Dynamic Service Composition: Why, Where and How

Eduardo Silva, Luís Ferreira Pires and 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, l.ferreirapires, m.j.vansinderen}@cs.utwente.nl

Abstract. We live in a society that is in its nature service-oriented: organizations and individuals get services from others, and provide services to others. This paradigm has been now applied to computer systems with the Service-Oriented Architecture, and it is gaining momentum, mainly motivated by the natural environment provided by the Internet to connect people and businesses. The Service-Oriented Architecture provides an architectural style for the creation, share, composition and execution of networked services. Given the actual dynamic, heterogeneous and distributed nature of computer systems, the composition of services requires mechanisms to support service description, advertisement, discovery, composition, and execution. In this paper we motivate the dynamic composition of networked services, presenting an overview on *why* this area is gaining importance; discussing *where* it has its most promising applications; and finally exposing our initial ideas on *how* dynamic service composition can be realized. To tackle these problems we present a life-cycle for the service composition task, and present our initial framework to support dynamic service composition.

1 Introduction

Nowadays we are observing a constant emergence of mobile computing devices, with powerful communication capabilities and increasing processing power. These devices are getting smaller and ubiquitous, and this tendency will continue. Recent studies [1] have concluded that in the upcoming years an increase usage of small devices, referred as *Internet-centric pocketable* devices, will overcome the usage of laptops, mainly for users with high mobility. Such a trend is triggering a change on the way software applications are provided, going from the traditional on-device software applications to Internet-based software applications. This class of Internet-based software applications will take advantage of the high processing power of back-end server systems, providing users with advanced functionality on their pocket computers, offered as services. Therefore, these trends are expected to cause an increase in the usage of Service-Oriented Architecture (SOA) [2]. The acceptance of SOA principles in the design of such distributed software systems will allow companies to sell and buy services based on subscription instead of product licenses. This idea of offering functionality as services (according to the SOA principles) is referred to as *Software as a Service (SaaS)* [3], and allows a client organization or user to use on demand services provided by other organizations or users. Such a change in the way software is provided (as a service) will mainly be possible due the high bandwidth available today, and the way software companies are

da Silva E., Ferreira Pires L. and van Sinderen M.
Dynamic Service Composition: Why, Where and How.
DOI: 10.5220/0004465000730085
In *Proceedings of the 2nd International Workshop on Enterprise Systems and Technology (I-WEST 2008)*, pages 73-85
ISBN: 978-989-8111-50-0

developing their services, by following open standards, which allows higher interoperability amongst different companies products.

In the context of end-users service provisioning new applications areas are appearing. A clear example is the creation of services on demand, taking into consideration the context (situation) and preferences of the user to adapt the service accordingly [4]. Users' preferences, behaviour, context, etc., vary with the user and his situation, so applications created targeting a large set of users, will not be optimally tailored for all their possible users. Having this idea in mind we claim in this paper that mechanisms for the dynamic composition of services are necessary in order to provide tailored services on demand to service users. We argue that SOA provides the basic principles to support dynamic service composition, but more mechanisms are necessary to improve the collaboration of the different parts of a service-oriented system.

The rest of the paper is organized as follows: Section 2 motivates the dynamic composition of services, why is it required and why is it possible; Section 3 shows some of the possible scenarios for dynamic service composition; Section 4 presents a possible life-cycle for dynamic composition of services; Section 5 presents our initial framework for dynamic service composition; Section 6 presents some related work on dynamic and automatic service composition; and finally Section 7 present some conclusions and discuss challenges to be addressed in the future.

