Optimized dynamic semantic composition of services

Conference Paper · January 2008

DOI: 10.1145/1363686.1364230 · Source: DBLP

Sorin M. Iacob
Thales Group
27 PUBLICATIONS **196** CITATIONS
SEE PROFILE

João Paulo A. Almeida
Universidade Federal do Espírito Santo
148 PUBLICATIONS **1,549** CITATIONS
SEE PROFILE

Maria-Eugenia Iacob
University of Twente
114 PUBLICATIONS **1,276** CITATIONS
SEE PROFILE

Optimized Dynamic Semantic Composition of Services

Sorin M. Iacob
Thales Research and Technology, D-CIS Lab
PO Box 90, 2600 AB, Delft, The Netherlands
sorin.iacob@icis.decis.nl

João Paulo A. Almeida
Departamento de Informática, CT-VII, UFES Av. Fernando Ferrari, s/n 29060-970 Vitória, Espírito Santo, Brazil
jpalmeida@ieee.org

Maria E. Iacob
University of Twente
P.O. Box 217, 7500 AE
Enschede, The Netherlands
m.e.iacob@utwente.nl

ABSTRACT

This paper proposes a design for the optimized dynamic (re)composition of services that supports various user requests and accounts for changes in user's context. The composition mechanism relies on semantic descriptions of services' functionality to compose services at runtime. The design-time component of our solution is a domain ontology that is used by the composition service for deriving different composition possibilities and for finding the appropriate similarity relations between different semantic constructs representing the user request and service descriptions. The composition mechanism combines a greedy optimization for the initial selection of candidate constituent services and a global optimization for reaching the final composition.

Keywords

Services, service composition, domain ontology, semantic similarity, context-awareness.

1. INTRODUCTION

Service-oriented architectures improve design efficiency by promoting the reuse of well-defined and coherent functionality which is made available through services. In these architectures, designers can achieve complex application behaviors by appropriately combining the functionality of several services, in what is called a service composition. End-user services often consist of a more or less complex composition of services, each of which may be defined by composing other services.

Since individual services may be heterogeneous in terms of technology, performance, usage conditions, etc., the process of discovering, connecting and coordinating individual services in a service composition is potentially time-consuming and error-prone. The challenges for service composition are aggravated in the case of ubiquitous and context-aware services, for which availability, performance and characteristics of the component services are (highly) dynamic and sensitive to the context of operation.

This paper proposes a design for an optimized runtime composition mechanism for services. We address the problem of finding the set of functions and their logical sequencing that is required for achieving a composition, automatically, i.e., without designer intervention.

Our approach consists of matching a semantic description of a desired service with a set of semantic descriptions of available (atomic) services. The matching entails the calculation of a so-called semantic similarity measure for each of the available services and the subsequent prioritization of these services depending on their similarity with the search phrase. The design-time component of our solution is a domain ontology that is used by the composition service for deriving different composition possibilities and for finding the appropriate similarity relations between different semantic constructs representing the user request and service descriptions.

The composition mechanisms presented is generic with respect to the particular domain ontologies used as well as independent of the services registered in the repository. Further, it is also independent of the composition operators used to compose services. To show that, we apply the approach to two different scenarios with different kinds of services compositions: a scenario which consists of the composition of multimedia streaming services, and a scenario which consists of the composition of (web) services for location-awareness.

The composition mechanism combines a greedy optimization for the initial selection of candidate constituent services and a global optimization for reaching the final composition. This optimization takes into account non-functional properties of the constituent services expressed in terms of a utility function. Our approach is illustrated by means of two examples. First a multimedia service scenario is used to motivate the necessity of optimization. Then a simplified example of a distance service is used to prove the feasibility of its implementation.

This paper is further structured as follows: section 2 presents some background on dynamic services composition; section 3 presents the multimedia streaming scenario, which motivates a runtime process that is capable of adapting a service configuration to changes in the user's or systems' contexts. Section 4 presents the architecture of the dynamic composition service and introduces the composition process. Section 5 presents a prototype to illustrate the application of our approach to a second example with the composition of location-based services. Section 6 discusses some related work. Finally, section 7 presents some conclusions and outlines topics for future work.

2. BACKGROUND

In design time service composition (e.g., [1]), the outcome is a composition description (or composition graph) that together with service descriptions (of the component services) can be executed by a runtime environment. In this case, the runtime environment is primarily responsible for executing a (composite) service, routing messages between component services, handling events, faults and exceptions and monitoring overall service execution.

Given that some component services may not be available at all times, the runtime environment should also be able to select individual services in a dynamic manner [12]. Typically, services are selected from a pool of candidate services, which is determined at design time. The actual service to be invoked is then obtained by evaluating a given query over a registry, or obtained from a brokering service. The choice for a particular service over similar ones is based on a set of (non-functional) requirements and user preferences expressed in the composite service description. Finally, a binding mechanism is used to homogenize the (possibly heterogeneous) inter-faces of component services.

