The SOA Paradigm and e-Service Architecture Reconsidered from the e-Business Perspective

Stanisław Ambroszkiewicz, Waldemar Bartyna, Marek Faderewski, Dariusz Mikułowski, Marek Pilski, Marcin Stepniak, and Grzegorz Terlikowski

Institute of Computer Science, Polish Academy of Sciences
Al. Ordona 21, PL-01-237 Warsaw
and Institute of Computer Science, University of Podlasie, PL-08-110 Siedlce, Poland
sambrosz@ipipan.waw.pl
http://www.ipipan.waw.pl

Abstract. A business service has well founded structure where its operations (corresponding to request-quote, order-contract, invoice-payment) are related to each other. These relations cannot be expressed in WSDL. The request-quote operation corresponds to SLA negotiations and can be performed in a universal description language such as OWL that can also express all the relations between service operations mentioned above. Generally, from the e-business perspective the following notions are important: *(1) Service architecture. (2) Communication protocols in e-business processes.* These notions are crucial for providing standards necessary for creating open, heterogeneous and scalable systems for realizing complex e-business processes. These notions are discussed in the paper.

Keywords: business process, service arrangement, service composition, ontology, task execution.

1 Introduction

The classic version of the SOA (Service Oriented Architecture) paradigm, see [1], may be summarized as follows. *SOA provides a standard programming model that allows self-contained, modular software components residing on any network to be published, discovered, and invoked by each other as services. There are essentially three components of SOA: Service Provider, Service Requester (or Client), and Service Registry. The provider hosts the service and controls access to it, and is responsible for publishing a description of its service to a service registry. The requester (client) is a software component in search of a component to invoke in order to realize a request. The service registry is a central repository that facilitates service discovery by the requesters.*

The key point of the SOA paradigm is service integration, that is, *"... other applications (and other services) can discover and invoke the deployed service ..."* In order to realize this vision, simple ubiquitous and automatic process modeling techniques are needed. The service integration is interpreted in several ways,

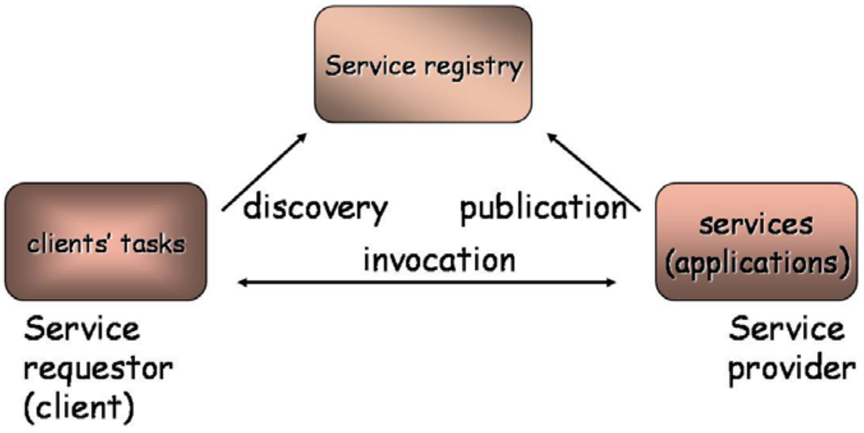


Fig. 1. The classical version of the SOA paradigm

e.g., as service composition, service orchestration, service choreography, process modeling, etc. These terms have been widely used to describe business interaction protocols. Generally, business processes comprise collaborative services that can provide much more complex functionalities, than the single services [2].

One of the classical definitions of a business process is following [3]:

It is a structured, measured set of activities designed to produce a specific output for a particular customer or market. It implies a strong emphasis on how work is done within an organization, in contrast to a product focuss emphasis on what. A process is thus a specific ordering of work activities across time and space, with a beginning and an end, and clearly defined inputs and outputs: a structure for action. ... Taking a process approach implies adopting the customers point of view. Processes are the structure by which an organization does what is necessary to produce value for its customers.

