An Ontology-Driven Semantic Bus for Autonomic Communication Elements

Jeroen Famaey[1⋆], Steven Latré[1⋆⋆], John Strassner[2], and Filip De Turck[1]

[1] Ghent University – IBBT, Department of Information Technology,
Gaston Crommenlaan 8/201, B-9050, Gent, Belgium
jeroen.famaey@intec.ugent.be

[2] Division of IT Convergence Engineering
Pohang University of Science and Technology, Pohang, Korea

Abstract. Recently, autonomics have been proposed as a solution to tackle the ever-increasing management complexity of large-scale computing and communications infrastructures. Over time, the control loops used to orchestrate the intelligent behaviour of autonomic management architectures have evolved from fully static to highly-dynamic loops comprised of loosely coupled management components. Communication and other interactions between these components is facilitated by a communications substrate. Additionally, in order to achieve truly autonomic behaviour, the interacting components need to be able to understand each other, justifying the need for semantically enriched communications. In this paper, we present a novel semantic communications bus that orchestrates interactions between the components of an autonomic control loop. It employs ontology-based reasoning in order to establish communication contracts, validate message consistency and support semantic topic subscriptions. Additionally, a prototype was designed, implemented and its performance evaluated.

Keywords: autonomic communications, autonomic elements, autonomic control loops, semantic communications bus

1 Introduction

The booming popularity of the Internet in recent years, has caused a great increase in size, complexity and heterogeneity of communication networks. In combination with more stringent and diverse end-user and service requirements, this leads to the proliferation of management complexity of such large-scale networks. To alleviate the problems associated with managing current and future communication networks, the autonomic communication networks paradigm has

⋆ 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)

been introduced [1],[2]. Its ultimate goal is to automatically adapt the network's services and resources in accordance with changes in the environment and end-user requirements [3]. Human network administrations specify high-level policies, which represent the business goals of the organisation. The autonomic network management system dynamically translates them into low-level network device configurations. Consequently, the increasing management complexity is handled by the system itself.

Since the introduction of autonomics in the computing and subsequently network communications arena, many autonomic management architectures and control loops have been devised. When first introducing the concept of Autonomic Computing, IBM proposed the MAPE control loop [4], which is a static loop consisting of four fixed components, named Monitor, Analyse, Plan and Execute. Recently, research has indicated the need for more dynamic and adaptive control loops. For example, several architectures have been designed that integrate these ideas, including CASCADAS [5] and FOCALE [6]. Dynamic control loops are necessary in order to achieve truly autonomic behaviour, as they enable the system to adapt its core functionality to changes in the environment according to policies. Both in CASCADAS and FOCALE, Autonomic Elements are composed of loosely coupled components that perform diverse functions, such as monitoring, planning, context management and learning. Together, these components form the adaptive control loops. Additionally, a substrate is needed in order to orchestrate behaviour and communication between them. In FOCALE, this idea was advanced in the form of a semantic *enterprise content bus* (ECB) [6], which is an extension of the *enterprise service bus* (ESB) [7] paradigm. In contrast to the standard ESB, the ECB can be used to orchestrate *content*, instead of merely *messages*. Therefore, it is capable of routing on the *meaning* of the message. Additionally, it is an intelligent mediator that transforms data into technology- and platform-neutral forms. Finally, the ECB supports different types of knowledge acquisition and distribution (e.g. push, pull and scheduled) and performs common processing (e.g. semantic annotation, filtering and storage) before content is delivered to components. This enables them to register interest in knowledge in a more precise fashion, thus reducing messaging overhead.

In this paper we extend these ideas and introduce a semantic bus for orchestrating communications between autonomic control loop components. Its semantics and filtering capabilities are centred around the use of ontologies [8] derived from the DEN-ng information model [9], [10]. Messages are structured using OWL [11], which facilitates ontology-based consistency checking and semantic filtering. The goal of this paper is to formulate an answer to several pertinent research questions. First, how do we model communications and other interactions between components of an autonomic control loop in a formal and semantic manner? Second, how can existing technologies and techniques be leveraged to implement such a semantic communications bus? And third, what is the impact on performance from injecting semantics into the bus?

