Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844
Vol. V (2010), No. 5, pp. 871-883

Natural Language based On-demand Service Composition

F.-C. Pop, M. Cremene, J.-Y. Tigli, S. Lavirotte, M. Riveill, M. Vaida

Florin-Claudiu Pop, Marcel Cremene, Mircea Vaida
Technical University of Cluj-Napoca,
Faculty of Electronics, Telecommunications and IT,
Cluj-Napoca, Romania
E-mail: {florin.pop, marcel.cremene, mircea.vaida}@com.utcluj.ro

Michel Riveill, Jean-Yves Tigli, Stéphane Lavirotte
Université de Nice - Sophia Antipolis,
Sophia Antipolis CEDEX, France
E-mail:{riveill, tigli, lavirott}@unice.fr

Abstract: The widespread of Web services in the ubiquitous computing era and the impossibility to predict a priori all possible user needs generates the necessity for on-demand service composition. Natural language is one of the the easiest ways for a user to express what he expects regarding a service. Two main problems need to be solved in order to create a composite service to satisfy the user: a)retrieval of relevant services and b) orchestration/composition of the selected services in order to fulfill the user request. We solve the first problem by using semantic concepts associated with the services and we define a conceptual distance to measure the similarity between the user request and a service configuration. Retrieved services are composed, based on aspect oriented templates called Aspects of Assembly. We have tested our application in an environment for pervasive computing called Ubiquarium, where our system composes a service according to the user request described by a sentence. The implementation is based on the WComp middleware that enables us to use regular Web services but also Web services for devices.
Keywords: Natural Language, Service Composition, On-demand, Middleware, Templates

1 Introduction

Background. Since Web 2.0 marked it's appearance as a concept in the fall of 2004 and introduced the principle of the Internet as a platform [19], the complexity and diversity of this platform grew together with the more enhanced features it was providing to its users. Given the fact that the software in the Internet era is delivered as a service, not as a product and there is no release cycle for the services, it is the user who's in charge of finding a service and using it.

In a near future services will be more diverse and widespread as computers will become ubiquitous. In the same way the information across the Web is structured, classified and then presented to a user that sends a natural language request to a search engine, so a collection of applications should be assembled, classified and deployed using the services that are found within a given context based on a similar unrestricted language request coming from the user.

The problem. In a context where various services with different functionality are available, it's possible to compose new services on-demand, based on a user request, only when the right selection of components is used.

On-demand service composition involves two operations: *service retrieval* and *service orchestration*. *Service retrieval* refers to identifying those specific services that are addressed by the user request or the closest functional match to the request. The transition from a natural language request to a list of services is a challenge that is even more difficult when no restrictions are added to control the request. *Service orchestration* or service assembly is the process of linking the retrieved services in a functional flow so that the user demand is fulfilled. Both mentioned problems make the object of this paper, but while service retrieval was the field of our research, for service orchestration we used an existing approach.

Scenario. We consider the following scenario to illustrate the purpose of dynamic service composition based on natural language requests. A handicapped person lives inside an intelligent house, surrounded by intelligent devices, sensors and actuators. Each device has its own inputs and outputs and is able to process different types of data, leading to a large number of possible functional combinations of those devices. The person in the intelligent house wants to use those devices by combining their functionality (e.g. link the output from a sensor to the input of an actuator), but his disability prevents him to physically interact with the devices or he simply lacks any technical knowledge. Therefore, he expresses his need using the natural language (either written or spoken): *"I want to use my remote control on the wheel chair to turn off the light, change the channel on TV and play some music on the media center"*. Each device the user addresses through his request (remote control, light, TV, media center) provides a different service with specific actions that can be used in various configurations. Finding a way to sort these configurations by relevance to the user's request is a key requirement for the imagined scenario. Also, when one of the devices the user wants to use is not present in the intelligent house or it was replaced with an updated version, the system should adapt and assemble a service that is the closest match to the user's need.

Approach. Dynamic service composition solves the problem of adaptation to different contexts and user preferences. Also, by composing services on demand, the learning curve required for the user to work with new configurations is reduced as the user *"gets what he wants"* from the application. Existing systems for dynamic service composition based on natural language requests either provide a restricted natural language interface or don't offer support for adaptation to structural and behavioral changes of the service configuration.