When the service composition graph is known, the only runtime process that remains to perform is service instance selection. The criteria for choosing a particular instance of a given service are therefore exclusively related to the non-functional properties of the service, all of which can be combined in a "cost" value. The optimization process is reduced thus to the minimization of a real-valued cost function. In case of dynamic service composition the service selection also needs to take into account functional properties. However, since the overall functionality of the composite service is supposed to be the same regardless the particular choice of elementary functionality, non-functional properties will play the most important role here. Nevertheless, the selection problem is much more complex than in the previous case, since there may be different service graphs that achieve the same functionality.

Dynamic service composition as presented in this paper attempts to select and compose, at runtime, those services that, in the right combination will provide the functionality that is required for obtaining the desired behavior. In order to automate the composition process, it is necessary to enrich the description of services beyond the syntactical description of interfaces and parameters. In this paper, we rely on the use of domain ontologies [13, 15], which not only allow for powerful discovery techniques, but also enable the design of mechanisms for dynamic composition [8, 17].

3. SCENARIO: DYNAMIC COMPOSITION OF MULTIMEDIA SERVICES

To highlight several aspects of service composition we present an example of a multimedia streaming service that delivers the latest edition of BBC World News, in the best possible quality (audio, video, text) to mobile users. The composite streaming service adapts dynamically in response to changes in the user's or system's context, by adding and/or removing the video stream from a multimedia transmission, or by including additional media encoders or processors.

The main components of the service (see Figure 1) are the media sources (M, e.g. audio, video, text), a set of processors (P, e.g. sub-samplers, color space converters, text-to-speech converters, A/V splitters, video key-frame extractors, etc), a set of encoders (E, e.g. mp3, H264.1, etc.), a set of transport services (T, e.g. rtsp, http, ftp), context sources (Xu – for the user context, Xn for the network context, Xs for the service context), and the end-user device (Ud). Note that the service graph may change dynamically since the choice of services from each category is not exclusive. In general, a composite service can be seen as a graph of media sources, processors, encoders, and transport components as illustrated in Figure 1.

In this example, a particular service configuration results as a function of user's request and different context parameters. We describe the different service configurations for different situations based on the user's context:

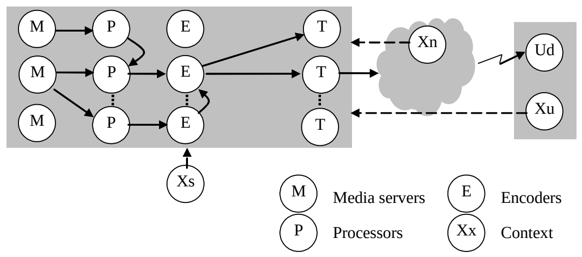


Figure 2: Adaptive multimedia streaming service

While the user is driving on the highway (fact that is sensed by Xu) the service delivers the BBC World News in audio-only mode (for safety reasons). This means that from the audio-video file the audio will be extracted with an a/v splitter. Since the edition includes some text-only news, a text-to-speech converter is also used. Xn senses that the UMTS connection used by Ud allows for a maximum of 64 kbps, so the first audio sequence is encoded mp3 at 64 kbps, and then fed to an RTSP server. Since the computer generated audio requires a much lower bandwidth (16kbps), this is put in a separate file and placed on an http server. The links to the two audio files (rtsp and http) are sent through the over-the-air (OTA) push protocol to the Ud, which then automatically connects to the two servers and plays the audio files (see Figure 2).

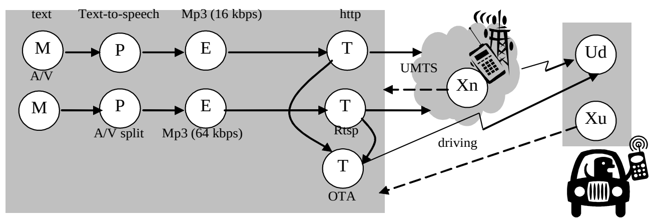


Figure 3: Service configuration (user is driving)

The user stops at his destination and alights from his car. This has the effect of changing his activity to "walking" (see Figure 3), which is signaled by Xu. As a result, the service adjusts to also deliver the video part of the transmission.

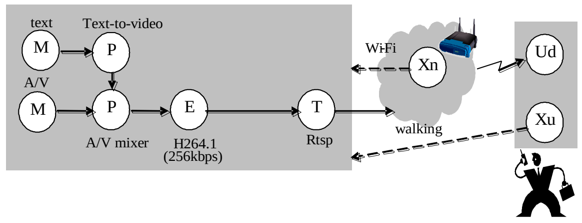


Figure 1: Service configuration (user is walking)

The text-only news is now inserted as subtitles. Xn senses that Ud connected to a publicly accessible wireless LAN, so a higher bit-rate is now available for the real-time multimedia transmission.