The rest of this paper is structured as follows. Section 2 gives an overview of related work in the field of semantic publish-subscribe systems. Section 3 explains the inner workings of adaptive, bus-driven Autonomic Elements and control loops. Subsequently, Section 4 further elaborates on the specifics of the proposed semantic communications bus. Section 5 discusses the implementation details and evaluation results of the designed prototype in more detail. Finally, conclusions are discussed in Section 6.

2 Related Work

There are a number of papers that research semantically enriched communications substrates. Most research has focussed on semantic publish-subscribe systems. Although, the ECB-inspired communications substrate proposed in this paper offers a wider range of functionalities, there is still a partial overlap. Therefore, this section gives an overview of existing semantic publish-subscribe systems.

Petrovic et al. proposed a subscription (query) language suitable for filtering large numbers of RSS (Really Simple Syndication) documents [12]. The paper describes a hybrid publish-subscribe architecture that consists of publishers, subscribers, and brokers. Publishers send data to a broker, and subscribers register their interest in receiving data with the broker. A graph-based matching algorithm is used for structural and constraint matching. Publications are represented as directed graphs; node and edge labels are both typed literals, enabling them to be related using an ontology. Queries are represented as directed graph patterns. In [13], 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. The ontology can include synonyms, a taxonomy, and transformation rules to equate different terms with each other. Our work is different from both of these approaches, in that the ontology used in [12,13] is limited to RDFS hyponym/hypernym relationships, whereas our approach can use different linguistic and functional relationships. In addition, we use OWL, as opposed to RDFS, which provides greater flexibility and representation of semantics. In [14], a semantic publish/subscribe system for RSS documents is also described. While it also uses an RDF graph, it is different from [12,13] in that it uses OWL Lite and can represent both equivalent relationships as well as hyponynm/hypernym relationships. Our work is different, in that our approach can use different linguistic and functional relationships to provide more powerful semantic matching.

In [15], the DARPA Agent Markup Language (DAML) and the Ontology Inference Layer (OIL), which later merged into OWL, were used to provide semantic publish-subscribe capabilities. Matching was defined using inferencing based on description logic. Topics are defined as ontological concepts and roles. Each publisher advertises instances of one or more topic classes, and each subscriber submits a concept description, which is a class definition, as a subscription. 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. [16]. 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 [17]. Published messages are represented by instances in an ontology. When a new message is published it is added to the ontology and run against all SPARQL queries (subscriptions). If the instance matches a query, the message is delivered to the associated subscriber. 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 seconds. In this paper, we aim to improve scalability significantly.

In [18], two extensions of the SIENA system are proposed. SIENA, along with other similar systems, represents each event as a set of attribute-value pairs; hence, the subscription is defined as a set of conjunctions of simple predicates on the attributes. 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 event in addition to the contents of the event. 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 [19], a semantic method for matching topics (keywords) is defined by using the WordNet lexical database. This is structured as an application built on top of the Java Messaging System broker. Their system enables queries to be expanded and matched to a taxonomy of topics, which in turn enables topics that are related to each other to be efficiently subscribed to. However, this system is limited to hyponyms and hypernyms. Our work is different in that we can use a larger variety of lexical and functional relationships.

Finally, the semantic communications bus proposed in this paper further differs from existing work. In line with the ESB principle, it supports a wider range of delivery mechanisms, such as unicast, deliver at most once and deliver at least once. Additionally, an ESB provides other functionalities, such as transformations of received information into a form more suitable for consumption by subscribers.