Another definition of electronic business process is given in IST CONTRACT Project [4]: *A business process specifies the potential execution order of operations from a collection of Web Services, the data shared between these Web Services, which partners are involved and how they are involved in the business process, joint exception handling for collections of Web Services, and other issues involving how multiple services and organizations participate. In other words it defines the composition of Web Services in order to provide higher functionalities.*

The terms *higher functionalities* and *customers point of view* from the above definitions may be interpreted as a client (customer) task the business process is supposed to accomplish.

Although languages and technologies such as WSDL (Web Service Description Language), BPEL4WS (Business Process Execution Language four Web Services), and OWL-S (Web Ontology Language for Services) seem to be adequate for modeling and execution of the electronic business processes, automated process composition is still a challenge in Information Technologies.

The notion of *e-business process* (as the main point of the discussion) is strongly related to the several crucial issues, such as the SOA paradigm, and the concept of e-business service in SOA. The classic version of the SOA paradigm has its origins in software engineering. Service is viewed as server application (in the client-server model) waiting to be invoked by clients. From the point of view of e-business process, a service may be also active and looking by itself for clients tasks that can be accomplished. This corresponds to the reverse auctions in business practice.

A business service has a well founded structure where its operations (corresponding to request-quote, order-contract, invoice-payment) are related to each other. These relations cannot be expressed explicitly in WSDL. Hence, the concept of service architecture as well as communication protocols in e-business processes should be discussed.

According to the SOA paradigm, services and clients are engaged in several activities, i.e., publication, discovery and service invocation. Usually, the invocation is preceded by a negotiation for a service level agreement (SLA) that is (by its nature) "output" based, i.e. the result of the service, as required by the client, is the subject of the agreement. The SLA negotiations require semantic interoperability, that is, understanding between the client and the service provider. Actually all activities related to how a service is used by a client (also publication and discovery) require understanding between the client and the service provider. These activities must be realized in the form of communication protocols as it is usually done in distributed systems. Since the system is supposed to be open and heterogeneous, the semantic interoperability cannot be hard-coded in the protocols. It must be incorporated in the contents of the protocol messages. The contents must be expressed in a generic declarative language. The client task as well as the service description, and the output (final result) description of service performance must be expressed in this language. The important question is what this language is about? In other words, what is the grounding, i.e., the real semantics (not an abstract model-theoretic one) of the language. In our approach the grounding is a class of XML documents that services process (input documents and output documents). Once the documents are processed, they have precisely defined impact in the real world. Since this impact is hard (perhaps not possible) to describe formally, the proposed grounding is simple and sufficient to provide semantic interoperability. OWL was chosen as the language for describing XSD schemata of the documents processed by services. In our previous research projects the language was Entish [5], our own invention. The rest of the paper id devoted to describe our approach more precisely.

The structure of the paper is as follows. In Section 2 our approach to e-businesses is presented with OWL as the description language. Section 3 presents our discussion on active services and Task Oriented Architecture. Since the paper presents the work in progress, the conclusion is short and indicates only some future work.

2 Our Approach to SOA with OWL as the Description Language

The key assumption of our approach is that a service may be described externally, that is, by the local change in the environment, caused by a performance of the service. The very *local change* is usually described by a precondition and a post-condition (effect) of the performance. The environment should be represented in a formal way, and then described in a formal language. In the case of business processes, the environment is represented by documents that are exchanged and processed by the partners involved in the process. The language describes the documents. The crucial notions are: *environment of interaction between services and clients*, its *representation* denoted by (Service Environment Representation, SER for short), and *a language describing the representation.* The language is grounded (has semantics) in the environment representation (SER). The OWL syntax may be used as the syntax of the description language. Although OWL has well defined model-theoretic semantics (see http://www.w3.org/TR/owl-semantics/), the proposed semantics for the description language (based on the OWL syntax) has another semantics; it is SER. For this very reason the description language used in our approach is denoted by OWL+SER. OWL+SER is complementary to WSDL where the input and output documents, to be processed by a service operation, are defined. Actually, these documents are also in SER. OWL+SER allows to specify service precondition and effect as OWL formulas in the same way as it is done in OWL-S, however, in the case of OWL+SER the precondition formula describes the input documents of the WSDL operations of the service, whereas the effect formula describes the corresponding output documents of the operations.

