Flexible Composition of Smart Device Services

Mathieu Vallée[‡][¶] **Fano Ramparany**[‡] **Laurent Vercouter**[¶]
[‡]France Télécom R&D, TECH/ONE
28, Chemin du Vieux Chêne, 38240 Meylan, France
Email: mathieu.vallee@rd.francetelecom.com
[¶]École Nationale Supérieure des Mines de St-Étienne, G2I/SMA
158 cours Fauriel, 42023 Saint-Etienne Cedex, France
Email: Laurent.Vercouter@emse.fr

***Abstract*— Pervasive computing environments involve a variety of smart devices and home appliances. Communities of these devices form a ubiquitous information processing system to which the physical environment should provide an intuitive and flexible user interface. As each of these devices offers specific services, their number and variety make it difficult for users to control and monitor them all and tend to overcharge humans cognitive load. In this paper, we present an approach aimed at automatically composing device services, so that users can benefit from higher level services. We also ensure that context information such as users location, current needs and preferences is properly taken into account within the composition process. We believe that building mechanisms for synthesizing context-aware high level services is a mandatory step towards Ambient Intelligence.**

Keywords: Ambient Intelligence, Context-awareness, Service composition, Semantic services.

INTRODUCTION

Current progress in computer systems is mainly driven by hardware components miniaturization and ever increasing integration. This trend is enabling the provisioning of everyday objects with computing resources and makes silicon technology less and less visible in devices already computer enabled. However, the multiplication and heterogeneity of smart devices raise concerns about the amount of time and attention users must devote to interact with them, and strongly calls for new developments aiming at ”relieving the interaction cognitive load” [1]. Ambient Intelligence (AmI) pushes forward a vision where technology is integrated into everyday objects with the intent of enabling their adaptation to users and their context [2]. By favouring communication among objects and by enabling objects to sense the physical environment, AmI seeks to make users' interaction with their surrounding environment simpler and more intuitive.

We address the design of AmI systems using a service-oriented approach, in which devices in the environment provide independent, loosely-coupled, low-level services. Assembling or composing services enables a Service Oriented Architecture (SOA) to supply clients with more complex services. Such a facility offers several advantages including that of a modular and flexible approach to designing services. Using such an approach makes it possible to replace a component by another component. Services can thus better adapt to changing situations. Taking into account contextual information as well as users' preference is of paramount importance in this task. In this article we introduce our work on the automated composition of AmI services. We draw our inspiration from formalisms and techniques that have been developed in the framework of Web services composition. Our main contribution is in extending these techniques for adapting solutions to users needs, environmental conditions and any other contextual information.

In the following, we first introduce the role of service composition in an AmI environment. We next elaborate on our service composition system and algorithms. We then discuss our approach and we conclude on its advantages and perspectives.

I. THE ROLE OF COMPOSITION IN A SERVICE INFRASTRUCTURE

As mentioned above we aim at developing mechanisms for dynamically composing basic services in an open and ever changing environment. The result of the composition process will provide users with higher level services that satisfy their current needs, and that optimally adapt to the current context.

A. The Service Composition Architecture

Our service composition mechanism is viewed as an ”add-on” to an underlying Service-Oriented Architecture (SOA). This approach does not require any modification on services to have them collaborate by themselves. We designed our service composition function as a new component able to exploit service models, explore the space of composition alternatives in order to find an appropriate composition schema, and use this schema to enact the services involved.

Figure 1 presents the overall architecture and puts the emphasis on the role of the service composition system within the SOA.

The architecture is centered around a `Service Infrastructure` which keeps track of available devices and manages the services they offer. Typical Ambient Intelligence services include measurement services, data processing services and interface services. In such a SOA, services publish a description of their functionalities, so that they can be discovered and dynamically invoked by other services.

A `Context Management` system collects and maintains information about context. It is fed by rough sensors measurements, which it interprets, aggregates and stores. Like a knowledge base, it can answer queries submitted by other services.

The Composition System component lies on top of the Service Infrastructure and is composed of three elements. It initiates a services composition process in response to a change in the context which reflects new user's needs. In our illustration, a Personnal Assistant component supports user's interaction with the AmI environment. It interacts with the composition system to fulfill a user need which require a service composition.

