Defining and Prototyping a Life-cycle for Dynamic Service Composition

Eduardo Silva, Jorge Martínez López, Luís Ferreira Pires, Marten van Sinderen

Centre for Telematics and Information Technology
University of Twente, The Netherlands
P.O. Box 217, 7500 AE Enschede
{e.m.g.silva, j.martinezlopez, l.ferreirapires,
m.j.vansinderen}@cs.utwente.nl

Abstract. Since the Internet has become a commodity in both wired and wireless environments, new applications and paradigms have emerged to explore this highly distributed and widespread system. One such paradigm is service-orientation, which enables the provision of software functionality as services, allowing in this way the construction of distributed systems with loosely coupled parts. The Service-Oriented Architecture (SOA) provides a set of principles to create service-oriented systems, by defining how services can be created, composed, published, discovered and invoked. In accordance with these principles, in this paper we address the challenge of performing dynamic service composition. The composition process and its associated tasks have to be precisely defined so that the different problems of dynamic service composition can be identified and tackled. To achieve this, this paper defines a life-cycle for dynamic service composition, which defines the required phases and stakeholders. Furthermore, we present our prototype in which the different phases of the dynamic service composition life-cycle are being implemented. This prototype is being used to experiment with and validate our initial ideas on dynamic service composition.

1 Introduction

During the last years we have observed an increased use of the Internet, on both wired and wireless environments. Nowadays cellular phones are often sold with a data communication contract, which allows phone users to access numerous applications. Recent studies [1] show that in the upcoming years the continuously increasing use of small portable communication devices, referred as *Internet-centric pocketable* devices, will overcome the use of laptops, especially for users with high mobility. This tendency brings the Internet *always* and *everywhere*, triggering new opportunities and application areas. As a consequence, many of the biggest software industry companies are investing in new developments in Internet-based application service provisioning. *Software as a Service (SaaS)* [2] is a concrete example that is being adopted by some major software companies. This Internet-based methodology for software delivery provides the means to deliver services on-demand, as the user requires them, moving away from some classical approaches to software distribution, such as license-based. These new developments are based on the Service-Oriented Architecture (SOA) [3], which provides a set of principles to address the creation, share and use of services. A service is

realized by a given application, or system, and represents the external behaviour of the application, or system. This is the basic definition of service, which is going to be used throughout this paper.

With the emergence of Internet-based application services and open principles for service creation and use, such as SOA, new opportunities and approaches for service creation are appearing. One of these approaches is service composition, which focus on the creation of value-added services from existing services. Service composition should in principle reduce the development time of new services, while promoting the re-use of available application services. Furthermore, by using service composition it should be possible to create personalised services on-demand, based on specific user requirements. The creation of service compositions on-demand based on particular user requirements, context and preferences, characterizes *dynamic service composition*. Dynamic service composition will possibly support service developers at design-time, easing their task on service creation, and end-user, at runtime. Many people are working on different aspects of service composition in general, and much effort is being spent on some issues concerning dynamic service composition in particular. However, not much effort is being spent on the precise definition of the dynamic service composition life-cycle. A more precise definition of the dynamic service composition life-cycle should allow one to identify and reason about the concerns and requirements of this problem. In this paper we focus on the definition of such a life-cycle, and identify the research challenges that need to be addressed in each phase of this life-cycle. This paper also presents the prototype we are building to support dynamic service composition, which implements the life-cycle phases we have identified. This prototype should allow us to identify new problems that need to be tackled in this area.

The paper is organized as follows: Section 2 motivates dynamic service composition by discussing a target application scenario; Section 3 presents a service composition life-cycle, aiming at identifying and addressing the different phases of the dynamic service composition process; Section 4 presents the prototype we are developing to implement the phases of the dynamic service composition life-cycle; Section 5 discusses some related work; and Section 6 presents our conclusions and topics for future work.