In fact, we follow the IOPE (Input Output Preconditions and Effects) approach from OWL-S. However, contrary to OWL-S and its *ProcessModel*, the internal service structure cannot be specified; it is treated as a *black box.*

OWL+SER is also used to define tasks to be accomplished by (composite) services. Tasks are expressed in a declarative way as a pair of OWL formulas (initial situation formula and intention formula) describing the client initial situation, and a final situation intended by the client. This is similar to *(precondition, effect)* of the service description in OWL-S. OWL+SER serves also to arrange necessary conditions (to be fulfilled by the client) for service invocation in order to accomplish the clients task.

2.1 Service Architecture

WSDL is regarded as the standard for describing web services. General structure of service in WSDL 2.0 consists of the following elements:

- The data types used by the web service.
- The abstract interface as a set of abstract *operations*, each operation representing a simple interaction between the client and the service. Each operation also specifies a message exchange pattern that indicates the sequence

  in which the associated messages are to be transmitted between the parties. Usually, there are input and output messages.
- The communication protocols used by the web service.

The usual non automatic use of a business service is decomposed onto the four following phases:

1. The service requester sends a query to a service specifying roughly what she/he wants from the service, and then gets back a quotation (a pro forma invoice) from the service provider. The quotation specifies details of what (and how) the service can be performed for the requester.
2. Having the quotation, the requester creates an order and sends it to the provider. Usually, the provider replies with an appropriate contract.
3. If the service is performed according to the contract, the provider sends expertise or carriage letter to the requester, whereas the requester sends back a delivery note (or acknowledgement of receipt) or a complaint.
4. Finally, the service provider sends invoice and the requester realizes the payment for the service performance.

Sometimes, after sending the order, the service provider sends back an invoice. In this case the payment must be done before service delivery.

In order to automate the use of a service, all those four phases above must be automated. For any specific type of service this may be done by a dedicated software application implemented, for example, in BPEL. Note that in each of the phases above, there is a document flow, that is, query and quotation in the first phase, order and contract in the second phase, carriage letter, delivery note, and complaint in the third phase, and finally invoice and payment. These documents can be created in XML. For each of the phases, the document flow can be described in WSDL. Finally, the complete process of using a single service can be implemented in BPEL. This can be done (hardcoded) separately for any service type in a dedicated way. However, our goal is to do so generically, that is, to construct tools that allow automation of service use for any service type.

Note that all the above phases are interrelated with each other, that is, a request (in the second phase) is created on the basis of the quote from the first phase, whereas the carriage letter, delivery note, complaint and payment are strictly related to the contract. Each of the phases must be implemented as a separate WSDL operation. There are no means to explicitly express these interrelations in WSDL. An implicit way to do so is to use the same types of elements in documents from different phases. However, to automate the service use these interrelations must be explicitly expressed in a formal language; in our approach the language is OWL+SER. These interrelations, as OWL formulas, can be processed automatically.

The first phase is of special interest for the automation in question. Actually it concerns SLA (Service Level Agreement) negotiations. The agreement is about the service output, i.e. the result of the service required by the client. Hence, the negotiations are about the contract, delivery, complaint, and payment. It is natural to have a generic language for such negotiations independent of the type of service to be used. The language should describe documents to be processed in the next phases; it is clear that it must be OWL+SER. Then, the negotiation may proceed as follows. Given its task, the client sends its *intention formula* (a OWL formula) to a service, describing roughly what is to be accomplished by the service. This intention formula corresponds to the query in the first phase of business service. Service sends back a commitment consisting of two OWL formulas: *precondition formula*, and *effect formula*. This corresponds to the quote from the first phase. The precondition allows to create, by the client, XML-document required as input by the service. Actually, the precondition formula proposes several options (one to be chosen by the client) of the task accomplishment, and describes the input document corresponding to the client choice. The effect formula specifies the service output with relation to the input. Actually, the effect formula implies (logically) the client intention formula. It means that the intention formula describes (roughly) the output document, whereas the effect formula describes the output completely, usually, in several options. If one of the options is chosen by the client, SLA is set and its terms are incorporated in the order and in the contract (or invoice) in the next phases.

