QoS-based Testing and Selection of Semantic Services

Jan Schaefer Reinhold Kroeger
Design Computer Science Media Department
RheinMain University of Applied Sciences
Wiesbaden, Germany
{jan.schaefer|reinhold.kroeger}@hs-rm.de

Simone Meixler Uwe Brinkschulte
Department of Computer Science
Johann Wolfgang Goethe University
Frankfurt am Main, Germany
{meixler|brinks}@es.cs.uni-frankfurt.de

Abstract:
The importance of context-awareness is constantly increasing. Users want systems to react dynamically to their current context (e.g. their location). For emerging home service platforms, this represents a key element, as it allows systems to increase the users' comfort tremendously by acting on sensed and deducted situations. As such systems feature a lot of dynamic interaction between services, it has to be ensured that service selections and bindings simply *work*. This paper proposes an approach for QoS-based testing and selection of semantic services, which employs a service tester that uses genetic algorithms to check and monitor QoS properties. These properties are used to complete the semantic descriptions of services running on the platform, which are managed by a semantic service registry.

1 Introduction

The increasing amount of IT hardware and software (e.g. "apps") in personal living spaces allows users to experience more and more services both online via the Internet and offline in their homes: personal computers, smartphones, tablets, home theater devices and smart homes are becoming more and more intertwined with each other and online services (Internet and/or cloud services) forming an *Ambient Assisted Living* (AAL) home service platform. The term *Ambient Assisted Living* (AAL) was coined by the European Commission's Information Society in 2004, as AAL research activities were prepared in a special support action project that was part of the 6^{th} European Framework Programme [Gmb04]. AAL-related technologies introduce a complexity that users can or do not want to administrate themselves anymore. The increase of networked electronics and software can be compared to the increase of assistive and media technologies in cars during the last two decades. Unlike in cars, however, personal electronic devices and applications are much more open to (deliberate) modifications and data exchange, which makes securing their handling even more difficult. Finding and selecting the right service when using this home

service platform can become a time-consuming and even dangerous process (from an IT security perspective), if there are multiple service providers and a malicious or malfunctioning service is selected. In addition, usually multiple applications are active in parallel requiring binding decisions almost constantly.

Static binding between client consumer and provider, which usually takes place once at the application's compile/link time, is not a feasible approach here. Even dynamic binding with its implementation-dependent, tight coupling between consumer and provider is not sufficient. As smart homes consist of heterogeneous, distributed systems with varying computation capabilities, they represent an ideal environment for service-oriented computing with its loose coupling thus supporting the required runtime dynamics, which explains the advent and propagation of home service platforms in general. However, the service interface descriptions and, even more importantly, any existing service properties offered by service providers (typically name-value pairs) are still statically defined, as they do not reflect the runtime state of the provided implementations. This results in syntactic service look-ups, in which the binding only depends on the service interface and its statically defined properties. To support the definition of extended and dynamic service descriptions that are integrated with a common context model – an ontology – for the platform, the ability to define and process semantics is needed. The approach presented here allows clients to define semantic queries for services, which can be processed by the service platform. As these queries can use the semantics of both services and platform, it is possible to build adaptive, context-aware applications, in which clients can react to changing service and context properties, as service properties reflect the runtime state of the platform and its components.

With respect to the functional properties a service defines (e.g. regarding its input and output parameters), clients (i.e., users or applications) might require non-functional service capabilities such as *Quality of Service* (QoS) properties. In this case, the clients' needs can be formulated explicitly or as part of their preferences. Service properties, on the other hand, are typically either defined statically or dynamically gathered at runtime (e.g. by application instrumentation). The first approach is both hard to achieve and insufficient as it doesn't reflect the user experience: a service might be fast if accessed in one geographic region but slow in another. The second approach, however, might also be inapplicable as ongoing runtime instrumentation might lower the service's performance too much impairing end user experience. This paper proposes a third approach: The integration of semantic service descriptions with service testing and probing capabilities. Here, service implementations register a test interface, which is used to determine runtime QoS properties, before the service is used for the first time. The resulting QoS properties become part of the service's description and can be used either for service look-up or selection of the fittest service if multiple services fulfill the client's requirements. The subsequent service probing adapts automatically to the currently offered services as well as to the usage of these services. We also consider to keep the testing effort within reasonable bounds.

