating in the context dissemination process, subscription filters can be generated in 62.68 msec. Even the SWRL-based approach with summarization is able to scale to large and complex managed network dimensions. Although increasing the number of AEs does lead to an increase in reasoning time, the experienced increase is still tolerable. If the number of AEs involves is increased from 5 to 75, the reasoning time approximately doubles (e.g., if the last 100 seconds of context is kept the reasoning time increases from 210 msec to 478 msec). While these latter values correspond to a con-

siderable increase in reasoning time, it is safe to assume that a collaborative network of 75 AEs will perform complex management tasks, whose execution is in the order of seconds, so that high reasoning times for the context dissemination process are allowed.

## 7. Conclusions

In this article, we proposed a context dissemination process for autonomic collaborative networks. Entities in an autonomic collaborative network generate a continuous stream of knowledge, which we call context. The proposed context dissemination process enables the automated generation of subscription filters, which define the context of remote autonomic elements a local autonomic element subscribes to.

To enforce this, the context dissemination features a number of contributions. First, it contains translation algorithms that translate the contextual requirements of decision making components into a semantic interpretable definition to be stored in an ontology. Second, it contains two distinct subscription filter generation algorithms that generate the set of subscription rules according to policies stated by the network provider. The two generation algorithms either use a combination of OWL and SWRL or RDF and SPARQL and differ in the trade-off between offered expressivity and experienced performance. We discussed how the policies that steer the generation algorithms can support complex operations on the context such as defining semantic subscription filters. Finally, the performance of the context dissemination process can be significantly improved by including a fuzzification and pruning algorithm that summarize the semantic information that is stored in the ontology.

A performance evaluation of a prototype of the context dissemination process was carried out, focusing on the reasoning time required to generate the subscription filters. The results showed that the expressivity of the OWL/SWRL-based generation algorithms comes with an important cost in reasoning time. However, by applying the summarization algorithms, their performance can be considerably improved. As a result, the obtained reasoning times are in the order of tens of milliseconds for the considered managed network sizes. For example, a management network with 16 AEs involved in the context dissemination process is able to generate the subscription filters in approximately 72 msec. As such, the results show that the context dissemination process is able to react in a timely manner to context changes, which makes it deployable in on-line collaborative management environments. As an alternative to the OWL/SWRL-based generation algorithms, the more scalable RDF/SPARQL-based generation algorithm. While this algorithm has less expressivity power in defining the policies, it can generate the subscription filters in the order of a few milliseconds, even for large managed network sizes. For example, the subscription filters of a management network with 16 collaborating AEs can be generated in 18 msec, when using the OWL/SWRL-based generation algorithms.

In summary, the results show that a reasoner configuration set can be found for every network dimension that enables an on-line deployment of the context dissemination process and thus automates the dynamic generation of subscription filters. There is a trade-off in choosing the best configuration sets between expressivity and performance. The network and service provider can choose which approach is best suited for the collaboration scenario at hand, taking into account the presented results.

## Acknowledgements

Steven Latré is funded by grant of the Fund for Scientific Research, Flanders (FWO-V).

[1] B. Jennings, S. van der Meer, S. Balasubramaniam, D. Botvich, M. O Foghlu, W. Donnelly, J. Strassner, Towards autonomic management of communications networks, Communications Magazine, IEEE 45 (2007) 112 –121. 10.1109/MCOM.2007.4342833.

[2] J. Strassner, Policy-Based Network Management: Solutions for the Next Generation (The Morgan Kaufmann Series in Networking), Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[3] P. T. Eugster, P. A. Felber, R. Guerraoui, A.-M. Kermarrec, The many faces of publish/subscribe, ACM Computing Surveys 35 (2003) 114–131. 10.1145/857076.857078.

[4] C. Bouras, V. Poulopoulos, Enhancing meta-portals using dynamic user context personalization techniques, Journal of Network and Computer Applications 35 (2012) 1446 – 1453. 10.1016/j.jnca.2011.10.005.

[5] L. Tamine-Lechani, M. Boughanem, M. Daoud, Evaluation of contextual information retrieval effectiveness: overview of issues and research, Knowledge and Information Systems 24 (2010) 1–34. 10.1007/s10115-009-0231-1.

[6] A. Bikakis, G. Antoniou, P. Hasapis, Strategies for contextual reasoning with conflicts in ambient intelligence, Knowledge and Information Systems 27 (2011) 45–84. 10.1007/s10115-010-0293-0.

[7] M. Bakhouya, J. Gaber, P. Lorenz, An adaptive approach for information dissemination in vehicular ad hoc networks, Journal of Network and Computer Applications 34 (2011) 1971–1978. 10.1016/j.jnca.2011.06.010.

[8] W. Abdou, A. Henriet, C. Bloch, D. Dhoutaut, D. Charlet, F. Spies, Using an evolutionary algorithm to optimize the broadcasting methods in mobile ad hoc networks, Journal of Network and Computer Applications 34 (2011) 1794–1804. 10.1016/j.jnca.2011.01.004.

[9] Distributed Management Task Force, Configuration management database (CMDB) federation specification, 2010. Document Number: DSP0252 - Version 1.0.1.

[10] I. Jurisica, J. Mylopoulos, E. Yu, Ontologies for knowledge management: An information systems perspective, Knowledge and Information Systems 6 (2004) 380–401. 10.1007/s10115-003-0135-4.

