CORE
Metadata, citation and similar papers at core.ac.uk
Provided by The IT University of Copenhagen's Repository

Goal-Driven Context-aware Service Composition

Lian Yu, Arne Glenstrup[1], Yang Zhang, Shuang Su
School of Software and Electronics, Peking University, Beijing, 102600, PRC
lianyu@ss.pku.edu.cn, panic@itu.dk
[1]IT University of Copenhagen, Denmark

Abstract

Two important aspects are associated with service composition. One is to understand the needs and constraints for a new added-value composite service, and otherwise it would lead to an ad-hoc effort for service composition. The second is to reflect the changes of computing environment to the service composition to catch up the on-demand of users. This paper introduces a goal-driven approach to specify the user requirements and demands that guides the service composition, and proposes context awareness to adapt to a dynamically changing environment. Computing contexts, including physical context, user profile and computed results, are gathered by various services, and imported into an ontology based a context repository. A Goal Description Language, Context Condition/Effect are designed to describe the dynamic semantics of goal requirements and service capability. A planner is designed and implemented to dynamically compose services based on the current contexts, and a service runner is designed and implemented to invoke proper services based on the contexts and interactions with users.

Keywords: Context Awareness, Goal Driven, Ontology, Service Composition, Reasoning

1. Introduction

With the proliferation of computing environments and the qualitative improvement in the characteristics of networking systems, the number and variety of computing services has grown significantly. Services are expected to become more intelligent and complex: they are supposed to perform almost any imaginable task on behalf of users, accomplish sophisticated procedures and transactions without human assistance, and sometimes even express ambient awareness and proactivity. However, users' expectations often surpass real features of current services and bring higher demands for services' usability. Ideally, users want to get complex and value-added functionality in a single click or even without any interference.

This leads us to the situation in which services tend to become more and more complex. Services will suffer in stability and reliability while getting more and more complicated. At the same time, users demand added value functionality that is specific functionality provided to a certain user with respect to her specific needs. A good value-added service should be customized to meet the requirements of each particular user.

A goal-driven approach allows domain experts to specify their needs in a systematic way, and users just select their target goals as they want. Context-awareness helps services to acquire additional knowledge about the current situation and surrounding environment. Adaptability allows services to adjust their behavior with respect to certain conditions and effects.

Context awareness is a term from ubiquitous computing or pervasive computing which expresses adaptation to continuous changes in the environment [3] . For the purpose of making composed *services in context* automatically adapt to changing contexts, it is also imperative to take context information into consideration in service composition. Context information considered in this paper includes physical context information (e.g., light, noise, weather, location and status), user context information (e.g., name, age, sex and preference) and computing context information (e.g., service availability and service computing results). The context information above can be obtained and adopted in the service composition process.

Combination of goal-driven and context-aware approaches is an integral solution for building value-added services for a variety of users and circumstances. This paper proposes a goal-driven approach to plan a proper service sequence, and performs service execution based on current contexts.

1.1. Motivation Example

To illustrate our approach, we consider a service-oriented medical diagnosis system as a use case for context aware service composition (CASC). We assume that patients come with handheld mobile devices that can connect to the Internet, such as PDAs or G3 mobile

[1] The research is supported by Danish Strategic Research Council (No 2106-08-0046) and by the National Science Foundation of China (No. 60973001).
978-1-4244-9142-1/10/$26.00©2010 IEEE