We start with an initial set of services that are discoverable across a network. The user requests a completely new service using an unrestricted natural language sentence. In order to find specific devices to satisfy the request, we use semantic concepts and define a conceptual distance between the request and a service configuration. Concepts are leafs on a lexical tree that is generated by deriving and generalizing a notion. Once the services that match the request are identified, some aspect oriented advices are used to connect the services so that when a service disappears from the context or a new service is made available, the service configuration adapts.

Outline. This paper is organized as follows: the next section examines the problems a dynamic service composition system should solve in order to be usable in the modern context of Web services. Section 3 describes the solution we propose including the principles that lead to this solution, while section 4 focuses on the design and implementation patterns we used, along with the test results. Section 5 examines some of the existing dynamic service composition approaches. The paper ends with conclusions and further research.

2 An overview of the problem

On-demand dynamic service composition based on natural language requests raises some challenges that need to be studied before a solution is to be proposed.

Service retrieval. The first problem we approach in this paper is finding and retrieving a particular service. The large variety of Web services useable for composition needs to be classified in a way that would make it machine meaningful and semantic-rich for the search to provide the best results.

One Web extension, called Semantic Web [3] is focused on enabling better Web service interoperation. The Semantic Web's purpose is to bring structure to the meaningful content of Web pages, creating an environment where software agents roaming from page to page can readily carry out sophisticated tasks for users. One dialect of the DAML [14] family of Semantic Web markup languages was proposed in [17] for the markup of Web services. This so-called semantic markup of Web services creates a distributed knowledge base (KB) that provides a means for agents to populate their local KBs so that they can reason about Web services to perform automatic Web service discovery, execution and composition and interoperation. But the semantic mark-up uses a narrow, predefined vocabulary as identified in [16], which makes possible only the retrieval of those Web services for which the vocabulary is known. Queries or requests from Web services or user requests, using another vocabulary than the predetermined vocabulary are not suitable to find or retrieve such a Web service. Therefore, a narrow vocabulary for the semantic mark-up of Web services is not appropriate to be used in combination with natural language requests.

Service composition. The second problem that needs to be solved is the actual composition of the retrieved services. There are 3 types of systems for dynamic service composition according to [12].

Template-based systems [18,21] are using a service template to compose an application. They can handle complex interactions between components and allow some level of flexibility by choosing different sets of components. The drawback of this approach is that they cannot compose applications for which templates are not available.

Interface-based systems [7] allow the user to submit a set of inputs and outputs for the application he is requesting. These systems have a higher adaptability than the template-based systems, but certain applications cannot be represented as a set of inputs and outputs (e.g. an email sending service does not output any data).

Logic-based systems [20,23] extend the interface-based approach by adding extra information into interface information using first order logic or linear logic. A user requests an application by submitting a first order formula representing the logic that must be satisfied by the application. They are more adaptable than the template-based systems since they donÕt require service templates and offer support for more varieties of services than the interface-based systems. Their main disadvantage is given by the fact that they are not extensible and are not suitable for a distributed environment.

Adaptation. The last, but the most important problem that needs to be solved by a dynamic service composition system is adaptation. We distinguish two types of changes that require adaptation [2]: structure changes and behavior changes. Structural adaptation consists in modifying the retrieved services while preserving the global behavior of the application. The behavior describes the sequence of operations to be executed to fulfill the user request. Structural changes are triggered when a retrieved services disappears from the context or is replaced by another (for example the analog TV is replaced by a digital TV). Behavior changes are related to the user who may decide that the current service does no longer satisfy his need.

3 Proposed solution

Our dynamic service composition system was inspired out of the user's need to interact with the intelligent devices that surround him. This user-machine interaction should be as natural as human interaction, through unrestricted natural language. Intelligent devices are entities that provide services to the user and have networking capabilities. Roughly, there are two types of intelligent devices: *basic devices* that are simple service producers (e.g. a light that offers the illuminating service, a TV that offers the tuning service that allows changing the TV channels) and *controller devices* that can consume services by other devices, acting as interfaces for a composite service (e.g. a mobile phone, an ultra-mobile PC, a netbook). Hybrid devices that implement both previously mentioned functionalities can also be imagined.