2 Motivation

Service composition is mainly motivated by the principles provided by the Service-Oriented Architecture (SOA) [3]. The Organization for the Advancement of Structured Information Standards (OASIS) defines SOA as [4]:

> *A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations.*

According to the SOA principles, service developers can create new services, and make them available to potential service users. To make a service available, a service description document has to be produced, which describes the service properties, operations

and how the service can be invoked. Fig. 1 shows the basic interactions in a service-oriented architecture and the different parties involved in the architecture. SOA is not an implementation technology, but rather a set of concepts that can be implemented using different concrete technologies. Currently, the most popular technology to implement SOA is Web services [5, 6]. Many aspects of Web services have been standardized, making this technology mature and highly accepted by the industry.

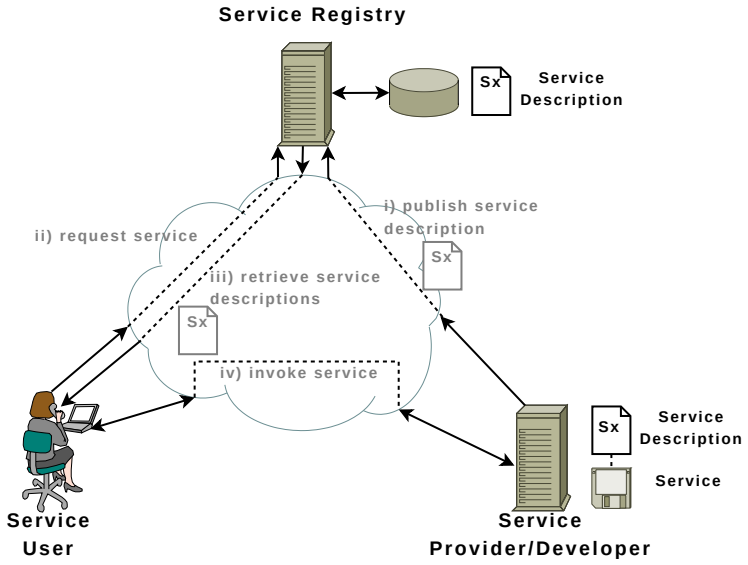

Fig. 1. Service-Oriented Architecture elements and interactions

SOA fosters the re-use of available services as components in service compositions. Traditionally, service composition is performed during design-time, resulting in service compositions that have to be used during runtime as they have been designed. However, new application scenarios have been identified that could benefit from dynamic service composition. We define dynamic service composition as *the process of creating a service on demand (at design or at runtime) to match specific user requirements and preferences by composing existing services*.

Amongst the scenarios in which dynamic service composition can be beneficially applied is *mobile computing*, which is presented in [6]. This scenario involves two main system parts, namely the user's mobile device and the back-end application server. Services in this scenario are called *Field Web services*. The user has a *field* mobile device, which is used to input the user's service request, store the user's profile and service preferences. The user's mobile device is also used as a *sensor* for the user's context. Provided with such a field mobile device, the user can create a service request and forward this request to the back-end server system. The back-end server system performs

all the necessary operations to provide the user with a service that matches the requirements expressed in the service request. This architecture allows advanced functionality and added-value services to be provided to the user on small *pocketable devices*. Another potential benefit is the increase of the battery life-time of the mobile devices, since some power-consuming processing is shifted to the back-end server system. Saving battery life-time is a key issue in mobile computing nowadays.

3 Dynamic service composition life-cycle

The life-cycle of dynamic service composition defines the required phases and the stakeholders that participate in the service composition process. Fig. 2 presents the life-cycle for the dynamic service composition process considered in our work.