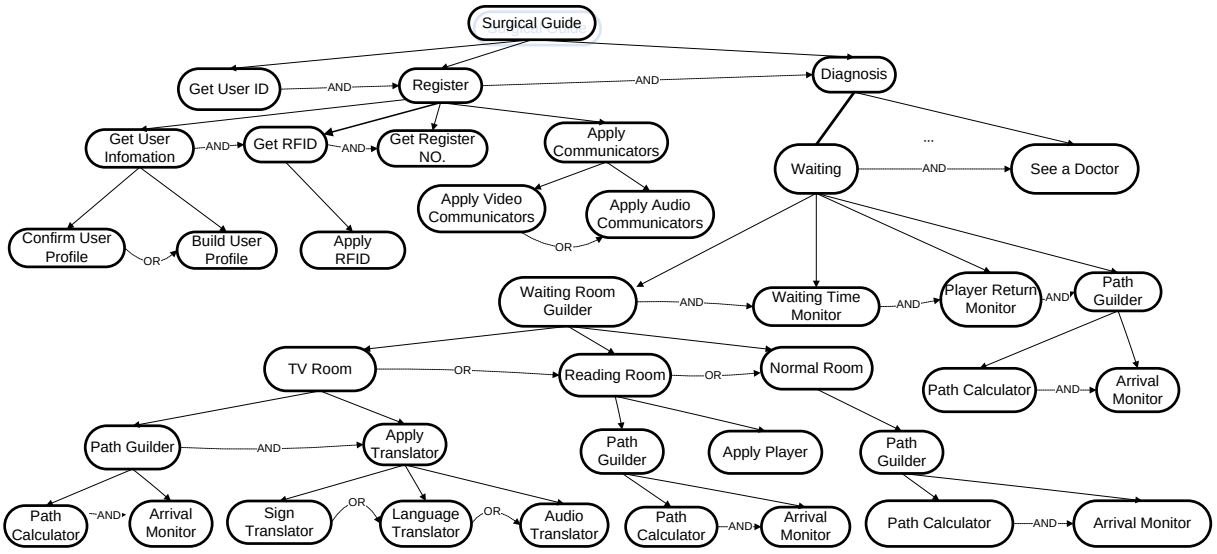

Figure 1. Goal Tree of "Surgical Guide"

phones. The hospital provides the system and the wireless network coverage creating a pervasive environment that has a number of web services running in it performing specialized medical tasks.

Further, we assume a patient wants to get a surgical diagnosis guide service. Traditionally, medical guide services are provided manually. But this wastes a lot of human resources, and sometimes it is inconvenient to patients due to the unavailability of human's help. Here the CASC system can enhance quality and efficiency of a hospital diagnosis guide service. When the user confirms her goal at each place with the hospital guidance, she is then presented, on her handheld mobile device, with a service composition result path that has to be performed to achieve the goal and the services in the service path are then automatically executed one by one.

For illustration, Figure 1 shows the goal tree of "Surgical Guide". One of the scenarios and its resolution given by CASC system are listed as follows:

Scenario: The patient visits the hospital for the first time. At that moment, many patients are waiting for surgical treatment. The patient is in favor of waiting in a quiet waiting room. CASC will recommend a service composition approach as: *"Get User ID"* → *"Build User Profile"* → *"Apply RFID"* → *"Get Register Number"* → *"Apply Video Communicators"* → *"Path Calculator"* → *"Arrival Monitor"* → *"Waiting Time Monitor"* → *"Path Calculator"* → *"Arrival Monitor"* → *"See a Doctor"*.

Figure 2 shows how the CASC system performs planning according to certain contexts and obtains an appropriate service path and executes the services.

(1) Context World is the container of the context facts, which is constructed according to the context domain description script. The domain concepts' definition and the facts in the context world are the foundation of planning to achieve the goal. Context information can be imported into a certain context world during the planning and running by context acquisition services and user inputs from UI.

