IDECSE: A Semantic Integrated Development Environment for Composite Services Engineering

Ahmed Abid[1] , Nizar Messai[1], Mohsen Rouached[2], Thomas Devogele[1] and Mohamed Abid[3]

[1] LI, University Francois Rabelais Tours, France
ahmed.abid@etu.univ-tours.fr,
{nizar.messai,thomas.devogele}@univ-tours.fr
[2] CCIT, Taif University, Saudi Arabia
m.rouached@tu.edu.sa
[3] CES Laboratory, Sfax University, Tunisia
mohamed.abid@enis.rnu.tn

Abstract. In this paper, we introduce IDECSE a new integrated approach for composite services engineering which is based on Semantic Web and Data Mining. IDECSE considers semantics in all the composition steps: user query, semantic classification of services in the registry, composing services, and verifying the composition process. By considering semantics for describing, discovering, composing, and monitoring services, IDECSE addresses the challenge of fully automating the discovery, composition and monitoring processes while reducing development time and cost. IDECSE appeals for data mining techniques, namely Formal Concept Analysis, for classifying and mining services into service registry in order to anticipate relevant services search and reduce services search space.

Keywords: Web Service Composition, Semantic Web, Data mining.

1 Introduction

Web services are software applications that can be advertised, located and invoked across the Web. Nowadays, an increasing amount of organizations implement their business core and outsource their services over Internet. Thus the ability to effectively select and integrate different services at run-time is an important step towards the development of Web service applications. If no single Web service can satisfy the functionality required by the user, there should be a possibility to create and compose new Web services from existing ones. Considerable academic research and industrial efforts have focused on various aspects of Web service composition ranging from service discovery, to composite service specification and deployment. In this context, important initiatives have been conducted to provide tools and languages that allow an efficient integration of heterogeneous services. Standard languages such as UDDI[4], WSDL[5], and

[4] http://uddi.org/pubs/uddi
[5] http://www.w3.org/TR/wsdl

SOAP[6] were proposed to define standard ways for service discovery, description, and invocation. WSBPEL[7] has focused on representing service compositions where the process flow and bindings between services are known a priori. Later on, following the emergence of the Semantic Web and the fast growth of its related technologies, enhancing Web services description by a semantic level has became one of the basic requirements for efficient services discovery and composition. Since then, several standardization efforts have been done to provide languages which allow to semantically describe Web services on the one hand, and which support efficient automation of the discovery and composition tasks by formal reasoning on services description on the other hand. Standard languages, mainly Web Ontology Language for Services (OWL-S)[8] and Web Service Modeling Ontology (WSMO)[9], were proposed to allow considering semantic aspects in the description and reasoning about services. Based on such languages, many frameworks was proposed for services composition and deployment [1–4]. Despite these efforts and progresses, Web services composition remains a challenging task for the following reasons. First, the number of available services is dramatically increasing. This requires composition frameworks to be accurate, scalable, and reliable to look up and select the most appropriate services for users requirements. Second, services are developed by different organizations based on different types of models and platforms. This heterogeneity in services modeling creates semantic gaps between the presentation of their specification. This requires composition frameworks to provide efficient tools to support bridging such gaps. Third, Web services can be created and updated rapidly. This requires the composition frameworks should be able to dynamically detect and interact with such changes at run-time. Fourth, specifying composition requirements needs the use of high-level languages that are easy to understand, in order to allow end users to express their functional and non-functional requirements in an effective way. Fifth, in case of failure to fulfill user's goal, the composition process should be able to iteratively refining the goal specification in an intuitive way to build composite services. Finally, run-time monitoring and adaptation strategies are primordial to ensure the correctness and the scalability of the composition environments. While numerous composition approaches have been developed, very little has been done towards dealing with these challenges. In this context, this paper introduces IDECSE, Integrated Development Environment for Composite Services Engineering, which considers semantics in all the composition steps: analyzing user query, semantic classification of services in the registry, composing services, and verifying the composition process. This approach aims to provide an easy way to specify functional and non-functional requirements of composite services in a precise and declarative manner, to guide the user through the composition process, while allowing modification or feedback, and finally to enable generating outputs in a deployable language. The rest of the paper is organized

[6] http://www.w3.org/TR/SOAP
[7] http://www-106.ibm.com/developerworks/webservices/library/ws-bpel
[8] http://www.w3.org/Submission/OWL-S
[9] http://www.w3.org/Submission/WSMO

as follows. Section 2, presents the architecture of the proposed framework and details its modules. Section 3 briefly reviews the best known existing approaches before comparing them to IDECSE. Section 4 concludes the paper and outlines current and future work.

