2009 Fourth International Conference on Internet and Web Applications and Services

A service-oriented middleware for composing context aware mobile services

João Paulo Sousa
Departamento de Informática e Comunicações
Instituto Politécnico de Bragança
Bragança, Portugal
jpaulo@ipb.pt

Eurico Carrapatoso
Faculdade de Engenharias
Universidade do Porto
Porto, Portugal
emc@ipb.pt

Benjamin Fonseca
CITAB/Universidade de Trás-os-Montes e Alto Douro,
Vila Real, Portugal
benjaf@utad.pt

Abstract— **Recent advances in wireless networks and mobile devices have brought about new scenes for the provision of services to end-users. Besides traditional services, new ones may be provided that transparently adjust and adapt to the user context. The user would have more choice and flexibility if, besides using the services, he could also compose his own services in an ad-hoc way. This paper presents iCas, an architecture to create context-aware services on the fly and discusses its main components. Also an application scenario is briefly described.**

Keywords: Context-aware, Services composition, Semantic Web, Web Services

I. INTRODUCTION

It is predictable that in the near future the network mobile environment will be characterized by interaction between services and that those services will be provided to users dynamically and transparently. In this scenario, the use of captured contextual information related to issues such as location, current activities, objects in the neighbourhood and device features, plays a crucial role in the simplification of the interaction between humans and the digital world.

Often the user only assumes the role of consumer of services provided by third parties. For those users a set of useful services and information is provided, but they are aimed at a general market, leaving aside users that would like to take advantage of more personalized services. Our goal is to create an open infrastructure for a mobile network environment, in which a user can receive in his mobile device (e.g. PDA, netbook, notebook) context-aware information (e.g. location, time, neighbourhood, user profile) and have a set of useful services sensitive to his current context. The user can also compose services dynamically in real time to create a new highly personalized service with more features and use or share it as many times as he wants.

The remainder of this paper is structured as follows: section 2 discusses related work, section 3 introduces an ontology to describe context. Section 4 discusses the several approaches to composing Web Services and the OWL-S ontology. Section 5 presents the iCas architecture and describes the details of each component. Section 6 describes the scenario for using iCas, followed by the first performance evaluation, in section 7. Finally, we provide some conclusions and future work, in section 8.

II. RELATED WORK

A number of context-aware systems have been developed to demonstrate the usefulness of context-aware technology such as ParcTab [1], which was one of the first systems to offer a general context-aware framework; and ContextToolkit [2], which presents a modular context-aware framework with reusable components. This allows the programmers to build more easily, interactive context-aware systems based on sensors. These systems don't have an open context model because often the context is described in an object-oriented base and so the information is strongly coupled with the programming model.

More recently several studies appeared to support context-aware composition of services, one more generic and others dedicated to mobile environments [3] [4] [5] [6] [7].

In [3] the authors present a distributed architecture and associated protocols for service composition in mobile environments. This study emphasizes some factors that allow the composition of services in ad-hoc networks such as mobility, dynamic changing service topology, device heterogeneity, fault tolerance and reliability.

In [4] the authors propose a framework for dynamic composition of context-aware mobile services. The main features are service adaptation to the devices and network, and service adaptation to the user preferences and user location. However the study does not specifies which approach is used to compose new services.

The SOCAM [5] presents, a middleware architecture for building rapid context-aware services. It provides support for discovering, acquiring, interpreting and accessing context information. It also presents one of the first ontologies that define the main classes of context: person, location, activity and computer entity. Nevertheless, this architecture does not allow the composition of services. MyCampus [6] is a semantic web environment that uses agents able to find context information for enhancing everyday campus life. The MyCampus architecture is composed by eWallets (static knowledge containers), which support automated discovery and access to the context. The users can subscribe task-specific agents to assist them in different context tasks using the semantic information in eWallets. These agents are able to discover, execute and compose automatic semantic Web Services using the OWL ontology for services (OWL-S) [8].

