On the Support of Dynamic Service Composition at Runtime

Eduardo Silva, 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, l.ferreirapires, m.j.vansinderen}@cs.utwente.nl

Abstract. Network-based software application services are receiving a lot of attention in recent years, as observed in developments as *Internet of Services*, *Software as a Service* and *Cloud Computing*. A service-oriented computing ecosystem is being created where the end-user is having an increasingly more active role in the service creation process. However, supporting end-users in the creation of a service, at runtime, is a difficult undertaking. Users are heterogeneous, have different requirements, preferences and knowledge. Furthermore, and since we cannot assume that all users are technical experts, we conclude that highly abstract mechanisms should be available to support the service creation process. To tackle these issues and provide end-users with personalised service delivery, we claim that runtime automated service composition mechanisms are required. In this paper we present the DynamiCoS framework, which aims at supporting the different phases required to provide users with automatic service discovery, selection and composition process. In this paper we also present the developed prototype and its evaluation.

1 Introduction

With the Internet becoming ubiquitous, the use of network-based application services is being increasingly adopted and it is expected to grow in the upcoming years [1]. This is being reflected in many technology developments and innovations, such as, for example, the *Internet of Services*, *Cloud Computing* and *Software as a Service* (SaaS). This is leading to the emergence of large sets of services in different domains. At the same time, the use of mobile devices with powerful communications capabilities is increasing quite rapidly. In [2] it is reported that by 2013 more than 38% of the European population will access the Internet on their mobile device, which is an increase of 300% compared to the current situation.

These developments are allowing and *pushing* new, more adaptive and personalised, application services where the end-users play an active role in the service creation process. Supporting end-users in this kind of process is a complex undertaking. Different users have different preferences and request services in different situations (contexts), which requires different actions to be taken. Furthermore, end-users expect a high-level of abstraction in the service creation process, since they are not properly trained to

use complex (technical) tools. Given this, some degree of automation has to be provided to support the end-user in the service creation process. We claim that this can be achieved by using semantic-based service composition approaches. Semantic information enables the use of computer agents, which can automatically reason on the services and user specified requirements. This alleviates the user from the burden of some of the details and the complexity of the service composition process. We denote the problem of automatic service composition based on user requirements as *dynamic service composition*. To cope with this problem, we propose a framework for dynamic service composition provisioning called *DynamiCoS*.

The aim of the DynamiCoS framework is to provide all the necessary support to users, namely to achieve automated runtime service composition. To achieve this automated support, DynamiCoS uses ontologies (domain conceptualisations). The framework allows different service developers to publish their semantically annotated services in the framework. These semantic descriptions have to refer to the framework's ontologies. Users may have different domain or technical knowledge, which implies that their service request interfaces have to be defined accordingly. DynamiCoS tackles this problem by prescribing a service request that supports different user interfaces. A service request consists of a specification of *goals* the user wants the service to achieve. A *goal* is likewise used to describe services, specifying the activities (or operations) the services can perform. Goals of users and services are specified according to the framework ontologies (representation of the domain of knowledge), which allows matching services to be found, whenever these services realise the user goals.

This paper is further organised as follows: Section 2 characterises different types of users based on their knowledge; Section 3 presents DynamiCoS, a framework to support users in the composition process; Section 4 provides an overview of related work; and Section 5 provides our conclusions and directions for future work.

2 Users' Knowledge

An user denotes a person that makes use of some functionality of a system. In the context of our work we consider that an user is a person with limited technical skills. Users may have different characteristics according to their knowledge of the composition process. In this work we show that not all users share the same characteristics. We present a classification of users according to their knowledge in the domain of the service being composed and their technical knowledge on the tooling supporting the service composition process. An user may play two roles in our context, namely to create or execute service compositions. A certain user may play both roles.