As an answer to this request, a Plan Generation mechanism provides an abstract plan description which defines the elementary tasks to perform. These tasks specify service functionalities required to assist the user. In our current system, this mechanism uses a simple library of abstract plans designed for specific situations.

Starting from an abstract plan description, the Service Composition mechanism uses information about available services and context to discover services able to perform each task, and to check compatibility between the services to be composed. This mechanism creates a detailed plan description, specifying the services chosen and their relationship.

The plan Execution Management mechanism uses this detailed plan and triggers the execution of services on top of the service infrastructure. It also monitors the state of execution of the plan and might revise the composition process in case of any problem.

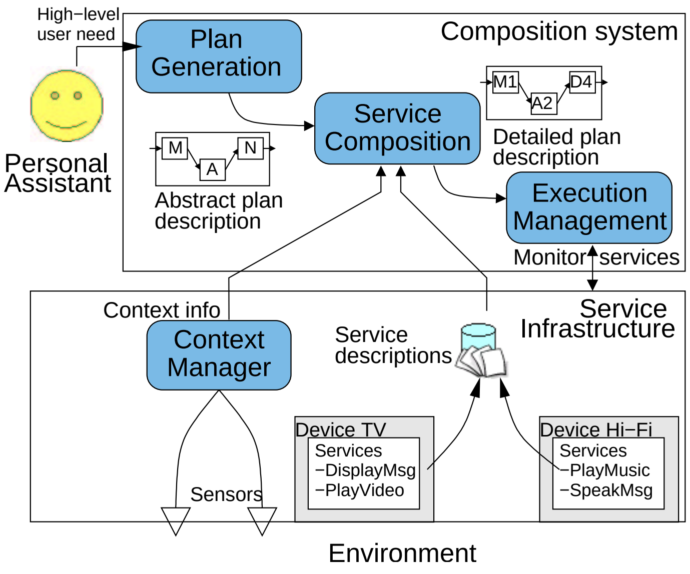

Fig. 1. Service composition architecture

This architecture enables to dynamically compose services in an AmI environment. In this proposal, our main concern is in the flexibility of service compositions with respect to the dynamics of the availability of services and of contextual information. Thus, we focus on the Service Composition mechanism which collects information from various sources in order to build and adapt a composition to current requirements. Our main assumption is that abstract plans, service descriptions and contextual information are available at the time of composition. We briefly discuss these assumptions as a conclusion to this paper.

B. The Oven Monitoring Scenario

The following scenario illustrates the use of our services composition in an Ambient Intelligence home environment supporting Albert, a dependent person, in using his home appliances.

Albert is in the kitchen to prepare his lunch. As soon as he leaves the kitchen, Albert's personal assistant checks if he has forgotten to switch an appliance off or to turn off a water tap. As Albert switched the oven on, the assistant infers that there is a need to monitor the oven and inform Albert about any abnormal or dangerous situation. This need is forwarded to the composition system.

To satisfy this need, the composition system has to build a high level service corresponding to an "home appliance monitoring" abstract plan. A graphical representation of this plan is shown in figure 2. It defines three tasks:

- a measurement task, which specifies the need to get measurements from the home appliance to monitor. For instance, an oven monitoring service that keeps track of an oven state (on/off, door open/closed ...) can provide such measurement.
- a measurements analysis task, which specifies a decision making process able to conclude on the abnormality of a situation, depending on the measurements. In the context of using an oven, "opening the door for more than 10 minutes while the oven is on" will be stated as an abnormal situation. Such a service can be customized for users according to their needs.
- a notification task, which forecasts to inform the user about the nature of an abnormal situation, using the most suitable interface.

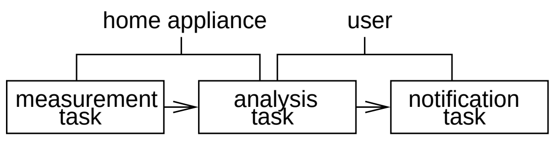

Fig. 2. Abstract Plan: Tasks represents sub-goals

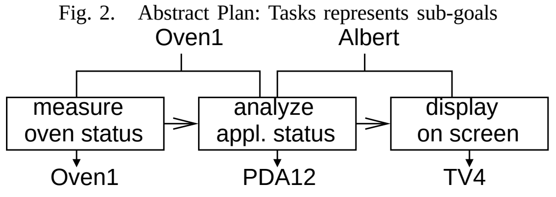

Fig. 3. Detailed plan: each task is achieved by a service

Several parameters complete this abstract plan description. Their values depend on the current situation. In our case, the user is Albert and the Home appliance is Oven1.