In [7] the authors present CACS, a framework that enables context-aware composition of Web Services. This framework supports capability matches and goal-driven composition services flow. The CASC architecture uses software agents to discover, compose, select, and automatically execute Web Services using OWL-S.

In [3], [4], [6], [7] we saw that these systems don't have an open model to describe context, which cause some lacks on sharing context knowledge and context reasoning with external systems. The [3], [4], [7] studies present architectures that support the automatic composition of services. The user makes a request to the architecture, most of the times to a software agent, that collects context information and tries to find the most suitable service, which agrees with the request's description. If the agent doesn't find the service or it doesn't exist, then the software agent decomposes the request into multiple sub-goals in order to find the matched services.

In all the cases that use automatic composition, it is a hard task to maintain the details about the rules of services' invocation. These approaches also do not have an open model to describe context, which causes some limitations regarding the sharing of context knowledge and context reasoning with external systems.

III. Semantic Model

Contextual information models based on ontologies have been explored in several architectures that support context-aware services (e.g. [5] [9] [10]). These models allow the cooperation among objects and the discovering, acquisition, inference, and distribution of contextual information.

To describe the context, we decided to use the semantic model SeCoM (Semantic Context Model), presented in [10].

The use of a semantic model brings several advantages:

- the possibility of having a high degree of expressiveness and formalism to represent concepts and relations in a context-awareness scenario; it allows reasoning about context;
- the use of a semantic information context model, based on Semantic Web standards, makes the exchange, reuse and, sharing of context information between context aware applications easier;
- it decouples the information context model from the programming model, unlike some architectures presented in the previous section.

SeCoM is composed of six main ontologies: Actor, Activity, Spatial, Spatial Event, Temporal Event, Device, Time; and six support ontologies: Contact, Relationship, Role, Project, Document, Knowledge.

IV. Web Services Composition

The composition of services allows developers and users to create new services or applications, based on a Service Oriented Architecture (SOA) that supports description, discovery and communication. One of the most used SOA technologies is Web Services, due to the advantages already known to the scientific community [11] [12] [13].

Web Services have often been used for the composition of services. Nowadays there are six approaches to the Web Services composition [14]: WSBPEL [15], Semantic Markup for Web Services (OWL-S) [16], Web Components [17], Algebraic Process Composition [18], Petri Nets [19] and Model Checking and Finite-States Machines [20]. The previous approaches intended to solve the problems found in services composition such as syntax and semantic verification, resource reservation, QoS or deadlocks. In [14] and [21] the authors compare several solutions, based on characteristics such as automatic composition, composition verification, scalability, goal satisfaction, connectivity and non-functional properties.

When the purpose is to implement the composition of mobile services, we have to consider some concerns such as the complexity of the services to be built. For this purpose, one must find a compromise between simplicity in service creation and flexibility. A more flexible service requires more complex rules and probably specific technical knowledge. In this case the simplicity offered to end users is lost.

To achieve this goal, we chose to compose services in an interactive way: the user gradually generates the composition with ad-hoc forward or backward selection of services. To use this approach for composing Web services requires that they can understand their features and how they interact together. WSDL specifies a standard way to describe the interfaces of a Web Service at the syntactic level. However, WSDL does not support the semantic description of services. OWL-S has appeared to fulfill this limitation and uses the OWL language to describe Web Services. OWL-S provides Web services with a set of markup language constructs for describing the properties and capabilities in an unambiguous interpretable form to the software agents. OWL-S is a framework that enables automatic discovery and matchmaking tasks, and composition and execution of Web Services.

OWL-S consists of the following classes: ServiceProfile - specifies how the services are announced to the world; ServiceModel - specifies how to interact with the service; ServiceGrounding - specifies the details of how an agent can access the service.

V. Proposed Architecture And Implementation

To support the composition of context-aware services on the fly and provide context-aware information to the users, we propose a service oriented architecture (SOA) based on ontologies. We divide the architecture into four essential engines to explore the potential of context, showed in Fig. 1.

