

REFERENCE ARCHITECTURE FOR A SEMANTIC SERVICE PROVISIONING PLATFORM

Jari Veijalainen, Mathias Weske
Hasso-Plattner-Institute, University of Potsdam
Prof.-Dr.-Helmert-Str. 2-3
D-14482 Potsdam, Germany
{j.veijalainen,mathias.weske}@hpi.uni-potsdam.de

ABSTRACT

This paper discusses architectural requirements for an inter-organizational, semantics-based service provisioning platform and presents a generic reference architecture. We deduce essential requirements for such an architecture from the inter-organisational environment and the technology in use. We also analyze the implementation options of the reference architecture and describe a prototypical implementation.

KEY WORDS

Inter-organisational computing (IOC), Service oriented architecture (SOA), Semantic web services (SWS)

1. Introduction

Computers have been used in business more than fifty years to store and manipulate data. The dream of fully automated business transactions began to realise during 1970's in the international banking community, when the first SWIFT system went into operation in 1977 [1]. Although new technologies have been introduced since then, the issue at stake in the seventies was and is today, how to use computers and communication networks in *fully automated inter-organisational interactions* (e.g. financial transactions). We call this activity *automated inter-organisational computing* (IOC).

The organisations in the context of IOC consist of humans, computer systems, and the relationships between them. Computer systems are always controlled by humans, and cannot take actions unless mandated by a human. An organisation can engage in interactions with other organisations through an *external message (type-based) interface* that is known to other organisations. The interactions between organisations take place in IOC exclusively by *exchanging message instances*, compatible with the external message interface, between the computer systems. *Executing a message* by a computer system means that some *actions* are performed by the organisation while it is processed. These can be real world actions (e.g. send a book to customer X), database transactions modifying the internal databases state of the

organisation (e.g. record an order Y for customer X), or generation and sending further messages to other organisations (debit credit card 4920....with 50 USD for order# 456).

In an *IOC environment*, consisting of organisations and a *network* being able to carry the messages above, the organisations are always *organisationally autonomous* against each other and can cooperate and compete at the same time on the global or domestic market [2]. The organisational autonomy is exhibited in mainly three forms towards other organisations and their computer systems. D(esign)-autonomous computer system is typically owned and under the control of the humans in the same organisation and it is structured as the people owning it deem best, no matter what other organisations in the IOC environment would deem necessary. E(xecution)-autonomous computer system might execute a message it receives from other organisation or it might refuse to execute it, and in the former case it can choose how to execute it (e.g. by asking further organisations to perform further actions by sending messages to them). Finally, a C(ommunication)-autonomous computer system can freely decide when it is reachable through the network, when not, and with which other computer systems it will communicate and when. For further analysis of these concepts, see [2,3].

Because there are many autonomous organisations with roughly the same power participating in an IOC environment, they must collectively agree upon the above message types. This is done by forming a *global designer* that is an organisation establishing the common standards for the *global homogeneous area* [2]. In the current stage of the development, a global designer consists of two parts, a *Global Language Designer* (GLD) and an *Operational Interface Designer* (OID). The former designs the formal *Inter-Organisational Language* (IOL) with which the external message interface of an organisation can be described (e.g. EDIFACT [4] or WSDL+WS-BPEL [5,6]). The latter agrees upon concrete interfaces using a particular IOL (e.g. EDIFACT message types in Electronic Data Interchange (EDI), WSDL files). From D-autonomy it follows, that the implementation of

the operational semantics (i.e. which actions an organisation performs locally which is asks the other computer systems to perform while executing a message) is not disclosed to other organisations. Thus, they must *trust* the organisation in order to believe that the externally described actions are really taken and nothing else, while executing a message.

An issue largely overlooked in technically oriented treatments of IOC is that automated IOC is primarily *contract-based* and *policy-guided*. Deciding which incoming messages to execute and how, i.e. E-autonomy, are mostly based on contractual and policy considerations.

The rest of the paper is organised as follows. In Section 2 we describe the latest phase in IOC, Web Services in the light of the above general issues. In Section 3 we will present technical requirements for IOC that is based on Semantic Web Service. In Section 4 we deduce a reference architecture for SWS environments supporting IOC and SOA [7]. Section 5 concludes this contribution.