The intelligent devices are connected to form an Ad-Hoc network inside the intelligent house, which they use to exchange information. This means that the devices can appear and disappear from the network structure on-the-fly: a device can auto-configure when it joins the network and then leave the network without notice. Network and device management is a task for the middleware that runs the intelligent house.

3.1 Semantic descriptions for Web services

WSDL [10] is intended for the functional description of Web services and the Semantic Web mark-up is limited to a narrow vocabulary, which is not suited for natural language requests. A lexical tree [16] would add too much semantic information to a service and would not be suited for embedded devices.

To overcome these limitations, we propose the use of general notions, called *concepts*, to describe the utility of a service. A service is not entirely identified by a single concept, but by an infinite number of concepts that are determined through the generalization of a notion. This notion will serve as a semantic description for an intelligent device that offers a service. We use the *television* notion to describe a TV, for example. Through generalization we find that both the *television* and *electronic equipment* concepts refer to the same device. To increase the precision, a lexical analysis is also conducted for the service description and the user request by the composition system. This way, the service description suffers little or no modifications due to the extra semantic information.

3.2 Linguistic processing

We consider the scenario where the user interacts with appliances and he expresses his need through a sentence: *"I want to use my phone to turn off the light, turn on the TV and play some music on HiFi"*. In order to retrieve the services required to satisfy the user need, the request goes through a linguistic processing module, responsible for:

- Text segmentation required to separate the words in the phrase (e.g. *switch off the light* is transformed into *switch*, *off*, *the*, *light*);
- Removing stop words that are considered to be irrelevant (e.g. *the*, *to*, *and*);
- Stemming (e.g. *lights* is transformed into *light*, *using* becomes *use*);
- Spell-checking to correct the misspelled words and the words "damaged" during stemming.

The output text segments for the user request in the considered scenario are: *want*, *use*, *phone*, *switch*, *light*, *turn*, *tv*, *play*, *music*, *hifi*.

3.3 The graph of concepts

The *text segments* together with the *service descriptions* are nodes in a graph, called the *graph of concepts*. The arcs in this graph connect each text segment to each service description. The weight of each arc represents the *conceptual distance* between the text segment and the service description. We introduced the conceptual distance to measure the relationship between two notions.

Measures of semantic similarity or relatedness are found in linguistic processing literature. According to the study published in [6], the Jiang and Conrath's measure proved to be best for practical usability. All compared distances are based on the information content of the lexical terms (the probability of encountering an instance). While this information may be valuable for other applications, it is of less importance for service composition and it adds to the complexity of the implementation, therefore making it slower. Our approach, the conceptual distance, is faster and less complex than Jiang and Conrath's measure, while the last is more accurate.

The conceptual distance is a numerical evaluation of how accurate two notions refer to the same concept. For example the words *phone* and *telephone* describe the same concept - a communication device, therefore the conceptual distance is null. On the other hand, the words *phone* and *electronic equipment* can describe the same concept - a communication device, but one of them is more general, therefore it can address more concepts, which leads to a non-null distance between these words.

Figure 1 shows an example of a graph of concepts where the text segments are *tv*, *light*, *hifi*, *phone* and the service semantic descriptions are *Television*, *Light*, *DVD*, *HiFi*, *Mobile Phone* and *PDA*. Figure 1.A. represents all the distances and figure 1.B. represent only the arcs with minimum conceptual distance.