2 Motivation

New service applications appear everyday, such as online map services, messaging services, location services, online shopping, etc. This is mainly triggered by the intensive use of the Internet, not only by companies but also by regular end-users, who can create applications and make them available. One of the most popular and successful examples is provided by the open source communities, which are most of the time a group of developers scattered all over the world working remotely (through the Internet) on common projects [5]. The result is a constant increase of available applications, which can be used by different users in different devices. Considering that such applications are made available, for example, on the Internet, new opportunities arise. One of the most interesting opportunities is the creation of new applications out of existing ones, *reuse instead of re-do*. The aim is to allow one to create a new application, in a given programming language, in a given system, and expose it to potential users without requiring them to use exactly the same set of technologies used to develop the application, but instead using the technologies that are more convenient to that application user. However to have such an open architecture all the different parties have to agree on common principles to allow interoperability between the applications. Service-Oriented Architecture (SOA) [2] provides such a set of principles to create distributed computing systems, which supports the creation of loosely coupled applications services in heterogeneous distributed systems.

2.1 Service-Oriented Architecture

The Organization for the Advancement of Structured Information Standards (OASIS) defines SOA as [6]:

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

Provided with such principles, developers can create functionality, and make it available to potential users. This functionality is provided as a service to possible users, by defining the service capabilities and how it can be invoked in a service description document. Figure 1 shows the basic concepts behind SOA, such as the different players and interactions required in this architecture.

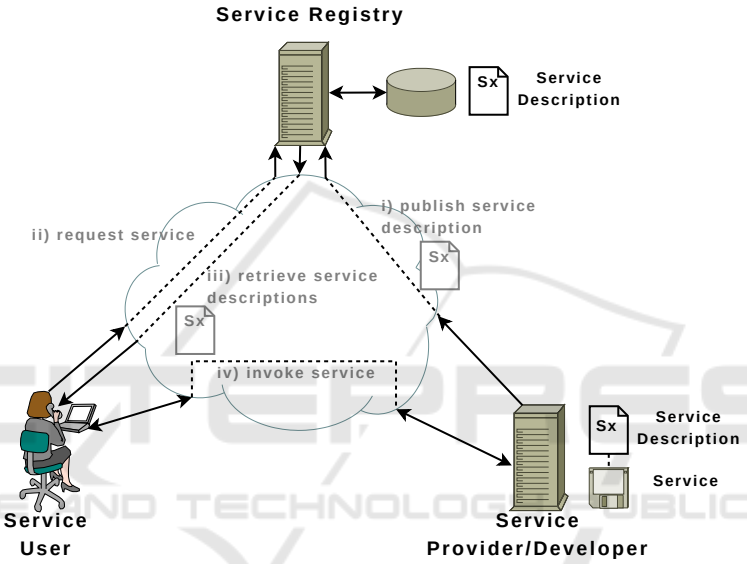

Fig. 1. Service-Oriented Architecture concepts and interactions.

SOA is not an implementation technology but a set of principles, that can be implemented in different concrete technologies. One of the most prominent and widely used technologies for implementing the SOA principles is Web services, which is a technology with high industrial acceptance, for which many standards and tools are available. This allows developers or service providers to create and publish services, and allow potential users (Service users) to discover services and possibly invoke them. Some of these standards are Web Services Definition Language (WSDL) [7], Simple Object Access Protocol (SOAP) [8], Universal Description, Discovery and Integration (UDDI) [9], and Business Process Execution Language (WS-BPEL) [10]. They allow one to describe services, exchange messages, publish/discover service descriptions and compose services, respectively. More standards have been developed, which aim at addressing all the additional issues concerning Web services systems.

2.2 Why Dynamic Composition

Traditionally service developers make application services from scratch, triggered by a specific request from the service users or by the identification of some potential (business) opportunity. This approach is time consuming, and leads, many times, to the duplication of already existing functionality. Following the SOA principles, service developers can instead create compositions of existing services to fulfill some given user needs. Nowadays there are plenty of tools to support developers in the task of service composition, and these tools tend to facilitate (ease and optimize) this task, enabling the re-use of existing services. However, the current approach consists of the creation of static service compositions, with fixed service end-points, targeting a specific group of service users. We argue that more dynamic composition mechanisms have to be developed to allow the creation of more personalized, adaptable and context sensitive services.