This paper is structured as follows: Section 2 presents required background information and is followed by Section 3, which presents related approaches. The subsequent Section 4 introduces the approach for QoS-based testing and selection of semantic services. Section 5 discusses the current state of and the next steps for the work presented here.

2 Background

2.1 The Web Ontology Language

The W3C specification for the *Web Ontology Language* (OWL) [Gro09] defines a family of languages to support the *Semantic Web* vision. While RDF already allows addressing resources and their properties via URIs thus creating a formal, triple-based representation of information, OWL adds machine-processable semantics enabling automated interpretation of this information by computers. Furthermore, OWL supports an inference mechanism – so-called *reasoning* – that allows the automated derivation of new information from existing descriptions of things and their relationships. To support reasoning and limit its complexity, which leads to increasing processing times, the OWL specification defines three OWL sublanguages with different levels of expressiveness: *OWL Lite*, *OWL DL* (includes Lite) and *OWL Full* (includes DL), where OWL DL was designed to provide maximum expressiveness while retaining computational completeness. Apart from these sublanguages, the OWL specification also supports several exchange syntaxes for specification and exchange purposes with varying degrees of human readability.

An OWL ontology contains statements consisting of *Resource-Property-Value* triples (e.g. *Father hasName 'John'*). A resource has a *Class* definition and features a number of *Individuals*, which represent the actual objects in a domain. Properties may feature *Domains* and *Ranges*: *Data Properties* associate resources with constants, whereas *Object Properties* associate resources with each other. In addition, properties may possess more detailed logical capabilities such as being functional, transitive, symmetric, reflexive, inverse and/or disjoint.

2.2 Genetic Algorithms

Genetic Algorithms are used for solving optimization problems and are based on the biological evolution theory. Terms that are often used for genetic algorithms are listed in table 1. Furthermore, the table shows the biological and IT specific translation of these terms.

Term	Biological Translation	IT Translation
Population	Set of individuals	Set of solution candidates
Parents	Mating subset of population	Selected individuals for generating new solution candidates
Fitness	Conformity of an individual	Quality of candidate solution
Chromosom	Properties of a individual	String
Gen	Part of a Chromosom	char
Allele	Characteristic of a gen	value of char
Locus	Location of a gen	Position of a char

Table 1: Genetic terms

From random variation new advantageous properties develop and will establish oneself in

the current environment. As the environment changes, properties can become disadvantageous. They may get replaced by properties that better fit to the current environment. For generating new solutions, genetic operators are used. Before using a genetic operator, individuals have to be selected. There exist many different types of selection methods. Fitness proportionate selection also called roulette-wheel selection is used as basis for our selection method, which is introduced in Section 4.4. After selecting two individuals, the genetic crossover operator can be used. Furthermore, random variation can be accomplished with the mutation operator. After creating the new population, the mutation operator can be used to choose randomly an allele and change its value. The reproduction operator is used to take individuals unchanged into the next generation. For the testing method, we have developed specific crossover, mutation and reproduction operators. A detailed explanation of these newly defined operators can be found in Section 4.4.

3 Related Work

With maturing tool support and increasing processing power, semantic (context-aware) services have gained enormous interest in recent years. Several approaches to semantics definition have been developed and compared in the last decade [CDM+04, ZKN06]. However, OWL-S, the *Semantic Markup for Web Services* extension for OWL, which provides additional concepts for specifying semantic processes and services, remains the one most widely used today, although it focuses on Web services environments. Due to its modularity, however, it allows the definition of *Service Groundings* for other service architectures.