The clues provided by this abstract plan and contextual information enable the composition mechanism to find that an oven state monitoring service can achieve the measurement task. A service able to analyze information about the oven, customized for Albert's specific handicap, is used for the analysis task. Finally, as Albert is listening to music in the living room, the notification task is assigned to a service which can display a message on a TV set located in proximity. A graphical representation of the detailed plan is thus displayed in figure 3. In fact the `notification task` could have been allocated to other devices, depending on the level of emergency. For example, instead of displaying a message on the TV screen, a voice message could have been emitted on the Hi-Fi set or an alarm triggered. The service composition mechanism will select the best option depending on the situation at hand.

While this scenario has been kept very simple for illustration purpose, our system is adapted to a wide range of Ambient Intelligence applications. For instance, we can use our approach to synthesize and control an ambient Jukebox that plays music on an adapted device, depending on context information such as the location of the user, the presence of other people or the noise level of a room. Other applications would allow a user to automatically monitor a room where children are playing, by composing the appropriate camera service with a display service she carries with her, perhaps abstracting the displayed data through an activity analyzer. We also envision assistance systems involving human helpers as services, where our composition system allocates some critical decision task to humans while dynamically selecting measurement services to supply them with appropriate information.

The following sections detail how we have implemented the `Service Composition` mechanism, whose main features have been sketched above.

II. Service Composer Mechanism

Our service composition mechanism relies on service descriptions to model the functionalities of services, and on task descriptions to model the needs to be filled by a service. Both service and task descriptions are expressed using a formalism described in section II-A.

Using these descriptions and the information obtained from the context manager, the composition system is able to build a detailed composition plan from an abstract plan. The abstract plan does not refer to any existing service, and is composed of tasks, thus expressing a set of needs and their dependency relations. The resulting detailed plan explicitly specifies the suitable services and their interactions.

Figure 4 gives an overview of the composition process. This is a two-step process. First, a `Task Matching` algorithm considers each task independently in order to find a list of suitable services for this task. This list is ordered according to the relevance of services. Secondly, a `Consistency Checking` algorithm selects one service for each task in order to make the whole composition globally consistent. We now go into more details on the service description language that supports these two algorithms.

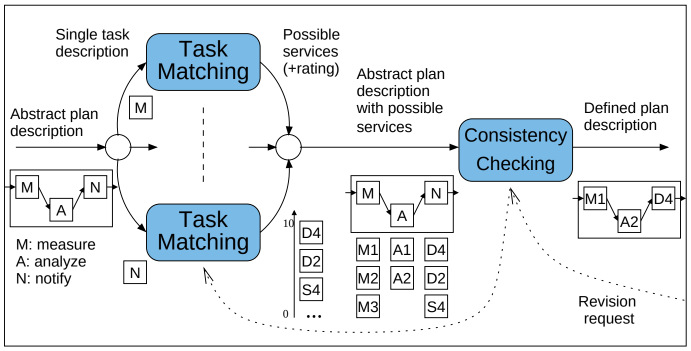

Fig. 4. Global composition process

A. Service Description Model

The need for such a service description has been widely addressed in the field of Web services, where extensive work has been done on how to make Web services usable by autonomous software agents.

This issue is related to the vision of the Semantic Web. In this vision machines can understand and manipulate data through the use of ontologies, which define knowledge domains in a formal manner. This has led to the concept of Semantic Web Services, for which Ankolekar et al. ([3]) have proposed a service ontology, named OWL-S [1]. This work, supported by the OWL-S coalition, has been founded by research on several AI fields, such as planning and software agent capacity descriptions.

Although OWL-S initially targeted at Web services, it is also of great interest for pervasive computing services, as illustrated in work conducted by Masuoka et al. [4] or Izumi et al. [5]. Our work investigates the use of such a semantic service description in conjunction with contextual information.

