Towards the Automation of the Service Composition Process: Case Study and Prototype Implementations

Mazen SHIAA[1], Paolo FALCARIN[2], Alain PASTOR[3], Freddy LÉCUÉ[4], Eduardo SILVA[5], Luís FERREIRA PIRES[5]

[1]Department of Telematics, Trondheim, Norway
Email: malek@item.ntnu.no
[2] Politecnico di Torino (Italy)
Corso Duca degli Abruzzi 24, I-10129. Torino, Italy
Email: paolo.falcarin@polito.it
[3]Bell Labs, Alcatel-Lucent France, Route de Villejust 91620 Nozay, France
Email: alain.pastor@alcatel.fr
[4]France Telecom R&D, France
4 rue du clos courtel, F-35512 Cesson Sévigné
Email: freddy.lecue@orange-ftgroup.com
[5]Center for Telematics and Information Technology,
University of Twente, The Netherlands
P.O. Box 217, 7500 AE Enschede
Email: {e.m.g.silva,l.ferreirapires}@ewi.utwente.nl

Abstract: Service Composition has been a challenging research area for many years. One of the key ideas in this area is the matchmaking (at the semantic level) of requested services and the portfolio of services registered in a given service repository. Accordingly, a composite service can be generated by selecting a set of existing service components that partially match the requested service, and composing these services. In this paper we introduce a general framework for the Service Composition process and our efforts to automate this process. This framework exploits the semantic annotation of services and their parameters. The main annotations we focus on are service inputs, outputs, goals, preconditions, effects, and non-functional properties. Matchmaking aims at making sure that the selected service components are capable of interacting with each other, while the composition aims at orchestrating these selected service components to fulfil the goals of the requested service. The framework takes care of service requests from both experienced service developers as well as ordinary end-users who are not necessarily familiar with service concepts and platforms. Three main prototype platforms have been developed to experiment with the framework.

Keywords: Semantic annotation, service composition, natural-language.

1. Introduction

Web service technologies have proliferated in the last years, so that nowadays a large number of web services are available on the web. By composing these web services, more services can be generated with added-value for the service end-users, creating new business opportunities. This justifies the tremendous research interest and efforts towards service composition. These efforts typically develop and elaborate on composition techniques that generate composite (enriched, value-added) services by combining several services through their interfaces, where these services are possibly provided by different providers in different domains. Automatic service composition is a special case of service composition that targets the automation of the service discovery, selection and composition processes. This paper deals with automatic service composition. The services aimed at by this work are both Telecommunication and Web-based Services. Any service can be a constituent of a composition. At an abstract level we consider that a service is just provided by a service component regardless of how this service component is realized and how it maps onto real distributed network components. These services can be stateless or stateful.

The work presented in this paper has been conducted within the context of the European IST-SPICE project (IST-027616) [1]. SPICE (**S**ervice **P**latform for the **I**nnovative **C**ommunication **E**nvironment) is a research project aimed at addressing the design, development and deployment of efficient and innovative mobile service creation and execution platforms for networks beyond 3G. The SPICE project vision is to design, develop and prototype an extendable overlay architecture that facilitates and accelerates the creation and execution of intelligent ambient-aware services for the above mentioned networks [2]. In SPICE, a Service Creation Environment (SCE) has been developed that consists of a set of tools for enhancing the automation of the service creation and composition process. In this paper we present the methodology and the prototypes that have been developed within the scope of the SPICE SCE for service composition.

Section 2 provides a motivation to the problem area dealt with in this paper. Section 0 describes the general framework, looked at from the end-user perspective and the service developer perspective. Section 4 presents our three main prototype implementations: the handling of end users natural-language requests, the automatic composition engine, and the goal-based validation of composite services. We conclude our paper in Section 5.