Even before semantic services were examined in particular, several context modeling approaches for home service platforms emerged. Gu, Pung, Zhang [GPZ04] propose using layered ontologies for context models in the home domain. It covers widely-acknowledged concepts that also appear in more recent publications (e.g. activities, locations, persons, devices including a custom service concept). However, the interaction of formal service definitions with runtime systems (actual groundings) and how user preferences influence the platform's behavior are not discussed. Daz Redondo et al. [RVC+08] propose the combination of OWL-S with an OSGi grounding called *OWL-OS*, although neither QoS properties nor semantic queries for service selection are supported (only key-value pairs). However, the authors use OWL-S service categories as a means to group services into aspects (e.g. Lighting), which can be used by clients during service look-up. Romero et al. [RHT+11] propose a platform based on the *Service Component Architecture* (SCA), which provides similar service and component abstractions as OSGi. The paper focuses on device integration over heterogeneous communication protocols into an event processing architecture, which is able to process data from a wide collection of devices (including sensor networks). However, a semantic abstractions for implementation details and support for the deployment of additional (3rd party) services are not presented.

Apart from home service platforms, a broad collection of service discovery and binding approaches have been developed in the past. The most widely used specifications for this today, especially in living environments, is *Universal Plug and Play* (UPnP) [UPn11].

Apart from its convincing technical capabilities, it lacks OWL's modeling capabilities as well as support for rules, queries and reasoning. Thus, it rather can be seen as a potential grounding for OWL-S services.

4 Approach

The work presented here proposes enhancements to home service platforms, which allow extending, testing and integrating the collection of applications and services dynamically based on semantic descriptions of service properties.

4.1 A Platform for Semantic Services

The approach presented here proposes adding semantic interfaces and continuous testing to applications and services that integrate with common OWL ontologies provided by a *Semantic Service Registry*. This registry allows extending an existing service registry for non-semantic services by adding the capability to manage ontologies and supporting semantic service queries. This extension supports the installation of non-semantic and semantic services in parallel. An overview of the surrounding platform is given in [Sch10]. In addition, the *Service Tester* component uses the descriptions of registered services to gather runtime QoS properties once after service registration. After that, the Service Tester is able to probe services on demand. The *Knowledge Module* stores the context and application ontologies including related rules, which are accessed by the other components. With changing system context, the contained ontologies are updated by incoming events resulting in constantly changing facts and, thus, also changing service properties. This dynamic can be used by clients in service look-ups, as bound services can differ between two subsequent look-ups due to interim model changes (e.g. state, location or performance change). The interaction between the involved components is displayed in Figure 1.

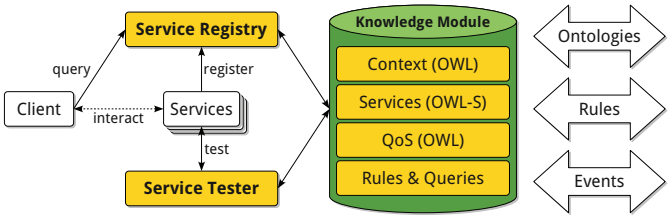

Figure 1: Service Registration and Testing

The context model has been constructed around the core concepts *Person*, *Computational Entitiy*, *Location* and *Activity* and is based on the context model from Gu et al. [GPZ04] introduced in Section 3, which has been extended with more detailed concepts for computational entities, services, activities and user preferences (among others) for this approach,

although the latter two are not covered here. Figure 2 presents a condensed version of the context model, in which the colored shapes represent some of the additions to the original model. The service-related ontologies contain service descriptions from service groundings (implementations) to formal service models and are defined using OWL-S. The *Service Registry* is responsible for registering and looking up services in the knowledge module and the platform's registry mechanism.

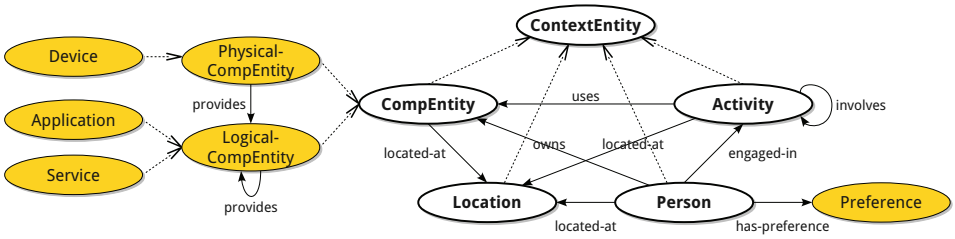

Figure 2: Context Model (condensed)

4.2 Service Registration

If a service implementation is installed by a service provider, it registers its implementation details – the service grounding – with the service registry assuming that the ontology describing the implementation integrates with the context and service ontologies. Subsequently, this integration allows the registry to find the service implementation, if the associated service is looked-up.