When a user selects the service composition IDE, the service discovery component gets the preferences, parameters configuration and interests. With this information and the OWL-S services descriptions, the service discovery and selection selects the services from the service repository to perform a context-based selection, and then delivers it as a list to the IDE.

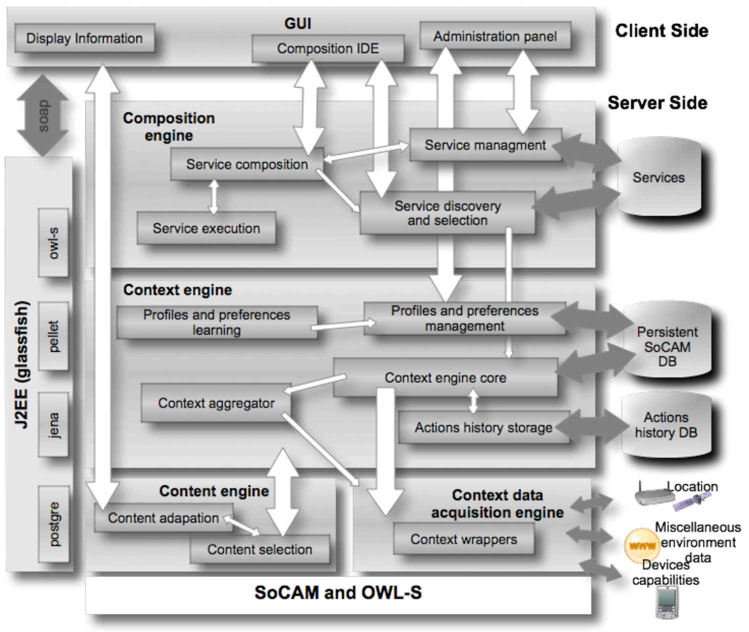


Figure. 1 Overview of iCas architecture.

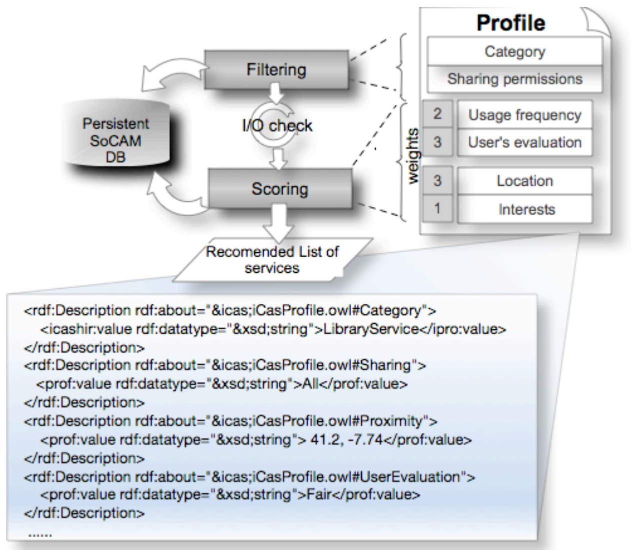
When a user starts a composition, maybe he knows clearly which tasks he wants to achieve with the composition or perhaps he starts to compose, choosing compatible services that can suggest the creation of a new service. In either situation the service composition is an ongoing process, where the user can add or remove services interactively.

Each time a service is selected to be part of the composition, the service discovery and selection module searches for services (Fig. 2) using data collected from the context engine core and returns further possibilities based on the current context and user policies. The search and selection is only possible due to the OWL-S service description, which allows creating relationships with other ontologies that can describe details about a service type and its features.

The search is performed using the description of the ServiceProfile class, which contains what the services can do, and specifies the input/output types, preconditions and effects. The first selection of services is performed using the ServiceProfile hierarchies, which choose the services from a particular category. Then a matching is performed, selecting the services whose input is syntactically compatible with the output of the current service.