2 IDECSE Framework

The IDECSE follows the generic architecture of Web services composition frameworks [5] which contain the following components: Translator, Process Generator, Evaluator, Execution Engine and Service Repository. Figure 1 shows the main parts of a composition framework in a high-level of abstraction (i.e. without considering particular algorithms, languages or platforms). The Service Requester consumes services, following their requirements, offered by service providers, whereas the service provider produces services and puts them into the Service Repository (steps 1 and 2). The Translator translates between the external languages used by the Requester and the internal languages used by the Process Generator (steps 3 and 4). The Process Generator produces plans that combine the available services from the service repository to satisfy the Request (step 5). The evaluator then evaluates all produced plans and returns the best one for execution(steps 6 and 7). Finally, the Execution Engine executes the plan and returns the result to the service requester (step 8).

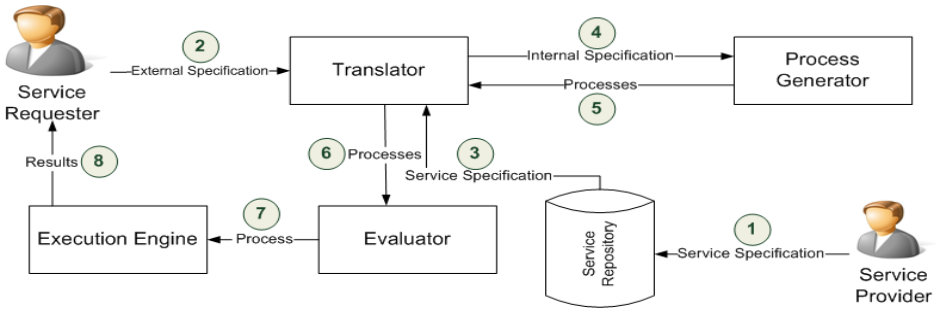

Fig. 1. General framework for Web service composition

IDECSE enhances this architecture to address the challenge of fully integrating semantics in all stages of the composition global life-cycle. First user requirements are more understood using refinement techniques such as generalization or specification of concepts from a given ontology. Second, IDECSE appeals for data mining techniques for classifying and mining services into service registry based on semantic relations. Third, IDECSE is based on two types of reasoners: a similarity-based reasoner and a logic-based one. The IDECSE architecture is depicted in Figure 2. It consists of five modules covering the global composition life-cycle (i.e. specification, classification, composition, deployment, and monitoring). These modules are described in the following sub-sections.

2.1 Service Request Module

The Service Request module translates user requirements to an internal language to be used either by the Service Classification module or the Service Reasoning module. The Graphical Query Editor relies on domain ontology to "understand" the requirements before enriching them through adding new ontology concepts based on semantic relations such as generalization, specialization, etc. The Query is then parsed to extract functional and non-functional requirements. Functional requirements are modeled using the IOPE Extractor, which extract the Input, Output, Precondition and Effects. Non-functional requirements are specified as QoS parameters. Extracted user requirements are then modeled as a new requested service called S_R. Given a domain ontology O, a user query Q modeled as S_R, consists of a set of provided inputs $S_{R_{in}} \subseteq O$, a set of desired outputs $S_{R_{out}} \subseteq O$, a set of preconditions $S_{R_{pre}} \subseteq O$, a set of effects $S_{R_{eff}} \subseteq O$, and a set of quality of service constraints $S_{R_{qos}} = \{(q_1, v_1, w_1), (q_2, v_2, w_2), ..., (q_k, v_k, w_k)\}$, where $q_{i(i=1,2,...,k)}$ is a quality criterion, v_i is the required value for criterion q_i, and w_i is the weight assigned to this criterion such that $\sum_{i=1}^{k} w_i = 1$, and k the number of quality criteria involved in the query. We can model S_R as $S_R = \sum IOPE + \sum QoS$.

