Automatic Service Composition

Dr. Donald Robinson
Northrop Grumman Corporation

CS307, Software Engineering (Spring 2009)
Video-conferences on Fridays

One of the inherent problems of service oriented computing is rapidly discovering and composing multiple business services to accomplish some user-defined end goal. The objective of this project is to create a modular program that will take a needed business service and available information sources as an input, and then access a list of available services and automatically determine which services must be composed to achieve the needed task.

This problem is one that is commonly researched in the field of service oriented architectures (SOA) but has corollaries in other domains. One such domain is in the domain of tasking multiple military units in the execution of a mission. Each unit may have a set of tasks that it is able to execute. A commander needs an automated way to identify which assets need to be tasked, and how they should transfer their associated information to execute the objective. For instance, a military unit may need to "take the hill". To do this, an intelligence function should be tasked to gather information about the location of the enemy on the hill. Furthermore, this intelligence function may involve tasking multiple information collection assets to gather the information. Additionally, the intelligence that is gathered would need to be translated into a data format that an artillery unit could use to shell a particular location. In the field of service oriented computing, the common method used to expose a web service and the associated binding information is the WSDL. In terms of registries of available tasks to accomplish a military mission, UML documents are often used as well as a more formal database referred to as the Unified Joint Task List (UJTL). Many other tools are available to expose services but most will rely on some form of underlying XML format.

The desired program for this project should have several components. The first should be a module that would capture the request from the user. In the simplest of forms, this would specify completely the output parameters of the services that would be composed in a SOA or the specific tasks that are available from the task lists. This module might consist of a dynamically populated drop down menu of available outputs from all of the services or might take a more generic text request from a user. We will call this the ***request interface***.

The second module of the program would contain a method for interfacing with various databases containing lists of services. These services are likely to be contained in multiple databases. This module should be designed so that another database can be quickly added. For example, if one were searching through UDDI registries, then there should be the ability to rapidly add another UDDI registry to the set of services that can be queried. Similarly, if one is using a database of UML models describing candidate services, then there should be a way to rapidly add new models that can be queried. We will call this the ***service interface***.

In reality the logic behind how these services can be combined might be enormously complex. Ontologies are often used to not only translate the request into a

more computable set of requirements needed from the services, but also, are often used to derive the particular information flows that must be present to successfully compose a set of services or resources. Hence, the third module should be a reasoning mechanism which should interact with the first 2 modules. This reasoning mechanism should be able to take the request from the request interface, break it down into more distinct pieces, iteratively use the service interface to query for and identify candidate services that can be properly composed. We will call this the ***reasoning mechanism***. The most basic implementation of this reasoning mechanism might be to just take the set of specific inputs and outputs specified in the request interface and query directly for them and related ones in the service interface. An ontology of services is not necessary in this implementation but could be added at a later time.

The last component of this program is the ***user interface***. In the user interface, there should be some form of authentication of a user. Once authenticated, the user should be able to input the necessary data into the request interface and initiate the composition request. Then the output of the reasoning mechanism, which is the list of services as well as their associated information, should be displayed to the user. This information should be presented so that at a quick glance a user can understand the services that need to be composed, but can then optionally drill down to more specifics about each service.

Besides the aforementioned modules, there are other constraints that must be put on the system. The first is modularity. Composing services in a SOA has very different details than composing capabilities available in UML models. The fundamentals are the same, but the details are different. Hence the program must be developed so that if someone develops a new request interface that is tailored to a specific data type, then it can be easily replaced in the program. Similarly, as more sophisticated reasoning mechanisms are developed, they should be easy to replace in the system.

One additional constraint is the computing time. In the case of mission execution as mentioned previously, the time constraints are critical. A commander may have only a matter of seconds to identify what resources are available and can be composed together to accomplish his request. Thus, all efficiencies in extracting the necessary information from the user as well as providing the end result should be explored.

It is assumed that the 4 modules of this project can be initially designed and implemented in simple forms with additional capabilities and refinements being added as time allows. Other elements to consider would be the transformations necessary and available to translate outputs from one service into usable inputs from another service. These transformations contribute to the overall information flow which is necessary. Furthermore, some compositions will require services being executed in a particular sequence. This must be conveyed to the user as well.