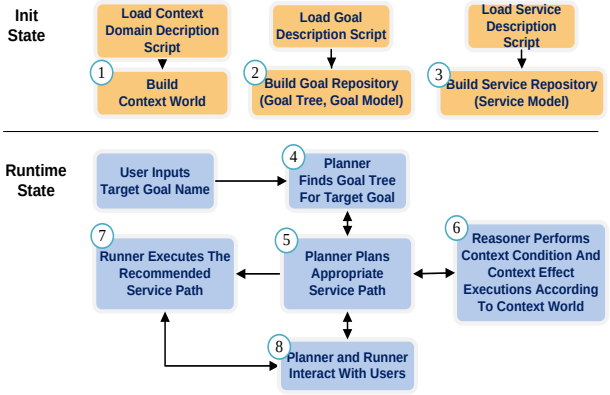

Figure 2. Work Flow of CASC System

(2) Domain requirement is abstracted as Goal. A Goal Description includes the description of the conditions and constraints of a certain task and how to achieve the task by achieving a set of sub-goals. Figure 1 shows a goal tree of the final goal "Surgical Guide". Goal Descriptions can be created by domain experts, and imported into this system. The details about Goals will be described in Section 2.

(3) The semantics of services in context and the interaction rule between services and this system are described by the Service Description, which can be imported into this system. Semantics of services mainly provides the formal semantic description about input, output, precondition and effect. The details about Service Description will be described in Section 2.

(4) At run time state, the system serves end-users beginning with the users input of target goal name, and the planner will find the target goal.

(5) The planner starts the planning process from the target goal, using a planning algorithm to plan an appropriate service path for the end-users according to context information.
(6) During the planning process, context conditions in the goal model and the service model will be executed to obtain the condition evaluation result according to the facts in the context world. Context effects in the service model will be executed to modify the facts in the context world.
(7) The service path obtained by the planner will be executed by a service runner.
(8) During the process of planning and running, the system will interact with end-users to get user input or affect users.

1.2. System Architecture

The system architecture is depicted in Figure 3, which includes three layers: Planner and Runner Layer, Context Reasoner Layer, and Service Layer.

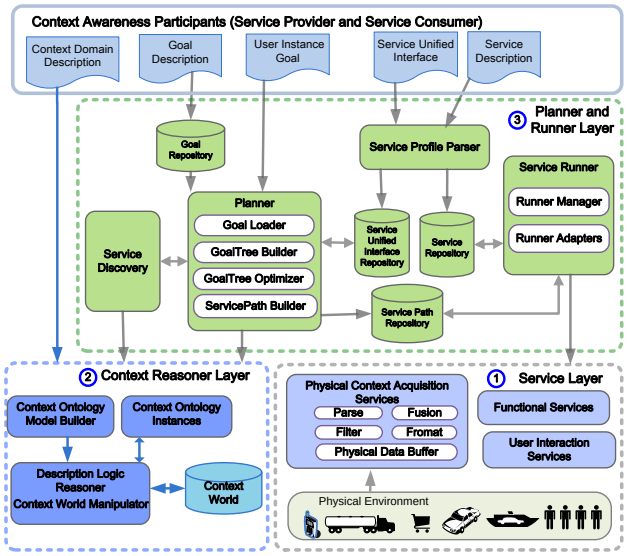

Figure 3. Architecture of CASC System

Service Layer: This layer is responsible for gathering context information and serving end-users. In this system, the physical context information is obtained by Physical Context Acquisition Services, which is the upper level of Physical Sensor systems. Context information from the physical environment can be gathered by sensors, e.g., infrared sensors, RFID systems, and microphones. Physical Context Acquisition Services monitor the context data in the physical data buffer, and filter, fuse and format them into unified context information that can be imported into the context world for planning. User Information Gathering Services gather the profile information of users directly by user interface. Functional Services refer to all these services that work for end-users. The semantics descriptions of all of the services are the foundation of planning. The CASC system interacts with those services with customizable interfaces.

Context Reasoner Layer: The Context Reasoner Layer consists of reasoning related modules. The domain concept model can be constructed according to the Context Domain Description script, and the Context World can be instantiated as a facts' container. The modules of the context condition/effect builders, and the context condition/effect executors are responsible for interpreting or executing a context condition and effect script, which is written in a goal description or service description. All of the formatted context information can be imported into the context world by the context world updater. During the planning and running process, the instance set of each goal node is organized and manipulated by the Process Handler module, and services are selected by the Service Selector module.