2. Motivation

In the last years there has been a lot of interest in defining and implementing mechanisms and frameworks for service composition in the industry and the academia. Some of these developments apply principles of Service-Oriented Architecture (SOA) and Service-Oriented Computing (SOC) [3], and some build algorithms based on Artificial Intelligence (AI). The Semantic Web Services (SWS) initiative plays a key role in most of these developments, and consists of annotating web services with semantic information, such as, for example, service description, provider details, operations details, intentions, parameters, and invocation details. The annotation terms used in this case adhere to terminologies that have corresponding formal conceptualizations, which are called ontologies. Although many different approaches to service composition exist, we observe that all of them are triggered by a set of needs (characterizing the requested service) and that they all aim at producing a composite service (matching these needs) that is defined in some sort of workflow, orchestration plan or detailed service specification.

Among the most interesting approaches to automatic service composition are those that process composition requests expressed in natural languages, and those based on explicit requests that identify the goals (intentions) to be achieved by the composite service.

The natural language approach exploits the possibilities of deriving a formal specification of a service request for a service composition starting from an informal request expressed in natural language. This formal machine-understandable service request can then be used as input for obtaining a composite service through different composition strategies (goal-based, forward chaining, backward chaining, best coverage, etc.). Examples of approaches based on service requests expressed in natural languages are discussed in [4],[5],[6],[7],[8], and [9].

Different approaches based on the goals of the requested composite service can also be found in the literature. Some of these approaches apply techniques from Semantic Web Services and/or AI planning. In [10], semantic graphs derived from natural language descriptions are used for service composition. In [11] semantic interfaces that are annotated with goals describing the interactions between components are used to compose services.

3. Automatic Service Composition Framework

One of the goals of the SPICE SCE is to facilitate the composition of existing services to build new services. The benefits of service composition stem from the possibility of reusing the efforts invested in developing services, thereby enabling faster time-to-market and costs reduction in the service development process. This leads to direct and indirect benefits to service developers, platform operators and service providers.

Within the SPICE SCE we aimed at creating tools that facilitate automatic service composition. In the initial phase of the project, we focused on the development of a language suitable for specifying composite services, bearing in mind the challenges brought up by targeting both web-based and telecommunication services. This language should also support the semantic annotation of service descriptions with functional and non-functional properties. Furthermore, this language should comprise both a workflow-like notation to describe how the constituent services are orchestrated to provide the overall functionality of the composite service (similarly to WS-BPEL [12] and CDL [13]), and a notation to produce semantic descriptions of the composite service interface and its operations (similarly to WSDL-S [14] and OWL-S [15]). This resulted in the development of SPATEL (*SPICE Advanced service description language for TELecommunication services*), which is described in [12]. The details of SPATEL are outside the scope of this paper.

Figure 1 shows a simple example to give an impression of the semantic annotation and orchestration capabilities of SPATEL. The supported annotations are: *Inputs* and *Outputs* (IO) parameters, *Goals* (G) that describe the overall objectives, *Effects* (E) that describe the outcomes of the execution of an operation, *Preconditions* (P) that describe the conditions that have to be satisfied in order to allow the execution of a given operation, and *Non-Functional* (Q) properties to describe aspects related to quality-of-service, such as performance (e.g. delay), charging (cost), reliability, and resource usage.