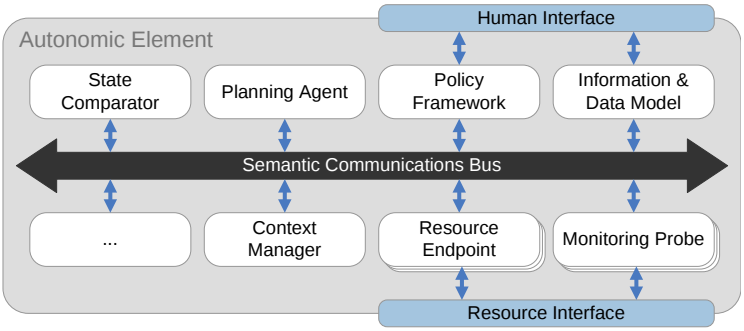


Fig. 1: Bus-driven Autonomic Element architecture

3 Bus-Driven Autonomic Elements

Bus-driven autonomic architectures discard the idea of a static control loop and introduce a more adaptive and dynamic approach to autonomic network management. In such architectures, the Autonomic Element is comprised of a set of loosely-coupled management components that all implement part of the necessary management functionality. An Autonomic Element is a self-organising management component that governs a subset of the network's resources [20]. It is capable of operating independently or collaborating with other Autonomic Elements in order to autonomously achieve higher-level goals. Each Autonomic Element exposes a specific set of management services and functionalities, which can be leveraged in order to achieve autonomic management behaviour.

Figure 1 shows an example Autonomic Element with a minimal set of management components. The functionality implemented by the example elements is roughly identical to that of the MAPE control loop [4]. However, due to the loose coupling, the ordering of components in the control loop is less strict. The advantages of this approach have been clearly demonstrated in the new FOCALE architecture [6]. It provides a diverse set of control loops that are used in different situations. For example, some loops use the available context information to perform large-scale adjustments, while others perform more fine-grained tuning within a specific context. All of these loops use a different subset of the available management components.

Additionally, the loose coupling allows new management components to be dynamically added to or replace existing components of the Autonomic Element. This increases management modularity and makes it possible for the Autonomic Element to dynamically adapt its exposed functionality based on the environment. For example, the MAPE-like functionality as shown in Fig. 1 could be extended with new components that are capable of orchestrating federations across management domains or performing contract negotiation.

It is obvious that a communication substrate is needed to glue the management components together [6],[21]. It is responsible for orchestrating both communication and collaboration between the management components within the Autonomic Element. Additionally, in order to achieve true autonomic be-

haviour, it needs to be capable of handling semantically-enriched queries. This will allow the communicating components to interpret and better understand the messages they receive and send. Furthermore, the introduction of semantics facilitates the verification of message consistency, along with enabling intelligent filtering and aggregation of information. Although the need for such a Semantic Communications Bus has been repeatedly expressed, no in-depth design has been proposed to our knowledge. Therefore, we have designed and implemented a Semantic Communications Bus for intra- and inter-Autonomic-Element interactions. Note that this paper focusses on interactions between components *within* Autonomic Elements, the latter will be further studied in future work. The conceptual ideas are discussed in the following section. Subsequently, the implementation is described in Section 5.

4 Semantic Communications Bus

In order for the management components of the Autonomic Element to be able to interpret received messages, they must be enriched with semantics. This allows the management components to analyse context information and deduce appropriate action by interacting with other components. In addition, it enables the Autonomic Element to implement and maintain a dynamic knowledge base. For example, as new information is discovered, it can be semantically validated and can then be added to the knowledge base; optionally, it can be distributed to other management components within the Autonomic Element and even other Autonomic Elements that have expressed interest in updates of such knowledge. The exchange of information and other interactions are facilitated by the semantic communications bus (SCB). Messages are semantically interpreted using an *ontology*, which models management component interactions within an Autonomic Element. Gruber defined an ontology as *"a specification of a conceptualization in the context of knowledge description"* [8]. It thus describes the concepts of a certain domain, together with their attributes and relationships, in a formal manner. The rest of this section first introduces an ontology-based model for semantically representing communications and other interactions between management components. Subsequently, the core functionalities of the SCB are discussed in more detail.