Finally a scoring is carried out using the weights of the evaluation parameters defined in the ServiceProfile and a particular evaluation policy, which depends on the service category.

The ongoing user composition is supported by the service composition function, which generates a workflow of



```
<rdf:Description rdf:about="&icas;iCasProfile.owl#Category">
   <icashir:value rdf:datatype="&xsd;string">LibraryService</ipro:value>
</rdf:Description>
<rdf:Description rdf:about="&icas;iCasProfile.owl#Sharing">
  <prof:value rdf:datatype="&xsd;string">All</prof:value>
</rdf:Description>
<rdf:Description rdf:about="&icas;iCasProfile.owl#Proximity">
   <prof:value rdf:datatype="&xsd;string"> 41.2, -7.74</prof:value>
</rdf:Description>
<rdf:Description rdf:about="&icas;iCasProfile.owl#UserEvaluation">
   <prof:value rdf:datatype="&xsd;string">Fair</prof:value>
</rdf:Description>
 ......
```

Figure. 2 Overview of iCas architecture.

services calls (Fig 3).

By the time that user finishes the composition, the entity service composition has created a composite service that contains a workflow. This workflow is a composite service that has the three key descriptions of an OWL-S service: service profile, grounding and model, as mentioned in the end of section IV. This newly composed service can be saved, executed or used into another service composition task. To store the service, the service composer component uses the service management, and to execute the service it calls the service execution.

The service management component deals with the services stored in the services container, providing operations such as adding, removing and sharing services using the policies properties. The service container only stores the OWL-S description of this service (service profile, model and grounding). The service functionality is still provided by a third party (e.g. e-learning platform Web service).

The service execution module, using the OWL-S API, provides an execution engine to invoke atomic processes described by WSDL or UPnP groundings, and composite processes that uses control constructs sequences, unordered, and split. All the execution processes that depend on conditional statements, such as if-then-else and repeat-until, are not supported by the API. When the service execution promotes a composition, it follows a workflow to call each individual service and exchange data between them, according to the flow constructed by the user.

The context engine is responsible for managing all related context data and for reasoning about context. All context information is stored in a permanent OWL ontology storage system. The context engine core uses the Jena API to store the RDF models of SeCoM using a Postgre DB. This engine is also responsible for extracting knowledge from the SeCoM ontology, using SPARQL queries and for making inferences to derive additional statements that are not described explicitly in the SeCoM model (e.g. “if a user is located in the library, so he is in university campus”, or if a user has interests in “ontologies” and context-awareness is related to “semantic web”, hence the user is also interested in “semantic web”).

The context aggregators keep in memory (non-persistent), highly changing dynamic data that is captured from various sources related to an entity (e.g. user, object). For each entity an instance is created that relates that entity with the data that come from the sources (e.g. user’s location and data sensor). This component removes the computational charge caused by the frequent data updates into the persistent ontology.

The profiles and preferences management is responsible for managing the explicit user profile and interests information. Using the administration panel this component allows the user or administrator to manage explicit context such as insert, update and remove profiles parameters and user preferences.

The actions history storage captures each action performed by the context engine core and stores it in actions history DB. The main actions are search, insert, update and remove, and they are saved in the following format: Action + target Triplet (e.g. update: Bob isMemberOf the Sciences Students Group).

The profile and preferences learning component can change preferences and profile data through a learning algorithm (e.g. if a student queries many times for a particular book in the library services, the theme category of that book is added to the hasInterestesIn property of the knowledge ontology). The profile and preferences learning is an independent component. It searches for particular actions stored in the actions history DB, and counts the number of times that an action appears and, accordingly, changes specific parameters defined to be learned. Although this is not an optimal approach, a good solution can only be achieved with a large-scale utilization of iCas architecture