Additional changes in the user, system, or network context can be envisaged that would require completely different service configurations. The only stable given in this dynamic configuration (composition) scenario is the user request, formulated as a semantic construct to represent the "latest BBC news at the best possible quality".

4. OPTIMIZED DYNAMIC SERVICE COMPOSITION

4.1 Assumptions

For our solution we assume that user requests, context parameters, and service descriptions are expressed as semantic constructs within a given domain ontology O(***C***, ***R***), where ***C*** is the set of concepts (or vocabulary), and ***R*** the set of relations (or ontological axioms). The actual design of such a domain ontology is beyond the scope of the present work. According to this assumption, the relation between service descriptions (or service requests for that matter) and ontology concepts can be sketched graphically as in Figure 4. In principle, a semantic construct for specifying a (composite) service is an aggregation of concepts, according to some arbitrary composition operators (denoted hereafter ⊕).

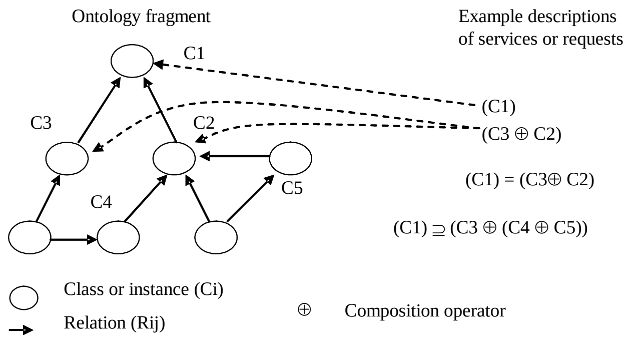


Figure 4: Service descriptions as semantic constructs in terms of an ontology fragment.

The existing relations in the ontology should allow certain transformations on these semantic constructs and the definition of a semantic similarity measure between different constructs. Based on this measure, equivalence relations between semantic constructs can be defined, such as: (C1) = (C3 ⊕ C2) or (C1) ⊇ (C3 ⊕ (C4 ⊕ C5)) in Figure 4.

4.2 Architecture for discovery and composition

The composition service is responsible for dynamically constructing composite services based on client requests. It can also be made responsible for recomposing the service to keep it "matched" with what the client initially requested if, for example, some of the service's constituent services suddenly become unavailable. In this case, the composition service proposes (re)compositions of a particular service to the requesting client. The client can then decide which (if any) of the proposed composite services it would like to bind to. Alternatively, the client could also delegate binding decisions to the composition service, which would require informing the composition service of its 'binding policy'.

Services (including context sources) advertise their service description to the discovery service when they wish to make themselves available to potential clients. During discovery, a client obtains information about the existence of services, their operations, parameters, and their semantics. The discovery and composition processes are inseparable in our architecture, such that there is a continuous interaction between the Service Composition Service (SCS), and the Service Discovery Service (SDS), as shown in Figure 5. A request for some complex functionality is generated, as explained earlier, by the end-user application (Ud), in terms of the selected domain ontology. The SCS processes this request in order to find possible equivalent constructs in terms of the same vocabulary, and then generates a set of requests for (atomic) services towards the SDS. The services returned by SDS are then composed and presented to the end-user.

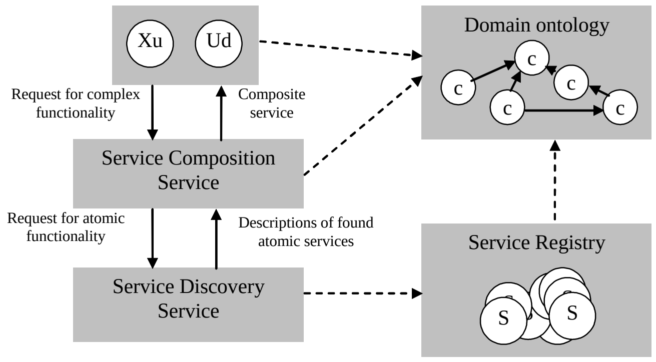

Xu
Ud
Domain ontology
c
Request for complex functionality
Composite service
Service Composition Service
Request for atomic functionality
Descriptions of found atomic services
Service Registry
Service Discovery Service
S

Figure 5: Architecture for discovery and composition

4.3 Composition mechanism

Given that the functionality of a (composite) service and the query for that service are expressed in terms of the same ontology, it is possible to estimate the "gap" between the functionality of the requested (composite) service and that offered by any of the available (constituent) services [14]. Dynamic service composition can then be defined, in an abstract way, as an iterative process by which new functions are added, at runtime, to an existing aggregation until a certain error threshold, measuring the gap between the desired and available functionality, is reached. After each iteration, the newly composed service is evaluated, and a utility measure (u) is calculated as a function of the semantic similarity between the required and achieved functionality, and some non-functional constraints (e.g. response time, or some generic costs). The context of the client can influence the set of constituent services that the SCS selects for a particular composition as well as the way in which they are arranged in the composite service's service graph. Once a service has been constructed, context changes might require the SCS to select new constituent services and use them to recompose the composite service.