Planner and Runner Layer: This layer is responsible for planning a service path and running services. The description of goals and services is interpreted and stored in a repository. The planning process is triggered by a user requirement from a user interface. During the planning process, Context Reasoner modules and relevant services will be invoked. The planner generates a service path according to the relevant goal and service definition as well as the current context for an end-user, and the runner is responsible for executing each service in the path and interacting with the end-user by communication devices.

2. Goal Modeling and Service Modeling

The Goal Model and Service Model are proposed to describe formal presentation of services and user goals in an easily-understandable and editable manner.

2.1. Goal Model

The Goal Model focuses on eliciting user requirements in an accurate and unified way. It formalizes the user goal, and defines the way of how to configure and save it. The Goal Model is the most basic and important element in the service composition planning process. According to the goal decomposition and context information, a service composition path is generated and shown to end-users.

Figure 4 shows the Goal Model. Properties of a Goal Model are Goal ID, Goal Name, Goal Description, Goal Type, Priority, Constraint, Effect and Goal Computation. Composite Goal contains information about its sub goals and their relation. There are three types of relation:
(1) Sequence: Realization of Composite Goal can be achieved by realization of all sub goals according to a strict order. If one of the sub goals fails, the Composite Goal fails.
(2) Concurrent: Realization of Composite Goal can be achieved by realization of all sub goals in a random order. If one of the sub goals fails, the Composite Goal fails.
(3) Alternative: Realization of Composite Goal can be

achieved by realization of any one of the sub-goals. The sub-goal that comes first will be taken into consideration first too.

The goal model is stored in an XML file called the goal configuration file.

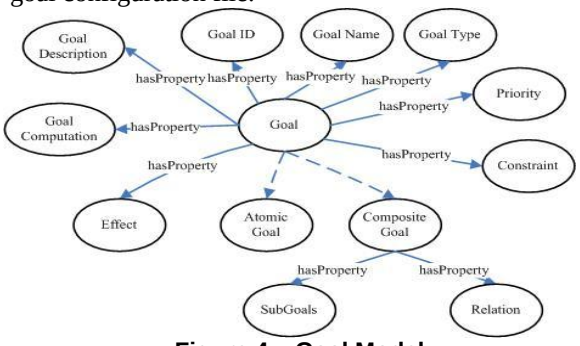

Figure 4. Goal Model

According to the relationship between a composite goal and its sub goals, a composite goal can be decomposed. This process is repeated until none of them can be further decomposed. After the decomposition process, a Goal Tree is generated.

2.2. Goal Model Configuration

Instances of a goal model are stored in an XML file called goal configuration file. It is edited by domain experts. During the service planning process, they are imported first. They are used to interpret goals, decompose composite goals and generate goal tree. The structure of the configuration file is shown in Figure 5.

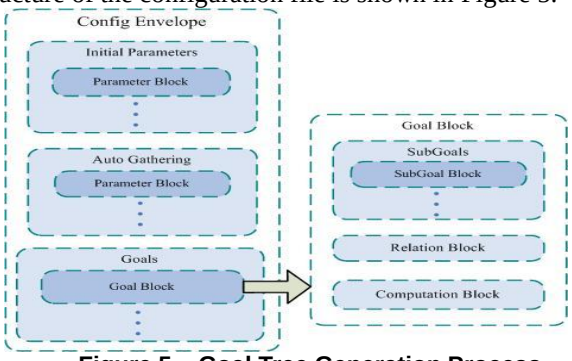

Figure 5. Goal Tree Generation Process

Each configuration file corresponds to instances of a high level composite goal and all its sub goals. The goal configuration file uses "Config Envelope" as its root element which consists of an Initial Parameters Block, an Auto Gathering Block and a Goals Block.