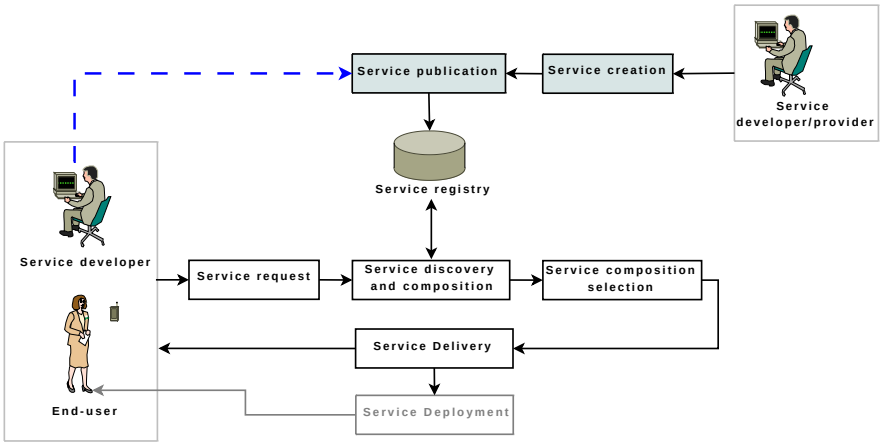

Fig. 2. Dynamic service composition life-cycle

We assume that three stakeholders participate in the life-cycle: *Service developers*, *Service providers*, and *End-users*. The life-cycle depicted in Fig. 2 consists of two main flows, namely *creation and publication of (atomic) services*, to be performed by service developers and service providers, and *(dynamic) service composition*, to be performed by end-users on demand, at runtime, or by service developers, at design-time. Each one of these flows is discussed below in terms of the phases and the issues to be solved in each phase.

3.1 Services Creation and Publication

Although the service provider and the service developer are two different stakeholders with well-defined roles, in this discussion we do not discern them. We assume that

the *service provider/developer* is usually a professional user or company, which focus on creating and providing services for a potential set of users. *Application Service Providers (ASPs)* [7] are organizations that play the combined roles of these stakeholders. A service provider/developer creates services, which may be built from scratch by programming new applications and making them available as services. A service may also be built by hand by re-using existing services in a composition, making the composition available as a service. The service creation phase consists on the *construction* of the service functionality and the respective service description document.

The service description document is used in the service publication phase to publish the necessary information to allow potential service users to discover the service. Furthermore, the service description document also has to contain all the required information to deal with the invocation of the service, such as the address of the service end-points and specific technical details (protocols, encoding, etc.). The service description document may also contain other non-functional information, such as quality-of-service, Service Level Agreements (SLAs), and possibly contractual conditions to use the service.

Service creation and publication are essential phases in the dynamic service composition life-cycle, since service composition can only take place if services that are candidates to be used in compositions are available and can be discovered for usage.

3.2 Service Composition

We assume here that an end-user or a service developer wants a new service to satisfy his specific needs. Fig. 2 shows an end-user, which usually has no technical skills on service composition and wishes a new service at *runtime*, and a service developer, which has technical skills on service composition, but wants to create a new service in a faster and more automated way given some specific requirements, usually at *design-time*.

The first phase of this process is the specification of a *service request*. The service request should provide enough information on the user requirements and preferences for the service. In the case of an end-user, additional context information can also be gathered to further adapt the service creation to the concrete user situation. In this phase, the end-user or the service developer interacts with the system that performs the dynamic service composition, but most of the other phases are expected to be performed transparently for these stakeholders. This is because we assume that dynamic service composition should be an automated process in which a service is created without requiring the direct intervention of a human user, as opposed to the service creation phase discussed in Section 3.1.

After the service request is defined, the *service discovery and composition* phase starts. The different services that could be used in a composition are discovered according to the composition algorithm. Service discovery is performed by invoking the interface provided by the service registry, based on information contained in the published service description documents. The service discovery phase depends on the publication phase discussed in Section 3.1. This implies that the information published in the publication phase should be compatible with the information required in the discovery and composition phase, which can be achieved by complying to open standards even if different organizations implement their own publication and discovery mechanisms. In

the service discovery and composition phase, an algorithm is often performed that takes the user service request to build candidate compositions of the services that have been discovered previously or whenever the algorithm needs them.