To explain the composition mechanism, we begin with the assumption that for any service request a set of (constituent) services can be retrieved that match, to a certain degree, subsets of the query, as explained further. The SCS first attempts to find a composite service that matches the original query. If the utility measure indicates a poor match, the SCS "breaks" the original query in smaller fragments by replacing individual terms with equivalent (composite) constructs derived from the domain ontology. The SDS then tries to find these simpler services, and the SDS selects from the returned set those for which the utility measure is maximized. This process (see Figure 6) is repeated until the utility measure reaches a prescribed value.

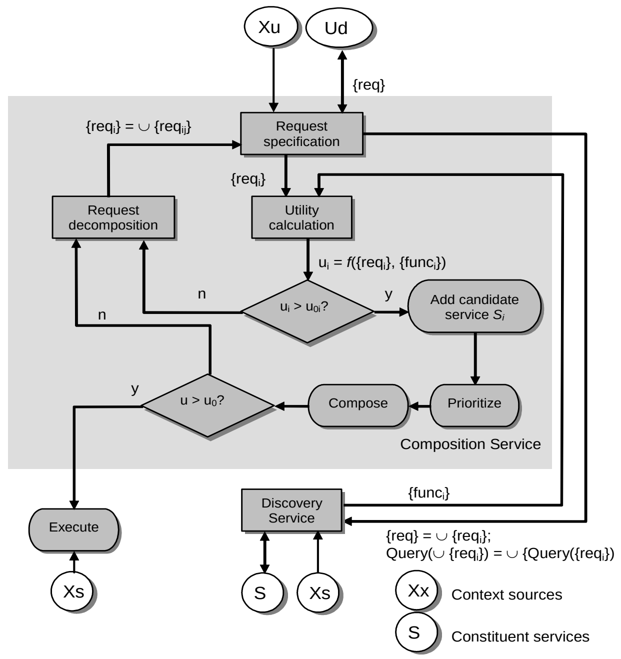


Figure 6: Process model for iterative service composition

4.4 Optimization of the dynamic composition

The composition mechanism combines a greedy optimization for the initial selection of candidate constituent services and a global optimization for reaching the final composition.

In the greedy step, all discoverable constituent services that match one or more search terms ("req_i" in Figure 6) are selected, and then the semantic similarity is determined between each service function description ("$func_i$") and the request. The matching descriptions are then prioritized according to their similarity (i.e., the number of matching terms). The search terms for which no appropriate constituent service was found are further decomposed into finer-granularity terms, according to the domain ontology used. In the global optimization step a composition is attempted from the highest-ranking constituent services with non-overlapping descriptions, and then the utility measure is computed. If the utility measure is above the required value, the composition is considered valid and the composite service is returned. If not, the candidate constituent services selected in the greedy step are discarded, the original request is further refined, and a new query is generated for finer-granularity services. The algorithm can be summarized in the following steps:

1. Given the request *req* and the threshold similarity s_0 find the set M of all services having the semantic description *func*, such that $s(func, req) \geq s_0$;
2. If $M \neq \varnothing$ calculate the utility measure for all services $func_i$, i = 1,…, m in M. Go to step 5.
3. Else:
   a. write $req = req_1 \oplus req_2 \oplus \ldots \oplus req_n$. For j = 1 to n repeat steps 1 and 2 to obtain the sets M_j;
   b. With all services $func_{ij}$ in M_j build the semantic constructs $func_i = func_{1i} \oplus func_{2i} \oplus \ldots \oplus func_{ni}$
   c. For all $func_i$, i = 1,…, m calculate the utility measures u_i.
4. Select the service $func_0$ for which is $u_0 = \max \{u_i\}$.

The above heuristic combines the greedy selection in steps 1 and 3b with the global optimization in step 4. The highest complexity is $O(n^2)$ in step 3b.

During execution, context parameters (including the context of the client and that of candidate constituent services) are monitored, and the utility measure is computed at regular intervals. The composition service re-invokes the global optimization step when the utility drops due to context changes (which may alter either the service request or non-functional service parameters).

4.5 Semantic similarity and utility measures

In [14], the similarity between two semantic constructs is defined as the angle of two vectors that characterize the hyponymy relations between the concepts used in the two constructs under comparison. In principle, such a similarity measure can be used in the SCS as well, and other graph similarity measures proposed in the literature (e.g., [6]) are also suitable. In general, the similarity between semantic constructs of concepts in an ontology fragment can be defined as a function that is proportional with the inverse of the graph distance in a directed (acyclic) graph (see e.g. [5] for more details regarding distance in directed graphs).