4.1 Semantic Interaction Model

Figure 2 depicts the core concepts of the designed ontology. The general structure and many concepts of the ontology are modelled after the DEN-ng information model and its classes [9], [10], [22]. At the ontology's heart is the `Message` concept, which represents any type of message sent through the bus. It is the equivalent of the DEN-ng Message class and is a subclass of Event and thus Entity. Every message is sent by a single `ManagementComponent`, which is depicted by the `hasSource` property. The bus supports several communication mechanisms, such as broadcast, unicast and multicast. Therefore, the `hasTarget` property

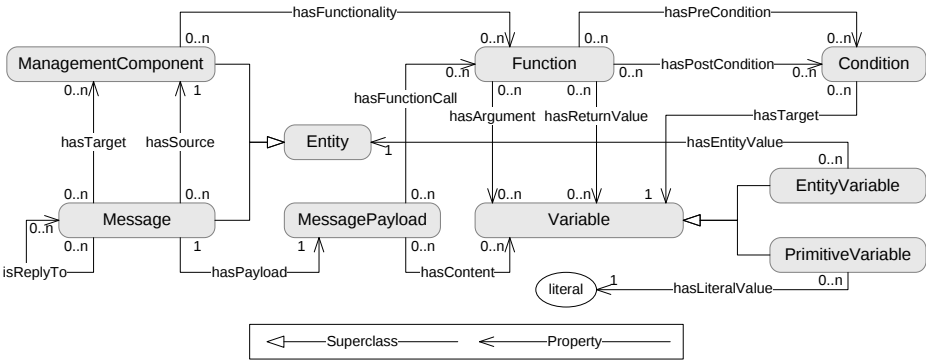


Fig. 2: The main concepts of the core ontology; sub-concepts of Entity and Condition have been omitted for clarity

supports 0, 1 or an arbitrary number of targets. Additionally, the message has a payload which represents its content and consists of variables and/or function calls. The `Variable` concept represents information of different types, while the `Function` concept identifies function calls. The different types of variables are modelled as sub-concepts of `Variable`. The depicted ontology contains two variable types. However, new types of variables can be defined by creating sub-concepts of `Variable` or of one of its existing sub-concepts. The `EntityVariable` contains as its value an `Entity`, which comes directly from DEN-ng. It represents any type of physical or logical resource; including devices, software components, protocols, persons and even events. The `PrimitiveVariable` concept has a literal value, such as an integer, boolean or string. Function calls can be performed by sending messages with `Function` OWL individuals as payload. A function consists of an optional set of arguments, return values, preconditions and post-conditions. Every condition applies to a single variable and constrains the state of that variable before or after execution of the function call. The set of conditions associated with a OWL `Function` individual thus represents a contract that models the obligations and benefits of the function caller and callee [23]. By including them in the ontology, semantics are attached to them and the effects of a function call on the environment can be unequivocally determined by management components. This approach facilitates semantic service and functionality discovery, which will be further studied in future work.

In a bus-driven autonomic element, such as the one described in Section 3, the semantic interaction model can be used to achieve communications between the different components attached to the bus. In this case, functions are used to invoke operations on specific components, whereas messages act more as a notification system to one or multiple subscribers, where the publisher of the message does not need to know who the recipient(s) may be. In an autonomic communications scenario, typical examples of messages are those sent by a monitoring component that updates information about the resources it is monitoring or an autonomic control loop that informs everyone on the bus about the decisions it has made. Throughout the rest of this section, we provide exemplary messages

that can be used by a monitoring framework that monitors one or more servers (e.g. for performing autonomic node activation [24]).

Extensibility is supported in several ways. First, the core ontology of an Autonomic Element can be extended or changed by a human administrator that is responsible for the management domain. Such a change affects all management components of the Autonomic Element. Second, specific management components may introduce their own extensions by defining new concepts or extending existing ones. These extensions are defined per message type in an accompanying ontology and are linked to the core ontology but not integrated into it. Other management components that wish to communicate with it can then take these extensions into account, while others can ignore them.

4.2 Message Type Definitions

The core ontology described above allows management components to easily define new message types. A new message type is simply a sub-concept of `Message`. For example, a message type that contains information about one or more servers in the network can be defined using description logic syntax as follows:

$$Message \sqcap hasPayload = 1\ (MessagePayload \sqcap \exists\ hasContent\ (EntityVariable\ \sqcap\ hasEntityValue = 1\ Server))$$

What this states is that we define a new concept that must abide to two restrictions. First, it is of type `Message`. Second, it contains exactly one payload, which in turn has some content of type `EntityVariable` that contains information about exactly one `Server` entity.