2.1 Domain Knowledge

In the process of service composition users need to have some knowledge or idea of the service they want, i.e., what the desired service does, who provides it, etc. We refer to this knowledge as domain knowledge. Domain knowledge is obtained by learning, advertisement, interaction with the service providers, etc. Users may have this type of knowledge when they want to use a service in a given application domain, but often

they have limited knowledge and require some interaction with the elements of the application domain(s) to acquire the knowledge necessary to decide on the service to be requested.

In semantic services, the domain knowledge is explicitly represented in ontologies, which are domain conceptualisations produced by domain experts. This information can then be used to describe the services in the domain, find these services and reason on them.

For example, if a user wants to buy a phone online, he usually has an idea of what type of phone and the market, i.e., phone type/brands, price, stores that sell phones, etc. The application domain has the central concept of telephone store, to which resources (phones, phone stores, etc.) and actions the user can take (search phone, buy phone, etc) are stated. At the end, the user knowledge of the domain will allow him to decide what decision to take, and possibly to compose actions (or services) to realise the desired service.

2.2 Technical Knowledge

Service composition is being used in different domains and applications nowadays, mainly by professional users or developers, which can handle complex tooling and understand the composition process, and details associated with this process. For example, many companies nowadays use Web services technology, and apply WSDL (Web Service Description Language) to describe services in the company, and BPEL (Business Process Execution Language) to compose and coordinate services. However, end-users are not expected to know these technical details. A service composition environment for end-users has to provide a higher level of abstraction to their users, so that users without technical background can make use of this environment.

For example, if the end-user wants to find a phone and then buy it, this whole operation may consist of two services (find and buy services). The supporting tooling has to allow the end-user to find suitable services and then help him with the composition process, by automating this process or by suggesting possible next actions to the users. The supporting tooling may depend on the type of application domain. For example, if the application domain is e-health, where a caregiver decides the sequence of activities a given patient has to perform (e.g., measure blood pressure and if it is to high send a message to the patient's doctor), the caregiver has knowledge on the domain, i.e., he knows the services available in the domain. On the other hand, if an end-user wants to buy a phone, the user may not know the domain, i.e., the supporting environment has to deliver the necessary suggestions and guide the user towards the creation of the service (composition) he wants.

2.3 Types of User

Table 1 defines a possible classification of types of users, based on their domain and technical knowledge.

A *Layman* is a user who does not know the application domain in enough detail, neither has knowledge on the tooling supporting the composition process. A *Domain Expert* is a user who knows the application domain, but does not have knowledge on the

Type of User	Domain Knowledge	Technical Knowledge
Layman	No	No
Domain Expert	Yes	No
Technical Expert	No	Yes
Advanced	Yes	Yes

Table 1. Types of Users

technical tooling that supports the service composition process. A *Technical Expert* is a user who has knowledge on a service composition tool, but does not know the details of the application domain. An *Advanced* user is a user who has technical knowledge on the tooling supporting the composition process, and furthermore knows the application domain. Because this classifications depends on the domain being considered, and sometimes the user uses services from several domains, a user can be, sometimes, of more than one of the types identified. Furthermore, the user may also not have a direct mapping to one of the identified user types, there may exist other types. Our intention is not to identify all the possible user types that may exist. We use the identified user types to motivate that the may exist users with different requirements and characteristics. Based on this we believe that supporting environments should be created according the target population of users they have.

At the moment, DynamiCoS addresses mainly the users that have some knowledge about the application domain, i.e., *Domain Experts* and *Advanced* users. However, we are working in a methodology that will make DynamiCoS adaptable to different types of users, with different types of knowledge.

3 DynamiCoS framework

DynamiCoS[1] (**Dynami**c **Co**mposition of **S**ervices) supports the different phases and stakeholders of the dynamic service composition life-cycle, as discussed in our previous work [3]. Fig. 1 shows the DynamiCoS framework architecture. It also indicates (between parenthesis) the technologies used in the implementation of the different framework components in our prototype platform.