Let us consider a domain ontology $O(\boldsymbol{C}, \boldsymbol{R})$. For the purpose of this work however, it suffices to assume that one such suitable similarity measure between constructs $K_i = (C_{i1} \oplus C_{i2} \oplus \ldots \oplus C_{im})$ and $K_j = (C_{j1} \oplus C_{j2} \oplus \ldots \oplus C_{jn})$ is defined as a real valued function $s: \boldsymbol{C}^m \times \boldsymbol{C}^n \rightarrow [0,1]$, with $s(K_i, K_j)$ having the following properties:

i) $s(K_i, K_i) = 1$;

ii) $\forall i \neq j$, $s(K_i, K_j) \in (0, 1) \Leftrightarrow$ □ $\exists R_{ij} \in \boldsymbol{R}$. Otherwise $s(K_i, K_j) = 0$;

iii) $s(K, K_1 \oplus K_2 \oplus \ldots \oplus K_n) = \alpha_1 \cdot s(K, K_1) + \alpha_2 \cdot s(K, K_2) + \ldots + \alpha_n \cdot s(K, K_n)$, where $\alpha_i \in [0,1]$, $\forall i \in \{1,\ldots,n\}$ are normalization coefficients and $\alpha_1 + \alpha_2 + \ldots + \alpha_n = 1$.

If we further assume that the non-functional parameters of candidate services can be described through a scalar, the utility function can then be defined as:

$u: \boldsymbol{C}^m \times \boldsymbol{C}^n \rightarrow [0,1]$, $u(K_{req}, K_{func}) = \alpha_{nonfunc} \cdot s(K_{req}, K_{func})$, with $\alpha_{nonfunc} \in [0,1]$.

The choice for a scalar representation of the non-functional parameters is arbitrary, its only purpose being that of further qualifying candidate services. In practice, the non-functional parameters may be given more meaningful representations.

5. PROTOTYPE FOR COMPOSITION OF LOCATION-BASED SERVICES

In order to show the feasibility of our approach we have implemented a prototype of the composition service, which considers an example ontology in the domain of the location of users, as depicted in Figure 7 (represented here in UML). This model includes the following entities: User, Home, Office. A User is associated with his/her home and his/for office. The user's location is modeled by GeoLocation (which includes GPS coordinates), the user's current GSM cell is modeled by GSMCell and some additional calendar information is modeled by a Schedule associated with a User.

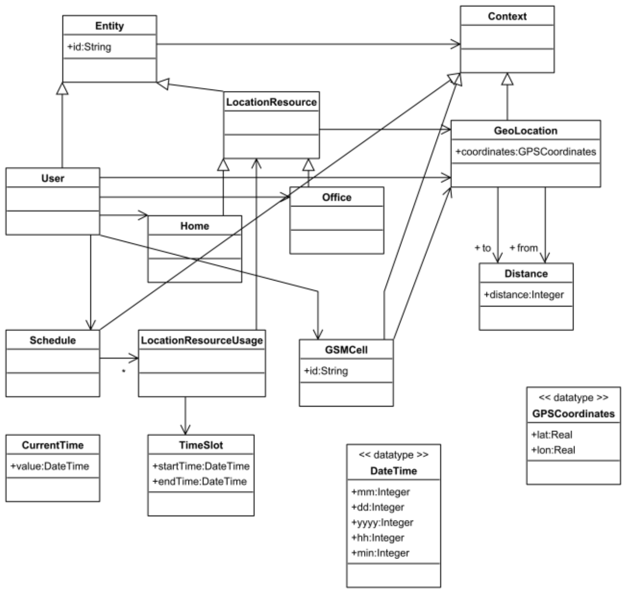


Figure 7 Ontology fragment used in prototype

The main elements of the implementation are the ontology fragment, the service composition service, the service discovery service (including a services repository) and an ontology browser as depicted in Figure 8.

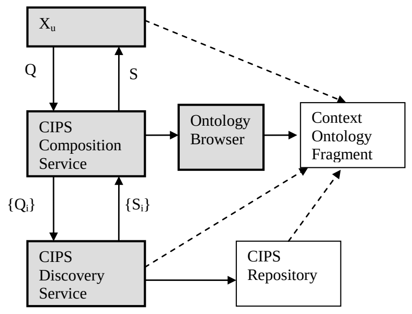


Figure 8. CIPS system components

The following assumptions are made to restrict the possible design choices of Context Information Provisioning Service (CIPS) prototype:

- There are two composition laws for generating semantic constructs, which are referred to as “sequential” and “parallel”. Sequential composition laws can be regarded as unary operations (assignments, mappings, etc.), while the parallel compositions are n-ary operations.
- The semantic (de-)composition laws result from the definition of the domain ontology, and constrain the service composition graph as follows: sequential composition forces the corresponding sequential service invocations and parallel composition requires independent invocation of the corresponding component service
- The similarity measure is defined as follows:

  - Two different specializations c_i, c_j of a class c are equivalent, i.e., have a similarity s_{ij} = 1, unless explicitly stated otherwise
  - For n different classes c_i, i = 1,…,n, which have an association relation (e.g. “part of”) with one and the same class c, a similarity measure $s(c_i, c_j)$ cannot be defined directly. The following relation, however, holds: $s(c, \oplus c_i) = \sum_i s(c, c_i) = 1$, and for practical purposes can be considered that $s(c, c_i) = 1/n$.
- The utility function results from the scalar multiplication of the semantic similarity measure between a (sub-)query and a (composed) service signature, and a weighting factor calculated from the non-functional aspects of service execution.