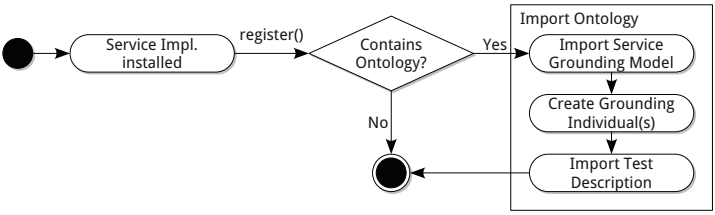

Figure 3: Service Registration Process

Apart from functional service properties (interfaces and parameters), services register additional context-related or QoS-related properties that can be specified as requirements by clients during service look-up. If these properties are dynamic, they are updated by the registry at runtime (e.g. by testing as described in Section 4.4). The registry is also responsible for removing service groundings, if service implementations are uninstalled or unavailable (e.g. caused by service failures). It is also possible for clients or services to register and update context or service ontologies without providing an implementation for them, as ontologies and implementations are decoupled.

4.3 Service Selection

If clients require a service, they can query the service registry for implementations of that service. This can be done using the platform's own service selection approach (if one exists), or by executing *SPARQL* queries [PE08] on the semantic service registry. The latter approach is required, if semantic properties are used for the service look-up. For the definition of look-ups, clients can use both functional and non-functional properties resulting in comprehensive queries. This can lead to situations, in which no service is able to meet these requirements. However, typically clients only define a limited set of requirements and end up with a selection of functionally equal services. In this case, the registry relies on the results of the service tests to select the fittest service. The service selection process is shown in Figure 4.

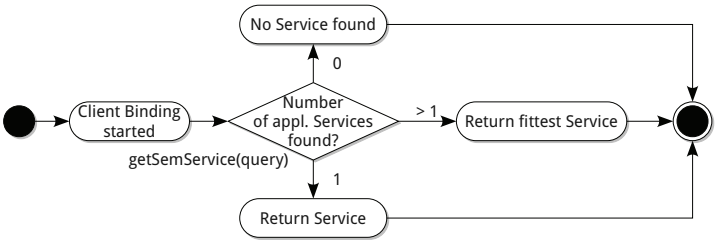

Figure 4: Service Selection Process

As an example, a client might look for a service providing off-site health monitoring. Here, services might feature more or less static properties such as the service provider's general trustability or service cost. Properties such as availability or response time, on the other hand, are dynamic properties, which would be collected by the service tester.

4.4 Service Testing

Given the new situation of being able to dynamically extend the service platform by services from different providers results in a great responsibility for testing these services. The need of testing is rooted in the fact that services can be added, deleted or substituted. Therefore no proven information exists how they perform on given client requests. As services of different service provider are supposed to interact, it is necessary to be able to rely on information given for these services (for example QoS Properties). Hence, a testing method should fulfill the requirements of adapting to currently offered services and client requests as well as being independent from the service provider offering a service. Therefore, we combine the approach of using a knowledge module with the approach of using genetic algorithms [MB10] for creating test sets.

QoS Property	Category	Valid Values
Response Time	1	$\leq 5ms$
	2	$6ms - 10ms$
	3	$11ms - 20ms$
	4	$\geq 21ms$

Table 2: Category Example

4.4.1 Test Case Adaption

In the following, a test set denotes a number of l different test cases. It should be taken care that a test set does not contain too many test cases because every test case results in a service-request and every service-request generates load for the service provider system. Therefore, it is necessary to find a suitable selection of test cases to compare and test services. As suggested in [Xia06], test cases are categorized by QoS properties and quality levels. Therefore, every QoS property is divided into different quality categories on the test system. These categories are used to distinguish between different QoS requirements. The test system manages m test sets for one category. A QoS property is e.g. the response time. Table 2 shows an example for such response time categories.