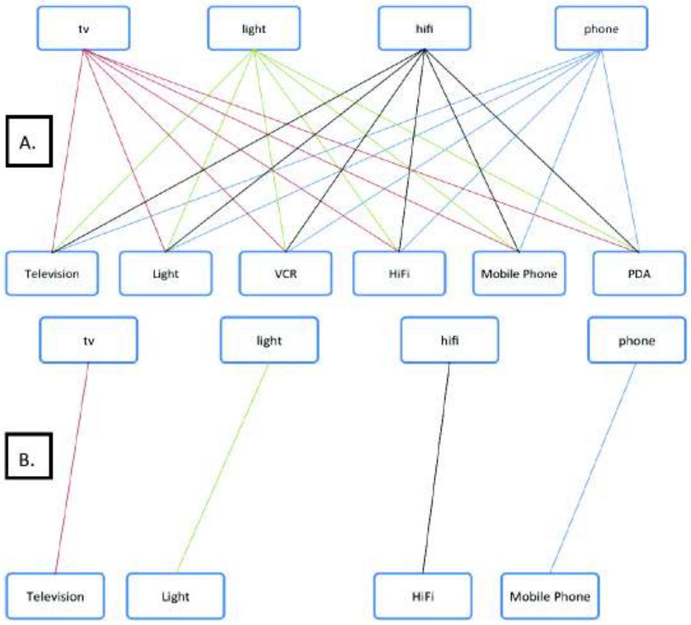

Figure 1: The graph of concepts

3.4 Knowledge structure

In order to evaluate the conceptual distance we need to find a way to classify the lexical basis of the English language. We used for this purpose a specialized dictionary called WordNet [11]. WordNet groups 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. For example, the semantic relations for nouns include:

- hypernyms: Y is a hypernym of X if every X is a (kind of) Y (mobile phone is a hypernym of phone);
- hyponyms: Y is a hyponym of X if every Y is a (kind of) X (phone is a hyponym of mobile phone);
- coordinate terms: Y is a coordinate term of X if X and Y share a hypernym (mobile phone is a coordinate term of cellular phone, and cellular phone is a coordinate term of mobile phone);
- holonym: Y is a holonym of X if X is a part of Y (mobile phone is a holonym of transmitter);
- meronym: Y is a meronym of X if Y is a part of X (transmitter is a meronym of mobile phone).

While semantic relations apply to all members of a synset because they share the same meaning, words can also be connected to other words through lexical relations, including antonyms and derivationally related, as well. Both nouns and verbs are organized into hierarchies, defined by hypernym or *IS A* relationships.

For example, the hierarchy for mobile phone is:

- cellular telephone, cellular phone, cellphone, cell, mobile phone
- radiotelephone, radiophone, wireless telephone
- telephone, phone, telephone set
- electronic equipment
- equipment

The words at the same level in hierarchy are synonyms of each other.

3.5 The concept hierarchy

The algorithm that evaluates the conceptual distance uses the WordNet lexicon to create concept hierarchies. A concept hierarchy is generated in 4 steps:

1. Find the synset that contains the concept for which the hierarchy is generated. Each word in the synset becomes a root for a tree in the concept hierarchy.
2. For each tree root, find the synsets that are in a relationship with the rootÕs synset. Each word in the related synset becomes a leaf for the tree, on the next level in hierarchy, branching from the root.
3. For each word on the current level in hierarchy, find the synsets related to the wordÕs synset and add the words in the found synsets as leafs for the tree on the next level.
4. Repeat Step 3 until the hierarchy is big enough so that the degree of generalization for the notion for which the hierarchy is built, corresponds to an accepted accuracy that produces best results. The bigger the hierarchy the longer it takes to generate it, but the smaller the hierarchy the more confusion can occur among concepts.

The hierarchy for the notion mobile phone is shown in Figure 2. The roots of each the tree are part of the same synset and each level in a tree represents the words from the synsets that are related to the word they are branching from.

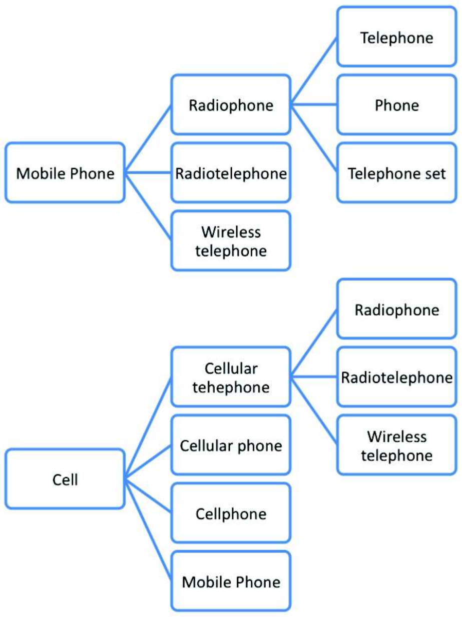

Figure 2: The concept hierarchy for the notion *mobile phone*

3.6 The conceptual distance

In order to evaluate the conceptual distance for two notions, a concept hierarchy is built for each notion. Then, the conceptual distance is calculated as follows:

- The minimum difference of levels between the common node of the 2 hierarchies and the node that represents the notion on which the hierarchy is built for, if such a common node exists.
- The maximum number of levels a hierarchy can have if thereÕs no common notion among the two.

Examples:

- D (Mobile Phone, Cell) = 0
- D (Radiotelephone, Radiophone) = 0
- D (Mobile Phone, Radiophone) = 1
- D (Mobile Phone, Telephone) = 2

3.7 Service retrieval

Finding and retrieving the closest services to the user request resumes to identifying the couples (*text segment*, *service description*) in the graph of concepts that have a minimum sum of conceptual distances. This makes it easier to take advantage of a service that only partially matches a request.

In order to find mentioned couples we need to apply tow transformations to the graph of concepts:

- Finding the sub-graph that has the minimum distance path that includes all the nodes. After this transformation each service description will be connected to 2 text segments.
- For each triplet (*service description*, *text segment 1*, *text segment 2*) remove the arc that has the maximum weight.

In order to find the minimum weight sub-graph, we use the Kruskal [15] algorithm that calculates the minimum spanning tree (MST). The nodes that contain the service descriptions resulted after the two transformations are applied, represent the services that are used to generate the composite service requested by the user. The transformed sub-graph for the graph in Figure 1 contains the nodes that are connected with the thick continuous line.

3.8 Service 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) [9] is part of the WComp [8] middleware for ubiquitous computing and besides the benefits that derive from the fact that is template-based, also offers support for auto-adaptation.

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 speciȚcations. The result of the weaver is projected in terms of pure elementary modiȚcations (PEMs) Ð 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.

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.

4 Implementation and results

We used the WComp [8] platform for ubiquitous computing as the middleware of our intelligent house. WComp uses the UPnP protocol to achieve device interconnectivity and interoperability. Each UPnP device has a software proxy that acts like a software component. Using this proxy, we can treat Web services for devices similar to the UI components of a GUI designer. We added some meta-data to the UPnP service description for each device to serve as semantic description.

Using WComp, we have simulated the following devices/services: TV set, described by the *television* notion; DVD recorder, described by the *DVD* notion; Mobile phone, described by the *mobile phone* notion; PDA, described by the *PDA* notion; HiFi, described by the *HiFi* notion; Lighting system, described by the *light* notion.

The interactions between these components were specified using AoA templates. Following, is an example of such a template that is used to connect the mobile phone to the TV:

```
Pointcut
   inputDev:=/mobilePhone.*/
   outputDev:=/television.*/
Advice KeyToChannel(inputDev, outputDev):
      input.^key_Pressed ->
            ( output.set_Channel ; CALL )
```

The first 3 lines describe (defining the pointcut as in aspect programming), using filters in the AWK language, the components involved in the interaction: a mobile phone (*inputDev*) and a television (*outoutDev*). The filters of type */instanceName.*/* will find components that have their name prefixed by *instanceName*. Line 4 declares a composition schema that uses the previously described components. Lines 4-5 specify the composition mechanics: call the *tv.set_ Channel* method when the *mobile phone* fires the event *key_ Pressed*.

The service composition system implements a UPnP device that offers the service of designing composite services for the user in order to fit seamlessly with the WComp middleware. Requests from the user are captured by a WComp assembly of components, and then sent using the UPnP protocol to the service designer along with a description of the context where the devices are located. The service designer queries the devices for service descriptions (semantic meta-data), and then finds only those services that are relevant to the user request. Instances of the devices that provide the named services are interconnected based on the rules described in the AoA templates.

