HAL open science

Service Composition based on Natural Language Requests

Marcel Cremene, Jean-Yves Tigli, Stéphane Lavirotte, Florin-Claudiu Pop, Michel Riveill, Gaëtan Rey

To cite this version:

Marcel Cremene, Jean-Yves Tigli, Stéphane Lavirotte, Florin-Claudiu Pop, Michel Riveill, et al.. Service Composition based on Natural Language Requests. International Conference on Service Computing, IEEE Computer Society, Sep 2009, Bangalore, India. hal-00481765

HAL Id: hal-00481765

https://hal.science/hal-00481765

Submitted on 7 May 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution - NonCommercial - NoDerivatives 4.0 International License

Service Composition based on Natural Language Requests

Marcel Cremene*, Jean-Yves Tigli†, Stéphane Lavirotte†, Florin-Claudiu Pop*, Michel Riveill† and Gaëtan Rey†
* *Communications dept., Technical University of Cluj-Napoca, Romania*
Email: cremene@com.utcluj.ro, florin.pop@com.utcluj.ro
†*I3S Lab., "Rainbow" team, University of Nice Sophia-Antipolis, France*
Email: tigli@polytech.unice.fr, lavirott@unice.fr, riveill@unice.fr, gaetan.rey@unice.fr

***Abstract*—The easiest way for a user to express his needs regarding a desired service is to use natural language. The main issues come from the fact that the natural language is incomplete and ambiguous, while the service composition process should lead to valid services. In this paper we propose a natural language service assemblage method based on composition templates (patterns). The use of templates assures that the composition result is always valid. The proposed system, called NLSC (Natural Language Service Composer), was implemented on the top of a service-oriented middleware called WComp and tested in an intelligent home environment.**

***Keywords*-natural language; service composition; patterns;**

I. Introduction

Service composition means to create new services by composing existent ones. In the classical approaches, this composition is done by a human expert because the composition task requires an understanding about the service semantics.

Automatic service composition means to replace the human expert by a software application that will create/compose the new service. In this paper we are interested in user-driven automatic service composition. The easiest way for a (non-expert) user to express his needs regarding a desired service is to use natural language. Thus, instead of trying to predict all possible services, which is practically impossible, we are trying to offer to user the possibility to use natural language request in order to drive the service composition. Our application field concerns pervasive services but it can be extended to any kind of services.

The main problem that we address here is related with two different domains: Natural Language Processing (NLP) and Service Composition. A NLP problem is that unrestricted natural language may be ambiguous and incomplete. Some requests may contain indications about *how* to achieve a goal (i.e. "connect the phone with the TV") but other requests (i.e. "I want to keep the light level constant") will indicate just a *goal*. On the other hand, the service composition process must create valid and functional services, without limiting too much their complexity. Thus, our objective is not an easy one because we are trying to create complex but valid services, starting from ambiguous requests.

In order to solve the problem stated before we propose a system called NLSC (Natural Language Services Composer). Our solution is based on the use of semantic distances in order to make a connection between the natural language and the services; and composition templates in order to assure that the composition result are always valid. Thus, the use of composition patterns compensates the natural language ambiguity without losing the user-friendliness and the flexibility.

The next section presents the conclusions about the related work and shows the main limitations of the existent approaches. The proposed solution tries to overcome these limitations. Section three presents the proposed solution: the NLSC (Natural Language Service Composer). Section four contains the implementation, which is based on the top of WComp middleware [1] and presents some examples that we have used in order to test our solution. In the section five we evaluate our work. The last section contains the conclusions and some future work.

II. Related work

In or study about the state of the art we have insisted more on the service composition aspects because our intention is to reuse existent solutions from the NLP field. We have focus on semantic-based service composition and natural language-based service composition. Some existent solutions are described below.

A solution based on restricted natural language and sentence templates. The system described in [2] is based on the similarity that exists between an ontology-based service description and a formal representation of the user request. The sentence analysis is based on templates such as: *if* ... *then* ... *else*, *when* ... *do* and others. Verbs are used in order to identify the action and its parameters. The user request is processed and finally transformed into a flow model. One of the major limitations of this solution is the use of restricted natural language (limited set of words and also sentence structure). The user should use keywords (*if, then, else, while, or, and*, etc.) similar to programming languages. It is not clear how can we create a new service if the user request specify only a goal, without giving indications about *how* to achieve it.