Management components that connect to the SCB first register the message types they are capable of sending. These message types consist of a definition as described above and an - optional - accompanying ontology, which allows introducing new concepts in the definition (e.g. a specific type of `Server` or another type of `Message` payload next to `Function` or `Variable`) and are also added to the core ontology. The bus stores them in a repository that links each management component to a set of message types. This has several advantages. First, it allows the semantic communications bus to check the semantic validity and consistency of the message types defined by the component. Second, this set of message types represents a contract by which the management component must abide. The component implicitly agrees that it will only send messages of the types it registered with the bus. The rest of this section further explains the consistency checking of registered message types. The validation of the messages themselves is discussed in Section 4.3.

In order for a message type to be consistent and valid, it must satisfy two requirements. First, it must be a sub-concept of the `Message` concept in the core ontology (cf. Fig. 2). Second, it may not cause any inconsistencies when being added to this core ontology and the imported accompanying ontology. Both of these requirements can be checked using an ontology reasoner. The semantic bus performs several actions when validating the consistency of new message

types. First, it creates a copy of the core ontology specifically for the newly registered management component. This prevents inconsistencies from arising between message types of different management components. Second, the message types defined by the management component are added to the ontology. Third, the reasoner is asked to classify the ontology. The SCB supports message type versioning to avoid inconsistencies caused by earlier versions: the versioning system ensures that only the most recent message type and accompanying ontology are added to the core ontology. If the classification process detects any inconsistencies the registration failed, and the component is not allowed to use the bus. Finally, the bus checks if all defined message types are classified as sub-concepts of `Message`. If this is not the case, the registration process fails as well. Otherwise, the registration is successful and the management component can start using the bus.

Clearly, the algorithm described above is time-consuming, as the core ontology is copied, all defined message types are added to it, and the entire ontology is classified. However, this process only occurs when a new management component is plugged into the Autonomic Element. This only occurs rarely, so the execution time is of lesser importance.

4.3 Message Instantiation & Validation

Management components can send messages onto the bus by creating OWL individuals that are members of the message types defined by the source component when it registered with the SCB. The SCB validates this by classifying any sent messages and checking their membership with the defined message types. If the created OWL individual is a member of at least one message type defined by the source component, it is considered valid and the SCB forwards it. Hence, this validation process consists of an ontology reasoner performing realization reasoning, which checks if the OWL individual belongs to a specific class. As this can be a time-consuming step, the SCB allows to turn the message validation off.

As an example, we define an OWL individual of the message type shown in Section 4.2. The message and its payload can be created as follows:

$$m : Message \qquad p : MessagePayload \qquad v : EntityVariable \qquad s : Server$$
$$(s, \text{“10.10.0.1”}) : hasIPAddress \qquad (v, s) : hasEntityValue$$
$$(p, v) : hasContent \qquad (m, p) : hasPayload$$

4.4 Subscription

As is the case with classic publish-subscribe mechanisms, our semantic communications bus supports the use of subscriptions. By registering a subscription, a management component indicates interest in specific messages. All messages that belong to a message type to which a component subscribed are delivered to it. However, in contrast to classical subscription mechanisms, the SCB does not require publishers to explicitly declare the topic a message belongs to. Rather,

the subscriber actually defines the structure of the message types in which it is interested. A subscription is thus specified the same way as a message type definition (cf. Section 4.2). For example, a management component can define a message type subscription stating it is interested in all messages that relate to a specific server:

$$\begin{aligned} Message \sqcap hasPayload = 1\ (&MessagePayload \\ &\sqcap \exists\ hasContent\ (EntityVariable \\ &\sqcap hasEntityValue = 1\ (Server \sqcap hasIPAddress = \text{"10.10.0.1"}))) \end{aligned}$$

Note that, although this subscription is considerably different than the message type definition presented in Section 4.2, it will still match with the message of that specific type as the definition is a subset of that message type. Obviously, this approach adds a great deal of flexibility compared to the classical topic hierarchies. Now, subscriptions can relate to any part of the message, such as its content, source or targets. For example, a management component can define a message type subscription stating it is interested in all messages sent towards himself as follows:

$$Message \sqcap\ \exists\ hasTarget\,(ManagementComponent \sqcap\ hasId \ni myId)$$

This approach also makes it easy for a component to indicate interest in all messages, merely by creating a message type that is equivalent to the `Message` concept.