2. The era of semantic web services

Whereas HTML was designed to represent and store information, primarily for humans, web services were designed to support interactions between applications running in different computer systems. The goal is thus the same as in EDI, i.e. *application-to-application interoperability*. The conceptual background of web services is object-oriented distributed computing. One assumes that within a computer system there is an object (type) that offers an external interface (cf. above) with several operations. “An operation is a sequence of input and output messages, and an interface is a set of operations” [5]. An operation is invoked by a message that carries the input parameters for the operation. An operation invocation often also returns a response message that is specified as part of the interface description. The entire web service specification consists of services, interfaces, bindings, elements, and type definitions [5]. These are the *meta-level concepts* of this particular IOL, Web Services Description Language (WSDL), that have a specific syntax and meaning.

A description in WSDL can be understood also as a set of protocol specifications for point-to-point protocols, where each operation corresponds to one protocol. It does not, however, contain any formal description, what should happen in terms of taken actions when a request or response message arrives at a recipient. The meaning of the specified elements of a concrete web service specification is given in “documentation” that can be attached to various concepts above. In this respect the modeling power is at the same level as in EDIFACT (i.e. the meaning of the messages and actions taken must be given in a natural language). Thus, programmers are needed to write the programs that realise the meaning of the message at the computer system of an organisation.

WSDL is, however, more flexible than EDIFACT due to the integrated type system. The latest version 2.0 [5] also makes possible to declare elements optional (cf. EDI message types) and fixes the handling rules for these elements.

What is also different to basic EDI (though not to XML/EDI [8]) in Web Services environment, is the Universal Description Discovery & Integration (UDDI) [9], where service specifications defined by service providers can be stored in service registries and searched and retrieved by service requestors.

Concerning operational interface designer (OID), in the EDI environments the message types must be defined and agreed upon by and OID, in the world of web services the service specification is compiled by the service provider alone. The clients have to comply with the message specifications in order to use the service. Thus, if the client wants to use several web services offering the same or roughly the same functionality (e.g. flight reservations) it must use different message types when communicating with these services. The complication in the above example is caused basically by the fact that there is primarily no OID in the web service environment that would specify what is a standard interface like for a certain type of service (like flight reservations). Another aspect is the missing semantics description that would make it possible to achieve more flexibility and automation in web service environment at the client side. As stated in the OWL-S white paper: “The Semantic web should allow users to locate, select, employ, compose, and monitor Web-based services automatically.” [10].

The above tasks can be understood as high-level requirements for the *semantic annotations* that should, evidently, be added to the service specification expressed with WSDL – or with another, as powerful IOL. Whereas UDDI offers standardised keywords [9] that refer to the actions performed during message execution, a more complex reasoning in discovery and composition is based on ontologies; “An ontology is a formal explicit specification of a shared conceptualization” [11]. Shared above refers to a community that in this case should be interpreted as a set of autonomous organizations forming an *IOC community* (cf. above). We analyze the semantic approach below.

3. Architectural choices for semantic web service (SWS) environment

We consider an SWS environment as an instance of an IOC environment. In this vein, we have already discussed above several requirements that an architecture supporting semantic web services should have. Although the definition of Service Oriented Architectures (SOA) is still in under discussion [7,12], we consider SWS environments to conform to SOA principles. We analyze

the requirements for SWS environment further below from these two perspectives.

3.1. SWS Environment with or without OID

The service requestors and service providers normally interact directly in a SOA while requesting message execution, i.e. *service delivery*. Consequently, in this case the semantic annotations should be processed either by *service providers* or by *service requestors* or by both. This would mean that the capability to process the semantic annotations would be distributed to a plethora of computer systems. The main source of these annotations would be primarily the service providers, but service requestors might also compile and develop their own ontologies and abstract service specifications. In other words, there would be no OID homogenising the semantic annotations. The power relations inside the environment are further discussed in [2,13].

Another approach is to introduce a *centralised platform* that is capable of processing *semantic requests* based on the semantically annotated service specifications and deliver services to the requestors. Because a registry (e.g. UDDI) is in any case needed for efficient service discovery, the platform could be build around the registry. A requestor needs in this case to send semantic requests to the platform or a “usual” request to an application that then transforms it to a semantic request. The platform then processes the request by possibly composing a more complicated service and asking service providers to enact the individual atomic services.