In this prototype, the services published in the repository offer a single operation, the purpose of which is to supply information based on certain input parameters (a query operation).

Both a query and a service are described by the tuple:

```
<output_param_list, [input_param_list]>
```

Input and output parameters refer to elements defined in the ontology. We have registered the following services for illustration purposes:

```
getUserLocation(user: User): GeoLocation;
getDistance(geoLocation1: GeoLocation,
   geoLocation2: GeoLocation): Distance;
getUserCell(user: User): GSMCell;
geoCode(cell: GSMCell): GeoLocation;
getLikelyLocationResource(user: User, currentTime: CurrentTime):
LocationResource;
getLocation(locationResource: LocationResource): GeoLocation;
getUserHome(user: User): Home;
getHomeLocation(home: Home): GeoLocation;
getUserOffice(user: User): Office;
getOfficeLocation(home: Office): GeoLocation;
```

5.1 Composition service

The Composer interface (shown here in Java) provides an operation that, given a ServiceQuery, returns a list of Compositions (Composition is a class generated from the Composition metamodel depicted in Figure 9, according to the model-driven implementation which is discussed in section 5.2):

```
public List<Composition>
compose(ServiceQuery s);
```

The compose operation is defined recursively, with two operators applied to decompose a query: a parallel operator, and a sequential operator:

```
compose ( service query ) : compositions
search repository
if (direct match found)
     return service in composition
else if (no services found)
     search the ontology for classes that are
     related to the output types in the
     service query

     if (no related classes found)
            return empty composition
     else
            // attempt parallel composition
            break the query into a subquery for
```

```
each input type
for each subquery
     call compose recursively
if (found)
     return resulting compositions
     in parallel

// attempt sequential composition
break the query into a subquery from
each related class
for each subquery
     call compose recursively
return resulting compositions in
sequence
```

The parallel operator involves the independent invocation of services in parallel, with the establishment of multiple output parameters. The sequential operator consists of the concatenation of invocations in time, with the assignment of the values of output parameters to the input parameters of subsequent invocations.

5.2 Model-driven implementation

The service composition which is produced by the composition service is represented by using an instance of the Ecore (EMF 2.2.0 [7]) metamodel defined in Figure 9. This metamodel is used to generate an Eclipse (3.2) [7] plug-in which provides model representation, serialization and visualization capabilities. A services composition consists of a number of invocations. Invocations can be classified into RecursiveInvocation (in which case a composition of services is invoked), a ServiceInvocation (in which case a specific "atomic" service is invoked) and an UnboundInvocation (which is used to represent invocations which are not yet assigned to a specific service or service composition.)

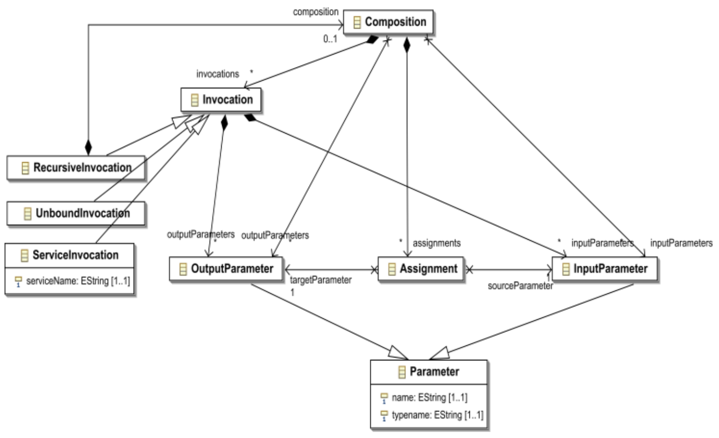
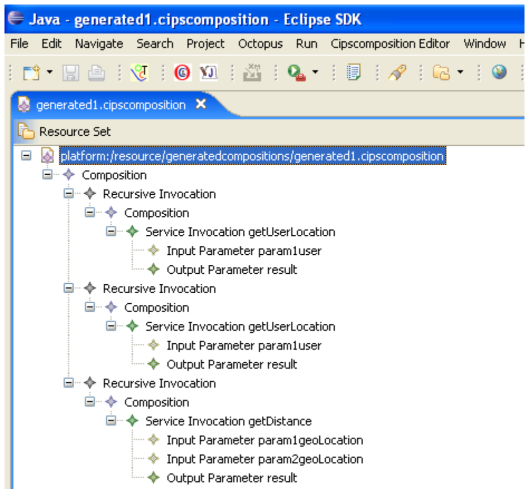


Figure 9. Service composition metamodel

5.3 Results of the composition process

Figures 10 and 11 show the results of the composition process as visualized in our Eclipse plug-in. These compositions compute the distance between two users in different ways, and are generated automatically by the composer to satisfy the same query (i.e., a query that requires a composite service with two input parameters of type User and one output parameter of type Distance).