Following OWL-S point of view, we consider a service as a process which consumes inputs and produces outputs. This process is mainly described by its global service type (e.g. DisplayService) and its inputs and outputs types (e.g. Text Message). Types are concepts defined in ontologies, and are not usual data types (like String). Unlike keywords, these concepts enable reasoning on the service functionalities with respect to abstract ontologies. Another OWL-S feature we use is the ability to enrich a service description with preconditions (that need to be satisfied for service execution) and effects (produced by the service when executed). OWL-S provides a framework for describing Web Services, but some enhancements are necessary for using such a framework in AmI. An example is given in figure 5, for a service that displays a text message on a screen. In the following, we explain the description and the choices we have made.

serviceType: defines the type of a service. Unlike previous work ([6] [7]), we avoid using this type as a unique description of the service functionality. It rather gives a preliminary vague classification, and does not require a huge and precise ontology of all service types, as underlined in [8].

[1]http://www.daml.org/services/owl-s/

hasInput / hasOutput: define the inputs / outputs of the service, and `parameterType` describes their respective type. We distinguish two main classes of input / output types in our ontologies: binary data and physical data. Binary data can be transmitted between services within an information system. Physical data is exchanged between the environment (for instance the user) and the system. By introducing physical data, it becomes easy to model the exchanges not only between services but also with the physical environment. It is a key feature for Ambient Intelligence.

hasLocal: defines additional "local" parameters of a service. In our model, those parameters refer to elements of the real world which are relevant to the use of the service, such as devices and users. As with inputs/outputs, local parameters have a type. Their value can be either set by the description (with `parameterValue`) or deduced at composition time.

hasInputProperty / hasOutputProperty: define properties of inputs / outputs. An output property is an effect involving an output parameter, and an input property is a precondition involving an input parameter. When performing a service composition, an output and an input can establish a dependency relationship between services. I/O properties, as well as types, are used to check the consistency of this link.

hasContextCondition: defines a precondition referring to the context of use of the service. This feature is essential in an AmI environment, where the real service functionality is not only defined by static semantic properties but also by the current context, such as user's location.

```
(Service S11)
  (serviceType S11 Display)
  (hasInput S11 in11)
   (parameterType in11 TextData)
  (hasOutput S11 out11)
   (parameterType out11 VisualInfo)
  (hasLocal S11 dev11)
   (parameterType dev11 Screen)
   (parameterValue dev11 SCREEN_12)
  (hasInputProperty S11 '(accessi-
bleBy in11 dev11)')
  (hasOutputProperty S11 '(percepti-
bleBy out11 usr11)')
  (hasContextCondition S11 '(percep-
tibleBy dev11 usr11)')
```

Fig. 5. Sample description for a "display on TV screen service"

B. The Task Matching Algorithm

The `Task Matching` is a matchmaking algorithm, similar to that presented by Paolucci et al. [9]. Given a task description and a set of service descriptions, the matchmaking algorithm searches for services that are appropriate to fulfill the needs specified by the task. The matching is flexible, as the algorithm computes a kind of matching score between a task description and a service description.

While Web services matchmaking mainly relies on a semantic matching of service and parameter types, our algorithm also includes mechanisms to take into account input / output properties, and more significantly, context conditions required to use the service.

We consider four evaluation criteria to compute a matching score between a task and a service description:

Task and service types matching: both types are defined in an ontology. The algorithm considers their relationships (based on the subsumption algorithm [10]) to find how close a service functionality is from the task's requirement.

Parameter types matching: the algorithm also performs a semantic matching on the parameters' types (I/O and locals). It identifies corresponding parameters in the two descriptions and computes an overall matching score between the sets of task and service parameters.

Context condition evaluation: taking into account current values of local parameters they involve, the algorithm checks whether context conditions are valid in the current situation, and assigns a matching score reflecting the number of condition that hold.

I/O properties comparison: I/O properties are expressed as logical expressions. One way to compare them involves the implication operator. We consider that a task output property (resp. service input property) should be a logical consequence of one or more service output properties (resp. task input properties). To evaluate the matching score between I/O properties, the algorithm checks whether all, some or no properties of task and service can be related this way.

The order in which to apply the criteria is sensitive, as the first criteria quickly selects a reduced list of services and the last two ones depends on the parameter matching. Thus, each of these criteria is evaluated sequentially for one task, filtering and sorting candidate services at each step. The result of the `Task Matching` algorithm is then a list of services suitable to realize a task and sorted according to their global relevance.