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 graph of concepts. All the relevant services are identified and then composed.

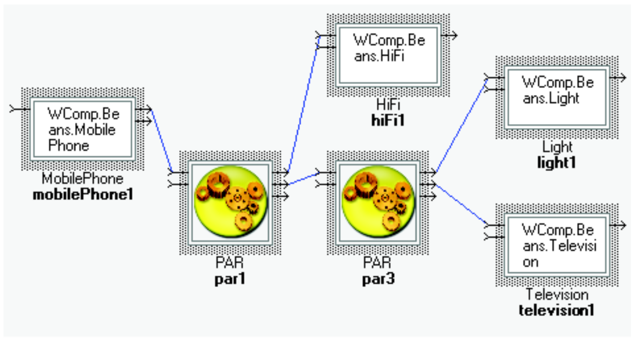

Figure 3: The dynamically composed service for Scenario 1

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.

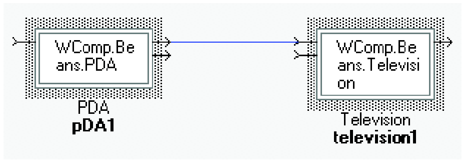

Figure 4: The dynamically composed service for Scenario 2

The WordNet (free download) dictionary is installed on a local machine thus the word search is fast. The main complexity of the algorithm is given by the conceptual distance computation. The time necessary in order to compute the distance between two concepts (similarity), based on the English dictionary, 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). Thus, if we have M user concepts and N service concepts, we need a time equal to $\frac{M*N}{2} * 6$ ms. 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).

In this section we have shown that we are able to create, in practice, new services, on-demand, using real devices, by applying the patterns available for the selected set of services. The scenarios discussed above were tested in a dedicated environment for ambient computing, called *Ubiquarium* [24]. Real devices were used (a part of the services are real devices and another part are virtual ones).

5 Related work

Composing Web services on the basis of natural language requests. The solution described in [4] and [5] assumes that the user requests are expressed with a controlled subset (a narrow vocabulary) of natural language. The sentence that represents the userÕs request is transformed into a flow model using templates (e.g. *if ... then ... else*, *when ... do*). Verbs are used to identify the action and its parameters. Each available service is paired with a well-defined set of keywords. OWL-S annotations are used to provide operation semantics and an ontological classification of Web services. The operations act as nodes of a direct acyclic graph and the relations among their IOPEs (Inputs, Outputs, Preconditions and Effects) establish arcs. The graph is translated into an executable service at invocation time.

The example the authors use to sustain the proposed solution uses the phrase *"If there is any cinema showing "Big Fish" in Turin then send a SMS to Dario containing "Lets go to the movies tonight!""* An *if ... then* template is used to identify the flow model. In the next step, called *context focus*, the service types are identified: *cinema*, *SMS*. The verb (*send*) triggers the parameter retrieval stage where IOTypes recognizers based on format (Date, Time, Telephone NumbersÉ) and values (City namesÉ) are used to extract the actionÕs parameters.

This solution establishes a synergy between the semantic service descriptions and the Natural Language interpretation of user requests, through a common ontology and a consistent lexical vocabulary. Therefore it canÕt be used in active environments where new components that act as black-boxes appear and disappear from the context dynamically. Also, the use of a controlled subset of natural language makes it non intuitive for the user as he is restricted to the use of templates when expressing a request.

Semantics-based dynamic service composition. Papers [12] and [13] propose the *CoSMoS* model and the *SegSeC* platform for dynamic service composition. Their idea is to transform the semantics of the user request into a semantic graph. Nodes in the semantic graph represent operations, inputs, outputs and properties of a component, as well as their data types and concepts. Arcs (labeled links) represent the relationships among the nodes. Concepts, entities representing abstract ideas actions are used to annotate the semantics of the operations, inputs, outputs and properties of components. The user request is parsed and the components addressed by the user form a workflow.

The example in [13] uses the phrase *"Print directions from home to restaurant"*. The semantic graph contains the predicate (*print*), the target of the action (*direction*) and the parameters (*home*, *restaurant*). The workflow, containing the retrieved components, is executed as soon as it satisfies the user request. This analysis takes place in a step called semantic matching and consists in a test that verifies that all the links that appear in the user request also appear in the graph that models the workflow.

