EasyApp: Goal-Driven Service Flow Generator with Semantic Web Service Technologies

Yoo-mi Park[1], Yuchul Jung[1], HyunKyung Yoo[1], Hyunjoo Bae[1], and Hwa-Sung Kim[2]

[1] Service Convergence Research Team, Service Platform Department, Internet Research Laboratory, ETRI, Korea
{parkym,jyc77,hkyoo,hjbae}@etri.re.kr
[2] Dept. of Electronics & Communications Eng., KwangWoon Univ., Korea
hwkim@kw.ac.kr

Abstract. EasyApp is a goal-driven service flow generator based on semantic web service annotation and discovery technologies. The purpose of EasyApp is to provide application creation environment for software programmers to make new application semi-automatically by enabling the semantic composition of web services on the Web. In this demo, we introduce key technologies of EasyApp to overcome the problems of the previous work on semantic web service technologies. Demonstration of use case 'hiring process' shows that EasyApp helps software developers make easily a service flow with key technologies: ontology-based goal analysis, semantic service annotation, semantic service discovery, and goal-driven service flow generation.

Keywords: goal-driven service flow generation, semantic web service annotation, semantic web service discovery, semantic web service composition.

1 Introduction

Semantic Web Service is a new paradigm to combine the Semantic Web and Web Services. It is expected to support dynamic computation of services as well as distributed computation. Ultimately, on Web Service, it leads to goal-based computing which is fully declarative in nature [1-3].

The previous researches on Semantic Web Services are OWL-S [4,6] and WSMO [5,6]. They suggested new service models and description languages with ontology for goal-based computing. However, these semantic web service approaches using the new models and languages require expertise on ontology and lots of manual work even for experts. In addition, they dealt with WSDL-based web services rather than RESTful web services which are commonly used in the industry recently. These limitations make it difficult to respond fast to the dynamically changing web service world that exist more than 30,000 services [8,9].

In this demo paper, we introduce a goal-driven service flow generator (i.e., EasyApp) with novel semantic web service technologies that can be applied to the currently existing web services without any changes. Towards automatic service composition, especially, we have considered semi-automatic service annotation, goal-driven semantic service discovery, and automatic service flow generation based on our goal ontology as its key enabling technologies. The key technologies can be applied for both WSDL-based services and RESTful services. In the use case ‘hiring process’, we show that the software developer makes service flows which satisfy his/her goals on EasyApp where our novel semantic web service technologies are embedded.

2 EasyApp

2.1 System Overview

EasyApp is a goal-driven service flow generator based on semantic web service annotation and discovery technologies. It provides application creation environment for a user to make new application by enabling semantic service composition of web services on the Web. Potential users of EasyApp are both expert software programmers and beginners.

Fig.1 shows service architecture of EasyApp and its logical functional entities that are ‘goal analyzer’, ‘semantic service discovery’, ‘service flow generator’, and ‘source code generator’. On EasyApp, a user inputs ‘user goal’ and finally gets a service flow satisfied with the given goal. The user goal is parsed by referencing ‘goal pattern library’ and decomposed into corresponding sub-goals by the ‘goal analyzer’. Each sub-goal is a criterion of the ‘semantic service discovery’. The ‘semantic service discovery’ discovers and ranks relevant services among semantically annotated services stored in the registry. With the result of semantic service discovery, the ‘service flow generator’ makes up a basic service flow. After that, the user can refine the service flow manually and generate a template code in Java and XML by ‘source code generator’ automatically. EasyApp is strongly devoted by semantic service annotator, which is explained in the next section.

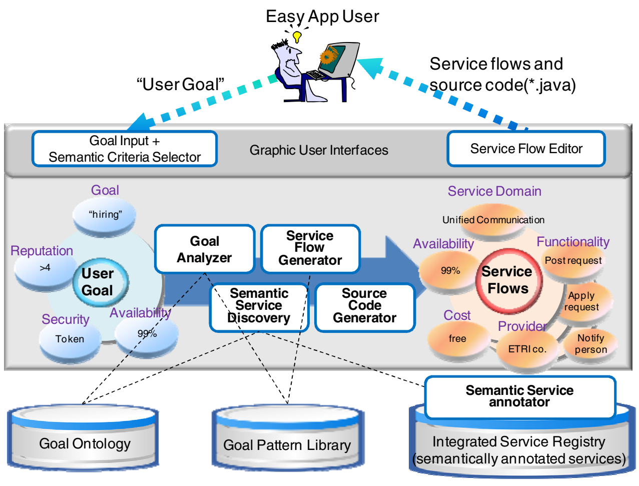

Fig. 1. Service Architecture of EasyApp

2.2 Key Technologies

We address the key technologies of this demo as follows:

- Semantic service annotation: It involves tagging of services with additional semantic information so that service annotation becomes more meaningful and clear. We focus on the semantic annotation of web services, especially by considering their data semantics, functional semantics, and non-functional semantics about a target service as follows:
  - Data semantics : input and output parameters
  - Functional semantics: name, category, provider, location, description, and country
  - Non-functional semantics: availability, cost, security level, user rating, and response time

  We provide a semi-automatic semantic service annotation environment by supporting automatically crawled meta-information of each web service and a step-wise annotation on web user interface.
- Ontology-based goal analysis: It generates a set of sub-goals which can fulfill the user-specified goal with goal ontology and goal pattern library. In case of goal ontology, the goals widely used in the communication field are selected and constructed as OWL ontology. The goal ontology covers synonyms, hypernyms, and hyponyms. The goal pattern library stores a set of sequential sub-goals that can meet high-level goals such as "hiring a person", "making group meeting", and "planning business trip".
- Semantic service discovery: It performs a goal-driven matchmaking on semantically annotated services. In case of semantic service discovery, we employ the goal-driven service matchmaking which considers textual similarity, functional similarity [7], and constraints of non-functional properties (NFP) simultaneously. It can effectively deal with various types of service description and dynamically changing QoS of web services according to the user-specified goal.
- Goal driven service flow generator: With the result of semantic service discovery, it composes the discovered web services automatically without user's intervention. It can be achieved by referring to goal pattern library. The goal pattern library is built with well-structured goal patterns which are composed of goal and relevant sub-goals.

3 Use Case – Hiring Process

Suppose that a software programmer wants to develop a 'hiring process for a company' using some web services in the internet and intranet. The 'hiring process for a company' should handle three party communication channels among requester, applicant, and broker. Even in a small company, there is a person who requests to hire a new employee, and a person who is responsible for hiring a person, and applicants who want to apply the company. Fig. 2 shows a process of making new 'hiring process' application on EasyApp with a simple goal.

First of all, a developer catches a keyword in mind describing his/her goal, which can be ‘job’, ‘work’, or ‘hiring’. He/she inputs the keyword in the goal box (1). Based on the given keyword, EasyApp extends keywords (e.g. ‘vocation’, ‘occupation’, ‘career’, ‘recruiting’, ‘resume’, etc) as its substitutes from goal ontology and then looks up the relevant goals with the extended keywords from goal pattern library. After goal analysis, EasyApp suggests several candidate goals which concretize the user’s keyword to the developer. The suggested goals are ‘hiring a person’, ‘recruiting people’, ‘offering a job’, ‘getting a job’, and ‘making a resume’. He/she can select an appropriate goal, which is ‘hiring a person’ among them (2). Then, he/she can select additional criteria (non-functional semantics mentioned in Section 2.2) (3) to make his/her requirement clearer. When he/she clicks ‘search’ button (4), EasyApp decomposes the goal into sub-goals using goal pattern library and makes up a basic service flow that is composed of sequential sub-goals. Then, EasyApp discovers relevant services which satisfy sub-goals through semantic service discovery. During the semantic service discovery, top-k services are ranked based on a weighted sum of the textual similarity score given by keyword search and the functional similarity that represents the goal achievability of the given service. The top-k services are re-ranked by the weighted sum of each NFP’s weight and its importance. After the discovery, a service flow is displayed in the service flow editor (5).