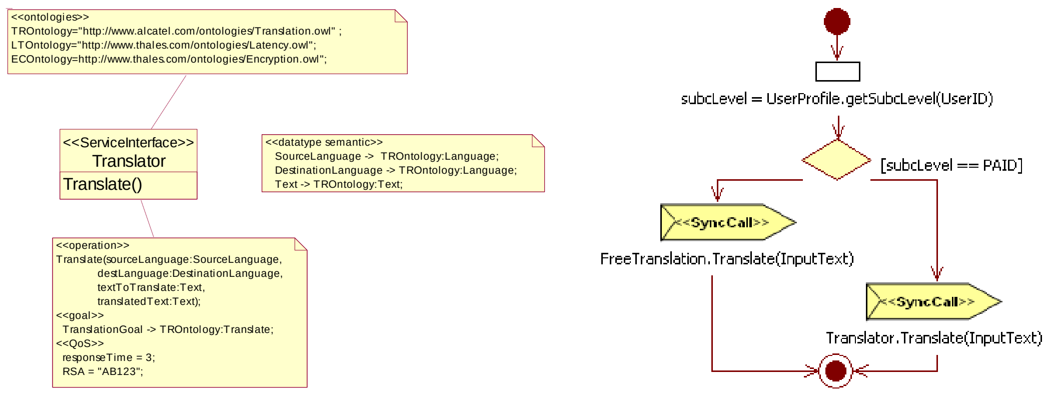
```
<<ontologies>>
TROntology="http://www.alcatel.com/ontologies/Translation.owl" ;
LTOntology="http://www.thales.com/ontologies/Latency.owl";
ECOntology=http://www.thales.com/ontologies/Encryption.owl";
```
```
<<ServiceInterface>>
Translator
Translate()
```
```
<<datatype semantic>>
  SourceLanguage ->  TROntology:Language;
  DestinationLanguage -> TROntology:Language;
  Text -> TROntology:Text;
```
```
<<operation>>
Translate(sourceLanguage:SourceLanguage,
          destLanguage:DestinationLanguage,
          textToTranslate:Text,
          translatedText:Text);
<<goal>>
  TranslationGoal -> TROntology:Translate;
<<QoS>>
  responseTime = 3;
  RSA = "AB123";
```
```
subcLevel = UserProfile.getSubcLevel(UserID)
[subcLevel == PAID]
FreeTranslation.Translate(InputText)
Translator.Translate(InputText)
```

Figure 1 A SPATEL example specification: the left-side shows the semantic annotation of a Translate service, and the right-side shows an orchestration of a composite service that uses the Translate service.

3.1 – The proposed framework

Our proposed framework for automatic service composition supports two scenarios initiated by two different stakeholders: the end-user and the service developer, respectively. In the scenario initiated by an end-user, the approach consists of providing a service on-the-fly that complies with the end user needs expressed in a service request uttered in some natural language, taking into account the user's context, like, e.g., presence and location. In the scenario initiated by a service developer, the main objective is to simplify and automate the service selection and composition process. To achieve this, the service developer formally specifies a service request, which is used by the framework to discover a set of services, and then to guide the service composition process to obtain composite services that match the service request. Service request and existing services are both described based on common ontologies (mainly domain and service ontologies), which allow service discovery and enable the composition and interoperability of the discovered services.

Figure 2 depicts our proposed framework and the supported scenarios.

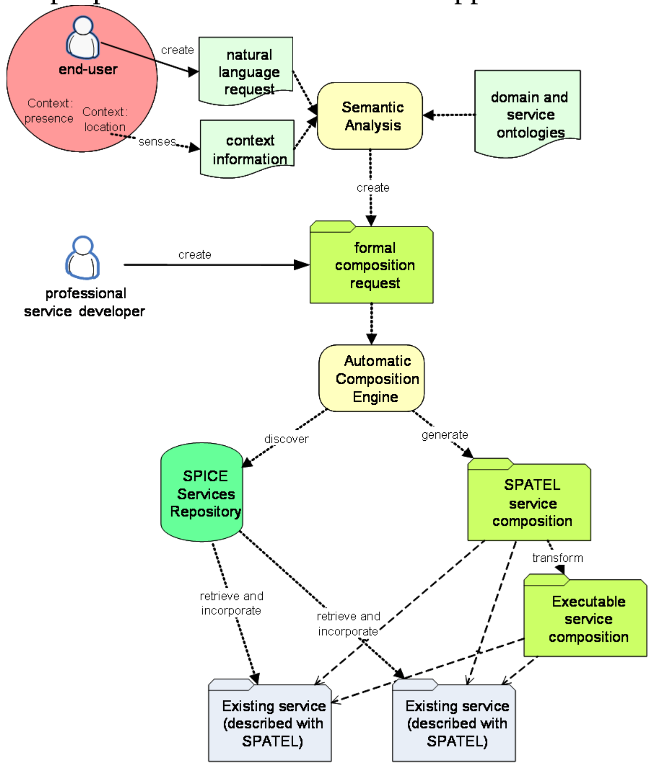
end-user
Context: presence
Context: location
create
senses
natural language request
context information
Semantic Analysis
domain and service ontologies
create
professional service developer
create
formal composition request
Automatic Composition Engine
discover
generate
SPICE Services Repository
SPATEL service composition
transform
Executable service composition
retrieve and incorporate
retrieve and incorporate
Existing service (described with SPATEL)
Existing service (described with SPATEL)

Figure 2 Proposed framework.