Assuming that different services are available that can be discovered and composed, we claim that more dynamic mechanisms can be used to achieve *on demand* service composition, given a specific user service request. This is the essence of dynamic service composition: *perform the composition of existing services on demand to match specific user requirements and preferences*. In this paper we motivate dynamic service composition based on a specific user service request, so taking the user request, context and preferences into account in the service composition process. Dynamic service composition may also address, such as, for example, the adaptation of a service composition in case a service component is unavailable, implying that an alternative service is discovered to replace the unavailable one, however this is not the focus of this paper. In [11] other research challenges have been identified in the area of service composition, however is clear that much focus is given to the creation of more dynamic mechanisms for service composition.

To achieve dynamic service composition, frameworks to coordinate all the phases of the service composition life-cycle are required. If such frameworks are available, users will be able to develop more personalized services, according to their needs.

3 Scenarios for Dynamic Service Composition

The most natural scenarios for service composition are *Internet-based*. This is mainly justified by the big number of applications that are available, which can be exposed as services (e.g., web services). Considering that the providers of these services publish their descriptions in a service registry, other users or service developers can discover and make use of these services. For example, if there are services that provide lists of hotels and lists of taxi companies given a location, a client user may on-demand create a new service that allows to book a hotel and given the location of the hotel book a taxi to take him from the airport to the hotel. Another clear example concerns inter-organisational (business) computing. If different organisations provide specialized services to each other, each organisation can focus on their own expertise and simply outsource some services by reusing other organisation's services to achieve the functionality they require. This inter-organisational cooperation allow partners to reduce the cost and possibly optimize their products, since they can focus on the problem to be

solved, avoiding to tackle all the less important issues that are required to solve it. The main issue in this situation, most of the time, is not the service composition process at the technical level, but the contractual issues. This is in our opinion one of the greatest barriers to the actual usage of service-oriented architectures in inter-organizational systems.

Another interesting scenario is concerned with *mobile computing*, in which mobile devices are provided with some functionality, but rely on back-end systems to perform the most complex computations and provide the necessary services. In [12] these services are described as *Field Web services*. The idea of this scenario is to provide the *field* mobile user device with the necessary functionality to interact with the back-end systems, and perform all the more complex computing tasks on the back-end systems. This is an emerging idea, and is gaining a lot of attention from different parties, especially from telecom service providers. If service users are provided with basic frameworks that allow them to discover and compose available services *anywhere and anytime* according to their context and preferences, companies may create a great source of revenue by providing such personalised services. This hybrid system (mobile clients and back-end server system) has a lot of potential applications, and fits the current trend of moving user client to mobile platforms. Another advantage that can be foreseen in applying SOA principles in mobile computing environments is that SOA provides a natural environment for task distribution, which allows one to save battery life of mobile devices. This issue is a very well known problem in mobile computing areas, since it is often a bottleneck for the usage of mobile devices. In [13] several ideas and challenges to the application SOA principles in mobile environments are discussed, as well as how more mature SOA principles applied to wired environments can be adapted to mobile environments.

These examples differ in their nature and application areas. The *Internet-based* scenario seems to be the most natural and also the most suitable scenario for service composition at the moment. However, the *Mobile computing* scenario, due to the intrinsic mobility of the users, provide interesting application opportunities to be explored. We expect that both scenarios will increase the usage of service-oriented architecture techniques, mainly triggered by the flexibility provided by such architectural approach. Dynamic service composition techniques will allow one to address the personalization, context and preferences of the users in any of these scenarios. This implies that efforts have to be made to allow the dynamic composition of services.

4 Service Composition Life-cycle

To present our framework for dynamic service composition we first introduce the notion of *service composition life-cycle* for dynamic service composition. Figure 2 shows the different phases and the different perspectives of the service composition life-cycle.