The authors of [13] admit that their solution is not suited for environments where a large number of components are deployed. The platform lacks the feature of providing a solution in the case where the workflow doesnÕt satisfy the userÕs request. If the generated workflow doesnÕt match exactly the user request, then the dynamic service composition fails. Also, the ability of the implementation to discover certain components is to be questioned because itÕs limited to work with a narrow set of keywords and it lacks a vocabulary.

Web service with associated lexical tree. The invention claimed by Alcatel [16] relates to a method to mark-up a Web service in order to allow finding and retrieving said service via a natural language request. A lexical tree, built by deriving the service description, finding synonyms and related forms of the derived keywords, is associated to each service. Finding a service based on the user request resumes to comparing the natural language query to the lexical tree of each Web service. This method of retrieving a Web service proves to be the most appropriate when dealing with natural language requests. The invention however doesnÕt exploit the full potential of this finding, as it lacks service composition.

6 Conclusion

This paper proposes a new method for assembling services on demand, starting from the user request expressed in natural language. We use a semantic analysis of the user request, in order to identify the services described by concepts that are related to concepts from the user request. Retrieved services are then composed, based on composition patterns, called AoA (Aspects of Assembly). The uses of patterns, which assure that the new service is always valid, compensate the ambiguity of the natural language.

Another important advantage of the AoA patterns, comparing to other existent pattern-based approaches, is the fact these patterns may be superposed by composition. Thus, a large number of combinations are possible using a given set of patterns. Additionally, for a given set of services, the AoA mechanism applies only the patterns that lead to valid services.

The solution was implemented on the WComp platform and tested in a dedicated ambient computing environment, called *Ubiquarium*, using real and virtual intelligent devices/services.

The service composition is user driven, by natural language (voice) and allows the user to get the service on-demand. From this point of view, our solution is less restrictive than the other solutions described in the state of the art section.

An important advantage of our solution is the reuse of WordNet free dictionary, which is acting like 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. Otherwise, the creation of a rich ontology is a very costly task and our solution succeeded to avoid it by reusing WordNet. This choice has another important advantage: it solves the problem of dealing with different ontologies and does not need to impose a common ontology (the only requirement is to use English).

A particular aspect of our proposal is the mixed approach: semantic and pattern-based. This approach combines the advantages of the both: 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 architectures that are valid and work correctly.

As future work, we intend to extend our solution for dynamic service adaptation (at runtime) and this should be feasible because WComp was designed for dynamic service reconfiguration in pervasive environments.

Acknowledgments

This work was supported by the EcoNet project code 18826YM and the romanian national project PNCDI II code 1062, and 1083 financed by UEFISCSU.

Thanks to other members of the Rainbow team for fruitful discussions and feedback: Vincent HOURDIN, Daniel CHEUNG-FOO-WO, Eric CALLEGARI.

Bibliography

[1] A. V. Aho, B.W. Kernighan, P. J. Weinberger, *The AWK Programming Language*, Addison-Wesley, 1988.

[2] Anastasopoulos M.; Klus H.; Koch J.; Niebuhr D.; Werkman E., *DoAmI - A Middleware Platform facilitating (Re-)conŢguration in Ubiquitous Systems*, In System Support for Ubiquitous Computing Workshop. At the 8th Annual Conference on Ubiquitous Computing (Ubicomp 2006), Sep 2006.

[3] Berners-Lee, T.; Hendler, J.; Lassila, O., *The Semantic Web*, Scientific American Magazine, May 17 2001.

[4] Bosca, A.; Ferrato, A.; Corno, F.; Congiu, I.; Valetto, G., *Composing Web services on the basis of natural language requests*, IEEE International Conference on Web services (ICWS'05), pp. 817-818, 2005.

[5] Bosca, A.; Corno, F.; Valetto, G.; Maglione, R., *On-the-fly Construction of Web services Compositions from Natural Language Requests*, JOURNAL OF SOFTWARE (JSW), ISSN : 1796-217X, Vol. 1 Issue 1, pag 53-63, July 2006.