3.2 – The end-user scenario

The scenario initiated by the end-user is described as follows:

- The end-user issues a service request in natural language.
- The end user's request is analyzed by a semantic analyzer tool.
- The main "service needs" are determined, and a formal service request is generated.
- The constituent services corresponding to these needs are discovered by using the Automatic Composition Engine (ACE), which browses a service repository that contains the semantic descriptions of the available services within a given domain.
- The discovered constituent services are assembled automatically based on semantic tags [8], generating a service composition. Semantic tags are attached to the outputs and inputs of each operation of the constituent services.
- From this assembly, an executable business process (executable service composition) is generated, expressing the logic of the calls to the constituent services.
- The newly created service can be deployed and offered to the end-user, for example, by generating a client application and notifying the end-user.

3.3 – The service developer scenario

The scenario initiated by the service developer is described as follows:

- Service developer issues a service request in formally and forwards it to the ACE.
- The ACE discovers the constituent services from the service repository, assembles them, produces a service composition and deploys this composition, similarly to the end-user scenario. However, the following differences can be observed:
  - The ACE may produce a list of composition candidates that provide the requested functionality (as SPATEL orchestrations), instead of a single composition.
  - Hence the service developer selects among (or validate) these composition candidates.

4. Prototype implementations

As described earlier, we have been implementing our framework addressing different scopes or functionality. Three main proof-of-concept prototypes have been worked out: the handling of end users natural-language requests, the automatic composition engine, and the goal-based validation of composite services.

4.1 – Natural-language (NL) request-based prototype

The implemented framework is sketched in Figure 3. The description of the framework is presented by simple example.

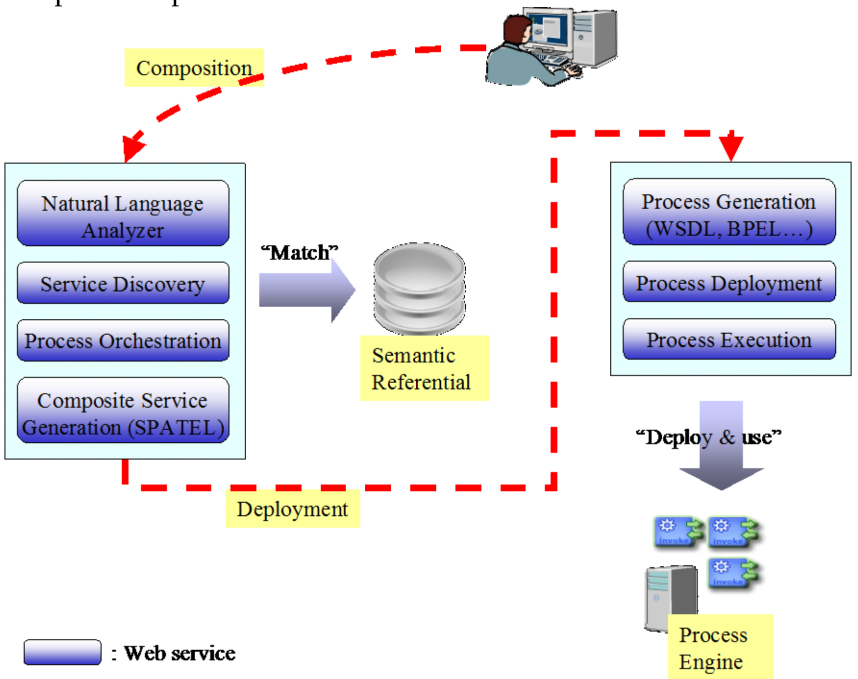

Figure 3Global view of the natural-language request-based prototype

As an example, we consider the following end-user request: "I want the weather at Paris translated in English and send it to the mobile of John.Doe". The NL request is analyzed in order to extract from it the "needs of services" that will be composed to make the final composite service, which covers the goal of the request [9]. The above sample is "understood" as follows by the Semantic Analyzer:

translate(meteo(Paris), english) and send(SMS, mobile(John.Doe))