As a result of the discovery and composition phase, several compositions may be generated that match the service request, which means that the generated services may have to be selected. This happens in the *service composition selection* phase. In the case of an end-user, a single service (composition) should be returned, which implies that this phase is performed by the system that supports the composition process. This is because we have assumed that an end-user does not have the technical skills necessary to select the most appropriate composition. However, the end-user may be asked to indicate which properties should have the highest priority in the selection of a composition (e.g., the aggregate cost of using the services in the composition, or the performance of the composition in terms of response delay). In any case, this phase should be as transparent as possible for the end-user, i.e., the resulting composition should be selected only based on the service request and the user's preferences and context. In the case of a service developer, a list of services that match the service request may be returned. Mechanisms to rank the generated service compositions may be used to facilitate the selection of the composition that is finally used.

Service delivery is the phase that follows service composition selection, and it is concerned with the activities that are necessary to allow the end-user to use the service composition. This phase is necessary because the resulting composition may still be represented in some (formal) technology-independent notation, while an executable representation is necessary to deploy and execute the composition as a concrete service. In the case of a service developer, a service composition description may also be required to allow the composition to be published as a new service, so these issues are also relevant to this phase.

Service deployment is a phase that applies only to the end-user case. The end-user expects a running service, so the selected composition has to be deployed to allow its instantiation and invocation by the end-user.

At the end of the life-cycle some actions may still occur, depending on the stakeholder. In the case of an end-user, the service composition is invoked to deliver the service requested by the end-user. In the case of a service developer, the list of services is returned so that the most suitable composition the developer needs can be selected. The service developer may possibly adapt the composition further, to include some additional functionality. Fig. 2 shows an additional phase for the case of service developers, in which the service developer publishes the composed service so that it can be used later by end-users or other developers.

4 Prototype for dynamic composition of services

This section discusses the prototype we are building to implement the different phases of the dynamic service composition life-cycle. We are mainly aiming at developing a modular, scalable and extensible architecture, to address each of the life-cycle phases, but also at supporting different concrete technologies, such as, for example, different service description languages.

In our prototype we are currently using Spatel [8], which is a language developed in the European project IST SPICE [9] in which this work is embedded. Spatel supports service description, creation, composition and execution. It also supports semantic description of services, through references to ontologies. Ontologies are formal representations of conceptualizations, and are necessary in our approach to automate the different tasks of the service composition life-cycle, providing the abstraction and transparency needed in the dynamic service composition process.

Fig. 3 shows our initial prototype for dynamic service composition by indicating how each phase of our proposed life-cycle for dynamic service composition is mapped onto the components of the prototype.

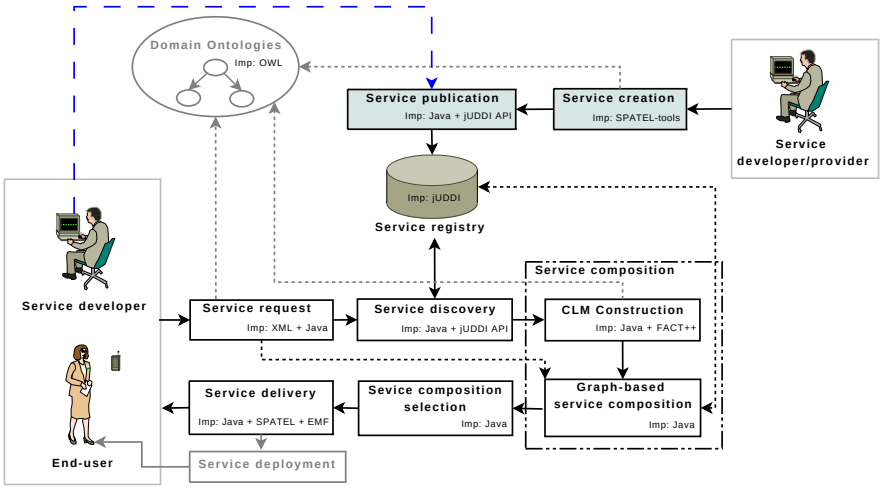

Fig. 3. Prototype for dynamic service composition

4.1 Service creation

The service creation process makes use of the Spatel tools, which offer a graphical interface to define the interface of services and the service end-points. This component also allows one to semantically annotate service operations with *inputs*, *outputs*, *preconditions* and *effects*, to define the *service goal*, which reflects the purpose of a service, and to define *non-functional properties* of a service, such as, for example, cost and response time. All these annotations refer to concepts formally defined in ontologies, which in our case have been produced in the scope of the SPICE project.