The Initial Parameters Block records the static context information, which includes an arbitrary number of Parameter Blocks. The Auto Gathering Block records the name and the access method of context that can be used during the service composition process, it also include an arbitrary number of Parameter Blocks. Each "Parameter Block" records the name of the context object and its acquisition approach. The Goals Block records all instances of the goal model, including the combination of all atomic goals and composite goals. Each instance of a goal model is described in a Goal Block.

A Goal Block contains a description of sub-goal block (SubGoals), relationship of sub-goal (Relation Block) and the computation information (Computation Block). A sub-goal block records the names of all sub-goals and access conditions, each sub-goal description is recorded as SubGoal Block.

2.3. Goal Tree Generation

According to the relationship between a composite goal and its sub goals, a composite goal can be decomposed. The process is repeated until none of them can be further decomposed. After the decomposition process, a Goal Tree is generated.

- First of all, composite goal which is identified as "001" can be further broken down into sub-goals "002", "003" and "005". The relations between sub-goals are Sequence. Based on the above information, we can construct a sub-tree which using "001" as root, as shown in Figure 6.

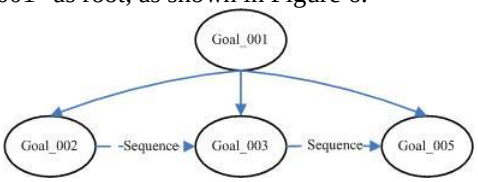

Figure 6. Goal Tree Generation Process

- Then, based on the goal configuration file, each goal will be further decomposed using the above approach in accordance with the sub-goal tree structure one by one. The process is repeated until none of them can be further decomposed.
- Finally, all the sub-trees are combined, creating a goal tree using the high-level goal as the root, as shown in Figure 7.

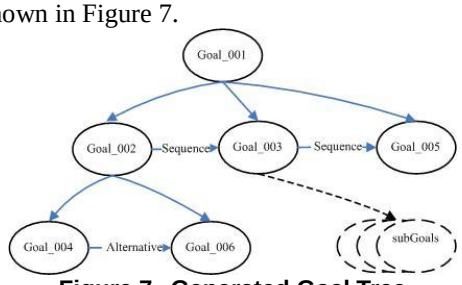

Figure 7. Generated Goal Tree

2.4. Service Model

The Service Model describes the service semantics adopted in this system; it focuses on input, output, precondition and effect.

Input: Input is declared with a variable name and its context object class defined in the context world model. These input variables can be bound with instances from the context world during the planning and running process.

Precondition: The precondition block describes the constraints which should be satisfied before executing the service. The Context Condition Description Language is used in this block.

Product: The product block describes the data that the service can produce and the effect that the service will have. The Context Effect Description Language is used in the effect block.

3. Context-aware Service Composition System

The CASC System adopts context-awareness in service composition to construct SOA based systems. The architecture of the CASC is presented in Section 1. This section will give the detailed design of the Planner and Runner Layer. Meanwhile, a Unified Description of Service Interface is proposed to associate parameters of services with ontology objects or ontology attributes in the context world.

3.1. Unified Description of Service Interface

The Planner and Runner need to prepare parameters for calling the implementation of a service. So it needs a mapping approach to link the parameters to ontology objects or ontology attributes. The Unified Description of Service Interface (UDSI) is used to solve this problem. UDSI is coded as an XML file.

UDSI consists of one or more Service Blocks. Each Service Block represents a service which includes one optional Engine Block and one or more Operation Blocks. Each Operation Block describes one operation in the service; it can bind different Ontology Blocks in different domains. Each Ontology Block contains mapping methods between ontology objects and service parameters (Input Block) plus returned results (Output Block).

3.2. Planner Module

Planning in the CASC is implemented as a Service Composition Planner Module (Planner) in this paper. The main function of the Planner Module is recommending a most suitable service sequence for users according to current contexts. The architectural design of the Planner module consists of the following components:

Goal Loader: Loads a Goal Tree based on the user's input. If the target Goal Tree does not yet exist, then calls GoalTree Builder to construct it.