The conclusion from the above discussion is the following. The first phase of service use (corresponding to SLA negotiation) can be done, in a universal automatic way, using OWL-SER language. Based on this, the next phases of service use can also be automated. The crucial points of the proposed approach are as follows.

- Service interaction environment is represented by XSD schemata of the XML documents processed in the order-contract phase, and in the invoice-payment phase of service use.
- The language (OWL+SER) describing the representation (i.e., XSD schemas) is based on the syntax of OWL. The grounding (semantics) of the language is constituted by the above XSD schemata.
- Query and quote (of the first phase of the service use) are expressed in this language.

An automation of the use of a single service is relatively simple. If, however, several single services must be composed to perform a complex task, then the problem is how this composition can be done automatically and in a universal way.

Once the first phase is realized according to our approach, the automatic service composition is possible to some extent. The composition is done in the following three steps.

- **Step 1:** Given an initial situation and a goal (that constitute together a request) expressed in OWL+SER, construct generic plans that may realize this goal starting with the initial situation.
- **Step 2:** Choose one generic plan and discover appropriate services. Then, arrange them into a workflow. The arrangement is done as query-quote phase in OWL-SER, and corresponds to SLA negotiations.
- **Step 3:** Present the workflow to the client specifying the initial situation more precisely. Once the initial situation is satisfied by the client, execute

the workflow, control the execution, and realize the distributed transaction, if the execution is successful.

Note, that if a business process is more complex, the client interaction may be necessary in the Step 3. In this case, depending on a client decision, a new task must be realized, and the Steps 1, 2, 3 must be repeated.

To realize the Step 1, an automatic reasoning for plan construction is needed. There is a considerable work done in this area, see for example PDDL (planning domain definition language) [6] for Estimated-regression planning, and SHOP2 domain-independent planning system [7].

The Step 2 is realized by sending queries (called also intentions in our approach) and getting back quotations (called also commitments in our approach). Step 3 may be realized by using a process modeling language, i.e., by BPEL.

In our approach we propose a universal protocol for accomplishing simple client's requests (that do not require client interactions) according to the Step 2 and Step 3. Generally, the protocol specifies message exchange between services, agents (realizing the requests on behalf of the clients) and service registry.

The protocol consists of two general phases: The first one is called query phase (corresponding to SLA negotiations) whereas the second one is called execution phase. The query phase consists of the following two steps:

1. The agent sends a query (intention) to the service specifying the desired output (effect).
2. Then, the service answers with the quotation (commitment), i.e., specification of the input (precondition) required to produce the desired output.

The execution phase consists of the following three steps:

1. The agent (dedicated for the task accomplishment) creates input data (an order) according to the quotation and sends it to the service.
2. The service receives the order and produces output data (a contract) that is sent back to the agent.
3. If the contract is fulfilled, the next phases (i.e., delivery note, and payment) follows.

The previous versions of our approach (where Entish was used as the description language instead of OWL+SER) were verified by several prototype implementations. The first implementation (called enTish [8]) focused merely on composition of services that process data, i.e., services were ordinary software applications. The last completed implementation (called ELA-enT [9]) is dedicated to realization of Electronic Market for simple services. The complete description language Entish can be found in [5] and on the project web site [8].

3 Active Services and Task Oriented Architecture

The classical version of the SOA paradigm has its origins in software engineering. From the point of view of e-business processes (that are supposed to be also software applications), the concept of service need not be related to RPC (as it is in