In the context of service-oriented systems different perspectives (or parties) have to be considered in the service composition creation and execution life-cycle. We admit that there are two main perspectives in this life-cycle: the *Service user* and the *Service developer/provider* perspectives. Other authors distinguish the service developer from the service provider. However, to simplify the discussion, we assume that the service

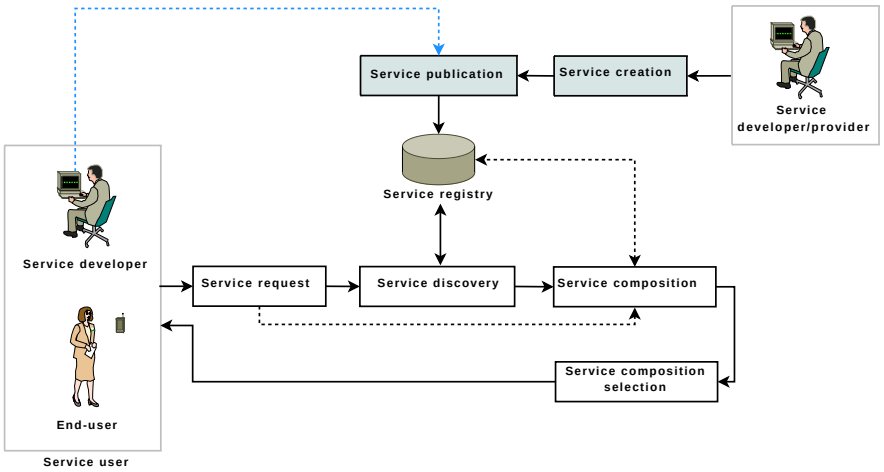

Fig. 2. Service composition life-cycle.

developer and the service provider are the same entity in our life-cycle. The service user can be an end-user, a person without much technical knowledge, or can also be a service developer, who makes use of services that are possibly provided by other service developers/providers to create new value-added services.

From the perspective of the service developer/provider, two main phases can be identified: *service creation* and *service publication*. The service creation phase basically focuses on the creation of the application service. An application service may be a new application created from scratch, or may consist of wrapped legacy or existing applications exposed as a service with a well-defined interface. The service creation can alternatively consist on the construction of a new service composition, meaning that a service developer/provider simply re-uses existing services to compose a new value-added service. After the creation of a new service, a service descriptions for this service should be published in a service registry. The publication phase consists on the publication of the service description document, so the service can be later discovered by possible service users.

From the perspective of the service user, several phases can be identified: *service request* specification; *service discovery*; *service composition*; *service composition selection*. The service request specification consists of the definition of the desired service properties and goals. This information is used to perform service discovery, and to drive the service composition and selection processes. Two main approaches to service discovery can be considered: discover all the relevant services for the composition, based on the service request; or iteratively discover the required services during the service composition process. A combination of these two approaches can also be considered, in which all the relevant services are considered first and then extra services are discovered on-demand at composition time, in case the set of discovered services is not sufficient to complete the service composition process. Independently of how the service discovery phase is implemented, it is always made based on information specified in the service

request. The next phase is the actual service composition, where an algorithm for the creation of a service composition plan is used to match the user service request for a composition. Given the set of discovered services, different alternative service composition plans may be generated. In this case, the next phase consists on the selection of a service composition, again based on properties of the user service request such as, for example, cost, reliability and response time, and his context and preferences. Additionally, and not stated in the figure, for the end-user case a service deployment phase must also exist, so the service can be deployed to be ready for execution. In Figure 2 there is another possible phase in the perspective of the service user, mainly the service developer - the *Service publication*. This phase consists on the publication of a service that is created dynamically for a given service user. This motivates the use of dynamic service composition mechanisms to support not only end-users, but also service developers, who can then publish the generated service compositions.

In this service composition life-cycle we ignored several issues, such as for example service binding, service deployment. We intentionally ignore these details, leaving them open to be addressed in the concrete frameworks for dynamic composition of services, since these operations can be specified in different phases of the life-cycle.

5 Framework for Dynamic Service Composition

Figure 3 presents our initial framework for dynamic service composition, following the life-cycle presented in the Section 4.