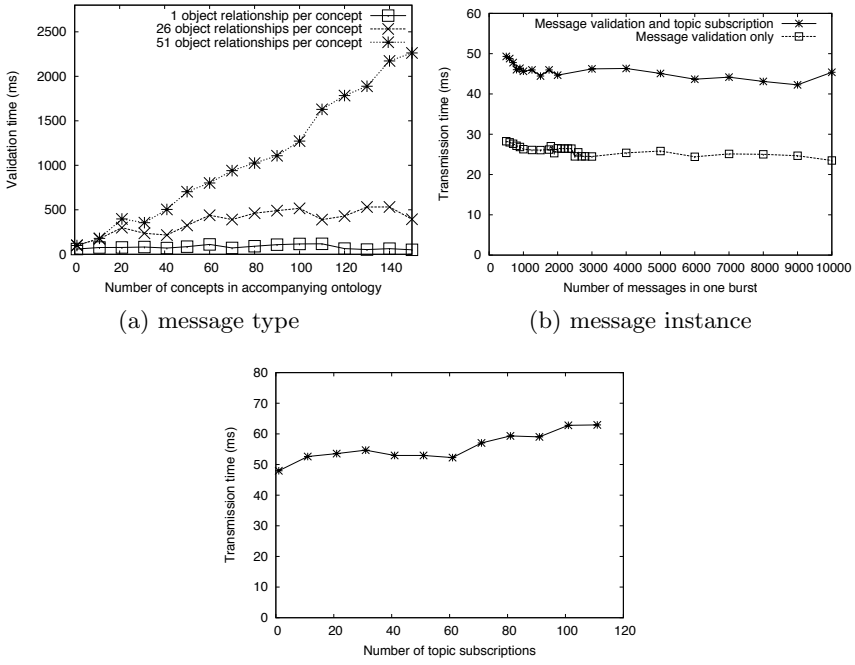
Obviously, a management component can indicate interest in several message types. Every message that satisfies at least one of them is admitted to the component by the bus. Checking whether or not a message satisfies a specific subscription definition is as simple as determining if it is a member of the message type defined by that subscription. This can be done in the same way as described in Section 4.3.

5 Evaluation results

A prototype was implemented to validate the performance of the SCB. We focus on the message sending functions of the SCB. The implementation is based on OSGi, which is a Java-based, modular platform that supports at-runtime starting and stopping of software bundles. Therefore, it is highly suited for implementing the loosely-coupled interactions needed by our bus-driven autonomic architecture. The SCB itself is based on the OSGi Event Admin bundle, which is an event-driven publish-subscribe mechanism. Our implementation uses the OWLAPI[3] library for representation and the Pellet[4] library for reasoning.

The implementation, as described above, was used to measure performance of the main functions of the SCB. Performance was measured in terms of execution time (i.e. checking consistency and validity) or transmission time (i.e. sending

[3] `http://owlapi.sourceforge.net`
[4] `http://clarkparsia.com/pellet`



(a) message type

(b) message instance

(c) topic subscription

Fig. 3: Performance, in terms of validation and transmission time of the prototype detailed in Section 5 for an increasing message type complexity (a), message burst size (b) and number of topic subscriptions (c).

messages). All tests were performed on a machine with an AMD Athlon 64 X2 Dual Core 5200+ processor with 1 GiB memory. Each test was repeated 20 times, we present average values; all standard deviation values were less than 3% when compared to the corresponding average values. Figure 3a shows the time required to validate a single message type (cf. Section 4.2) as a function of the number of concepts in the core ontology and the number of object relationships linking to each concept. For this purpose, randomly generated concepts and relationships were added to the accompanying ontology (cf. Fig. 2). Figure 3b plots the time required to send a single message as a function of the burst size, which is defined as the total number of messages sent over the SCB simultaneously. Finally, Figure 3c illustrates the impact of the number of topic subscriptions on the time needed to send a message (cf. Section 4.4).