C. The Consistency Checking algorithm

The `Task Matching` stage provides a list of potential services for each task independently. From this point, the system has to consider the interaction between services that could be assigned to each task, in order to build a globally consistent plan.

1) Consistency of the Dependency Links between services: The initial abstract plan defines dependency links between the tasks, stating which output of a task feeds which input of a subsequent task. Each time a service is selected to perform a given task, its links with the other services selected for the plan have to be checked.

We define a notion of the consistency of an I/O link on the basis of information defined in our description model. A link is called type-consistent if the type of output provided by a service is equivalent or more specific than the type of input that the subsequent service can handle. For instance, this ensures that there will be no attempt to link a service producing a text message with a service consuming video data. A link is called property-consistent if each property of the input can be deduced from properties of the output (as it is done for task/service properties matching (cf. sectionII-B). Thus, an I/O link is called consistent if it is both type-consistent and property-consistent.

2) Constraint Satisfaction Problem: In the final combination of selected services each link between services has to be consistent. We model the search for such a combination as a constraint satisfaction problem (CSP) [2]. We consider a variable for each task, whose domain is a set of values corresponding to the set of potential services to realize this task. CSP solving consists in choosing a value for each variable while ensuring that constraints between variables are satisfied by the chosen value. In our case, the choice is constrained by the consistency of links between subsequent services.

Although CSP solving is NP-Complete, we benefit for general purpose methods that greatly enhance performance [11]. More importantly, the algorithm tests values in the order given by the matching relevancy score, which reduces computation to a few consistency checking, especially when a task is strongly constrained by the current context. At last, the `Consistency Checking` uses the first solution to the CSP to build a detailed plan description, composed of services consistent with each other.

D. Composition Revision

A salient feature of service composition for Ambient Intelligence is the ability to quickly react to condition changes and update the composition scheme. In particular, we should be able to replace a service, in case it has unexpectedly disappeared (due to network, device failure or conflicting use by another application), or when a change in context or a runtime interoperability failure between services has occurred.

Our approach is particularly efficient in revising an already computed composition to handle such changes. For instance, if one service is no more available, the `Consistency Checking` algorithm will attempt to find a new solution to the CSP starting from the failing (but still approximate) solution, using very efficient local search methods. In case no new solution exists with the known set of services, the `Task Matching` algorithm will search for new services to replace the failing one, possibly extending its search in a larger area (in a way similar to the work of Chakraborty et al. [7]). This way, new services are added to the variables domains, creating new values to be considered in the CSP.

E. An AmI Service Composition Prototype

In order to assess the validity of our approach, we have implemented the plan instantiation algorithm introduced in the preceding section and applied it to concrete services corresponding to the scenario presented earlier in this article.

Various services descriptions and the abstract plan descriptions have been expressed using the OWL-S description framework. Although such descriptions still need to be manually edited, the use of the Protégé [3] ontology graphical editor along with its OWL-S plug-in [4] speeds up this operation.

The composition algorithm described in section II has been implemented in Java. We used the OWL-S API [5] to parse OWL-S description files. Reasoning upon service and parameter types is provided by the highly optimized Racer [6] engine, while we used the JTP [7] theorem prover to reason on service and task input/output properties. We also implemented a CSP solving algorithm based on the aima.search.csp [8] library.

III. Related works

Compared to related work that aims at dynamically building applications in the field of pervasive computing [12], we head towards more open and customizable environments through the use of a Service-Oriented approach. We do not require that applications comply to predefined APIs in order to be integrated into our system. This is key when we come to real life systems including various types of devices not primarily designed to interoperate. Instead, we favour the use of semantic descriptions of services to bridge the gap between the service level and our service composition system.

The issue of automatically composing services has been addressed in the field of Web Services, with foundation work around the concept of Semantic Web Services (SWS) ([13], [8], [14], [15]). Like Ranganathan et al. [16], we believe such an approach can be extended to the field of pervasive computing and Ambient Intelligence. However, while SWS work focuses on composing services using their static semantic descriptions, we propose a framework for considering not only semantically described functionalities, but also dynamic contextual information about the user and its environment to adapt the composition.