The result of the analysis of the original request is a set of "needs of services" or concepts, logically linked together and expressed in a "canonical" form. These concepts are the information by which the service repository is queried for a list of services that match. Each concept/set of concepts is searched in the meta-data of each service description (SPATEL description and ontology). During the selection of the discovered services the needs are updated with the preferences of the user or with his context. As an example a messaging service is discovered but it has three (matching) operations that can be selected:

SendMessage.SendSMS(phone_number, message)
SendMessage.SendFax(phone_number, message)
SendMessage.SendVoiceAnnouncement(phone_number, message)

If the user is driving, the SendVoiceAnnouncement operation will have the preference. The composition of the discovered services is based on the following steps:

- The recognition of the grammatical form allows determining the main goal of the request. In this example the main goal is identified as: send the weather by SMS. The determination of the main goal is a key factor to the success of an exact chaining of services that match the requested service.
- The recognition of service parameters allows for the chaining of services by comparing and chaining inputs and outputs of the services, through the comparison of their semantic tags [8]. For example, a data named "message" or "content", with the type "string", will have the same semantic tag "text". Within the chain of services inputs that cannot be related to output of other services are moved to the composite service input.

The chaining of services for the example is the following:

SendSMS(Translate(Meteo("Paris"), "fr_en"), GetMobilePhone("John.Doe"))

4.2 – Formal composition request-based prototype

The automatic composition engine prototype

Figure 4 shows the high level architecture in which we implemented and tested our automatic composition engine. In this architecture we start from a repository of services, containing descriptions of the services used in our different scenarios. The developer specifies a service request in terms of its inputs, outputs, preconditions, effects, and optional non-functional properties. The service request is then translated into a SPATEL.

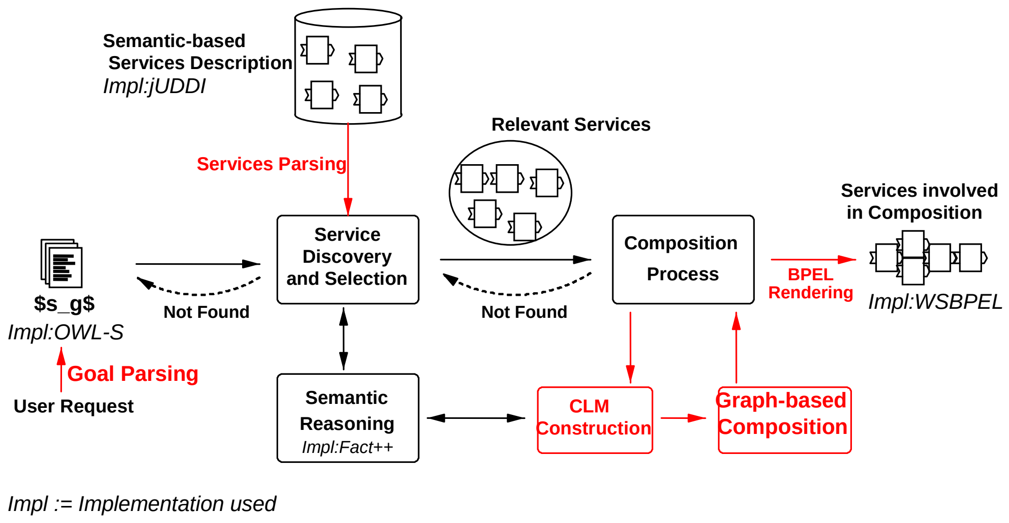

Figure 4 high-level description of the service-developer prototype

In the considered architecture we assume that a discovery process (i.e., Service Discovery and Selection Module) is in charge of retrieving relevant services based on the service request of the service developer. The latter module interacts with the Semantic Reasoning Module since some semantic inferences are performed to retrieve the relevant set of web services. The Service Discovery and Selection module aims also at parsing relevant web services in order to facilitate the CLM (Causal Link Matrix [17]) construction. Such a matrix aims at storing semantic connections between Web services.