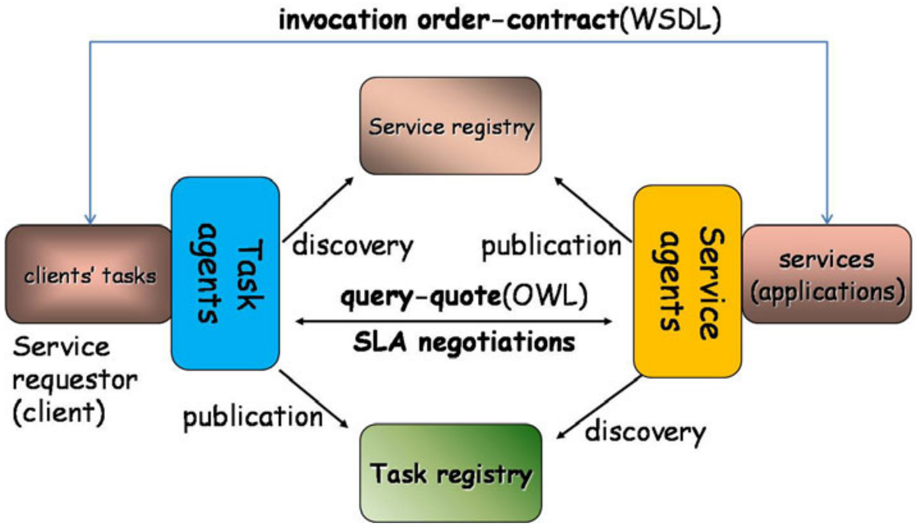


Fig. 2. SOA and TOA together

SOA) where a service is passive and is waiting for a client to be invoked. In other words, a service may be active and looking by itself for clients tasks that can be accomplished. This corresponds to the reverse auctions in business practice. In the SOA paradigm the services must be described (as it is done in WSDL and OWL) in order to be published in a service registry, and discovered there by clients. Task oriented architecture, as opposed to Service Oriented Architecture, requires tasks to be expressed declaratively. These tasks can be published by the clients in a special task registry, and discovered there by services. Unfortunately the use of the term *Task Oriented Architecture* is a bit restricted by the US patent [10].

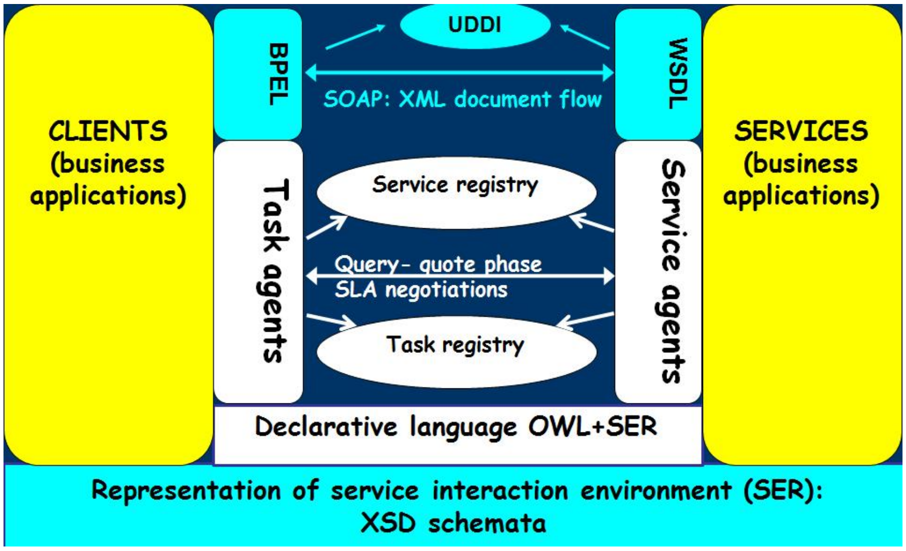
In Fig. 2, the classic version of the SOA paradigm (presented in Fig. 1) is augmented with its TOA counterpart. That is, several components are added: Task repository for task publication and discovery, as well as task agents and service agents, representing clients and service providers (respectively) in business processes. These agents together with their interactions, constitute a multiagent system that is interesting to investigate by itself. One of the fundamental points of our approach is the separation of the query-quote phase (corresponding to SLA negotiations) and its generic realization in declarative language (as shown in Fig. 2) from the execution phase (invocation order-contract in WSDL). The execution phase is realized by direct interaction between client and service on the basis of the service level agreement done in the query-quote phase. All the interactions corresponding to task publication and discovery, service publication and discovery, as well as SLA negotiation are delegated to task agents and service agents. It seems that this very separation makes the revised SOA paradigm more generic and flexible.