Other languages than Spatel may be used in the service creation phase as long as these languages support semantic annotations.

4.2 Service publication

The component responsible for service publication analyses the service description document, extracting the necessary information to publish the service in the service registry. The operation of extracting the information from a service description document depends on the used description language, which means that different interpreters have to be available to parse the supported description languages. Since we use Spatel for service description, in order to parse a Spatel service description document we generate a Java API from the Spatel Ecore model with the Eclipse Modeling Framework (EMF). This allows us to extract the semantically annotated properties from the Spatel service description document, namely Inputs, Outputs, Preconditions, Effects, Goals and Non-functional properties. These semantically annotated properties are always considered for semantic service description in our prototype, so independently of the description language a service is always published in the service registry in this same way.

The extracted information is organized and published in an UDDI-based registry. We use jUDDI [10] as service registry, which is a Java-based implementation of the Universal Description, Discovery, and Integration (UDDI) specification [11] for Web Services. Our purpose is to make use of the jUDDI API for publication and discovery of services, creating the necessary UDDI models to store the semantic annotations mentioned above.

4.3 Service request

The service request component allows the specification of a service request, using a set of semantic annotations that describe the properties of the desired service. The properties considered are the same as those used for service publication: Inputs, Outputs, Preconditions, Effects, Goals, and Non-functional properties. All these annotations refer to ontologies that are valid in the application domain being considered, in our case telecom services. This process is implemented using a simple interface that allows the construction of a (XML-based) service request document with the following structure:

```
<ServiceRequest>
    <Inputs>..</Inputs>
    <Outputs>..</Outputs>
    <Preconditions>..</Preconditions>
    <Effects>..<Effects/>
    <Goals>..</Goals>
    <Non-functional>..</Non-functional>
    <Ontologies>..</Ontologies>
</ServiceRequest>
```

4.4 Service discovery

In our approach the component responsible for service discovery is goal-based. The list of candidate services for the service composition is established before the actual composition is performed. The service request is analysed, and the goal annotations are extracted. Given these annotations, the service registry is queried through the jUDDI API Inquiry function for services with goals that are semantically related to the goal of the service request. This is possible because both the services and the service request

are described using the same properties, and ontologies. When the service registry is queried, not only semantically exact matches are retrieved, but other partial semantic matches are also possible, such as *Plugin*, *Subsume* and *Intersection* [12].

4.5 Service composition

Once the list of services with matching service goals is discovered, they are first analysed and organized in a formalism called *Causal Link Matrix (CLM)* [12], and after that they are composed using a graph-based algorithm [13].

CLM allows one to model all possible semantic connections, or causal links, between the discovered services. We use CLM^+, which is an extension of CLM, in order to capture the services' non-functional properties as well. Non-functional properties are used later in the composition, and in the selection phase.

To construct the CLM^+ matrix we use FACT++ [14], which is a reasoner for Description Logics (DL). This reasoner is used to infer DL relations such as Consistency and Subsumption. Based on these relations the CLM^+ matrix can be constructed, and once the CLM^+ matrix is available the actual service composition process takes place. In our prototype, the service composition process is implemented in Java, using a graph-based composition algorithm. The algorithm dynamics consists of finding a combination of services, previously organized in the CLM^+ matrix, which makes it possible to match the service request. The process starts by analysing the CLM^+ matrix for services that provide the requested outputs. Once this is done, the algorithm proceeds with a backwards composition strategy, resolving the inputs of the services of the graph. The composition process consists of a matching the inputs of the services in the graph with the outputs of services from the CLM^+ matrix. If multiple services match a given service input, an alternative composition graph is created, representing an alternative service composition. During each step in the algorithm, the aggregated non-functional properties in the composition graph are checked, to verify whether they match the requested non-functional properties. If a composition graph does not match the requested non-functional properties, it is not further considered and is discarded from the set of valid service compositions. The algorithm finishes (i.e., the composition is complete) when all requested inputs and goals are resolved.