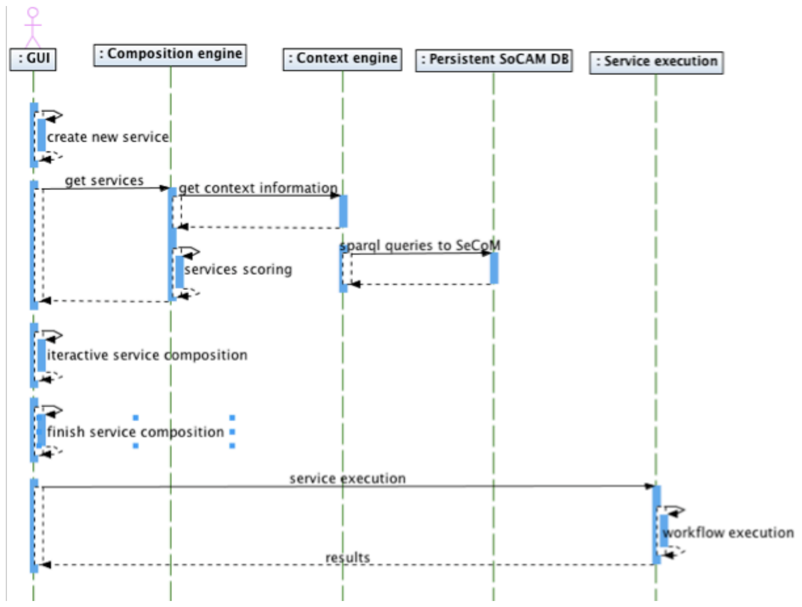


Figure. 3 Overview of iCas architecture.

and the collecting of user feedback. In the future the algorithm may also evolve to an AI algorithm, searching for patterns in the database.

The context data acquisition engine collects data from several sources, such as location devices, sensors and external services, and prepares the data to be used by the content engine and context engine (e.g. convert units values from a data sensor, or transform the coordinates user's location to a referential location (room 2.1)).

The content engine is composed by two components: the content selection is a timer function that periodically selects the user interests information from the context engine and delivers it to the content adaptation module for transformation. To be able to consult information in arbitrary devices, the information content must be provided in a device-independent way. iCas provides the context information as RSS feeds that are adapted by the content adaptation component. To do that this component adapts the information to the user's device features, using XHTML Modularization [22].

The iCas system is implemented integrally in Java (JDK 1.6.0). The iCas middleware architecture is composed of:

- Composition engine and context information system: Glassfish v2, JAX-WS 2.1, JAXB 2.1, Jena 2.5.4 and OWL-S 1.1.
- Context, profiles and preferences management DB: Postgre 8.2.8.
- Actions history storage management DB: Postgre 8.2.8.
- Ontologies models: SeCoM and OWL-S.

All four engines are implemented in the Glassfish v2 application server, which provides the functions to the GUI client through HTTP, as Web Services. This configuration was chosen to support the ad-hoc composition of services in mobile devices, bringing the reasoner's computational requirements to the server side.

VI. Example of Application

We chose a university campus as a scenario for using iCas (Fig. 4).

This architecture aims the support students and teachers in their campus life, helping them to keep updated and improve their social and pedagogical interaction. When a student arrives at the campus and connects his pda/netbook/laptop to the wireless network he will have to authenticate. This authentication is used to identify the user in a wireless system and in the iCas architecture.

The campus university already has a location system based on the wireless network, which is used to locate the users inside the campus. Besides the service location, the campus also has other services that can provide useful information integrated to the iCas system. Some of the most important services are: an e-learning platform that provides news about lessons, classes contents and others pedagogical information; library services; academic services that can provide administrative information such as official news and administrative services.

The main features of iCas consist of providing context-aware information and the dynamic composition of services. For this purpose the user's GUI client has four panels: informative, services composition, maps and administration. In the information panel the user can consult campus information based on his context (e.g., activities, events, news). To compose services in an ad-hoc away the user can use the services composition panel. If the user uses any service that has location output format, information will appear on the maps panel. Any task related with administration, such as changing user profile data and other explicit information, has to be done in the administration panel.