GoalTree Builder: Builds Goal Tree based on service composition objectives and the goal configuration files.

GoalTree Optimizer: Prunes the Goal Tree according to whether the constraint of the goal meets the current context environment, and removes any service path that cannot be implemented in the current situation.

ServicePath Builder: Matches services for goals based on the current context, and this process is continued until the ServicePath Builder has access to an executable service path.

3.3. Runner Module

Running in CASC is implemented as a Service Composition Runner Module (Runner) in this paper. The main function of Runner is responsible for running the service path that is recommended by the Planner dynamically. The Runner dispatches all kinds of services in a unified manner according to the Unified Description of Service Interface.

Runner Manager: It is the entry point of the Runner Module, which receives requests for running a service path and manages concurrent Service Path Unit threads.

Service Path Execution Unit (SPEU): Completes the scheduling request from the Runner Manager. It executes services in the service path using an appropriate Adapter that implements a Runner Adapter Interface according to the service engine. SPEU works until all services in the service path have been executed successfully.

Web Service Handler: Prepares parameters for services using the Unified Description of Service Interface.

Runner Adapters: Generate an appropriate Web Service client.

3.4. Demonstration

First, a patient triggers the system to execute the service composition function. Then, during the CASC planning phase, CASC plans for patients according to his special needs and the current context information. CASC gets the user's profile from the patient and finally gives him a resulting service composition path. During the running phase, CASC executes each service in the service composition path until all services have already been finished. Context information is also used to guide service running in the CASC running phase. Figure 8 shows some of implementation screenshots, including a user specifies its target goal; the system assists the user to create its profile, and the system gives the planner results and allows the user to interact with the system during the planning execution.

4. Related Work

Axel van Lamsweerde [6] puts forwards a guide tour for goals which compares the main approaches to goal modeling, goal specification and goal-based reasoning in the many activities of the requirements engineering process. He gives a general picture of goal and goal driven concepts and the necessity of using a goal driven manner to elicit user requirements.

Axel van Lamsweerde [7] presents the issues of handling obstacles in goal-oriented requirement engineering. After that he proposes a way of managing conflicts in goal-driven requirement engineering [8] , which is very important in goal modeling and goal

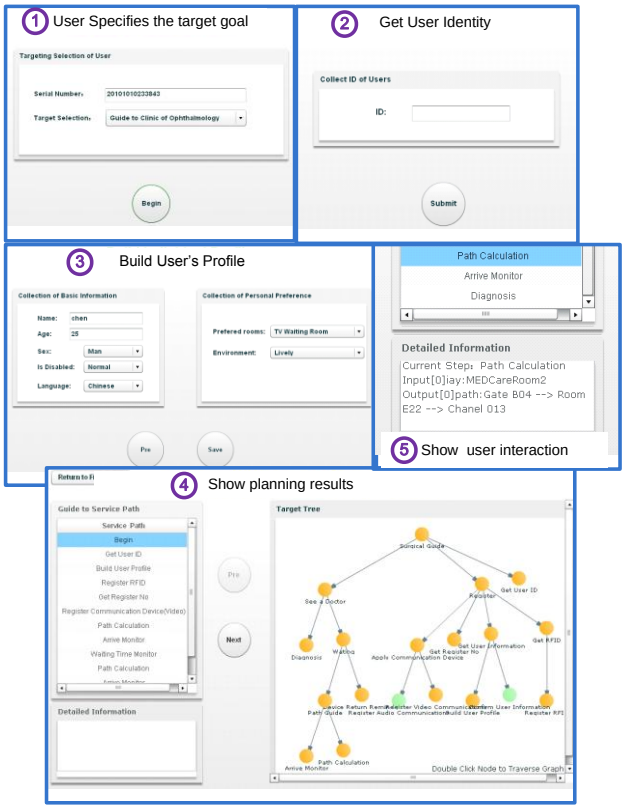

Figure 8. Planner and Runner at Run-Time

reasoning. Finally, he describes the whole roundtrip from research to practice proceedings [9] .