The results depicted in Fig. 3a clearly show that the time needed to validate a message type depends on the number of property relationships in the accompanying ontology and less on the actual number of concepts. For a small number of object relationships attached to each concept, the validation time does not increase much when the number of defined concepts increases. However, even for an ontology with 150 concepts and 51 property relationships per concept the val-

idation time is just over 2 seconds. As we do not expect the number of property relationships to grow very high and message type validation is only performed when a new component is plugged into the Autonomic Element, such execution times make it feasible to use an ontology driven message type definition.

Figure 3b shows that the time needed to transmit a message depends little on the burst size. If no topic subscriptions are defined, and consumers thus receive all messages, the transmission time converges to about 25 ms per message, which means the SCB can send about 40 messages per second. Subscribing to a topic also introduces some overhead as an additional reasoning step is needed: in this case the transmission time converges to about 45ms, which corresponds with approximately 22 messages per second. As sending a message over the default OSGi Event Admin takes only 0.04 ms, most of this delay is caused by the reasoner. The added benefit of formal validation and topic subscription, offered by the SCB, is therefore not viable for time critical applications that require the sending of hundreds of messages per second. However, we believe that the message frequency will be considerably lower in an autonomic element system, where messages are typically sent because of an update of monitor information. Furthermore, we believe this delay can be further decreased by introducing additional optimizations, such as incremental reasoning and grouping of messages.

As can be seen in Figure 3c, the number of topic subscriptions per consumer of the SCB does have an impact on the total transmission time but the increase is not high: increasing the number of topic subscriptions up to 110 only leads to an increase of transmission time of 18ms. Therefore, the SCB makes it possible to transmit messages even when consumers have subscribed to a lot of topics. In practice, the number of topic subscriptions is likely to be much lower: such a high number represents topic subscriptions that are narrowly defined. It is realistic to assume that in such a case, a consumer would opt for only a few topics, of which the subscriptions are defined more broadly.

6 Conclusion

In this paper, we presented a novel semantic communications bus (SCB) for orchtestrating interactions between loosely-coupled autonomic management components. Using an ontology, the SCB supports the exchange of semantically-enriched messages, which in turn accommodates the interpretation of exchanged information and requested functionality. Additionally, it provides mechanisms for checking the validity and consistency of these messages. Finally, autonomic components using the SCB can register interest in specific types of messages in a semantic way. Obviously, this approach achieves greater flexibility than classical publish-subscribe topic hierarchies, as it allows subscriptions to relate to any part of the message structure.

In order to validate the feasibility, applicability and performance of the designed SCB, a prototype was implemented using the OSGi platform. The implementation was used for a preliminary performance evaluation. The results show that, while performing ontology based reasoning does have an impact on per-

formance, the additional overhead still allows sending messages quickly through the bus with transmission times between 25 and 45 ms for a first prototype. As the message validation and topic subscription algorithms focus on maintaining the accompanying ontology as small as possible, ontology reasoning times in the order of milliseconds can be achieved, which is reasonably fast for an ontological approach. Furthermore, we believe that we can further decrease these transmission times by applying more specific reasoning algorithms (e.g. incremental classification) and by grouping messages, message types and topic subscriptions.

In future work, we are planning to further optimize performance of our prototype. Additionally, the interaction model will be extended in order to support semantic interactions and contract-based collaborations not only within but also across Autonomic Element. Finally, the ontology-based function definitions will be further extended in order to achieve semantic service and functionality discovery.

Acknowledgment. We would sincerely like to thank Bert Vankeirsbilck, Pieter Simoens and Femke Ongenae for their valuable feedback and fruitful discussions.