Figure 3, shows that our framework makes uses of ontologies for service creation and description, service request description and also for service discovery and the construction of service compositions. In computer science, an ontology consists of a formal specification of a conceptualization of a given domain. This formalization allows

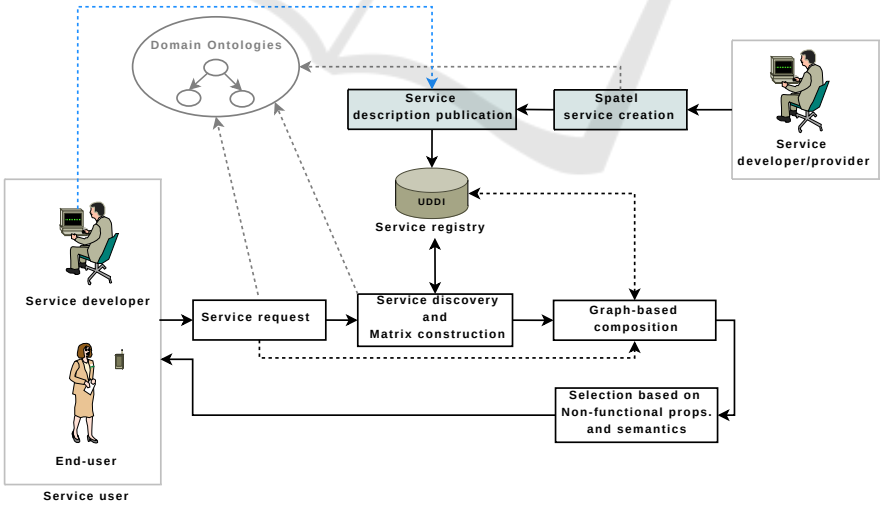

Fig. 3. Framework for dynamic service composition.

the description of a domain at a semantic level. This semantic level description enables automatic reasoning, i.e., without human intervention. In our framework we make use of this possibility to perform the service composition based on a service request and the service description of the existing services, both concepts described in common ontologies.

Our framework is being implemented in the context of the SPICE (Service Platform for Innovative Communication Environments) project [14]. In the SPICE project a language called Spatel [15] has been defined to support the creation, composition and execution of services and service compositions. Another property of this language is that it allows semantic annotations to be associated to service operations and properties. Provided with the Spatel language, a service developer can create new services, and semantically annotate them according to ontologies of the service domain. In our framework this is done by the *Spatel service creation* module. Another language could be used that support semantically annotated services, such as for example SAWSDL [16]. After the service is created, such annotations can be used in the *Service description publication* module, to publish the necessary information to enable service discovery. These modules reflect the life-cycle service developer/provider perspective in the framework.

From the perspective of the service user, the first step we consider in the framework is the definition of the *Service request*. The service request has to express the goal(s) of the service user for his service, so that the necessary discovery and composition of existing services can take place. We define a service request, in an XML-based format, as follows:

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

At the moment we are experimenting with simple stateless services, not taking into account complex service behaviors. The service request above has seven different types of annotations. The service *Inputs*, *Outputs*, *Preconditions* and *Effects* (IOPEs) refer to specific input, output, precondition and effect parameters that the service composition has to contain and satisfy. The *Goals* annotations describe specific goals that the service composition has to fulfill, such as, for example: translate, bookTicket, findDoctor. The *Non-functional* properties specify some additional requirements that the service composition has to fulfill, such as, for example: maximum cost of the service composition and minimum level of security. The *Ontologies* lists the ontologies used to specify the service request properties. This means that each property has to be specified following a defined concept in a valid domain ontology. An example of a service request is the following:

```
<ServiceRequest>
    <Inputs>
        <"LanguageOnt#Language" name="srcLang">
        <"LanguageOnt#English" name="trgtLang">
        <"LanguageOnt#Text" name="txtToTrans">
        <"TelecomOnt#PhoneNum" name="destNumber">
    </Inputs>
    <Outputs>
        <"TelecomOnt#AckSMS" name="AcknowledgmentSMS">
    </Outputs>
    <Preconditions/><Effects/>
    <Goals>
        <"GoalOnt#translate">
        <"GoalOnt#sendSMS">
    </Goals>
    <Non-functional>
        <"NFPOnt#Cost" value=0,50 EUR>
    </Non-functional>
    <Ontologies>
        <"GoalOnt" "TelecomOnt" "NFPOnt" "LanguageOnt">
    </Ontologies>
</ServiceRequest>
```