Given the complexity of the dynamic service composition life-cycle and its stakeholders' heterogeneity, we have made the core components of the framework technology-independent. In the framework, a service and a service composition are represented as a tuple and a graph, respectively. A service is represented as a seven-tuple $s =< ID, I, O, P, E, G, NF >$, where ID is the service identifier, I is the set of service inputs, O is the set of service outputs, P is the set of service preconditions, E is the set of service effects, G is the set of goals the service realises, NF is the set of service non-functional properties and constraint values. We assume in this work that services are stateless *request-response*, i.e., they consist of a single activity or operation. A service composition is represented as a directed graph $G = (N, E)$. Graph nodes N represent services, i.e., each node $n^i \in N$ represents a discovered service s^i. A node can have multiple ingoing and outgoing edges. Each graph edge E represents the coupling between a service output/effect (or a user input for the services) and a service

[1] http://dynamicos.sourceforge.net

input/precondition, i.e., $e_{i \to j} = n^i_{O/E} \to n^j_{I/P}$, where $i \neq j$ (a service cannot be coupled with itself).

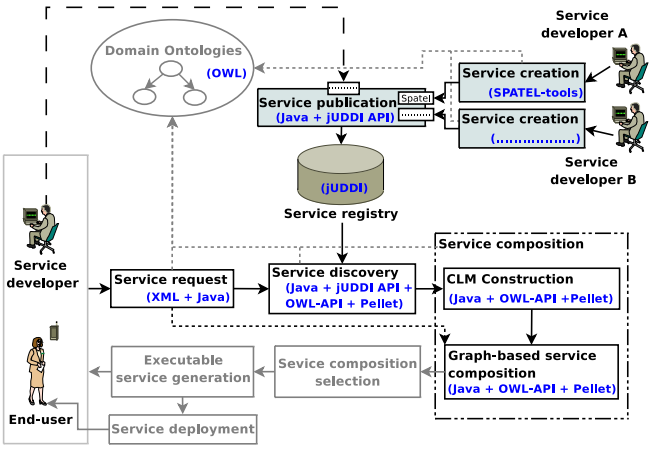

Fig. 1. DynamiCoS framework

3.1 DynamiCoS modules

DynamiCoS consists of the following modules:

Service creation The creation of a new service "from scratch" by a service developer is performed outside the DynamiCoS framework. However, to comply with the capabilities of the DynamiCoS framework, the services have to be semantically described, in terms of inputs, outputs, preconditions, effects ($IOPE$), goals (G) and non-functional properties (NF), using the framework domain ontologies' semantic concepts.

Service publication To support the publication of services described in different languages, the DynamiCoS framework has a two-step publication mechanism. First, there should be an interpreter for each supported service description language. The interpreter reads the service description document and extracts the necessary information for publication (ID, I, O, P, E, G, NF). This makes the service representation in the framework *language-independent*. Second, the extracted service information is published in the service registry using the DynamiCoS generic service publication mechanism. The service registry allows one to publish, store and discover semantic services.

Service request The user interface to define the service request may have different forms, as long as it gathers the user goals and the expected outputs/effects. Optionally it can also gather the inputs/preconditions and non-functional properties that the

service should support. The number of parameters defined in a service request may depend on the type of end-user expertise. If the end-user has technical knowledge and domain knowledge, all the parameters (G, $IOPE$,NF) may be used. However, if the end-user has no technical knowledge, a simpler request may be created, for example, only based on the goals the user wants to achieve and the expected service outputs. The service request parameters are defined as references to semantic concepts available in the framework ontologies. Therefore, the service request consists of a set of semantic annotations (I, O, P, E, G, NF) that describe *declaratively* the desired service properties.