Let us assume that there is a platform that supports SWS, a number of *user terminals (requestors)* and a number of *atomic service providers* hosting web services. Then we can distinguish four different organizational settings, depending on where the organizational borders across the technical infrastructure are placed.

- a) All entities are controlled by different organizations, i.e. users are autonomous individuals (customers) that control their devices and do not belong to the atomic service provider or end service provider (platform controlling) organization and the latter are also separate, autonomous organizations (service market case)
- b) Users controlling terminals belong to the same organization that controls the platform, but atomic service providers are autonomous organizations and different from the platform controlling organization (internal business process support within an organization, access to external services)
- c) Users controlling terminals are as in a), but the end service provider and the atomic service providers belong to the same organization (distributed service provision within one organization towards customers)
- d) Finally all controlling entities belong to the same organization (company internal SWS platform use)

The difference the organizational border makes is that interactions that go over an organizational border must have an implicit or explicit contract and authorization attached to them and often also these interactions have a price tag, i.e. those organisations that want others to execute a message, should pay for the actions taken. The same control sphere also means that, in principle, various conflicts can be resolved more easily than if they conflict reaches over organizational border.

The organisation which is in control of the platform always is itself an OID or organises and OID in the above cases.

3.2. General architectural requirements for a platform

The functional requirements of such a platform are

Management of semantically annotated service descriptions, composed services, and the related ontologies, goals, as well as actor accounts. This functionality includes inserting new service descriptions, atomic service providers and perhaps users onto the platform, inserting new ontologies and checking the consistency of the resulting ontologies (OID’s task), modifying the existing artefacts and deleting them. The management must deal with possible conflicts and enforce proper authorization for the actors corresponding to the organizational setting. Depending on the capabilities, preparing composed service descriptions can be seen as a management task or as a step in service enactment. One of the permanent issues is how can a modification performed at an atomic service provider’s service interface be noticed at the platform in settings where the platform and the service provider are at different sides of an organizational border.

Contract negotiation, maintenance and service monitoring support for various actors in the environment. This is necessary between autonomous actors (cf. case a), but the mechanisms can be used also in other organizational settings, i.e. company internally. The concrete needs depend on the organizational setting.

Enactment of services on an external request, at external service providers, i.e. distributed service provisioning. The latter is necessary, because only a few services could be provided at the platform, the rest must be requested from the other sites (atomic service providers) and the service requests also come from the requestors over an open network. The way enactment is done depends essentially on the *dynamic composition capabilities* of the platform. It might only select from the existing compositions and enact, it could compile a composition based on the request and service description and enact this, or in addition, negotiate a contract during enactment. It can even compile a new composition, if the enactment of the previous fails for some reason. And finally, it can ask another platform instance to enact a part

of the request, if it cannot find a matching composition or atomic service description for it.

Dependability, security and privacy provisioning for the actors and data in the environment. This addresses mostly the technical properties the platform has. Dependability properties [14] are closely related with the way the platform is implemented and are necessary to guarantee that the contracts are kept.

Support for interoperability with the traditional web service-based architectures (and preferably with various flavours of SOA [7,12,15]) and with other semantic web service platforms (mediation, enactment). This is an inherent requirement in all organizational settings, and follows from the D-autonomy of the actors in the environment.

These general functionalities can be realized in many ways. There are also dependencies between various functionalities. Especially the semantics of the service descriptions and related ontologies have direct influence on how dynamic the service enactment can be. The semantic service descriptions can also be used to provide reliability (on-the-fly recovery by finding a contingency plan) and maintainability. The contracts are negotiated between actors and stored at the platform. The negotiation can happen, when an external service provider makes its service known at the platform and states what are the terms and conditions for its usage (price, response time, etc.) and which of them are perhaps dynamically negotiable during enactment. The contracts can also be used during service enactment to select among possible *service implementations* providing the semantically same service and monitor, whether a contract is kept by an atomic service provider.