The use of context as a mean of adapting the behavior of a system to better interact with users has also been widely investigated. However, existing works [17] [18] usually demonstrate context adaptation in closed systems, where it can be handled through hard coded context rules. As we wish to expand context-awareness to open, dynamic system with evolving service availability, our approach tends to establish soft dependencies between services and the situations in which they can be used. Instead of defining precise rules to use specific services in specific situations, we rather define a ”context of use” of a service. We believe that the flexibility that is obtained by this way is the key to more efficient and robust systems where solutions can be found with only partial information on the considered services.

IV. Conclusion and Discussion

The work described in this paper contributes to the Ambient Intelligence vision by developing mechanism for automating

[2]a clear introduction to CSP is given in [11]

[3]http://protege.stanford.edu

[4]http://projects.semwebcentral.org/projects/owlseditor/

[5]http://www.mindswap.org/2004/owl-s/api/

[6]http://www.sts.tu-harburg.de/ r.f.moeller/racer/

[7]http://www.ksl.stanford.edu/software/JTP/

[8]http://aima.cs.berkeley.edu/

the composition of simple services offered by small smart devices in the user's physical environment. We aim at bridging the gap between these basic services and high level user's activities through dynamic and context-sensitive service composition.

We build upon work from the Semantic Web Service field and propose a mean to semantically describe services in an AmI environment. We extend these descriptions by taking context information into account, in order to ground services into the physical environment.

Based on these service descriptions, we propose a composition algorithm suitable to build up relevant service composition in a given situation, using contextual information. Our algorithm dynamically selects services using a semantic and contextual matchmaking algorithm and consistently compose them using Constraint Satisfaction Problem solving.

While our system enable dynamic and flexible composition of smart device services in an AmI environment, we can underline a few issues related to our approach.

A first concern is the availability of the semantic descriptions. As the semantic approach avoids the use of commonly agreed description standards, it enables descriptions to be collected from various sources (e.g. device vendors, services providers or users). Still, it might not be obvious that accurate and complete descriptions will be available for every device, and that the ontologies they rely on can be easily compared. We argue that the flexibility of our approach is still relevant when only limited or no description are available, as default information can be considered when none exists. Although it alters the quality of results, approximate solutions can still be found. A next step would be to dynamically complete service descriptions, considering for instance its interaction with other services or clues provided by users.

Another assumption required by the system presented is the preexistence of a library of abstract plans designed for possible situations (e.g. monitoring someone's kitchen). Creating and maintaining such a library might not be possible in real-life problems, where an unlimited number of situations may occur. While this work assesses the benefits of a semantic approach to adapt service composition, we are working on more elaborated systems in which composition and re-composition will be even more closely related to the users' activities. To this purpose, we plan to design our composition system as a multi-agent system (MAS). MAS are systems whose global behavior arises from interactions of autonomous entities (called agents). These systems present appealing features for AmI. Specifically, MAS enable to take into account heterogeneous information, for instance on services or on context, as agents use abstract models and communication languages to represent and share information. Thus, the semantic descriptions presented in this paper can serve as a basis to multi-agent reasoning and planning. Composition choices do not come anymore from predefined plans, but will be designed dynamically and collaboratively, taking context into account.

Finally, we underline that performance is an important issue in a pervasive computing context. Although we have implemented a prototype to demonstrate the use of our system, we do not provide performance results on our work at this stage. While focusing on flexibility as a first step, we aim at proposing an efficient, robust and scalable AmI composition infrastructure. To that purpose, we believe that a MAS approach will bring a suitable solution to AmI challenges. Decentralization favors local and reactive computation and avoids the bottleneck of a centralized system. Reconfigurable systems whose structure depends on situations can be created using flexible organization and re-organization capabilities. From these MAS features, we expect to come up with flexible, adaptable and reconfigurable systems.

Acknowledgement

The authors are grateful to Gilles Privat and Olivier Boissier for their contribution to the ideas presented here, and John Watlington for his valuable comments on earlier drafts of this paper.

References

[1] Gilles Privat. Des objets communicants a la communication ambiante. *LCN 4-2002*, 2002.

[2] W. Weber, C. Braun, R. Glaser, Y. Gsottberger, M. Halik, S. Jung, H. Klauk, C. Lauterbach, G. Schmid, X. Shi, T. F. Sturm, G. Stromberg, and U. Zschieschang. Ambient intelligence - key technologies in the information age. In *49th IEEE International Electron Devices Meeting*, 2003.