2.2 Service Classification Module

To deal with the important number of Web services and instead of considering the whole service registry, this module allows to classify available services semantically into classes according to their similarities. Its second role is to return only relevant services to S_R from the registry. The module contains four components which are Service Projector, Service Description Extractor, Service Similarity Calculator and Relevant Service Selector. The Service Projector selects services capabilities based on syntactical and semantic description for each service (i.e. WSDL and OWL-S description for each service) into one interface. The Service Description Extractor, extracts useful parameters from service capabilities. The Service Similarity Calculator Sub-Module is based on data mining techniques for classifying services into classes according to their relevance and similarity in order to anticipate relevant services search and reduce services search space. The Formal Concept Analysis (FCA) [6] formalism and its extension to complex data called Similarity-based Formal Concept Analysis (SFCA) [7] are used as data mining techniques for this purpose. This module contains three main components:

1. the Context Builder is responsible for preparing the input data-set to the classification module. It selects the main properties of Web Services and creates a tabular representation where the rows correspond to the Web Services, the columns correspond to the services capabilities (descriptions) such as type of input or the ontology that the input refers to, and finally table cells contain values of these properties for each service.

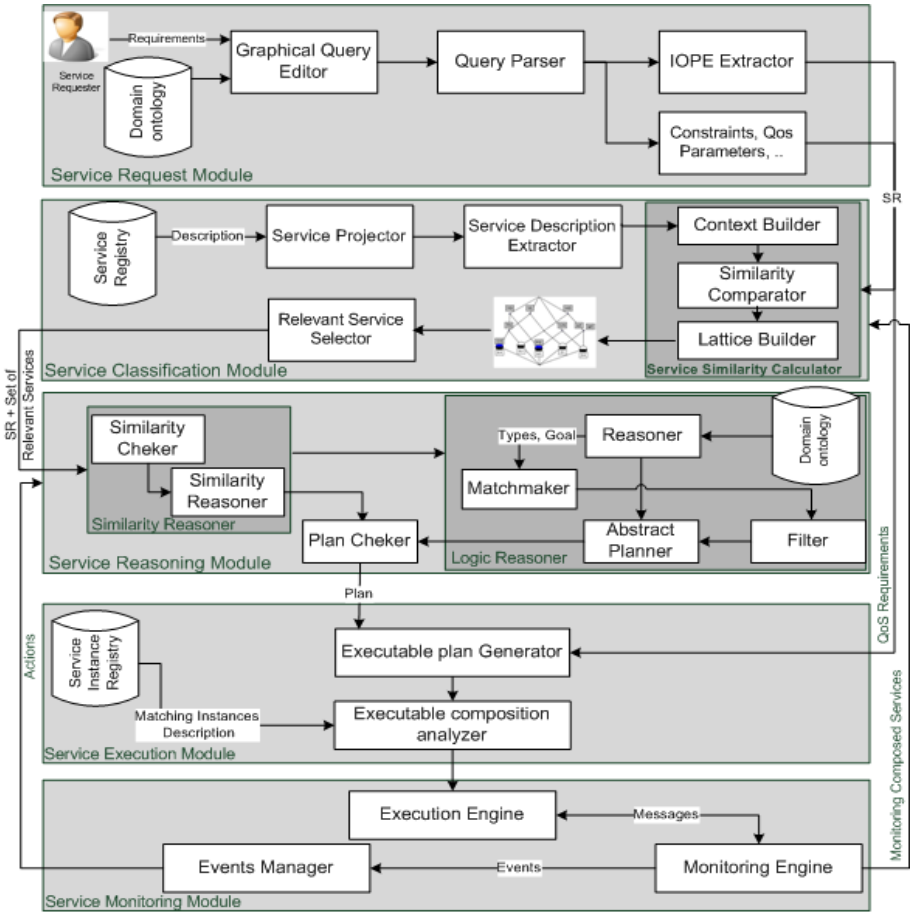

Fig. 2. IDECSE Architecture

2. the Similarity Comparator relies on a set of mathematical formulas based on the semantic distance between concepts from the Context Builder and then between services in the registry.
3. the Lattice Builder enables to compute the lattice structure corresponding to the input table generated by the Context Builder. The lattice structure reflects services grouping possibilities based on their common or similar properties.