The architecture is completed by a Semantic Reasoning module, which provides a vital infrastructural support to two components of the architecture i.e., the Service Discovery and Selection module, and CLM Construction module. The main function of this module is to infer some properties on input and output parameters of the services, for instance to check satisfiability or subsumption of service parameters by means of a reasoner (Fact++ [17]). The power of this module is therefore crucial to the performance of the overall architecture.

The Composition Process module is the core module of the overall Architecture. From the set of services returned by the Service Discovery and Selection Module and the service request, the CLM Construction module elaborates the CLM of the considered composite service. The CLM Construction module is related to the Semantic Reasoning module since the latter module requires semantic reasoning to infer values of causal links between web services, i.e. exact or partial match. The graph-based composition module is responsible for computing correct composition of composite service. These compositions have to match the service request functional, but also the non-functional properties. At the end, the composition process renders the compositions in an executable format (e.g. WS-BPEL).

SCE Add-on for goal-based validation

Validation in system development is the process of ensuring that we are building the right system that meets the requirements of the user. In the context of automatic service composition, validation is aiming at ensuring that the outcome of the automatic composition process is meeting the service developer’s requirements. The validation process is based on the semantic annotations expressed in the formal composition request. As a first step, we have been considering goal-based expressions to be validated. These expressions reason about the service goals of the requested composite service.

As shown in Figure 5 the Add-on (called validation wizard as it iteratively takes several goal-based expressions from the service developer) is developed as a back-end to the ACE. It reasons about the outcome of the composition process (one or more composite services) using the goals ontology. To ensure modularity the interface between the ACE and the validation wizard is the composition candidates, which are specified in SPATEL.

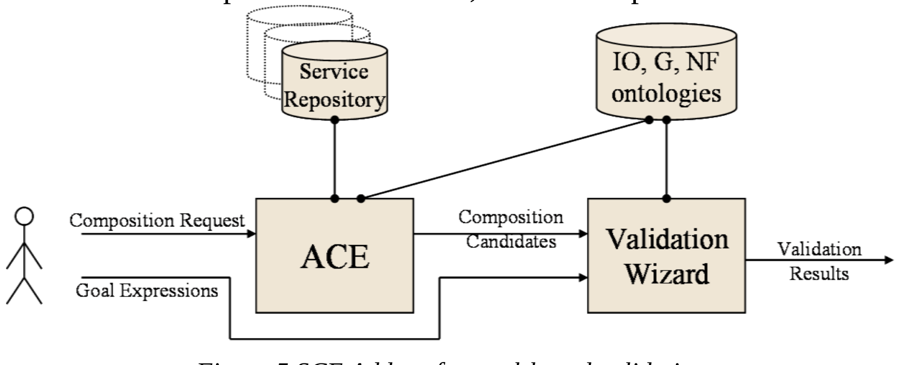

Figure 5 SCE Add-on for goal-based validation

By validating composite services using expressions on service goals we can verify how the goals of the composite service are expressed in terms of the goals of the constituent services involved in the orchestration. Also, we can verify that certain goals are achieved in the composition, as well as verify that certain goals are achieved in the correct order and with the correct parameters. We chose to use simple Boolean and pseudo-code expressions to denote goal expressions. The following examples give quick look of the expressions the current prototype capable of validating:

- GoalA AND GoalB
- GoalA OR GoalB
- NOT GoalA

- GoalA THEN GoalB   (GoalA directly followed by GoalB)
- GoalA MAXIMUM n TIMES
- Combination of the above expressions

These expressions are self-explanatory and are being constantly worked out and experienced with in our demonstration platform. The validation wizard parses through these expressions and validates them against the goal semantic annotation of the composite service in question, by taking into account the relations in the goal ontology.