Crossover Operator	Modified Crossover Operator
Two offspring	One offspring
Locus of a gen is important	Locus of a gen is not important
All alleles are handed on to the offspring	A distinction is made between dominant and recessive allels

Table 3: Crossover Operator Modifications

If a client discovers a non-functional error, he has to send the ID of the service causing this error to the test system, the corresponding input, and the violated requirements. The test system uses these errors as test cases and generates test sets with this information. An error counter is attached to every test case. The value of the error counter reflects the up-to-dateness of a test case. When an error is reported, the corresponding error counter will be increased. In case of a new error, a test set will be chosen randomly and a test case to be substituted will be selected. For the selection, it has to be checked if the test set contains test cases having an error counter of value zero. In this case, one of them will be chosen randomly to be substituted. Otherwise, the selection of a test case will be based on a probability reverse proportional to its error counter. Test sets cannot be generated by errors only, but also by recombination of existing test sets. The advantage of recombinations is that good test cases can be unified to one test set. Hereby, is it possible to create better test sets from already existing ones. The approach presented in this paper is based on a genetic algorithm. This algorithm shows differences to standard genetic algorithms to reflect the considered application. Individuals are given by test sets of a QoS property category.

Test cases form the alleles and the fitness of a test set (individual). t_j is denoted as $f(t_j)$ and equals the sum of the error counters belonging to test cases of t_j . Let ec_i be the

error counter of test case i. Using the fitness, we can calculate the probabilities of the individuals. The probability p_j of an individual j depicts its possibility to survive, because an individual is chosen for the reproduction operator as well as for the crossover operator with probability $p_j = \frac{f(t_j)}{F}$, where $f(t_j) = \sum_{i=1}^{l} ec_i$ and $F = \sum_{j=1}^{l} f(t_j)$

For the crossover operator, two individuals have to be chosen. The crossover operator will produce one offspring from these two individuals. Not all alleles will be handed to the offspring, it will be distinguished between dominant and recessive alleles. The selection of alleles, which will be part of the subsequent generation, is affected by the probabilities calculated based on the error counters. The probability to include a test case (allele) in the next generation is proportional to the value of its error counter. First of all, the sum S of all error counters for both test sets has to be provided to calculate the probability $S = \sum_{j=1}^{2} \sum_{i=1}^{l} ec_{ij}$, where ec_{ij} is the error counter of test case i in test set j. Afterwards, the probability $p_{ij}^C = \frac{ec_{ij}}{S}$ can be calculated. Because we do not put back a selected allele, the values of S and p_{ij}^C change after every selection. Test cases handed on to the next generation by the crossover operator are called *dominant* alleles while the others are called *recessive*. Changes made to the crossover operator compared to the standard crossover operator are listed in Table 3.

The reproduction operator moves an individual to the next generation without any changes. The only modification of this operator is not to allow an individual to be chosen twice. Therefore the value of F as well as p_j change after every selection. A major problem of the crossover and reproduction operator is their seldom appliance to test sets having a low fitness value. Even with this low fitness value, these test sets might contain some test cases with high error counter values. Due to low error counter values of the majority of test cases, the overall sum nevertheless would be low. These high rated test cases would rarely have a chance to be handed on to the next generation without the mutation operator. For every mutation a test set has to be chosen from the current generation. The selection of test set j from all available test sets is done with probability $p_j^M = 1 - p_j$, where p_j is defined as described before. A test set is chosen for mutation with a probability reverse proportional to its fitness. This allows to extract test cases with high error counter values from test sets with a low fitness. From the chosen test set, a test case s_i will be taken with the probability p_i^{MST}. This probability is proportional to the value of the corresponding error counter ec_i

Mutation Operator	**Modified Mutation Operator**
Random selection of an allele which is to be mutated	The probability of selecting an allele to be mutated depends on its error counter
New allele is chosen randomly	New allele is chosen from parent generation with the help of the Fitness and error counters
Mutation is not cancelled	Mutation will be cancelled if it causes a duplicate allele

Table 4: Mutation Operator Modifications

of test case $s_i : p_i^{MST} = \frac{ec_i}{\sum_{j=1}^{l} ec_j}$. A test case e_k from the next generation has to be chosen to be substituted by test case s_i. The test set in which the substitution takes place is chosen randomly. If the test case s_i is already an element of this test set, the mutation will be cancelled. This avoids the presence of duplicate test cases in a test set. Test case e_k, which has to be substituted, will be chosen with probability $p_k^{MET} = 1 - \frac{ec_k}{\sum_{j=1}^{l} ec_j}$. This results in test cases with a low error counter having a higher probability to be substituted. The modifications of the mutation operator are summarized in Table 4.