4.6 Service composition selection

In the service composition process, several alternative service compositions that match the service request may be generated. This is possible since alternative services can provide the same (or similar) functionality. It is therefore necessary to rank the generated composition graphs according to some criteria. The ranking of the generated composition graphs can be made based on the graphs' aggregated non-functional properties (e.g., total cost), and the semantic connections of the services in the composition graph. For the end-user case the best service composition is selected, and for the service developer case all generated service compositions are stored in the ranked order.

4.7 Service delivery

In our case, the service delivery phase consists of translating the service composition from our graph formalism to Spatel. We do this by using the EMF and the Spatel Ecore models. We have not yet addressed this issue in detail, but the objective is to create a Spatel description of the generated composition service, so that the service developer can further adapt it to his needs, and the end-user can invoke and execute the composition.

The service deployment phase will be considered in our future work.

5 Related work

Service composition has received a lot of attention from industrial players and academia. Different aspects of service composition are being addressed, including the (partial) automation of service composition methodologies. However, the integration of the different parts of the life-cycle for dynamic service composition has not been addressed that often. This is in our opinion a very important step to create and evaluate suitable solutions to the dynamic service composition.

In [15] the problem of interleaving web service discovery and composition is addressed, by considering only simple workflows where web services have one input and one output parameter. In this case the web service composition plan is restricted to a sequence of limited web services corresponding to a linear workflow of web services. In our framework we propose a formalism to support the composition of services with multiple inputs and outputs, and also address the other phases of the life-cycle of the service composition process.

In [16] an algorithm for automatic composition of services is presented. The service composition is considered as a directed graph, where nodes are linked by the semantic matching compatibility ($Exact$, $Subsume$, $PlugIn$, $Disjoint$) between input and output parameters. Based on this graph, the shortest sequence of web services from the initial requirements to the goal can be determined. This approach computes the best composition according to the semantic similarity of output and input parameters of web services, but it does not consider any non-functional properties of the service composition. We consider this to be a very pertinent point to take into account, since the selection of the most suitable service compositions is often based on such properties (for example, costs and security).

In [17] a semi-automatic composition process is proposed to perform the composition of web services. This approach supports the system user in the selection of web services during each step in the composition process, and to create flow specifications to link them. The discovery process consists of finding matching services, which consist of web services that provide outputs that can be fed as input to the services of the service composition. After selecting all the services, the system generates a composite process [18]. The composition is executed by calling each service separately, and passing the results between services according to the flow specifications. This process allows more control over the composition process, which is sometimes desirable for service developers. However, since the composition process is semi-automatic, end-users without

technical knowledge probably cannot make use of this approach. Our framework deals with the composition process in a more abstract and automatic way, which allows its usage by both service developers and end-users.

6 Conclusions and future work

In this paper we motivate the dynamic composition of services, focusing on how to address all the necessary phases of the dynamic service composition life-cycle. We propose a life-cycle following the SOA principles, further extended with the necessary phases to perform dynamic and automatic composition of services. The life-cycle focuses on creating new services (i.e., service compositions) based on a service user service request. The discovery, composition and selection phases are transparent to the end-user or service developer.

Based on the proposed life-cycle we are developing a prototype implementation. The prototype is based on the use of ontologies to allow the automation of the service discovery, matchmaking and composition processes. We propose a goal-based discovery and graph-based composition algorithm, using non-functional properties for service composition optimization and ranking of the generated service compositions. The whole process of publication/discovery and composition is language-independent, meaning that different description languages, supporting semantic annotations, can be published. The generated service compositions can also be delivered in different execution languages.