[11] Y. Xue, H. H. Ghenniwa, W. Shen, Frame-based ontological view for semantic integration, Journal of Network and Computer Applications 35 (2012) 121 – 131. 10.1016/j.jnca.2011.02.010.

[12] M. Roantree, J. Shi, P. Cappellari, M. F. O'Connor, M. Whelan, N. Moyna, Data transformation and query management in personal health sensor networks, Journal of Network and Computer Applications 35 (2012) 1191 – 1202. 10.1016/j.jnca.2011.05.001.

[13] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, B. Ohlman, A survey of information-centric networking, Communications Magazine, IEEE 50 (2012) 26 –36. 10.1109/MCOM.2012.6231276.

[14] E. Renault, W. Drira, H. Medhioub, D. Zeghlache, Management and semantic description of objects for the future internet, in: Second International Conference on Ubiquitous and Future Networks (ICUFN) (2010), pp. 291 –296. 10.1109/ICUFN.2010.5547187.

[15] M. Petrovic, H. Liu, H.-A. Jacobsen, G-ToPSS: Fast filtering of graph-based metadata, in: 14th international conference on World Wide Web (WWW) (2005), pp. 539–547. 10.1145/1060745.1060824.

[16] M. Petrovic, I. Burcea, H.-A. Jacobsen, S-ToPSS: semantic toronto publish/subscribe system, in: Proceedings of the 29th international conference on Very large data bases (VLDB) (2003), pp. 1101–1104.

[17] H. Li, G. Jiang, Semantic message oriented middleware for publish/subscribe networks, in: Sensors, and Command, Control, Communications, and Intelligence (C3I) Technologies for Homeland Security and Homeland Defense III (2004), volume 5403, pp. 124–133. 10.1117/12.548172.

[18] J. Wang, B. Jin, J. Li, D. Shao, A semantic-aware publish/subscribe system with RDF patterns, in: 28th Annual International Computer Software and Applications Conference (COMPSAC) (2004), pp. 141–146. 10.1109/CMPSAC.2004.1342818.

[19] J. Skovronski, K. Chiu, An ontology-based publish-subscribe framework, in: International Conference on Information Integration and Web-based Applications Services (2006), pp. 49–58.

[20] D. Jones, J. Keeney, D. Lewis, D. O'Sullivan, Knowledge-based networking, in: Proceedings of the second international conference on Distributed event-based systems, DEBS '08, ACM, New York, NY, USA, 2008, pp. 329–332. 10.1145/1385989.1386034.

[21] A. Carzaniga, D. S. Rosenblum, A. L. Wolf, Design and evaluation of a wide-area event notification service, ACM Transactions on Computer Systems 19 (2001) 332 – 383. 10.1145/380749.380767.

[22] J. Keeney, D. Roblek, D. Jones, D. Lewis, D. O'Sullivan, Extending siena to support more expressive and flexible subscriptions, in: Second International Conference on Distributed Event-Based Systems (DEBS) (2008), pp. 35–46. 10.1145/1385989.1385995.

[23] R. Brena, J. Aguirre, C. Chesnevar, E. Ramrez, L. Garrido, Knowledge and information distribution leveraged by intelligent agents, Knowledge and Information Systems 12 (2007) 203–227. 10.1007/s10115-006-0064-0.

[24] S. Latré, S. van der Meer, F. De Turck, J. Strassner, J. Won-Ki Hong, Ontological generation of filter rules for context exchange in autonomic multimedia networks, in: 12th IEEE/IFIP Network Operations and Management Symposium (NOMS) (2010), pp. 575–582. 10.1109/NOMS.2010.5488448.

[25] M. Serrano, J. Serrat, J. Strassner, M. O Foghlu, Management and context integration based on ontologies, behind the interoperability in autonomic communications, Extended journal publication of the SIWN International Conference on Complex Open Distributed Systems 1 (July 2007).

[26] M. O'Connor, A. Das, A method for representing and querying temporal information in OWL, in: A. Fred, J. Filipe, H. Gamboa (Eds.), Biomedical Engineering Systems and Technologies, volume 127 of *Communications in Computer and Information Science*, Springer Berlin Heidelberg, 2011, pp. 97–110. 10.1007/978-3-642-18472-7_8.

[27] T. Wang, B. Parsia, J. Hendler, A survey of the web ontology landscape, in: I. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe, P. Mika, M. Uschold, L. Aroyo (Eds.), The Semantic Web - ISWC 2006, volume 4273 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2006, pp. 682–694.

[28] J. Famaey, S. Latré, J. Strassner, F. De Turck, Semantic context dissemination and service matchmaking in future network management, International Journal of Network Management 22 (2012) 285–310. 10.1002/nem.805.

[29] L. Ma, Y. Yang, Z. Qiu, G. Xie, Y. Pan, S. Liu, Towards a complete OWL ontology benchmark, in: Y. Sure, J. Domingue (Eds.), The Semantic Web: Research and Applications, volume 4011 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2006, pp. 125–139.

[30] M. Horridge, S. Bechhofer, The OWL API: A Java API for Working with OWL 2 Ontologies, in: R. Hoekstra, P. F. Patel-Schneider (Eds.), OWLED, volume 529 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2008, pp. 11–21.

[31] E. Sirin, B. Parsia, B. Grau, A. Kalyanpur, Y. Katz, Pellet: A practical OWL-DL reasoner, Web Semantics: Science, Services and Agents on the World Wide Web 5 (2007) 51–53. 10.1016/j.websem.2007.03.004.