Service discovery The discovery of candidate component services is performed before the service composition phase. Service discovery is based on the service request parameters. The service discovery process consists of querying the service registry for all the services that *semantically* match the defined service request parameters. Since DynamiCoS uses a semantics-based service discovery and composition, it discovers not only exact matches with the service request G and $IOPE$ semantic concepts, but also other services with partial semantic matches, e.g., services with parameters that are semantically subsumed by the service request parameter concepts, i.e., $RequestedConcept \sqsupseteq DiscoveredConcept$.

Service composition To perform service composition, DynamiCoS first processes the set of discovered services and organises them in a so called *Causal Link Matrix (CLM)* [4]. The CLM stores all possible semantic connections, or *causal links*, between the discovered services input and output concepts. CLM rows (Equation 1) represent the discovered services input concepts ($DiscServs_I$). CLM columns (Equation 2) represent service inputs concepts plus requested service outputs ($ServReq_O$).

$$CLM_{rows} = DiscServs_I \tag{1}$$

$$CLM_{colu} = DiscServs_I + ServReq_O \setminus (DiscServs_I \cap ServReq_O) \tag{2}$$

We place a service s in the row i and column j position if the service has an input semantically related with the input on column i of the CLM and an output semantically related with the semantic concept on column j. Furthermore, we store in the matrix the *semantic similarity* of the service output i and the column semantic concept i, and the non-functional properties of the service.

Given a CLM, the composition algorithm has to find a composition of services that fulfils the service request. Our service composition algorithm is graph-based. Algorithm 1 presents the algorithm in pseudo code. The composition process starts by analysing the CLM matrix, checking if it contains the requested IOPE. After this, the CLM is inspected for services that provide the service request outputs. If there are services that provide the service request outputs, the algorithm starts by creating the initial matching nodes, otherwise it stops. If the service request outputs can be provided by the discovered services, the algorithm proceeds with a backwards composition strategy towards the requested service inputs. An *open* (or not yet composed) input of the graph is resolved at each algorithm iteration. The algorithm matches the *open* inputs of the services in the graph with the output concepts of services from the CLM matrix, or column

concepts. If multiple services exist that match a given service input, a new composition graph is created, representing an alternative service composition behaviour. The algorithm finishes when all requested inputs, preconditions and goals from all the alternative service compositions are resolved.

Algorithm 1: Graph Composition Algorithm

Input: $CLM, ServReq$
Result: $ValidComps$

```
   // Variables
1  activeG; // Graph that is active in the algorithm iteration
2  activeN; // Node that is active in the algorithm iteration
3  openG; // Set of open graphs
4  validG; // Set of completed and valid graphs
   // Initialisation
5  if CLM_{rows∪colu} ⊇ ServReq_{I,O} then
       // Create new graph instantiating the initial Node
6      activeG ← createNewGraph(ServReq);
7      createInitialNodes();
8      openG ← activeG;
9  else
       // Discovered services cannot fulfil the service request
10     Stop;
   // Graph construction cycle
11 while |openG| > 0 do
       // Close graph if it matches ServReq_{I,G}
12     if activeG_{I,G} ⊇ ServReq_{I,G} then
13         validG ← activeG;
14         openG ← openG \ activeG;
15         activeG ← openG^0;
16         activeN ← activeG_{openN^0};
17         break; // Goes to next openG
       // Checks CLM for services that match open inputs
18     foreach semCon ∈ activeN_I do
19         if CLM_{colu} ⊇ semCon then
20             activeN ← CLMmatchingNode;
21         else
22             openG ← openG \ activeG;
23             break; // No possible composition, goes to next open graph
       // Check if graph NF props comply with requested NF props
24     if activeG_{NF} ∩ ServReq_{NF} = ∅ then
25         openG ← openG\activeG;
26         break; // If Not, composition is not possible
       // prepare next cycle
27     openN ← openN \ activeN;
```

3.2 Prototype

In our prototype we have used a language for semantic service description called Spatel [5]. Spatel has been developed in the European IST-SPICE project [6], where the DynamiCoS framework was partly developed. The ontologies used in the framework are described in OWL [7]. We used four ontologies: *Goals.owl*, which contains the services' supported goals and also goals that the user can specify in the service request;

NonFunctional.owl, which defines non-functional properties that can be used to describe services; *Core.owl* and *IOTypes.owl*, which are used to describe services' $IOPE$ parameters. However, the framework is general enough to support the use of other ontologies to define other application domains.