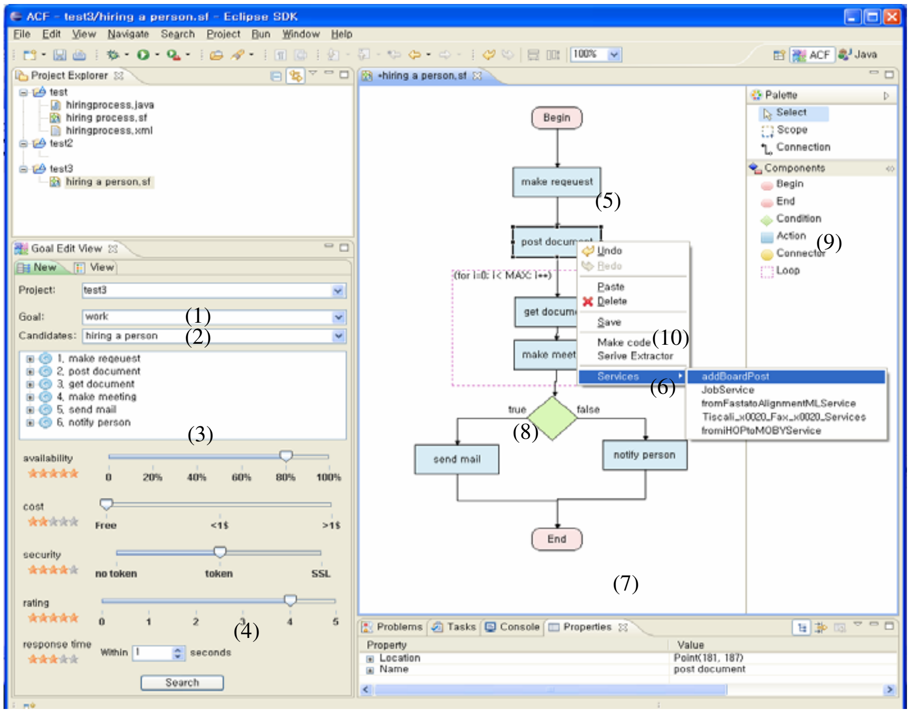

Fig. 2. Making service flow for ‘Hiring a person’ on EasyApp

In this use case, a service flow consists of the following sub-goals: ‘make request’ (for requester to make hiring request to the broker in company) → ‘post document’ (for requester to upload a required specification) → ‘get document’ (for broker to get the required specification) → ‘make meeting’ (for requester to meet the applicants) → ‘notify person’ (for requester to send result to the applicants). The sub-goals include finally ranked top-k services as a result of service discovery.

The developer can choose the most appropriate service (6) by referring to service properties represented in the ‘property’ view (7). He/she can modify the service flow in the editor when he/she wants by dragging & dropping activity icons (8) on palette (9). After the developer finishes the selection of services and the modification of service flows, he/she can obtain java code from the service flow (10). Finally, the developer gets a service flow for ‘hiring process’ on EasyApp.

4 Conclusions

In this demo, we present EasyApp, which is a novel and practical semantic web service composition environment. With EasyApp, software developer can make service flows that match the targeting goal regardless of web service programming proficiency. Further work is to employ semantic service mediation technology for on-the-fly service discovery and composition of web services.

Acknowledgments. This work was supported by the IT R&D program of KCC/KCA [KI002076, Development of Customer Oriented Convergent Service Common Platform Technology based on Network].

References

1. Fensel, D., Kerrigan, M., Zaremba, M.: Implementing Semantic Web Services. Springer, Heidelberg (2008)
2. Preist, C.: A Conceptual Architecture for Semantic Web Services. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 395–409. Springer, Heidelberg (2004)
3. Cabral, L., Domingue, J., Motta, E., Payne, T.R., Hakimpour, F.: Approaches to Semantic Web Services: an Overview and Comparisons. In: Bussler, C.J., Davies, J., Fensel, D., Studer, R. (eds.) ESWS 2004. LNCS, vol. 3053, pp. 225–239. Springer, Heidelberg (2004)
4. Web Service Modeling Ontology (WSMO), `http://www.w3.org/Submission/WSMO/`
5. Semantic Markup for Web Services (OWL-S), `http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/`
6. Lara, R., Roman, D., Polleres, A., Fensel, D.: A Conceptual Comparison of WSMO and OWL-S. In: European Conference on Web Services, pp. 254–269 (2004)
7. Shin, D.-H., Lee, K.-H., Suda, T.: Automated Generation of Composite Web Services based on Functional Semantics. Journal of Web Semantics 7(4), 332–343 (2009)
8. Seekda, `http://webservices.seekda.com/`
9. ProgrammableWeb, `http://www.programmableweb.com/`