enTish: an Approach to Service Description and Composition

Stanislaw Ambroszkiewicz[1,2] *

[1] Institute of Informatics, University of Podlasie,
al. Sienkiewicza 51, PL-08-110 Siedlce, Poland
[2] Institute of Computer Science, Polish Academy of Sciences,
al. Ordona 21, PL-01-237 Warsaw,
sambrosz@ipipan.waw.pl; http://www.ipipan.waw.pl/mas/

Abstract. A technology for service description and composition in open and distributed environment is proposed. The technology consists of description language (called Entish) and composition protocol called entish 1.0. They are based on software agent paradigm. The description language is the contents language of the messages that are exchanged (between agents and services) according to the composition protocol. The syntax of the language as well as the message format are expressed in XML. The language and the protocol are merely specifications. To prove that the technology does work, the prototype implementation is provided available for use and evaluation via web interfaces starting with www.ipipan.waw.pl/mas/ . Related work was done by WSDL + BPEL4WS + (WS-Coordination) + (WS-Transactions), WSCI, BPML, DAML-S, SWORD, XSRL, and SELF-SERV. Our technology is based on similar principles as XSRL, however the proposed solution is different. The language Entish is fully declarative. A task (expressed in Entish) describes the desired static situation to be realized by the composition protocol.

1 Our approach to service composition

Generally, there are two approaches to service composition. The first one is based on the assumption that services are composed, orchestrated, or choreographed in order to create sophisticated business processes, whereas the second one assumes that services are composed (typically on the fly) in order to realize clients' requests. Most of the existing technologies realize the first approach. The second approach is followed by academic projects, e.g., SWORD, XSRL, and our own project enTish. It seems that the service architecture corresponding to SOAP and WSDL is appropriate for the first approach. However, in our opinion, a different service architecture is required for realizing the second approach. The reason is that clients' requests are expressed in a declarative way in a formal language, so that it is natural to propose a universal protocol for the request realization. However, also in this case the service architecture based on SOAP and WSDL may be applied as it is done in XSRL[4].

* The work was supported partially by KBN project No. 7 T11C 040 20

We follow the idea of layered view of service architecture introduced in [2, 3]. Our service architecture comprises the following three layers: Conversation layer, functionality layer, and database management (executive) layer. The database management layer is the same as in [3], it influences the real world. However, the next two layers have different meaning. The functionality layer has exactly two interrelated components: Raw application, and so called filter associated with the raw application. Raw application implements a single operation, i.e., given input resources, it produces the output resource according to the operation specification. Note, that operation has exactly one output, although it may have several inputs. The associated filter works as follows. Given constrains on the output resource, it produces the constrains on the input resources. That is, given a specification of the desired output, the filter replies with properties that must be satisfied by the input in order to produce the desired output by the raw application. It is clear that these constrains must be expressed in one common language. The conversation layer implements a conversation protocol to arrange raw application invocation, as well as input / output resource passing to / from the raw application. The conversation protocol specifies the order for message exchange. Message contents is expressed in the common language.

Since our service architecture is different than the one that corresponds to WSDL and UDDI, we must revise the concept of service description language as well as the concept of service registry. It is natural that service description language should describe the types of service input / output resources as well as attributes of these types to express constrains. Note, that the language is supposed merely to *describe* resource types in terms of theirs attributes, not to construct data structures as it is done in WSDL. It is also natural to describe *What service does* in the language, i.e., the type of the operation the service performs. This type is expressed in terms of abstract function implemented by the operation. Usually, *What service does* is described in UDDI. We include this in our description language.

Since service has additional functionality performed by filter (i.e., a service may be asked if it can produce output resources satisfying some properties), the description language should be augmented with a possibility to formulate such questions as well as answers. Moreover, the clients' requests (tasks) should be expressed in the language.

We also want to describe some static properties of service composition process such as intentions, and commitments; this corresponds to the functionality of WS-Coordination.

The final requirement is that the language must be open and of distributed use. It means that names for new resource types, their attributes, and names for new functions, as well as for new relations can be introduced to the language by any user, and these names are unique (e.g., URIs). This completes the requirements for the description language called Entish. Since our technology is supposed to realize the declarative approach, we need a universal protocol for realizing the requests (tasks) specified in our description language.