The main architectural options of a semantic web services providing systems are

- a) Division of the platform functionality into particular subsystems, specification of the platform internal interfaces, as well as external interfaces visible at the external platform border
- b) Selection of the semantic annotation concepts and language, i.e. ontology concepts and language(s) (meta-meta level, meta-level of MOF [16])
- c) Selection of the language to encode the service compositions so that they can be enacted *as workflows* by the services providing system, i.e. workflow language; in theory, this could be part of a) but in practice there are many reasons for this is language to be separate from a)
- d) Off-line pre-compilation of composed service descriptions vs. run-time composition
- e) Decentralized, distributed, or centralized registry; decentralized or centralized maintenance of the service descriptions and ontologies, as well as contractual data
- f) Decentralized control of the enactment of the composed service descriptions vs. centralized one

- g) Dependability and security level of the services providing system [14]

The options are obviously dependent on each other. A starting point is a fully centralized architecture, where all subsystems can be considered to be running within one physical computer. The requestor and external, atomic service providers have different computers connected to the network. This setting can fit into a certain organizational environment, but not for all. From this point on one can then think, what alternatives one has to make the architecture distributed, what the dependencies are, what organizational setting is in question, etc. It is beyond the scope of this paper to analyze all this. Some of the distributed architecture alternatives have been analyzed in [17].

3.3. Ontology language requirements

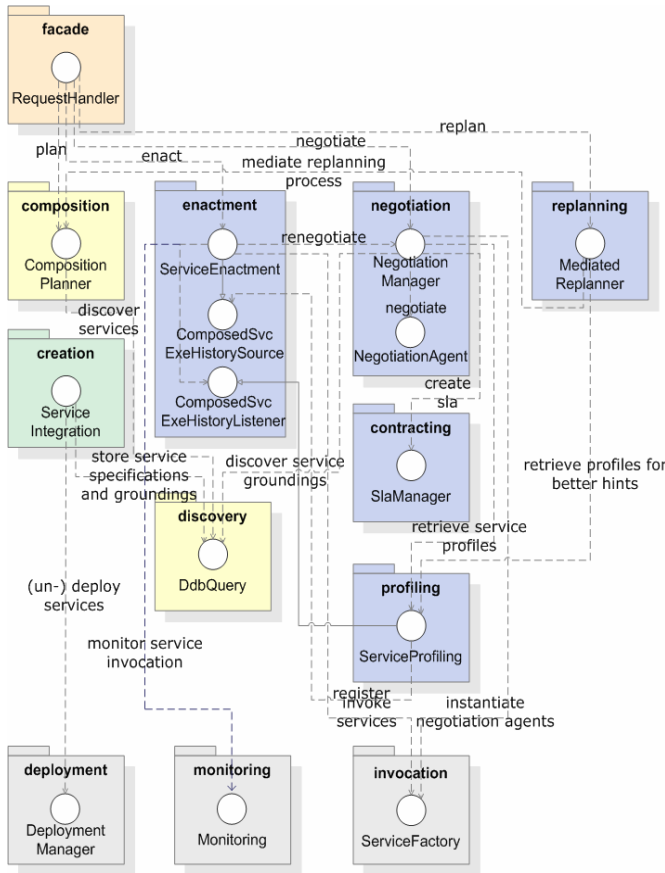
The ontology-language choice a) is important and has ramifications to many other points. A GLD (standardization body or research project) designs the meta-meta level concepts and the corresponding ontology language. Main requirements for it are [18 p. 68]: a well-defined syntax; a well-defined semantics; efficient reasoning support; sufficient expressive power; and convenience of expression.

Currently there are several approaches to this that are in a research phase. Web Ontology Language (OWL) [18, pp. 67-92] is based on RDF and RDF schema. Based on OWL, OWL-S was developed [10,19]. OWL-S addresses the needs to reason, compose and enact web services. Upper-level ontology consists of Service Profile (provider, functional description (precondition&input param, effects&output param), and service properties), Service Model (how to interact with the service, conditions), and Service Grounding (how it can be concretely enacted). Searching for a service specification happens by clients defining a profile and matching it with the service profiles in the registry.