A solution based on lexical trees associated to services. Usually, semantic mark-up languages use narrow, predefined vocabulary, which makes possible only the retrieval of those Web services for which the vocabulary is known. The solution patented by Alcatel [3] proposes a method to mark-up web services in order to allow finding and retrieving web services via natural language requests. The idea is to create a lexical tree, built by deriving the service description, and associate it to the web service. The lexical tree is created using synonyms and related forms of the derived keywords. Finding a service based on the user request resumes to comparing the natural language query to the lexical tree of each web service. However, this method address only the service retrieval issue and not service composition.

A solution based on a semantic component model and semantic graphs. The paper [4] propose the *CoSMoS* model and the *SegSeC* platform for dynamic service composition based on semantic graph and ontology. A semantic graph is a regular graph having as nodes the concepts (words) and as arcs the relations between these concepts. The user request is processed using a NLP parser like BEELINE [5], and transformed into a semantic graph. The same type of model, the semantic graph, is used also for describing the services. The semantic graph nodes represent: operations, inputs, outputs, properties of a component and data types.

The authors admit that their solution is not well suited for a large number of components because the composition time increases exponentially with the components number. The platform cannot find a solution if the service semantic graph does not match exactly the user request. Even if it is not specified by the authors, their natural language is in fact restricted: there must be only one predicate, the word set is restricted by the ontology, the sentence must have a specific structure.

Our conclusion about the state of the art is that the most important limitations of the existent solutions are the following: the vocabulary and the request form are restricted (the language is not really "natural"), the developer needs to create dedicated ontology, which is costly; and the composed services have usually simple structures because complex structures are difficult to validate.

III. Proposed solution

A. General architecture

Our system, called NLSC (Natural Language Service Composer), is based on the following principles:

- It uses a Natural Language Processor, based on existent NLP tools, in order to transform the user request into a machine readable, formal, request. This formal request will be the input for our Service Composer.
- It reuse an existent free English dictionary, like WordNet, instead of creating a new ontology.
- It is based on a service platform called WComp [1]. This platform is targeted mainly for intelligent environment applications. WComp was designed for supporting dynamic assembling of services provided by hardware devices. Web services and UPnP services in general may be used through this platform also. The AoA (Aspects of Assembly) mechanism, which comes with WComp, allows the developer to create composition patterns and use them at runtime in order to modify the service architecture.
- The formal request (the NLP output) will be used in order to select the services and also the AoA patterns. Once we have selected the services and the patterns, the WComp platform is able to create almost instantly the new, composed service.

Figure 1 describes the NLSC architecture. The NLP (Natural Language Processor) is composed by a set of tools necessary for user request analyze. The NLP input may be textual or based on voice recognition. The NLP transforms the user request (natural language) into a formal request that is, basically, a list of concepts (extracted from the user request).

The formal request is used by the SC (Service Composer) to compose a service, on demand. SC is built on the top of the WComp platform and uses the AoA (Aspect of Assemblies) patterns. We motivate our preference for WComp platform mainly because of the flexibility provided by the AoA pattern support. Contrarily to other pattern-based approaches, AoA patterns can be combined/superposed. Thus, a large number of valid combinations (services) may be created.

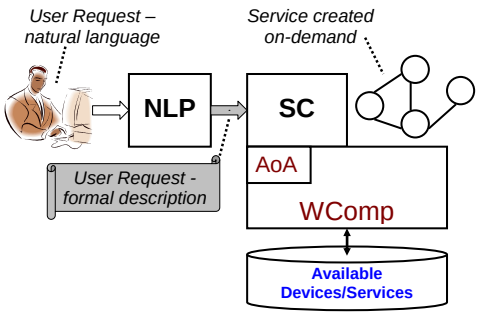

Figure 1. NLSC system architecture

The easiest way to explain our solution is to see it as a processing sequence, that starts from the user demand and it is finalized in a form of a new service.

B. NLSC sequence description

In this section we describe the operations that are executed by the NLSC system, in order. The first operations are related to the NLP module and the last are executed by the SC module.

1) NLP: the input.: The NLP input is a sentence that is either, written by the user or obtained from an existent voice recognition system.

2) NLP: the linguistic preprocessing.: The sentence is processed as it follows, in order:

- The sentence is decomposed in a word collection.
- The link words are eliminated. In order to do that, we use a list of link words.
- For each word, we apply a stemming procedure: the verbs are passed to infinitive, the nouns are passed to single form.
- We apply a spell checking in order to compensate some errors produced by our stemming algorithm.