For simplicity (i.e., for avoiding reasoning) as well as for making the prototype implementation feasible, we assume that the requests are extremely simple; in fact they are expressed as formulas that represent abstract plans and initial situations. In the next step of our project a *distributed* reasoning for plan generation will be implemented, so

that the requests will have a form of arbitrary formulas. The plan realization is done by the protocol called entish 1.0.

To prove that the requirements for the service description language and composition protocol can be satisfied we provide the prototype implementation available from http://www.ipipan.waw.pl/mas/ .

2 Walk-through example

The working example presented below constitutes an intuitive introduction to the description language and the composition protocol. The services described in the example are implemented and are ready for testing via the www interfaces.

A client was going to book a flight from Warsaw to Geneva; the departure was scheduled on Nov. 31, 2002. It wanted to arrange its request (task) by Nov. 15, 2002. With the help of TaskManager (TM for short), the client expressed the task in a formal language; suppose that it was the following formula:
phi =
"invoice for ticket (flight from Warsaw to Geneva, departure is Nov. 31, 2002) is delivered to TM by Nov. 15, 2002"

Then, the task formula (i.e., *phi*) was delegated to a software agent, say agent0. The task became the goal of the agent0. The agent0 set the task formula as its first intention, and was looking for a service that could realize it. First of all, the agent0 sent the query: *"agent0's intention is phi"* to a service registry called infoService in our framework. Suppose that infoService replied that there was a travel agent called FirstClass that could realize agent0's intention. Then, the agent sent again the formula *"agent0's intention is phi"* however, this time to the FirstClass. Suppose that FirstClass replied with the following commitment:
"FirstClass commits to realize phi,
if (order is delivered to FirstClass by Nov. 15, 2002 and
the order specifies the flight (i.e., from Warsaw to Geneva, departure Nov. 31,2002)
and one of the following additional specification of the order is satisfied:
(airline is Lufthansa and the price is 300 euro)
or (airline is Swissair and the price is 330 euro)
or (airline is LOT and the price is 280 euro))"

Let *psi* denote, the formula after *"if"* inside (...) parentheses. The formula *psi* is the precondition of the commitment. Once the agent0 received the info about the commitment, the agent0 considered the intention *phi* as arranged to be realized by FirstClass, and then the agent0 put the formula *psi* as its current intention, and looked for a service that could realize it. Let us notice that the order specified in the formula *psi* could be created only by the client via its TM, that is, the client had to decide which airline (price) should be chosen, and the complete order was supposed to include details of a credit card of the client. Hence, the agent0 sent the following message to TM: *"agent0's intention is psi"* Suppose that TM replied to the agent: *"TM commits to realize psi, if true "* The agent0 considered the intention *psi* as arranged to be realized by TM. Since the precondition of the TM commitment was the formula *"true"*, a workflow for realizing agent0's task was already constructed. Once TM created the order and sent it to FirstClass, the FirstClass would produce the invoice and send it to TM. It was supposed

(in the protocol) that once a service realized a commitment, it sent the confirmation to the agent0. Once the agent0 received all confirmation, it got to know that the workflow was executed successfully. In order to complete this distributed transaction, the agent sent synchronously the final confirmation to the all services engaged in the workflow. This completes the example. The complete enTish documentation is available at the project web site http://www.ipipan.waw.pl/mas/

References

1. S. Ambroszkiewicz. Entish: a simple language for Web Service Description and Composition, In (eds.) W. Cellary and A. Iyengar. Internet Technologies, Applications and Societal Impact. Kluwer Academic Publishers. pp. 289- 306, 2002.
2. Santhosh Kumaran and Prabir Nandi. Conversational Support for Web Services: The next stage of Web services abstraction. http://www-106.ibm.com/developerworks/webservices/library/ws- conver/?dwzone=webservices
3. F. Leymann and D. Roller.Workfbw-based applications. IBM Systems Journal, Volume 36, Number 1, 1997 Application Development http://researchweb.watson.ibm.com/journal /sj/361/leymann.html
4. Mike Papazoglou, Marco Aiello, Marco Pistore, and Jian Yang. XSRL: A Request Language for Web Services. http://eprints.biblio.unitn.it/archive/00000232/