The Web Service Modeling Ontology (WSMO) [20], developed mainly by DERI [21], is another approach. Starting from it, DERI and many EU-projects have developed the Web Service Modeling Language (WSML) and Web Service Execution Environment (WSMX) [22]. WSML has several subsets, WSML-Core (a subset of OWL), WSML-Rule, WSML-DL, and WSML-Full. The top layer contains the concepts ontologies (provide terminology, expressed in WSML), goals (a pre- and post-condition pair that describes user intentions), web services (computational entities with name, pre- and post-conditions, input&output parameters at corresponding ports, goal reference), and mediators.

The latter resolve various differences within the global homogeneous area spanned by various web standards and between it and the local environments. These include

Fig. 1: The reference architecture of SWS platform [23]



mediation of data structures (of the exchanged data), business logic (of the parties), message exchange protocols and dynamic service invocation. The mediation is assumed to happen between two or a few parties, i.e. it is assumed that there is no one OID that would dictate how standard service descriptions would look like [13].

A third approach is Semantic Web Services Framework (SWSF) [24]. “The Semantic Web Services Language (SWSL) is used to specify formal characterizations of Web service concepts and descriptions of individual services. It includes two sub languages. SWSL-FOL is based on first-order logic (FOL) and is used primarily to express the formal characterization (ontology) of Web service concepts. SWSL-Rules is based on the logic-programming (or “rules”) paradigm and is used to support the use of the service ontology in reasoning and execution environments based on that paradigm.”[24]. The ontology is called the First-order Logic Ontology for Web Services (FLOWS), and it is expressed using SWSL-FOL. FLOWS contains not only ontology for individual service specifications, but also ontology for process (workflow) concepts and exceptions. The ontology has a full axiomatisation in FOL.

The fourth proposal is WSDL-S [26]. It extends WSDL with semantics (e.g. pre- and post-conditions) relying on the extensibility of WSDL 2.0.

It is beyond the scope of this paper to investigate in detail how the proposed approaches would influence the architectural choices.

4. A Platform Reference Architecture

Reference architecture for the central components, called together a *semantic web service platform*, is presented in a Fig. 1. The boxes in Fig. 1 are software modules with an open interface specification, based on XML. The implementation of these *subsystems* can vary, as long as the specified interface behavior is obeyed.

4.1 Service Creation

Depending on what is the modeling power and reasoning capabilities of the ontology language are, the web services architecture can be more or less dynamic. Because modern ontology languages are rooted in various first-order logics (e.g. Description logic(s), F-logic [18]), they have well-defined formal semantics based on model theory and have algorithmic reasoning support.

Ontology is a monolithic entity in that respect that in order to make reasoning possible, all service descriptions subject to the reasoning (matching the semantic goal and service precondition+source state, end state+postcondition; see below) must be represented as concepts and integrated into the same ontology. This introduces through the backdoor an OID at the service instance level, because ontology that contains conflicts is useless

Functioning of the platform in [23] is based on WSMO [20,22]. The (imported) ontology contains the embedding of the web service to the domain ontology. *Capability* defines what the web service does (cf. actions above). The *precondition* determines when the “information space”, referred to by the input parameter values has such a state that the web service can be successfully enacted (cf. OWL-S, WSDL-S, SWSF). *Post-conditions* define the resulting output parameter values in the information space after the execution. *Assumptions* refer to the “world state” before the service execution and *effects* to the “world state” after it. The local system state is assumed to belong to the “world state”, but an assumption can also refer to physical world state (cf. real actions):

In meta-choreography part there is a description on how the service level agreements (see below) can be negotiated for this web service. The interface specification also allows describing the protocol machine and messages (orchestration) towards other web Services that this web Service needs in order to reply to the request (cf. E-autonomy).

The service registration is assumed to happen in the reference architecture with the help of the Service Creation/Integration component. This component checks also the consistency of the ontology stored into the

registry (called Discovery Database (DDB)), after the insertion of the new web service specification. The right super concept in the concept hierarchy must be found or new concept created, if this is necessary from modeling point of view. In WSML, a concept and its super concept have the same signature, consisting of a list of (attribute, attribute type) –pairs.

Using a SWS platform requires that those atomic service providers that want their web services to be available on it must *register* their services there, in a similar fashion as if they would register them at a UDDI. The difference is that their web service specifications must have directly syntax and semantics represented according to the IOL used by the platform, or they must be translated into this form during the registration. As opposed to EDI, *mediators* [13] could be used to map the individual service semantics at external interfaces to the global one, facilitating the reasoning in the global homogeneous area.