For service publication we have implemented a Spatel interpreter. The interpreter imports the Spatel service description by using a Java API generated from the Spatel Ecore model with the Eclipse Modelling Framework (EMF). The service is then published in a UDDI-based service registry that has been extended with semantic support. We use jUDDI [8] as service registry implementation, which is a Java implementation of the UDDI specification for Web services. jUDDI offers an API for publication and discovery of services. We have extended jUDDI with a set of UDDI models (*tModels*) to store the set of semantic annotations (I, O, P, E, G, NF) that describe a service in our framework.

For testing we have created two interfaces to specify the service request: a simple Java-based graphical interface, and a web-based interface. Both allow the specification of the different parameters ($IOPE$, G, NF) of the service request. The information introduced by the end-user is then transformed to an XML-based representation and sent to the composition framework.

Service discovery is performed based on the service request information. The $IOPE$ and G annotations are extracted from the service request and the service registry is queried through the jUDDI API Inquiry function. To discover and reason on semantically related concepts/services we use the OWL-API [9] and Pellet [10].

The CLM matrix is constructed by using the OWL-API, which allows one to handle and perform semantic inference on OWL ontologies by using a semantic reasoner, in our case Pellet [10]. The service composition algorithm is implemented in Java.

In the project website (http://dynamicos.sourceforge.net) we provide more information about the framework and prototype.

3.3 Evaluation and Validation

In a forthcoming publication we will present details on the evaluation and validation of DynamiCoS. The performed evaluation focuses mainly on performance and feasibility of the dynamic service composition process. We have concluded that the several phases of the service composition process can be automated, by using semantic descriptions. However, semantic reasoning is expensive in terms of processing time. Despite that, we consider that such expensive processing times are acceptable to support end-users in the creation of new service compositions on demand at runtime, since in other situations, mainly manual composition is used to tackle this problem, i.e., the user has to spend normally much more time performing a service composition. Furthermore, we expect manual composition, even with intuitive interfaces, to be too complex for most of the types of end-users.

4 Related work

Dynamic service composition has received a lot of attention lately. We refer to [11] for an overview of some existing approaches. Most of these approaches focus on some

phases of the dynamic service composition life-cycle, often the discovery and composition phases. However, some approaches cover most of the phases of the dynamic service composition life-cycle, as, e.g., METEOR-S [12]. METEOR-S provides mechanisms for semantic annotation of existing services, service discovery, and service composition. METEOR-S focuses mainly on design-time creation of service compositions, by developing *templates* that can have dynamic bindings at runtime. Our approach, as many other ones, has been inspired by METEOR-S, but we target an *on demand* runtime service composition creation, to support end-users at runtime. Kona et al. [13] propose an approach oriented to automatic composition of semantic web services. Similarly to DynamiCoS, they propose a graph-based service composition algorithm. The composition process is performed using a *multi-step narrowing algorithm*. The user specifies a service request, or a *query service*, specifying the $IOPE$ for the desired service. The composition problem is then addressed as a discovery problem.

Recently new approaches have emerged to support specifically end-users in the service composition process. These approaches, for example [14] [15] [16], mainly focus on using techniques for mashup, with intuitive graphical representations that allow end-users to create their own services. We argue that these approaches are applicable if the user of the composition environment has some technical knowledge on the composition environment, has a clear idea of the service he wants and knows the application domain. However, if the end-user does not have a clear idea of the service he wants, but only some vague ideas about the goals that he wants to be fulfilled by the service, an approach similar to the one we are proposing may be more appropriate.