This service request denotes a service that performs the translation of a text from a given language to English, and sends the result by SMS to a specific telephone number. Furthermore the service should not cost more that 0,50 EUR. This service request specifies the necessary inputs for the service, i.e., the text to be translated, the source and target languages for the translation, and the telephone number to which the message has to be sent. The output of the service request is a simple acknowledgment that the SMS message has been received. The goals are to translate and send an SMS message. Finally, the ontologies used to specify the annotations are also listed in the service request.

Once the service request is available the *Service discovery and matrix construction* module can be used to perform the discovery of the necessary services and organize them into a matrix that facilitates the construction of the actual service compositions by the *Graph-based composition* module. We perform service discovery based on service goals. For example, in the service request above two goals are specified: $GoalOnt\#translate$ and $GoalOnt\#sendSMS$. Based on these goals the service discovery module queries the service registry (an UDDI-based registry, extended with semantic support) for services with goal annotations semantically related with the service request goals. After retrieving all the matching services, they are organized in a *Causal Link Matrix (CLM$^+$)* [17], which is a formalism that allows the representation of all the possible semantic links between the discovered services. By semantic links we mean the connection between services inputs and outputs, which are described with semantic annotations using common ontologies, to allow their composition and interoperability.

Once the CLM$^+$ is created, the *Graph-based composition* module can perform the necessary service composition. The service composition algorithm consists of a graph-based algorithm that uses the service request specification to drive the composition process. It starts from the specified service request outputs and composes services back-

wards until all the requested service inputs are matched and all the requested goals are resolved. At each iteration, the composition algorithm checks whether the requested non-functional properties are met by the service composition; if these are not met the composition is discarded.

At the end of the process several service compositions may be generated. To help select a particular composition we rank the generated compositions according the specified non-functional properties and the services semantic links. This is an important issue, since if the service user is an end-user without any technological knowledge, he expects to obtain a running service. This implies that a particular composition has to be selected if alternative service compositions are possible. In the future we also intend to take the user's context into account in the selection of the most appropriate service.

We refer to [18] for a discussion in the CLM^{+} creation, the graph-based composition algorithm and the proposed ranking algorithm for service composition selection.

6 Related Work

The area service composition is a very active area of research nowadays. Different aspects of service composition are being studied. However the integration of the different parts of the process of service composition, from the service request to the actual runnable service composition, using dynamic and automatic mechanisms is still not addressed by many.

[19] address the problem of interleaving web service discovery and composition, 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 [20] 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 compute the best composition according to the semantic similarity of output and input pa rameters of web services, but it does not consider any non-functional properties of the composition services. We consider this to be a very pertinent point to take into account, since the selection of the most suitable service compositions are many times based on such properties (e.g.: cost, security, etc.).

In [21] a semi-automatic composition process is proposed to perform the composition of web services. This approach supports the system user on the selection of web services for each activity in the composition, and to create flow specifications to link them. The discovery process consists on finding matching services, meaning web services that provide outputs that can be fed as input on the services that exist in the service composition. After selecting all the services, the system generates a composite process in DAML-S [22]. The composition is executed by calling each service separately, and

passing the results between services according to the flow specifications. This process grant a higher control over the composition process, which is sometimes desirable for service developers. However, and since the composition process is semi-automatic, end-users without technical knowledge can't usually make use of this approach. Our framework deals with the composition process in a more abstract and automated way, which allow its usage by both service developers and end-users.

7 Conclusions and Future Work

In this paper we motivate dynamic service composition. We claim that given the current trends on computer and communication systems an increase usage of distributed application services is expected. This implies that new mechanisms and architectures are required to support such systems, and also to provide users with tools to use these new application services. The main architectural principles to support these ideas can be found in the Service-Oriented Architecture (SOA).