Finally, we obtain a list of words that we will call further "request concepts".

3) SC: the services selection.: The services selection is the most complex task of our system. Service selection means to select the M services from the N available, M $<$ N, based on the semantic matching between the request concepts (as explained before) and the concepts associated with the services. This means that, each service has a concept that describes it. The concept may be a single word (ex. TV) or a composed word (ex. mobile phone). We will call these concepts "service concepts" in order to make the difference with "user concepts".

The semantic matching is based on what we have defined as the "conceptual distance". The conceptual distance is a measure of the similarity between two concepts. In order to compute the semantic matching, we use a representation called "conceptual graph". The conceptual graph and its usage is described in the next paragraph.

The conceptual graph and the conceptual distance.: The conceptual graph nodes are the concepts (user concepts and service's concepts, defined before). The conceptual graph arcs connect each user concept to each service concept. The weight of each arc represents the *conceptual distance* between the concepts. The notion of conceptual distance is described below.

The conceptual distance is a measure of similarity between the concepts. We used for this purpose a specialized dictionary called WordNet [6]. WordNet contains nouns, verbs, adjectives and adverbs in sets of synonyms, called synsets. Each synset describes a different concept. Different senses of a word are in different synsets. Most synsets are connected to other synsets via a number of semantic relations.

Figure 2 describes an example of the conceptual graph (the incidence matrix), based on the conceptual distance (computed as described before). The service concepts are described by columns and the user concepts by rows. The values represent the conceptual distances: low values meas high concept similarity. The service concepts correspond to all available services (in our scenario, all available devices in the user room).

In order to select the service concepts that are similar to the user concepts, we need to apply the following two transformations to the conceptual graph:

	want	use	phone	switch	light	turn	tv	play	music	hifi
Television	5	5	1	5	5	5	0	5	5	5
Light	5	5	5	5	0	5	5	5	5	5
VCR	5	5	5	5	5	5	5	5	5	5
HiFi	5	5	5	5	5	5	5	5	5	0
Mobile Phone	5	5	1	5	5	5	5	5	5	5
PDA	5	5	5	5	5	5	5	5	5	5

Figure 2. The incidence matrix of the conceptual graph

- Find the minimum distance path in the graph (include all nodes). In order to find the minimum weight subgraph, we use the Kruskal [7] algorithm that calculates the minimum spanning tree (MST). After this transformation each service description will be connected to 2 text segments.
- For each triplet (service concept, user concept 1, user concept 2) keep the arc that has the minimum weight. We obtain a concept pair.

This algorithm helps us to select only the devices/services (from all available services) that are relevant comparing to the user request.

4) SC: the services composition.: We used a *template-based* service composition system because of its capability to handle complex interactions between components and the flexibility of choosing different sets of components. The system we used, called Aspects of Assembly (AoA) [1] is part of the WComp [1] middleware for ubiquitous computing.

These templates can be automatically selected either by the service composition system when satisfying a user request or triggered by context changes in a self-adaptive process and composed by a weaver with logical merging of high-level specifications. The result of the weaver is projected in terms of pure elementary modifications (PEMs) add, remove components, link, unlink ports. The AoA architecture consists of an extended model of Aspect Oriented Programming (AOP) for adaptation advices and of a weaving process with logical merging. For more details about the logical merging mechanism in case of multiple AoA applying on same components, the lecturer could refer to [1].

An AoA template is structured as an aspect with a list of components involved in composition (called "pointcut") and adaptation advice (a description of the architectural reconfigurations), which is specified using a domain specific language (DSL). We will examine some AoA templates and the composition process in detail in the next section.

IV. Implementation and Results

We used the WComp [1] platform for ubiquitous computing to connect the intelligent devices that offer services used to satisfy the user request. WComp uses the UPnP protocol to communicate with the devices. Each UPnP device has a

software proxy that acts like a software component, exposing the devices services. We added some meta-data to the UPnP service description for each device to serve as semantic description. The interactions between these components were specified using AoA templates [1].

Scenario 1. "I want to use my phone to turn off the light, turn on the TV and play some music on HiFi". This phrase contains many irrelevant words to the service composition system, but the relevant words are identical (except TV) to the service semantic descriptions. Irrelevant words have an effect of increasing the time required to process the conceptual graph. All the relevant services are identified and then composed.