Genetics provide a solution for adaption of individuals to a changing environment. This is one reason for taking genetic algorithms to address the issue of test case adaption. Another important issue is that genetics only takes the sum of all properties of an individual to calculate its fitness. This means an individual having a high fitness may even have low rated properties. The importance of handing on properties with a low rating is given by the changing environment. The rating of a property can improve as the system context changes. This reflects the changing ability of test cases to detect faults as the offered services and the requests change. We are confronted with the problem of an always changing search space and the consequence of never reaching a final optimum. The goal is to keep a variety of test cases and also to consider their rating changes over time. This implies giving newly registered test cases a chance to improve. Therefore, test cases with a low error counter should also get a chance to be integrated into the used test set.

4.4.2 Initial Test Setup

Initially, no test sets exist. However, services have to be tested from the start to find out which requirements they fulfill. Therefore, this phase will be conducted similarly to [TPC+03]. When a service is registered to the test system, the service provider has to deliver test cases for this service. Furthermore, the service provider has to deliver information about the service's QoS properties. Optionally, the service provider can specify, for which QoS property a test case is especially applicable. If no information is given, the test case will be seen as relevant for all QoS properties. The QoS property category of a test case is given by the QoS properties of the corresponding service. These initial test cases will be put into a pool belonging to the corresponding QoS property and category. There is a dedicated pool for every type of service and QoS property category. Furthermore, the pool can have test cases sent by a client. These test cases might not have caused any errors yet, but the client favors them to be tested. In this case, the client has to act like a service provider. The client has to send the type of service to be tested, the QoS property and the QoS property category. This enables testing a service not only with its own provided test cases, but also with other test cases provided by services of the same category or by clients.

The m test sets are generated with randomly chosen test cases from the corresponding pool. Test cases are removed from the pool as soon as they are used in a test set. As an improvement to the work of [TPC+03], services are also tested with inputs, which are currently used by clients and already have caused errors.

The step of generating a pool and m test sets is only necessary, if a service of a not existing functional domain is registered. Errors and inputs can be reported continuously to the service registry. Furthermore, the adaption and testing is a continuously repeating process of our test system.

5 Current State and Outlook

The approach presented in this paper is part of ongoing research, which focuses on enhancing the (self-)management capabilities of emerging home service platforms. Although the context model and the semantic service extensions presented here are fundamental elements of context-aware systems, they are just the basics for the creation of these self-management capabilities. As the introduction of semantics, just like service-orientation, decouple concepts from implementation technologies, the semantic service registry concept presented here is not limited to a specific implementation technology. For the prototypical implementation, however, the OSGi service platform [OSG09] was selected in combination with the *Jena Semantic Web Framework* [jen08] and the *Pellet OWL 2 Reasoner* [SPG+07] for ontology and SPARQL query processing. Here, the semantic service registry extends the default OSGi service registry and offers an additional client API for semantic service selection. Service registrations use either distinct or default OSGi API methods. In the case of the latter, OSGi service registration calls are intercepted and their associated bundles inspected for ontologies using OSGi's *Event Hook* mechanism that allows the inspection of OSGi services, whenever their runtime state changes (i.e., they are registered or unregistered). As clients have to formulate specific queries for semantic service look-ups, the semantic API offers an additional query-based method for this. At the current stage, however, the client still has to address the OSGi service interface of the targeted semantic service as part of the invocation, as abstract service queries including parameters cannot be mapped to arbitrary OSGi services currently, yet. For goal- or intention-driven semantic services, the service semantics have to be decoupled from service interfaces as well.