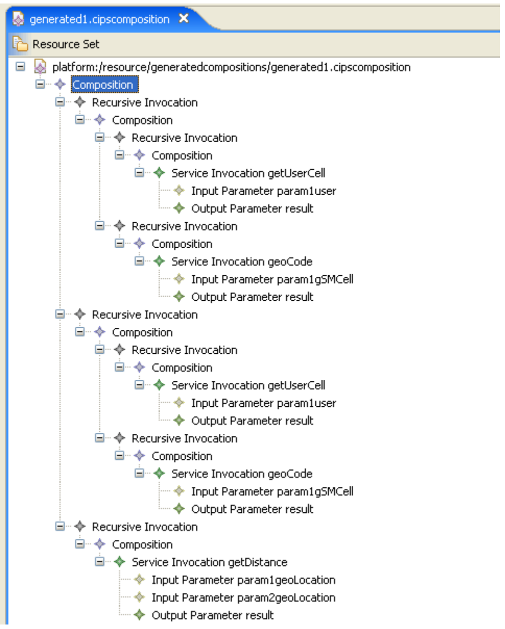
The composition in Figure 10 uses both parallel and sequential decomposition: parallel decomposition is used to invoke getUserLocation for each of the two users. Then getDistance is invoked with the location results obtained.


Java - generated1.cipscomposition - Eclipse SDK
File Edit Navigate Search Project Octopus Run Cipscomposition Editor Window
generated1.cipscomposition
Resource Set

```
platform:/resource/generatedcompositions/generated1.cipscomposition
  Composition
    Recursive Invocation
      Composition
        Service Invocation getUserLocation
          Input Parameter param1user
          Output Parameter result
    Recursive Invocation
      Composition
        Service Invocation getUserLocation
          Input Parameter param1user
          Output Parameter result
    Recursive Invocation
      Composition
        Service Invocation getDistance
          Input Parameter param1geoLocation
          Input Parameter param2geoLocation
          Output Parameter result
```

Figure 10 Automatically generated composition that provides the distance between two users

Figure 11 shows a composition that obtains the location of the users indirectly, by first determining the users' GSM cells (getUserCell) and, then, sequentially, using each cell identifier to determine a user's location.



```
platform:/resource/generatedcompositions/generated1.cipscomposition
  Composition
    Recursive Invocation
      Composition
        Recursive Invocation
          Composition
            Service Invocation getUserCell
              Input Parameter param1user
              Output Parameter result
        Recursive Invocation
          Composition
            Service Invocation geoCode
              Input Parameter param1gSMCell
              Output Parameter result
    Recursive Invocation
      Composition
        Recursive Invocation
          Composition
            Service Invocation getUserCell
              Input Parameter param1user
              Output Parameter result
        Recursive Invocation
          Composition
            Service Invocation geoCode
              Input Parameter param1gSMCell
              Output Parameter result
    Recursive Invocation
      Composition
        Service Invocation getDistance
          Input Parameter param1geoLocation
          Input Parameter param2geoLocation
          Output Parameter result
```

Figure 11 GSM cell identifier is used in the composition for deriving the user's location

6. RELATED WORK

When two or more services need to work together in order to achieve a complex processing, some additional information about their properties is required besides the syntactic and semantic descriptions of inputs, outputs, and transformation rules. This information includes: the set of functions required for the complex processing; the proper sequencing of operations; how to match the outputs of one service with the inputs of a subsequent one; what are the temporal relationships between the execution of the different services in a sequence of operations, etc. Depending on the emphasis put on these problems, several approaches have been proposed for solving the service composition problem, based on, e.g. finite state automata [3], planning [17, 18], agents [4], workflow analysis [9], etc. An extensive overview of the fundamental assumptions and concepts underlying current work on service composition, and of the key results in the area is presented in [10]. Fujii and Suda [8] proposed a similar approach to dynamic service composition based on the semantic similarity between queries and service descriptions, but assumed a static (and unique) graph representation of the query. In our approach, the composition process ensures a minimal number of services through the iterative decomposition of the input query, and accounts for dynamic changes in the user's environment.

Another approach to dynamic service composition is that of Zeng et al. [19], which selects particular service in a composite service composition model, based on the optimization of some quality measures derived from the nonfunctional properties of the component services. Nevertheless they assume a static composition graph, the optimization problem being thus less complex. Consequently, our approach may be seen as complementary to this one since we do not make explicit the dependency on non-functional parameters, but instead optimize the functionality of the composite service based on the semantic matching of service descriptions.