The ideas and the prototype presented in this paper are still under development, and several issues still need to be addressed. Apart from supporting service developers, at design-time, we also intend to support end-users, providing runtime composition of services. To achieve this we have to provide an even more abstract way to describe service requests. The support of end-users also requires the deployment of the generated service compositions. At the moment, the service discovery process is completely done before the service composition process, and is goal-based. This approach has benefits, but also drawbacks. For example, during composition time it may turn out that the previously discovered services cannot be combined to form a matching composition. In this case, on-demand service discovery during composition time is necessary. The proposed prototype is being finalized and further evaluations will be performed. We are currently setting up a demo scenario to evaluate the performance of the proposed prototype.

Acknowledgments. This work is supported by the European IST SPICE project (IST-027617) and the Dutch Freeband A-MUSE project (BSIK 03025).

References

1. Gartner: Gartner highlights key predictions for it organisations and users in 2008 and beyond. http://gartner.com/it/page.jsp?id=593207 (January 2008)
2. O'Reilly, T.: The open source paradigm shift. In: Perspectives on Free and Open Source Software, The MIT Press (July 2005) 461 – 481

3. Erl, T.: Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall (2005)
4. MacKenzie, C.M., Laskey, K., McCabe, F., Brown, P.F., Metz, R.: Reference model for service oriented architecture 1.0. Technical report, OASIS (October 2006)
5. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web services: concepts, architectures and applications. Springer-Verlag (2004)
6. Papazoglou, M.P.: Web Services: Principles and Technology. Prentice Hall (2007)
7. Tao, L.: Shifting paradigms with the application service provider model. Computer **34**(10) (2001) 32 – 39
8. Almeida, J.P., Baravaglio, A., Belaunde, M., Falcarin, P., Kovacs, E.: Service creation in the SPICE service platform. In: Wireless World Research Forum meeting on "Serving and Managing users in a heterogeneous environment". (November 2006)
9. Cordier, C., Carrez, F., van Kranenburg, H., Licciardi, C., van der Meer, J., Spedalieri, A., Rouzic, J.P.L.: Addressing the challenges of beyond 3G service delivery: the SPICE platform. In: 6th International Workshop on Applications and Services in Wireless Networks. (2006)
10. Apache: Apache juddi. http://ws.apache.org/juddi/
11. Clement, L., von Riegen, A.H., Rogers, T.: Universal description discovery and integration (uddi) version 3.0. http://uddi.org/pubs/uddi_v3.htm (October 2004)
12. Lécué, F., Léger, A.: A formal model for semantic web service composition. In: ISWC 2006. LNCS, vol. 4273 (2006) 385–398
13. Lécué, F., da Silva, E.M.G., Ferreira Pires, L.: A framework for dynamic web services composition. In: 2nd ECOWS Workshop on Emerging Web Services Technology (WEWST07), Halle, Germany, Germany, CEUR Workshop Proceedings (November 2007)
14. Tsarkov, D., Horrocks, I.: Fact++. http://owl.man.ac.uk/factplusplus/
15. Lassila, O., Dixit, S.: Interleaving discovery and composition for simple workfows. In: First International Semantic Web Services Symposium. (2004)
16. Zhang, R., Arpinar, I.B., Aleman-Meza, B.: Automatic composition of semantic web services. In: 1st International Conference on Web Services. (2003) 38–41
17. Sirin, E., Hendler, J.A., Parsia, B.: Semi-automatic composition of web services using semantic descriptions. In: 1st Workshop on Web Services: Modeling, Architecture and Infrastructure. (2003) 17–24
18. Burstein, M.H., Hobbs, J.R., Lassila, O., Martin, D.L., McDermott, D.V., McIlraith, S.A., Narayanan, S., Paolucci, M., Payne, T.R., Sycara, K.P.: Daml-s: Web service description for the semantic web. In: International Semantic Web Conference. (2002) 348–363