5 Final Remarks

In this paper we started by motivating that there are different types of users of service composition systems. Users may have different knowledge of the service composition application domains, and also of the technical tooling supporting the composition process. Based on this observation, we claim that supporting environments have to be created or adapted to match the user and his knowledge and expertise. To support the development of such supporting environments we propose DynamiCoS, which is a framework that supports more dynamic and automatic composition of services. To achieve automation in the composition process, DynamiCoS is based on semantic services. DynamiCoS is neutral with respect to the semantic service description languages used by the service developers. DynamiCoS supports service creation and publication by service developers at design-time, which make services available in the framework; and automatic service composition by end-users at runtime. We have experimented with DynamiCoS, showing that it is capable of providing *real-time* service delivery, and we observed that semantic reasoning is an expensive task in terms of processing time. However, these expenses may be worth paying for, since automated support of users in the creation of their own services can be enabled.

In the future we will investigate how the user properties can be used on the optimisation and personalisation on supporting him in the composition process. We are investigating mechanism to *guide* users through the process of specifying the service composition behaviour. These supporting mechanisms adapt according the user that is

being supported. This should enable the support different types of users, namely the ones identified in Section 2.3, specially the ones without domain knowledge nor technical knowledge. This process will require several interactions between the platform and the user, and a dynamic negotiation to match the user interests with a composition created out of the available services. To validate this we will perform some empirical evaluations of the proposed mechanisms using users in some suitable application scenarios.

References

1. Gartner: Gartner highlights key predictions for it organisations and users in 2008 and beyond (January 2008)
2. Forrester: European mobile forecast: 2008 to 2013 (March 2008)
3. E. Goncalves da Silva and L. Ferreira Pires and M. J. van Sinderen: Defining and prototyping a life-cycle for dynamic service composition. In: International Workshop on Architectures, Concepts and Technologies for Service Oriented Computing, Portugal (July 2008) 79–90
4. Lécué, F., Léger, A.: A formal model for semantic web service composition. In: International Semantic Web Conference. LNCS, vol. 4273 (2006) 385–398
5. 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)
6. 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: International Workshop on Applications and Services in Wireless Networks. (2006)
7. Smith, M.K., McGuiness, D., Volz, R., Welty, C.: Web Ontology Language (OWL) guide, version 1.0. W3C. (2002)
8. Apache: Apache juddi. http://ws.apache.org/juddi/
9. Bechhofer, S., Volz, R., Lord, P.: Cooking the semantic web with the OWL-API. In: International Semantic Web Conference. (October 2003) 659–675
10. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: a practical OWL-DL reasoner. Web Semantics: Science, Services and Agents on the World Wide Web **5**(2) (June 2007) 51–53
11. Rao, J., Su, X.: A survey of automated web service composition methods. In: International Workshop on Semantic Web Services and Web Process Composition. (2005) 43–54
12. Verma, K., Gomadam, K., Sheth, A.P., Miller, J.A., Wu, Z.: The meteor-s approach for configuring and executing dynamic web processes. Technical report, University of Georgia (June 2005)
13. Kona, S., Bansal, A., Gupta, G.: Automatic composition of semantic web services. In: International Conference on Web Services. (2007) 150–158
14. Liu, X., Huang, G., Mei, H.: A user-oriented approach to automated service composition. In: Web Services, 2008. ICWS '08. IEEE International Conference on. (Sept. 2008) 773–776
15. Ro, A., Xia, L.S.Y., Paik, H.Y., Chon, C.H.: Bill organiser portal: A case study on end-user composition. In: WISE '08: Proceedings of the 2008 international workshops on Web Information Systems Engineering, Berlin, Heidelberg, Springer-Verlag (2008) 152–161
16. Nestler, T.: Towards a mashup-driven end-user programming of SOA-based applications. In: iiWAS '08: Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services, New York, NY, USA, ACM (2008) 551–554