For service discovery, context information has been used to optimize the discovery process [2], for instance by only considering those services that are in the client's proximity [16].

7. DISCUSSIONS AND CONCLUSIONS

The proposed solution provides an optimal (re-) composition of services at runtime, based on various user requests, and taking into account rapid changes in user's context. The composition mechanism relies on the semantic description of services' functionality, and ignores the problems related to orchestration and choreography. The design-time component of this mechanism is the domain ontology that is used by the composition service to find the appropriate similarity relations between different semantic constructs representing the user request and service descriptions. The implementation has been realized in a prototype using model-driven development techniques. A complete prototype description is available in [11].

We have assumed that all necessary information exchange between services refers to a shared ontology fragment. We have considered a simplified ontology fragment in the demonstrator. Future work should investigate the scalability of the approach, which requires the availability of large ontologies and service repositories.

8. ACKNOWLEDGMENTS

This work is part of the Freeband A-MUSE project (http://a-muse.freeband.nl), which is sponsored by the Dutch government under contract BSIK 03025.

9. REFERENCES

[1] J.P.A. Almeida, M.E. Iacob, H. Jonkers, D. Quartel, "Model-Driven Development of Context-Aware Services,", Proc. of the 6th IFIP WG 6.1 Int'l Conf. Distributed Applications and Interoperable Systems (DAIS) 2006, LNCS, Vol. 4025, 2006, pp. 213 – 227.

[2] K. Arabshian, H. Schulzrinne, D. Trossen, D. Pavel, "GloServ: Global Service Discovery using the OWL Web Ontology Language", IEEE Int'l Workshop on Intelligent Environments, Colchester, England, June 2005.

[3] S. Ben Mokhtar, J. Liu, N. Georgantas, V. Issarny, "QoS-aware Dynamic Service Composition in Ambient Intelligence Environments", Proceedings of ASE'05, 2005.

[4] M. Berger, M. Bouzid, et al., "An Approach to Agent-Based Service Composition and Its Application to Mobile Business Processes", IEEE Trans. on Mobile Computing, July-September, (Vol. 2, No. 3), pp. 197-206, 2003.

[5] F. Buckley, F. Harary, Distance in Graphs, Addison Wesley, 1990.

[6] G. Chartrand, G., Kubicki, and M. Schultz, "Graph similarity and distance in graphs", Aequationes Mathematicae, Vol. 55 (1-2), 1998, pp. 129—145.

[7] Eclipse Metamodeling Framework, www.eclipse.org/emf/

[8] K. Fujii, T. Suda, "Dynamic Service Composition Using Semantic Information", 2nd ACM Int'l Conf. on Service Oriented Computing (ICSOC '04), Nov. 2004.

[9] R. Hamadi, and B. Benatallah, "A Petri net-based model for web service composition", Proc. 14th Australasian Database Conf. on Database Technologies 2003, Vol. 17, pp. 191-200.

[10] R. Hull, and J. Su, "Tools for Composite Web Services: A Short Overview", ACM SIGMOD Record, Vol 34, No 2, 2005.

[11] S.M. Iacob, J.P.A. Almeida, "Automatic Composition of Context Information Provisioning Services", A-MUSE D3.14, Telematica Instituut, The Netherlands, 2006.

[12] D. Karastoyanova, A. Houspanossian, M. Cilia, F. Leymann, A. Buchmann, "Extending BPEL for Run Time Adaptability", Proc. 9th IEEE Int'l EDOC Enterprise Computing Conf. (EDOC'05), 2005, pp. 15-26..

[13] Y. Kalfoglou, M. Schorlemmer, "Information-Flow-based Ontology Mapping", Proc. 1st Int. Conf. on Ontologies, Databases and Application of Semantics , 2002.

[14] V. Oleshchuk, A. Pedersen, "Ontology Based Semantic Similarity Comparison of Documents", 14th Int'l Workshop Database and Expert Systems Applications, 2003.

[15] The OWL Services Coalition, OWL-S: Semantic Markup for Web Services, 2003; www.daml.org/services/owl-s/

[16] O. Ratsimor, V. Korolev, A. Joshi, and T. Finin, "Agents2Go: An Infrastructure for Location-Dependent Service Discovery in the Mobile Electronic Commerce Environment", ACM Mobile Commerce Workshop, 2001.

[17] B. Srivastava, "Automatic web services composition using planning", Proc. KBCS, 2002, pp. 467-477.

[18] M. Vukovic, and P. Robinson, "Adaptive, planning-based, Web service composition for context awareness", Int'l Conference on Pervasive Computing, Vienna, April 2004.

[19] L. Zeng, B. Benatallah, A.H.H. Ngu, M. Dumas, J. Kalagnanam, H. Chang, Qos-Aware Middleware for Web Service Composition, IEEE transaction on Software Engineering, Vol. 30, No. 5, May 2004, p. 311-327.