Once the lattice is built, services are grouped into classes according to their similarities. The Relevant Service Selector identifies the most relevant classes of services from the lattice (lattice interpreter), which is more likely to answer the query. The set of relevant services can then be outputted with the appropriate rank with respect to the query needs (Service Ranking) to the next module.

2.3 Service Reasoning Module

The Service Reasoning module identifies the candidate composition plans that realize the goal through a similarity-based reasoner or a logic-based one. The Similarity Reasoner is based on the value of semantic similarity calculated by Service Similarity Comparator module. When the semantic similarity value between S_R and one or more existing services in the registry is higher than a given threshold then S_R is satisfied. Otherwise the Logic Reasoner is called to identify the plan of services that achieves the goal of S_R. The main components of the Logic Reasoner module are:

- Reasoner: checks the ontology consistency in addition to handling the maintenance of state including preconditions evaluation and effects application.
- Filter: avoids redundancy from the plan by identifying service types with potential relevance to the goal and checks the dependency relationships between each two consecutive service types.
- Matchmaker: allows querying the service registry for available services in order to match the preconditions of a Web service with the effects of another.
- Abstract Planner: It can be considered as the main component of this module and is responsible for generating a set of abstract plans.

To determine an Abstract Plan, the composition is reduced to a planning problem. A Plan is formalized as a proof of the goal to answer the user query. A Plan $\mathcal{P} = \{A_i\}_{i=1..n}$ is a sequence of n actions. Each action A_i applies on a state E_i to produce a state E_{i+1}: $\forall i \in \{0, .., n-1\}, E_i \wedge A_i \models E_{i+1}$. Starting from the initial state E_0 the plan $\mathcal{P}$ produces the goal G: $E_0 \wedge \mathcal{P} \models G$.

2.4 Service Execution Module

The service Execution Module translates the abstract plan into an executable one by associating to each service type its specific instances using the Service Instances Registry. The plan generated by the logic reasoner is considered as a template for the composite service and drives the process of matching each service type to a corresponding service instance. The Service Execution Module is mainly composed of two main components: The Executable Plan Generator considers non-functional requirements of the goal and enables to concretize the abstract plan generated by the Service Reasoning Module. The Executable Composition Analyzer generates executable code and invokes the execution engine in the Service Monitoring module. Different works was proposed in order to implement the Executable Plan Generator, we can use for example the algorithm presented in [8]. This algorithm takes as input a composition plan, the QoS permissible values imposed by the user, and their weights and generates as output a composition plan that satisfies the requirements of the user.

2.5 Service Monitoring Module

Monitoring deals with the actual execution of the composite service and is responsible for monitoring the execution and recording violation of any requirement of the goal service at runtime. If a violation event occurs, an adaptation

engine is triggered to handle this violation. There are two inputs to a monitoring framework, a set of constraints that the process must obey and events or messages generated during the execution. A processing engine ensures that all events comply with the constraints and reports any exception.