Further, the *grounding of the service* is facilitated at the deployment manager component. It means that the concrete service interface, URI of the service, and the handling rules of the concrete message instances are described for the infrastructure level. The handling requires in general code generation for a proxy that mediates between the abstract service description and the concrete service specification and its enactment. In addition, an internal reference is created that binds the proxy and the service description in the discovery database.

4.2. Service Composition

In general, one can think of several strategies for the service composition. (See [26,27] for a more detailed analysis). The most inflexible one is where the possible compositions (based on the general, stateless matching of service specifications) are compiled and stored into the registry. This might work well in environments where majority of requests fall in a few categories and only parameters change. This calls for *abstract compositions* that are instantiated with concrete services [28]. The approach does not work well in environments, where the mix of requests leads to wide variety of compositions; the approach requires also usually manual work.

The most dynamic approach is where the composition is done based on the incoming service request, i.e. while executing the request. In order to function at all, the composition must be performed in a fully automatic fashion. A necessary condition for this is that the service request is formulated in a machine-understandable way. Different approaches to web service ontology definition suit more or less well to this task.

The matching of needs and service specification in the registry (discovery database, DDB in Fig. 1) is based first on the concept hierarchy to which the goal instance

belongs (e.g. trips). Further, the (abstract) web service description that has a matching capability and a desired interface, are chosen as the starting service specification, if such a service specification exists. If the post-condition and the desired effect in the goal and in the first service do not match, the composing algorithm selects a further service specification that matches the first in pre- and post-conditions and checks, whether the post-conditions and desired effect in the goal now match with second service specification. This process is continued until output match is found or it is stated that there is no composition that could satisfy the goal. The process can also be performed backwards, starting from the post-condition and effect in the goal and also in parallel to both directions. The problem can be understood as a planning problem. A concrete algorithm is described in [29].

The benefit of the dynamic approach is that it can respond to any service request (goal) that refers to a suitable ontology. Further, the compositions once compiled for a goal can be stored and linked to the goal and thus reused without the need to perform the composition again. Another benefit is that the concrete parameter values are present when the composition is performed. This makes it possible to select e.g. only such services into the composition the cost of which is cheap enough (in the non-functional properties of the service description and goal this information can be given), or which will indeed provide services to the area where the person is going to travel.

A drawback of this approach is that the composition time is included into the response time of the service request and cannot necessarily be kept under an acceptable level. Another issue is that the goals must be specified in a precise manner (in WSML). The task is so complicated that an ordinary user is not able to do it without sophisticated tool support, if at all. In the reference architecture it is foreseen that there is a front-end application that transforms a simple user request to a semantically equivalent goal in WSML. The goals are also stored for further usage.

The result of the planning phase platform is an (enhanced) WS-BPEL program that describes the control flow and data flow between the atomic service descriptions composed, along the parameter values, and reference list to the deployed implementations (proxies).

4.3 Service Enactment

This is essentially a workflow engine. It gets the WS-BPEL program above to be executed. But in order to add flexibility into the enactment a dynamic Service Level Agreement (SLA) negotiation is perhaps performed with the atomic service providers that would offer the concrete services for the service descriptions in the WS-BPEL. If deemed reasonable and possible the negotiation is performed by Negotiation Manager and the results are

kept in an instance of SLA by the SLAManager subsystem. The SLAs basically only cover the ongoing enactment, i.e. are instance-related, although the architecture also supports fixed contracts recorded at service creation time.

The benefit of dynamic negotiation is that users might gain economically or in terms of QoS of the service enactment, while choosing the best provider (at that moment). The drawback is that negotiation time is added to the response time of the request observable by the user. Further, this approach requires an additional *dynamic negotiation interface* from the providers, whereas the standard WSDL-based services can be accessed from the platform without further requirements. If the service provider does not provide this kind of interface, then the only possibility is to use a fixed service agreement that is recorded into the Discovery database when the service is registered (cf. requirement g) above).

The dynamic planning makes also a dynamic recovery during the enactment through re-planning feasible. That is, if one or several atomic service requests fail when the composed service is enacted, e.g. because the server is down or because that airline does not have flights available, the composition can be re-planned during execution. The newly planned part of the program is then enacted again. A more detailed description of this feature can be found in [30].