To motivate the creation of mechanisms for dynamic service composition we provide two potential scenarios suitable for service composition: the *Internet-based* scenario, where several services can be published or advertised, and one can make use of them to compose new application services, reusing the existing services; and the *Mobile computing* scenario, which has a lot of potential with the emergence of mobile devices and communications. In the later scenario mechanisms are required to support mobile users, providing them with minimal functionality at the mobile terminal, and performing the complex tasks at the back-end server systems.

We conclude by providing some initial ideas on how to tackle the problem of dynamic composition of services. We discuss a dynamic service composition life-cycle, showing the phases, and perspectives that are necessary to support the process of dynamic composition of services. Based on this we present our initial framework for dynamic composition of services, from the service user request to the actual service composition.

In the future we plan to explore further our ideas and improve the proposed framework towards a generic framework to support dynamic service composition using different technologies in different application scenarios. The following research challenges have been identified:

1. The service request module has to accept user service requests in an abstract form, to support not so technical skilled users. It has also to collect context information and other user preferences, to be used in the composition process.
2. At the moment we perform service discovery based on the specified goals on the service request, but it is clear that other services may be needed at composition time to complete a service composition. Given this, mechanisms to make service discovery at composition time have also to be considered in the framework.
3. The use of ontologies is clearly required to allow such a dynamic mechanism for service composition. Nevertheless how and where these ontologies are defined is still fuzzy. This is an issue that may have very interesting research challenges.
4. The proposed framework is being prototyped, following the proposed modular architecture. The aim is to provide a very modular architecture so one can easily

extend it and support other service description languages, and service composition languages. We plan to evaluate the prototype in a specific scenario in the e-health domain, specifying for this a library of services and also the necessary ontologies to describe the domain.

Acknowledgements

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. Erl, T.: Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall (2005)
3. O'Reilly, T.: The open source paradigm shift. In: Perspectives on Free and Open Source Software, The MIT Press (July 2005) 461 – 481
4. Jorstad, I., van Thanh, D.: Service personalisation in mobile heterogeneous environments. In: Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services, IEEE Computer Society (February 2006) 70 – 75
5. Raymond, E.S.: The Cathedral and the Bazaar. O'Reilly & Associates, Inc., Sebastopol, CA, USA (1999)
6. 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)
7. Chinnici, R., Moreau, J.J., Ryman, A., Weerawarana, S.: Web services description language (wsdl) version 2.0. http://www.w3.org/TR/wsdl20/ (June 2007)
8. Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.J., Nielsen, H.F., Karmarkar, A., Lafon, Y.: Simple object access protocol (soap) version 1.2. http://www.w3.org/TR/soap12-part1/ (April 2007)
9. 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)
10. Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Business process execution language for web services, version 1.1 (May 2003)
11. Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F.: Service-oriented computing: State of the art and research challenges. IEEE Computer **40**(11) (2007) 38 – 45
12. Papazoglou, M.P.: Web Services: Principles and Technology. Prentice Hall (2007)
13. Sen, R., Handorean, R., Roman, G.C., Gill, C.: Service Oriented Computing Imperatives in Ad Hoc Wireless Settings. In: Service-Oriented Software System Engineering: Challenges And Practices. Idea Group Publishing (2005) 247 – 269
14. 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. (May 2006)
15. 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)

16. Sivashanmugam, K., Verma, K., Sheth, A., Miller, J.: Adding semantics to web services standards. In: 1st International Conference on Web Services. (2003) 395–401
17. Lécué, F., Léger, A.: A formal model for semantic web service composition. In: International Semantic Web Conference. LNCS, vol. 4273 (2006) 385–398
18. Lécué, F., Silva, E., Pires, L.F.: A framework for dynamic web services composition. In: 2nd ECOWS Workshop on Emerging Web Services Technology, CEUR Workshop Proceedings (November 2007)
19. Lassila, O., Dixit, S.: Interleaving discovery and composition for simple workfows. In: First International Semantic Web Services Symposium. (2004)
20. Zhang, R., Arpinar, I.B., Aleman-Meza, B.: Automatic composition of semantic web services. In: 1st International Conference on Web Services. (2003) 38–41
21. 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
22. 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