3 Related Works

A considerable number of research efforts have focused on various aspects of Web service compositions ranging from semantic service discovery to semantic specification, deployment, and monitoring. MoSCoE (Modeling Web Service Composition and Execution) [3] aims to provide a model-driven framework for an automatically composition of services. MoSCoE allows service providers to publish their services in a standard and semantic way and uses UML state machines for visually representing composite services. The Composition process in MoSCoE is based on the three steps of abstraction, composition and refinement. However user preferences and QoS were not addressed but only outlined as future work. In [9], authors combine semantic service descriptions with the invocations of the WSDL descriptions allowing to execute the composed services on the Web. The process includes matching services to the user at each step of a composition and filtering the possibilities by using semantic descriptions of the services. The generated composition is then directly executable through the WSDL grounding of the services. [10] presents a framework for service composition based on functional aspects, in which services are chained according to their functional and semantic description (IOPEs). The proposed framework uses the Causal Link Matrix (CLM) formalism in order to facilitate the computation of the final service composition as a semantic graph. The set of possible solutions are then pruned, at composition time, in order to rank the service compositions according to some fixed criteria. These criteria can be defined based on the semantic similarity of component services and/or the non-functional properties of the compositions calculated by aggregating the non-functional properties of the component services. In [11] and [12], Description Logics (DL) frameworks for Semantic Web service composition are proposed. The specification of Semantic Web services is reduced to preconditions, which define logical conditions that should be satisfied prior to the service invocation and effects, which are the result of the execution of the service. The proposed approaches for services composition use backward chaining search algorithms to find potential candidate services. Thus the composition process is done automatically and dynamically.

IDECSE builds on the approaches mentioned above and aims to provide a declarative approach to service composition engineering to achieve a full manipulation of the composition process. IDECSE appeals for data mining techniques for classifying and mining services into service registry in order to anticipate relevant services search and reduce services search space. It also considers monitoring and adaptation concerns, which are not incorporated in the above approaches. IDECSE provides an easy way to specify functional and non-functional require-

ments of composite services in a semantic and declarative manner, guides the user through the composition process, while allowing modifications or feedbacks.

4 Conclusion

This paper describes IDECSE, a new semantic integrated approach for composite services engineering. Compared to existing approaches IDECSE considers semantics in all the composition global life-cycle, addresses the challenge of fully automating the composition processes, uses data mining techniques such as SFCA for classifying and mining services, and proposes new reasoning, monitoring, and adaptation techniques. Our work in progress includes the improvement and implementation of the different modules of the proposed architecture. We also plan to extend the framework to include additional features such as failure handling, and an interactive visual environment for testing composite services.

References

1. Batra, V.S., Batra, N.: Improving Web service QoS for wireless pervasive devices. In: 2005 IEEE International Conference on Web Services, IEEE (2005) 130–137
2. Bener, A.B., Ozadali, V., Ilhan, E.S.: Semantic matchmaker with precondition and effect matching using SWRL. Expert Systems with Applications (2009) 9371–9377
3. Pathak, J., Basu, S., Lutz, R., Honavar, V.: MoSCoE: A framework for modeling Web service composition and execution. In: Data Engineering Workshops, 2006., IEEE (2006)
4. Kona, S., Bansal, A., Gupta, G., Hite, D.: Automatic composition of semantic Web services. In: ICWS 07. (2007) 150–158
5. Rao, J., Su, X.: A survey of automated Web service composition methods. In: Semantic Web Services and Web Process Composition. Springer (2005) 43–54
6. Ganter, B., Wille, R.: Formal Concept Analysis: mathematical foundations. 1999
7. Azmeh, Z., Hamoui, F., Huchard, M., Messai, N., Tibermacine, C., Urtado, C., Vauttier, S.: Backing composite Web services using Formal Concept Analysis. In: Formal Concept Analysis. Springer (2011) 26–41
8. Ko, J.M., Kim, C.O., Kwon, I.H.: Quality-of-service oriented Web service composition algorithm and planning architecture. Journal of Systems and Software (11) (2008) 2079–2090
9. Sirin, E., Hendler, J., Parsia, B.: Semi-automatic composition of Web services using semantic descriptions. WSMAI (2003) 17–24
10. Lécué, F., Silva, E., Pires, L.F.: A framework for dynamic Web services composition. In: Emerging Web Services Technology, Volume II. Springer (2008) 59–75
11. Qiu, L., Lin, F., Wan, C., Shi, Z.: Semantic Web services composition using AI planning of Description Logics. In: IEEE Asia-Pacific Conference on Services Computing, 2006, IEEE (2006) 340–347
12. Wang, Y.: Research on Web service composition algorithm using Description Logics. TELKOMNIKA Indonesian Journal of Electrical Engineering (1) (2014) 852–858