References

1. Dobson, S., Denazis, S., Fernandez, A., Gaïti, D., Gelenbe, E., Massacci, F., Nixon, P., Saffre, F., Schmidt, N., Zambonelli, F.: A survey of autonomic communications. ACM Transactions on Autonomous and Adaptive Systems **1**(2) (2006) 223–259
2. Jennings, B., van der Meer, S., Balasubramaniam, S., Botvich, D., Foghlu, M.O., Donnelly, W., Strassner, J.: Towards autonomic management of communications networks. IEEE Communications Magazine **45**(10) (2007) 112–121
3. Agoulmine, N., Balasubramaniam, S., Botvitch, D., Strassner, J., Lehtihet, E., Donnelly, W.: Challenges for autonomic network management. In: 1st IEEE International Workshop on Modeling Autonomic Communications Environments (MACE). (2006)
4. Kephart, J., Chess, D.: The vision of autonomic computing. Computer **36**(1) (2003) 41–50
5. Baresi, L., Ferdinando, A.D., Manzalini, A., Zambonelli, F.: The cascadas framework for autonomic communications. In: Autonomic Communication. (2009)
6. Strassner, J., Kim, S.s., Hong, J.W.K.: The design of an autonomic communication element to manage future internet services. In: 12th Asia-Pacific Network Operations and Management Symposium. (2009) 122–132
7. Christudas, B.: Service-Oriented Java Business Integration: Enterprise Service Bus Integration Solutions for Java Developers. Packt Publishing (2008)
8. Gruber, T.: A translation approach to portable ontology specification. Knowledge Acquisition **5**(2) (1993) 199–220
9. Strassner, J.: DEN-ng: achieving business-driven network management. (2002) 753–766
10. Strassner, J., Souza, J.N., 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 Systems Management **17**(1) (2009) 5–32

11. McGuinness, D., van Harmelen, F.: OWL web ontology language overview. [online] http://www.w3.org/TR/owl-features/ (2004)
12. Petrovic, M., Liu, H., Jacobsen, H.A.: G-ToPSS: fast filtering of graph-based metadata. In: Proceedings of the 14th international conference on World Wide Web (WWW). (2005) 539–547
13. Petrovic, M., Burcea, I., Jacobsen, H.A.: S-ToPSS: semantic toronto publish/subscribe system. In: Proceedings of the 29th international conference on Very large data bases (VLDB). (2003) 1101–1104
14. Ma, J., Xu, G., Wang, J., Huang, T.: A semantic publish/subscribe system for selective dissemination of the rss documents. In: Fifth International Conference Grid and Cooperative Computing (GCC). (2006) 432–439
15. Li, H., Jiang, G.: 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. Volume 5403. (2004) 124–133
16. Wang, J., Jin, B., Li, J.: An ontology-based publish/subscribe system. In: Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware (Middleware). (2004) 232–253
17. Skovronski, J., Chiu, K.: An ontology-based publish-subscribe framework. In: International Conference on Information Integration and Web-based Applications Services. (2006)
18. Keeney, J., Roblek, D., Jones, D., Lewis, D., O'Sullivan, D.: Extending siena to support more expressive and flexible subscriptions. In: Proceedings of the second international conference on Distributed event-based systems (DEBS). (2008) 35–46
19. Lien, Y.C., Wu, W.J.: A lexical database filter for efficient semantic publish/subscribe message oriented middleware. In: Second International Conference on Computer Engineering and Applications (ICCEA). (2010) 154–157
20. Hoefig, E., Wuest, B., Katalin, B., Mannella, A., Mamei, M., Nitto, E.D.: On concepts for autonomic communication elements. In: 1st IEEE International Workshop on Modeling Autonomic Communications Environments (MACE). (2006)
21. Strassner, J., Won-Ki Hong, J., van der Meer, S.: The design of an autonomic element for managing emerging networks and services. In: IEEE International Conference on Ultra Modern Telecommunications (ICUMT). (2009)
22. Strassner, J.: Introduction to DEN-ng. [online] http://www.autonomic-communications.org/teaching/ais/slides/0809/Introduction_to_DEN-ng_for_PII.pdf (2009)
23. van der Meer, S.: Architectural artefacts for autonomic distributed systems – contract language. In: Proceedings of the Sixth IEEE Conference and Workshops on Engineering of Autonomic and Autonomous Systems (EASe). (2009) 99–108
24. Famaey, J., Wauters, T., Turck, F., Dhoedt, B., Demeester, P.: Dynamic overlay node activation algorithms for large-scale service deployments. In: DSOM '08: Proceedings of the 19th IFIP/IEEE international workshop on Distributed Systems: Operations and Management, Berlin, Heidelberg, Springer-Verlag (2008) 14–27