Rim et al [10] propose a goal driven approach to understand the needs of different organizations for a new added-value composite service and to model the cooperative process supporting this service provisioning in a declarative, goal driven manner. They propose a goal model called Map which is used for service elicitation, distribution and orchestration.

Kangkang et al. [11] present a goal-driven approach of service composition which builds a task-oriented semantic representation model of web services. Juan Miguel Gómez et. al. [12] presents a best of breed ultimate engine which names GODO, it can use natural language processing and mapping techniques for orchestrating goals.

5. Conclusion and Future Work

This paper presents the design and implementation of a Context aware Service Composition (CASC) system to fulfill the increasing demands of environmental adaptation of SOA systems. The CASC system provides a methodology to model domain concepts, abstract domain requirements and service capabilities, and can calculate proper service orchestration according to changing contexts for end-users. During the design and implementation of this system, several techniques are designed or extended. A Context Ontology Model is proposed to model domain concepts in context. A Context Condition and Context Effect are proposed to describe dynamic semantics in an editable and easily-modifiable script manner. The Service Model and Goal Model are proposed to describe a formal presentation of services and a user goal in easily-understandable and editable manner. According to goal decomposition and context information, a service composition path is generated and shown to end-users.

References

[1] Manshan Lin, Heqing Guo, Jianfei Yin. Goal Description Language for Semantic Web Service Automatic Composition [C]. IEEE Computer Society Press, 2005: 190-196.

[2] Dmytro Zhovtobryukh. A Petri Net-Based Approach for Automated Goal-Driven Web Service Composition [J].

[3] B. Schilit, N. Adams, and R. Want. Context-aware computing applications. In proceedings of the IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'94), pp. 89–101. IEEE Computer Society 1994.

[4] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, Yarden Katz. Pellet: A practical OWL-DL reasoner [J]. Web Semantics: Science, Services and Agents on the World Wide Web. 2007,5(2): 51-53.

[5] Takashi Hattori, Kaoru Hiramatsu, Takeshi Okadome, Bijan Parsia, Evren Sirin. Ichigen-san: An ontology-based information retrieval system [C]. WWW Research and Development - 8th Asia-Pacific Web Conference (APWeb 2006), 2006.

[6] A. van Lamsweerde. Goal-Oriented Requirements Engineering: A Guided Tour [C]. 5th IEEE International Symposium on Requirements Engineering, 2001: 249-263.

[7] A. Van Lamsweerde, E. Letier. Handling Obstacles in Goal-Oriented Requirements Engineering [C]. IEEE Transactions on Software Engineering, Special Issue on Exception Handling, 2000: 978-1005.

[8] A. van Lamsweerde, R. Darimont, E. Letier. Managing Conflicts in Goal-Driven Requirements Engineering [C]. IEEE Transactions on Software Engineering, Special Issue on Managing Inconsistency in Software Development, 1998.

[9] A. Van Lamsweerde. Goal-Oriented Requirements Engineering: A Roundtrip from Research to Practice Proceedings [C]. 12th IEEE Joint International Requirements Engineering Conference, 2004: 4-8.

[10] Rim Samia Kaabi, Carine Souveyet, Colette Rolland. Eliciting Service Composition in a Goal Driven Manner [J].

[11] Kangkang Zhang, Qingzhong Li, Qi Sui. A Goal-driven Approach of Service Composition for Pervasive Computing [J].

[12] Juan Miguel Gómez, Mariano Rico, Francisco García-Sánchez, Rodrigo Martínez Béjar, Christoph Bussler. GODO: Goal driven orchestration for Semantic Web Service.