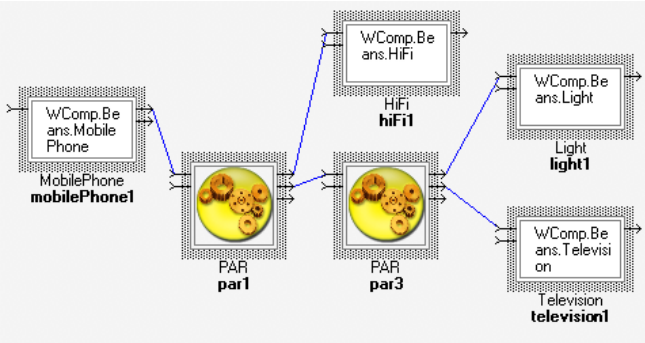

Figure 3. The dynamically composed service for Scenario 1

Scenario 2. "Use PDA for broadcasting". This user request is challenging for any composition system because it doesn't address the TV directly, but through the abstract concept of *broadcasting*. Due to the use of the specialized dictionary, the TV is found and then connected to the PDA.

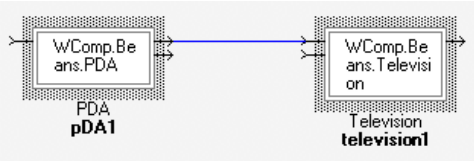

Figure 4. The dynamically composed service for Scenario 2

The main complexity of the algorithm is given by the conceptual distance computation. The time necessary to compute the distance between two concepts (similarity) was about 6 ms (on a Dell Latitude 830 laptop, CPU Dual Core 2.2GHz, 2G RAM) with the fastest algorithm that we have found (a WordNet *Similarity* implementation, the java package edu.sussex.nlp.jws.JiangAndConrath). For instance, we can create the concept graph for 10 user concepts and 64 services in about 2 seconds. The AoA application is very fast also: less than 1 sec. for up to 350 components.

V. Conclusion

This paper proposes a method for assembling new services on-demand, starting from the user request expressed in natural language. The original aspect of our proposal is the mixed approach: semantic and pattern-based. This approach combines the advantages of the both approaches: thanks to composition patterns, it allows us to build complex composite services, which are always valid and functional. With other approaches (interface, logic, semantic-based), that are not using patterns/templates, it is very difficult to create complex service structures/architectures that are valid and work correctly. In particular, our AoA patterns can be composed and this helps us to overcome the limitations of the traditional pattern-based approach (that is not very flexible).

Another important advantage of our solution is the reuse of WordNet free dictionary, which is acting like a huge, common ontology. Due to this, we can relax very much the limitations for the natural language, imposed by solutions where an ontology (usually restricted) must be created by the developer. For describing the services, their developer just need to use correct English words. This solves also the important issue of ontology integration.

As future work, we intend to extend the composition mechanism to dynamic services adaptation.

Acknowledgments

This work was supported by the EcoNet project code 18826YM and the national project PNCDI II code 1062. Thanks to other members of the Rainbow team for fruitful discussions and feedback: Vincent Hourdin, Daniel Cheung-Foo-Wo, Eric Callegari.

References

[1] J.-Y. Tigli, S. Lavirotte, G. Rey, V. Hourdin, D. Cheung-Foo-Wo, E. Callegari, and M. Riveill, "WComp Middleware for Ubiquitous Computing: Aspects and Composite Event-based Web Services," *Annals of Telecommunications (AoT)*, vol. 64, no. 3-4, Apr. 2009.

[2] A. Bosca, F. Corno, G. Valetto, and R. Maglione, "On-the-fly construction of web services compositions from natural language requests," *JSW*, vol. 1, no. 1, pp. 40–50, 2006.

[3] P. Larvet, "Web service with associated lexical tree, european patent, ep1835417," 2007.

[4] K. Fujii and T. Suda, "Semantics-based dynamic service composition," *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 12, pp. 2361–2372, 2005.

[5] G. Mann, "Beeline - a situated, bounded conceptual knowledge system," in *International Journal of Systems Research and Information Science*, 1995, pp. 37–53.

[6] E. by Christiane Fellbaum, *WordNet An Electronic Lexical Database*. http://wordnet.princeton.edu/: The MIT Press, 1998.

[7] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical Society*, vol. 7, no. 1, pp. 48–50, February 1956. [Online]. Available: http://www.jstor.org/stable/2033241