VII. Performance and Evalution

The implementation presented in the previous section is ongoing work. To get the first performance evaluation, we made some preliminary tests of some components that we consider critical to the viability of our approach.

In our test scenario we used two machines connected to the campus wireless network, with access points Cisco aironet 1100, which support IEEE 802.11g standard.

The machine 1 (M1) is an Intel Core 2 Duo 7400 (2.4Ghz) 3GB DDR2 with OS X 10.5.5, and runs the iCas architecture middleware described in Section V.

The Glassfish, that runs third party Web Services, is installed in machine 2 (M2), an Intel Core 2 Duo T8600 4 GB DDR2 with Linux (kernel 2.6.24) as its operating system. Some of the third party services installed in this

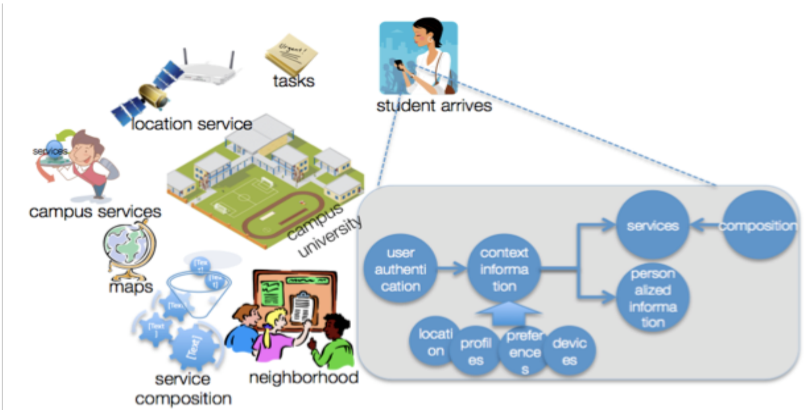


Figure. 4 iCas usage scenario in a university campus.

machine are services provided by the library, and e-learning platform.

We intended to get some preliminary results from the following main components that are exposed to computationally and I/O intensive processes: context engine core (inserting data and querying for derived contexts), service composition and service execution. We excluded services discovery and selections because the selection is highly dependent on the context engine core.

With the objective of gaining some feedback on how the context engine core performs, a high rate of data was inserted into a persistent ontology. The test was performed in M1 with 5000 users' information (username, hasName, hasSurname, hasFriend), which belongs to the two ontologies (actor.owl and relationship.owl). The time to add each user was less than 7ms; to update the same information it took less than 15ms.

To test the reasoning component we made several SPARQL queries, that generated information from three ontologies, and the results were between 10ms and 150ms. These preliminary results showed us that the use of persistent ontologies seems to be performing well.

To test service composition and service execution we ran a client in M1, which launched 500 threads. Each thread intended to simulate a user that orders a service composition and its execution. The resulting service consisted of two services joined in a pipeline. The services that were part of this composition were provided by the application server running in the M2 machine, and had an execution time of 20ms. Our intention was to see how the application server performed with a charge of service composition and execution. In this test all the requests were successfully completed and the average of time to finish the task was less than 300ms.

VIII. Conclusion

In this paper we have presented iCas, a service-oriented architecture that uses an ontological context model to provide personal and contextual information and to support the composition of context-aware services. The two major contributions of our work are the joint use of a semantic context model (SeCoM), to describe and explore the expression of contextual information, along with the support of dynamic composition, of context-aware services by the user.

We also present the first performance evaluation, which shows that our approach is viable. In the future we intend to finish iCas implementation and test it in a real scenario on a university campus. In this scenario we intend to determine how the context-aware mobile technologies can be used to improve pedagogical features and the socio-pedagogical interaction of various types of users.

References

[1] R. Want, B. Schilit, N. Adams, R. Gold, K. Petersen, D. Goldberg, J. Ellis, and M. Weiser, "The Parctab Ubiquitous Computing Experiment," Mobile Computing, pp. 45-101, 1996.