5. Conclusion

The paper presented a framework for automatic service composition for the end-user and the service developer. The automatic composition engine as a concept has been realized in two main use cases to handle the two types of composition requests; the natural-language and the formal service request.

Although the paper has not handled the aggregation of the non-functional properties for each composition and compares them to the requested QoS, our ACE prototype implementation takes care of calculating the overall cost for a composition candidate. In our current work we are applying different techniques to aggregate more complex non-functional properties.

References

[1] SPICE (Service Platform for Innovative Communication Environment) project homepage. On-line at http://www.ist-spice.org/

[2] S. Tarkoma, B. Bharat, E. Kovacs, H. van Kranenburg, E. Postmann, R. Seidl, A. Zhdanova, “Spice: A Service Platform for Future Mobile IMS Services,” in Proceedings of IEEE International Symposium on World of Wireless, Mobile and Multimedia Networks (WoWMoM 2007), June 2007, pp: 1-8, ISBN: 978-1-4244-0993-8.

[3] M.P. Papazoglou, D. Georgakopoulos, “Service-Oriented Computing,” Communications of the ACM, October 2003, Vol. 46, n. 10, pp. 25-28.

[4] J. Heinecke, F. Toumani; "A Natural Language Mediation System for E-Commerce applications: an ontology based approach"; 2nd International Semantic Web Conference (Workshop on Human Language Technology for the Semantic Web and Web Services)

[5] Wakeman, D. Weir B. Keller et al; "Composing Grid Services through Natural Language"; 4th UK UbiNet Workshop, May 2005

[6] Bosca, G. Valetto, R. Maglione, F. Corno; “Specifying Web Service Compositions on the Basis of Natural Language Requests”; ICSOC’05 3rd International Conference on Service Oriented Computing, Amsterdam, December 2005

[7] A Process For Automatic Discovery, Orchestration and Delivery Of Semantic Web Services, Philippe Larvet and Bruno Bonnin, INFORSID 2006, juin 2006, Hammamet (Tunisie).

[8] Philippe Larvet, “Automatic Orchestration of Web Services Through Semantic Annotations”, ICEIS 2007, 9th International Conference on Enterprise Information Systems, June 2007, Portugal.

[9] Philippe Larvet, Samir Tata and Nomane Ould Ahmed M'bareck, “Web Service Discovery: Dealing With Natural Language Requests and User Preferences”, WBPMO 2007, International Workshop on Business Process Management for Outsourcing, Maroc, June 2007.

[10] Keita Fujii and Tatsuya Suda, “Semantics-Based Dynamic Service Composition”, IEEE Journal on Selected Areas in Communications, Vol. 23, No. 12, Dec. 2005, pp. 2361-2372.

[11] Richard T. Sanders, Rolv Bræk, Gregor v. Bochmann, and Daniel Amyot; Service Discovery and Component Reuse with Semantic Interfaces. Springer LNCS. SDL 2005: Model Driven Systems Design: 12th International SDL Forum, Grimstad, Norway, June 20-23, 2005.

[12] OASIS Web Services Business Process Execution Language (WS-BPEL), http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf

[13] Web Services Choreography Description Language (WS-CDL), http://www.w3.org/TR/ws-cdl-10/

[14] Web Service Semantics (WSDL-S), http://www.w3.org/Submission/WSDL-S/

[15] Semantic Markup for Web Services (OWL-S), http://www.w3.org/Submission/OWL-S/

[16] M. Belaunde et al, “Advanced Language for Value added services composition and creation”, IST Spice Project Deliverable D5.1, August 2006. On-line at http://www.ist-spice.org/

[17] Lécué, F., Léger, A., “A formal model for semantic web service composition”, ISWC, 385–398, 2006.