Part of this research focuses on the development of a self-management module for service platforms, which relies on dedicated management ontologies and associated rules to manage the platform – applications, services and devices – autonomically. As monitoring and state assessment are critical for this task, this management module relies on sensors and tests (as described in Section 4.4) to gather runtime information required to compute an appropriate management action to improve the managed system's state. Here, the approach for service testing with test sets plays an important role, as the effectiveness of management actions could be tested in this setup as well. Apart from the self-management module, test sets are generated by occurring faults and with a genetic algorithm. This results in test sets, which adapt to the current faults and offered services. The advantages of this testing method, therefore, lie in dynamic service environments. Future work will implement, test, and compare this approach with other solutions. Furthermore we think about expanding the fitness function to include other parameters besides the error counter.

References

[CDM+04] Liliana Cabral, John Domingue, Enrico Motta, Terry R. Payne, and Farshad Hakimpour. Approaches to Semantic Web Services: An Overview and Comparison. In *European Semantic Web Conference*, May 2004.

[Gmb04] VDI/VDE Innovation + Technik GmbH. Ambient Assisted Living - Preparation of an Article 169 Initiative. `http://http://www.aal169.org`, September 2004.

[GPZ04] T. Gu, H.K. Pung, and D.Q. Zhang. Toward an OSGi-based infrastructure for context-aware applications. *Pervasive Computing, IEEE*, 3(4):66 – 74, Oct.-Dec. 2004.

[Gro09] W3C OWL Working Group. OWL 2 Web Ontology Language Document Overview (W3C Recommendation). `http://www.w3.org/TR/owl2-overview/`, October 2009.

[jen08] Jena - Semantic Web Framework for Java. `http://openjena.org`, January 2008.

[MB10] Simone Meixler and Uwe Brinkschulte. Test Case Generation for Non-functional and Functional Testing of Services. *Object-Oriented Real-Time Distributed Computing, IEEE International Symposium on*, pages 245–249, 2010.

[OSG09] OSGi Alliance. OSGi Service Platform Release 4 (Version 4.2) Core Specification. `http://www.osgi.org/Download/Release4V42/`, May 2009.

[PE08] Eric Prud'hommeaux and Andy Seaborne (Editors). SPARQL Query Language for RDF (W3C Recommendation). `http://www.w3.org/TR/rdf-sparql-query/`, January 2008.

[RHT+11] Daniel Romero, Gabriel Hermosillo, Amirhosein Taherkordi, Russel Nzekwa, Romain Rouvoy, and Frank Eliassen. The DigiHome Service-Oriented Platform. *Software: Practice and Experience*, 2011.

[RVC+08] R.P. Diaz Redondo, A.F. Vilas, M.R. Cabrer, J.J.P. Arias, J.G. Duque, and A.G. Solla. Enhancing Residential Gateways: A Semantic OSGi Platform. *Intelligent Systems, IEEE*, 23(1):32 –40, Jan.-Feb. 2008.

[Sch10] Jan Schaefer. A Middleware for Self-Organising Distributed Ambient Assisted Living Applications. In *Workshop Selbstorganisierende, Adaptive, Kontextsensitive verteilte Systeme (SAKS)*, March 2010.

[SPG+07] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A Practical OWL-DL Reasoner. *Web Semant.*, 5:51–53, June 2007.

[TPC+03] W. T. Tsai, R. Paul, Z. Cao, L. Yu, A. Saimi, and B. Xiao. Verification of web services using an enhanced uddi server. *Object-Oriented Real-Time Dependable Systems, IEEE International Workshop on*, 0:131, 2003.

[UPn11] UPnP Forum. About UPnP Forum. `http://upnp.org/about/`, August 2011.

[Xia06] Jinchun Xia. Qos-based service composition. In *COMPSAC '06: Proceedings of the 30th Annual International Computer Software and Applications Conference*, pages 359–361, Washington, DC, USA, 2006. IEEE Computer Society.

[ZKN06] Jiehan Zhou, J.-P. Koivisto, and E. Niemela. A Survey on Semantic Web Services and a Case Study. In *Computer Supported Cooperative Work in Design, 2006. CSCWD '06. 10th International Conference on*, pages 1 –7, May 2006.