[6] E. Budanitsky, G. Hirst, *Semantic distance in wordnet: An experimental, application-oriented evaluation of five measures*, 2001.

[7] Chandrasekaran S.; Madden S.; Ionescu M., *Ninja Paths: An Architecture for Composing Services Over Wide Area Networks*, CS262 class project writeup, UC Berkeley, 2000.

[8] Cheung-Foo-Wo, D.; Tigli, J.-Y.; Lavirotte, S.; Riveill, M., *Wcomp: a multi-design approach for prototyping applications using heterogeneous resources*, In 17th IEEE Intern. Workshop on Rapid Syst. Prototyping, pag 119Đ125, Creta, 2006.

[9] Cheung-Foo-Wo, D.; Tigli, J.-Y.; Lavirotte, S.; Riveill, M., *Self-adaptation of event- driven component-oriented Middleware using Aspects of Assembly*, In 5th International Workshop on Midd leware for Pervasive and Ad-Hoc Computing (MPAC), California, USA, Nov 2007.

[10] Christensen, E.; Curbera, F.; Meredith, G.; Weerawarana, S. *Web services Description Language (WSDL) 1.1*, Website, 2001
`http://www.w3.org/TR/wsdl`

[11] Cognitive Science Laboratory, Princeton University, *WordNet Đ a lexical database for the English language*, Website, 2006
`http://wordnet.princeton.edu/`

[12] Fujii, K.; Suda, T., *Component Service Model with Semantics (CoSMoS): A New Component Model for Dynamic Service Composition*, SAINT-W '04: Proceedings of the 2004 Symposium on Applications and the Internet-Workshops (SAINT 2004 Workshops). Washington, DC, USA: IEEE Computer Society, 2004.

[13] Fujii, K.; Suda, T., *Semantics-based dynamic service composition*, IEEE Journal on Selected Areas in Communications, Vol 23(12), pag 2361- 2372, Dec 2005.

[14] Hendler, J.; McGuinness, D., *The DARPA Agent Markup Language*, IEEE Intelligent Systems, vol. 15, no. 6, Nov./Dec. 2000, pp. 72Ð73.

[15] Kruskal, J. B., *On the shortest spanning subtree of a graph and the traveling salesman problem*, Proc. Amer. Math. Soc., Vol 7, 1956.

[16] Larvet, P., *Web service with associated lexical tree*, European Patent, EP1835417.

[17] McIlraith, S. A.; Cao Son, T.; Zeng H., *Semantic Web services*, IEEE Intelligent Systems, vol. 16, no. 2, Mar./Apr. 2001, pp. 46-53.

[18] Molina A. J.; Koo H.-M.; Ko I.-Y., *A Template-Based Mechanism for Dynamic Service Composition Based on Context Prediction in Ubicomp Applications*, In Proceedings of the International Workshop on Intelligent Web Based Tools (IWBT'2007), 2007.

[19] O'Reilly, T. *What Is Web 2.0*, Website, 2005
`http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html`

[20] Rao J.; Kungas P.; Matskin M., *Logic-based Web services composition: from service description to process model*, Proceedings of the IEEE International Conference on Web services, p.446, June 06-09, 2004.

[21] Sirin, E.; Parsia, B.; Hendler J., *Template-based Composition of Semantic Web services*, In AAAI Fall Symposium on Agents and the Semantic Web, 2004.

[22] *UPnP Forum*, Website, 2008
`http://www.upnp.org/`

[23] Wu D.; Parsia B.; Sirin E.; Hendler J.; Nau D., *Automating DAML-S Web services Composition Using SHOP2*, In Proceedings of 2nd International Santic Web Conference (ISWC2003), Sanibel Island, Florida, October 2003.

[24] Hourdin, V.; Cheung-Foo-Wo D.; S.L. ; J.Y.T., *Ubiquarium informatique: Une plate-forme pour l'etude des equipements informatiques mobiles en environnement simule.*, In Proccedings of 3-eme Journees Francophones Mobilite et Ubiquite (UbiMob), Paris, September 2006.