Profiling subsystem monitors the behaviour of the service providers during service enactment and checks whether the SLAs are obeyed by them (service instance). It also compiles statistical information that can be attached to the service description within the DDB and used while planning

The workflow engine obeying the control flow restrictions invokes the individual atomic services offered by the providers. The invocation component uses the deployed proxy for the concrete service instance, binds the parameters and lets the proxy call the concrete service at the URI attached to it. Monitoring observes the behaviour of the atomic service and collects the raw data on execution times etc. and hands it to the enactment component.

5. Conclusions

We have analyzed in this paper the semantic web services as the latest phase of inter-organizational computing (IOC). In these environments, a Global Language Designer (GLD) and Operational Interface Designers (OID) are needed. Standard message types were the first approach (in international banking); EDIFACT introduced a language to specify message types. Semantic web adds one more meta-level, i.e. meta-concepts on which an IOL is based. With the latter, individual semantic service specifications can be given. Since

EDIFACT there has been no guarantee that the language constructs defined for the same purpose by different groups relying on IOC would have the same structure, be it service specifications or messages. Although the machine interpretable service annotations, ontologies and pre- and post-conditions, infrastructure supporting them, promise a full automation when services are enacted, there are several prerequisites to be fulfilled. First, people must provide the operational semantics for individual message types at the external interface and probably compile the semantic annotations, including ontologies. A specially the latter seems to be a tedious task that requires high skills and is thus a costly point, making reusability of the ontologies an interesting issue. In this vain, also the goals specified in WSMO need to be defined and hopefully reused. The existence of formal goals requires a component that mediates between user issued simple requests and the formal goal into the architecture. Programs could embed the goals directly. Second, automatic composition of services seems to require that the composition algorithm is centralized and the annotation data needed by it as well. Further, the annotations should be presented with the same formal language and be semantically non-contradictory. The latter requirement introduces through “backdoor” a global designer to the level of individual service specifications. It is currently an open issue, whether service composition can be done using heterogeneous semantic service descriptions.

Another issue is, whether ontologies and goals could be generated from natural language descriptions automatically or semi-automatically. This raises the question: to which extent can the tasks of a GLD or OID can be automated?

The platform reference architecture is currently being prototypically implemented by the ASG project as a centralized solution. The management support should be further developed and dependability and performance aspects addressed. One of the crucial technical issues from this perspective is the time and space complexity of the automatic composition algorithms (matchmaking based on FLORA-2) and it is currently being investigated.

Acknowledgements

The work of the authors was partially funded by the European Commission under the contract No 004617 (Adaptive Service Grid). The first author is on leave from Univ. of Jyväskylä, FIN-40014, Finland.