3.1 A Revision of the SOA Paradigm

The main and starting point of the proposed SOA revision is that a service, in the SOA paradigm, should be considered as a business service. It has well defined structure corresponding to the following consecutive phases of service use: query-quote, order-invoice, payment, and so on. These phases may correspond to operations in WSDL. However the relations between the phases cannot be expressed in WSDL explicitly. Since a service may be active (as opposed to a passive service as it is in classic SOA), a declarative language is necessary to express client tasks. Once we have such declarative language; it may also be used to describe service interfaces in a declarative way. Although, the WSDL (by its name) is supposed to describe Web services, actually it serves to define XSD schema of the input and output documents processed by service operation.

Separation between WSDL and the description language is crucial in the proposed revision. Although this was done in Semantic Web services in OWL-S, the semantics of OWL is not directly related to the WSDL, that is, to the XSD schema of the documents processed by services. In our approach, introduction of the notion SER (Service Environment Representation) as the direct grounding (concrete semantics) of the description language, allows to explicitly express the relations between the phases of service use.

Fig. 3 summarizes the proposed approach and a revision of the SOA paradigm. The basis is the representation of service interaction environment consisting of XSD schemata of the documents processed by services and clients presented respectively in the right and the left side of the figure. In the next layer there is OWL+SER (grounded in the layer below) as a declarative language for expressing client tasks and service interfaces. In the third layer the separation between



CLIENTS (business applications)
BPEL
UDDI
SOAP: XML document flow
WSDL
SERVICES (business applications)
Task agents
Service registry
Query- quote phase
SLA negotiations
Task registry
Service agents
Declarative language OWL+SER
Representation of service interaction environment (SER): XSD schemata

Fig. 3. The general architecture of our approach

the execution phase (XML document flow by BPEL, SOAP, WSDL, and UDDI), and SLA negotiation phase (as a multiagent system) is shown.

4 Preliminary Conclusions

We have described our ongoing project that aims at creating tools to automate (as much as possible) modeling, planning, constructing, execution and control of sophisticated business processes. The work is in progress, and is based on the approach presented above. To verify our approach as well as the tools, two testing environments are being built. The first one concerns crisis management, whereas the second one for business processes related to real estate development.

The idea presented in the paper seems to be a novel approach to SLA negotiations as well as to e-business processes. A potential application of the proposed solution may improve automation of modeling, creation and executions business processes for accomplishing clients tasks.

Acknowledgements. The work was done within the framework of the IT-SOA POIG01.03-01-00-008/08-00 project supported by the European Commission and the Polish Government.

References

1. IBM Services Architecture Team, Web Services architecture overview: The next stage of evolution for e-business (2000), `http://www.ibm.com/developerworks/webservices/library/w-ovr/`
2. Peltz, C.: Web Services Orchestration and Choreography. Computer 36(10), 46–52 (2003)
3. Davenport, T.: Process Innovation: Reengineering work through information technology. Harvard Business School Press, Boston (1993)
4. Napagao, S.A., Biba, J., Confalonieri, R., Dehn, M., Kollingbaum, M., Jakob, M., Oren, N., Panagiotidi, S., Solanky, M., Salceda, J.V., Willmott, S.: Contract based electronic business systems state of the art. In: IST Contract Project (April 10, 2007)
5. Ambroszkiewicz, S.: Entish: A language for describing data processing in open distributed systems. Fundamenta Informaticae 60(1-4), 41–66 (2004)
6. McDermott, D.: PDDL - the planning domain definition language. In: The AIPS 1998 Planning Competition Committee (1998), `http://www.cs.yale.edu/homes/dvm`
7. Wu, D., Sirin, E., Hendler, J., Nau, D., Parsia, B.: Automatic Web Services composition using SHOP2. In: Workshop on Planning for Web Services (June 2003)
8. Project enTish, `http://www.ipipan.waw.pl/mas/`
9. Project ELA-enT, `http://ent.ipipan.waw.pl`
10. Apparatus, method and architecture for task oriented applications (Fujitsu Limited). Estimated Patent Expiration Date: October 30, 2018: US Patent 6442749 - US Patent Issued on August 27, 2002 (2002)