[3] Anupriya Ankolenkar, Mark Burstein, Jerry R. Hobbs, Ora Lassila, David L. Martin, Drew McDermott, Sheila A. McIlraith, Srini Narayanan, Massimo Paolucci, Terry R. Payne, and Katia Sycara. DAML-S: Web Service Description for the Semantic Web. In J. Hendler I. Horrocks, editor, *The Semantic Web - ISWC 2002: First International Semantic Web Conference*, volume 2342 / 2002, pages 348–363, Sardaigne, Italy, June 9-12 2002. Springer-Verlag Heidelberg.

[4] Ryusuke Masuoka, Yannis Labrou, Bijan Parsia, and Evren Sirin. Ontology enabled pervasive computing applications. *IEEE Intelligent Systems*, 18(5):68–72, Sep./Oct. 2003.

[5] Noriaki Izumi, Akio Sashima, Koichi Kurumatani, and Hideyuki Nakashima. Semantic services coordination for human and agent communities. In B. Burg, J. Dale, T. Finin, H. Nakashima, L. Padgham, C. Sierra, and editors S. Willmott, editors, *Agentcities: Challenges in Open Agent Environments*,. Springer-Verlag, 2003.

[6] A. Mingkhwan, P. Fergus, O. Abuelma'atti, and M. Merabti. Implicit functionality: Dynamic services composition for home network appliances. In *ICC'2004 IEEE International Conference on Communications*, 2004.

[7] Dipanjan Chakraborty, Filip Perich, Anupam Joshi, Tim Finin, and Yelena Yesha. A reactive service composition architecture for pervasive computing environments. In *7th Personal Wireless Communications Conference (PWC 2002)*, Singapore, October 2002.

[8] Katia Sycara, Massimo Paolucci, Anupriya Ankolekar, and Naveen Srinivasan. Automated discovery, interaction and composition of semantic web services. *Journal of Web Semantics*, Volume 1(Issue 1):27–46, September 2003.

[9] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia Sycara. Semantic matching of web services capabilities. In *Proceedings of the 1st International Semantic Web Conference (ISWC2002)*, 2002.

[10] Franz Baader, Ian Horrock, and Ulrike Sattler. *Handbook on ontologies*, chapter 1. Description Logics, pages 3–28. Springer, 2004.

[11] Stuart Russel and Peter Norvig. *Artificial Intelligence, A Modern Approach*, chapter 5. Constraint Satisfaction Problems. Prentice Hall, 1995.

[12] Manuel Roman and Roy H. Campbell. A middleware-based application framework for active space applications. In *Proceedings of ACM/IFIP/USENIX International Middleware Conference (Middleware 2003)*, Rio de Janeiro, Brazil, 2003.

[13] Sheila McIlraith and Tran Cao Son. Adapting golog for programming in the semantic web. In *Fifth International Symposium on Logical Formalizations of Commonsense Reasoning*, pages 195–202, 2001.

[14] Dan Wu, Bijan Parsia, Evren Sirin, James Hendler, and Dana Nau. Automating daml-s web services composition using shop2. In *Proceedings of 2nd International Semantic Web Conference (ISWC2003)*, Sanibel Island, Florida, October 2003.

[15] Mithun Sheshagiri, Marie desJardins, and Timothy Finin. A planner for composing services described in daml-s. In *Proceedings of Planning for Web Services Workshop in ICAPS 2003*, Trento, Italy, June 2003.

[16] Anand Ranganathan, Robert E. McGrath, Roy H. Campbell, and M. Dennis Mickunas. Ontologies in a pervasive computing environment. In *Workshop on Ontologies and Distributed Systems, Eighteenth International Joint Conference on Artificial Intelligence*, Acapulco, Mexique, August 9th 2003.

[17] Anand Ranganathan and Roy H. Campbell. An infrastructure for context-awareness based on first order logic. *Personnal and ubiquitous computing*, 7(x):353–364, 2003.

[18] Maja Vukovic and Peter Robinson. Adaptive, planning-based, web service composition for context awareness. In *Second International Conference on Pervasive Computing*, Vienna, April 2004.