[2] D. Salber, A. Dey, and G. Abowd, "The Context Toolkit: Aiding the Development of Context-Enabled Applications," 1999, pp. 434-441.

[3] D. Chakraborty, A. Joshi, T. Finin, and Y. Yeshadoi, "Service Composition for Mobile Environments," Mobile Networks and Applications, vol. 10, 2005.

[4] S. Panagiotakis and A. Alonistioti, "Context-Aware Composition of Mobile Services", IT Professional, vol. 08, pp. 38-43, 2006.

[5] T. Gu, H. Pung, and D. Zhang, "A service-oriented middleware for building context-aware services," Journal of Network and Computer Applications, vol. 28, pp. 1-18, /01// 2005.

[6] M. Sheshagir, N. Sade, and F. Gandon, "Using Semantic Web Services for Context-Aware Mobile Applications," in MobiSys 2004 Workshop on Context Awareness, Boston, 2004.

[7] L. Nan, Y. Junwei, L. Min, and S. Yang, "Towards Context-Aware Composition of Web Services," in Fifth International Conference on Grid and Cooperative Computing, Washington, DC, USA, 2006, pp. 494–499.

[8] W3C, "OWL-S: Semantic Markup for Web Services," http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/, 2004.

[9] H. Chen, F. Perich, T. Finin, and A. Joshi, "SOUPA: standard ontology for ubiquitous and pervasive applications," in Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004. The First Annual International Conference on, 2004, pp. 258–267.

[10] R. Bulcao and M. Campos, "Toward a Domain-Independent Semantic Model for Context-Aware Computing," in Third Latin American Web Congress (LA-Web'05), Washington, DC, USA, 2005, p. 61.

[11] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, "Web Services - Concepts, Architectures and Applications," 2003.

[12] H. K. Cheng, Q. C. Tang, and J. L. Zhao, "Web Services and Service-Oriented Application Provisioning: An Analytical Study of Application Service Strategies," Engineering Management, IEEE Transactions on, vol. 53, pp. 520-533, 2006.

[13] M. P. Papazoglou, "Service-oriented computing: concepts, characteristics and directions," Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on, pp. 3-12, 2003.

[14] N. Milanovic and M. Malek, "Current Solutions for Web Service Composition", IEEE Internet Computing, vol. 8, pp. 51-59, 2004.

[15] Oasis, "UDDI v3.0 Ratified as OASIS Standard," http://www.uddi.org/, 2005.

[16] A. Ankolekar, "DAML-S: Web Service Description for the Semantic Web," 2002.

[17] J. Yang and M. Papazoglou, "Web Component: A Substrate for Web Service Reuse and Composition," in CAiSE '02: Proceedings of the 14th International Conference on Advanced Information Systems Engineering, London, UK, 2002, pp. 21-36.

[18] R. Milner, F. L. Bauer, W. Brauer, and H. Schwichtenberg, "The polyadic pi-calculus: a tutorial," in Logic and Algebra of Specification: Springer-Verlag, 1993, pp. 203-246.

[19] R. Hamadi and B. Benatallah, "A Petri net-based model for web service composition," in ADC '03: Proceedings of the fourteenth Australasian database conference, Darlinghurst, Australia, Australia, 2003, pp. 191-200.

[20] T. Bultan, X. Fu, R. Hull, and J. Su, "Conversation specification: a new approach to design and analysis of e-service composition," in WWW '03: Proceedings of the 12th international conference on World Wide Web, New York, NY, USA, 2003, pp. 403-410.

[21] B. Srivastava and J. Koehler, "Web service composition - current solutions and open problems," in ICAPS 2003 Workshop on Planning for Web Services, 2003.

[22] W3C, "XHTML™ Modularization 1.1, W3C Proposed Recommendation," http://www.w3.org/TR/xhtml-modularization/, 2008.