References

- [1] Society for Worldwide Interbank Financial Telecommunication. Accessed on May 21, 2006 at <http://www.swift.com/>
- [2] J.Veijalainen, Issues in Open EDI. P.Ng, C. Ramamoorthy, L.Seifert, R.Yeh (eds), *Proceedings of the Second Intl. Conference on Systems Integration (ICSI'92)*, IEEE CS Press, NY, 1992, 401-412.
- [3] J.Veijalainen, *Transaction concepts in autonomous database environments* (Ph.D. thesis, GMD-Bericht Nr. 183, R. Oldenbourg Verlag, Munich, Germany, 1990).
- [4] ISO 9735 Electronic data interchange for administration, commerce and transport (EDIFACT) - Application level syntax rules (first edition 1988, amended 1990). ISO, 1990.
- [5] R.Chinnici etal, Web Services Description Language (WSDL) Version 2.0 Part 1:Core Language; W3C Candidate Recommendation 6 January 2006. Accessed on Jan. 31, 2006 at <http://www.w3.org/TR/wsdl20/>.
- [6] A. Arkin etal, Web Services Business Process Execution Language Version 2.0. Committee Draft, 21th December 2005. Accessed on May 21, 2006 at <http://www.oasis-open.org/apps/org/workgroup/wsbpel/>
- [7] T.Erl, *Service-Oriented Architecture; Concepts, Technology and Design* (Englewood Cliffs, NJ: Prentice-Hall, 2005).
- [8] R.Sen, T.Hull, H.Chinoy, *XML for EDI Making E-Commerce a Reality* (Boston: Academic Press, 2006).
- [9] L.Clement, A. Hately, C.von Riegen, T.Rogers(eds.), UDDI v3.02 UDDI Spec Technical Committee Draft, Dated20041019. Accessed on 28.1.2006 at <http://uddi.org/pubs/uddi-v3.0.2-20041019.pdf>.
- [10]The OWL Services Coalition, OWL-S, Semantic markup for web services; White paper 2004. Accessed on Jan. 27th 2006 at www.daml.org/services/owl-s/1.0/owl-s.pdf.
- [11]T.R.Gruber A translation approach to portable ontology specifications". *Knowledge Acquisition*, 5(?), 1993, 199–220.
- [12] M.P.Papazoglou, Service-Oriented Computing: Concepts, Characteristics and Directions; Keynote for the 4th International Conference on Web Information Systems Engineering (WISE 2003), December 10-12, 2003. Accessed on 29.1.06 at <http://ieeexplore.ieee.org/iel5/8885/28063/01254461.pdf?tp=&arnumber=1254461&isnumber=28063>
- [13] D.Fensel&C.Bussler, The Web Service Modeling Framework WSMF, *Electronic Commerce Research and Applications*, 1(2), 2002,113-137.
- [14] A.Avizienis, J.-C.Laprie, B.Randell&C.Landwehr, Basic Concepts and Taxonomy of Dependable and Secure Computing, *IEEE Transactions on Dependable and Secure Computing* 1(1), 2004, 11-32.
- [15] P.Helland., Data on the Outside Versus Data on the Inside, *Proc. 2005 Conf. on Innovative Database Research (CIDR)*, January, 2005. Acc. on Jan. 29, 2006 at www.only4gurus.net/miscellaneous/datainsideoutside.pdf
- [16] Meta-Object-Facility. OMG...
- [17] J.Veijalainen, S. Nikitin, V. Törmälä, Ontology-based Semantic Web Service platform in Mobile Environments, Proc. of MOSO Workshop, May 9, 2006 in Nara, Japan, 32-39. Available at <http://csdl2.computer.org/persagen/DLPublication.jsp?pubtype=p&acronym=MDM>.
- [18] S.Staab, R.Studer (eds.), *Handbook of Ontologies*. (Heidelberg/NY: Springer Verlag, 2004).
- [19] D..Martin et al. OWL-S: Semantic Markup for Web Services, 2003, ,Accessed on Jan. 30, 2006 at <http://www.daml.org/services/owl-s/1.1/overview/>,
- [20] D.Roman,U.Keller, H.Lausen, J. de Bruijn, R. Lara, M.Stollberg, A.Polleres, C.Feier, C. Bussler and D.Fensel, Web Service Modeling Ontology, *Applied Ontology* 1 (1),2005, 77–106.
- [21] Digital Enterprise Research Institute, University of Innsbruck. www.deri.org
- [22] ESSI WSMO working group. Accessed on May 21, at www.wsmo.org.
- [23] Adaptive Services Grid Consortium, ASG brochure. Accessed on Jan. 28 at <http://asg-platform.org/>
- [24] S.Battle etal. Semantic Web Services Framework (SWSF) Overview; W3C Member Submission, 9 September 2005, Accessed on Jan. 30, 2006 at <http://www.w3.org/Submission/SWSF/>
- [25] J. Miller etal, WSDL-S: Adding Semantics to WSDL - White Paper, 15.7.2004. Accessed on Jan. 31, at <http://lsdis.cs.uga.edu/library/download/wsdl-s.pdf>
- [26]U.Kuester, M.Stern, B.König-Ries, A Classification of Issues and Approaches in Automatic Service Composition, *Proc of WESC05, IBM Research Report RC23821*, IBM 2005, 25-34.
- [27]D.Kuropka, M.Weske, Die Adaptive Services Grid Plattform: Motivation, Potenzial, Funktionsweise und Anwendungsszenarien, *EMISA Forum* 26(1), 2006, 13-25.
- [28] C.Vetere&M.Lenzerini, Models for semantic interoperability in service-oriented architectures. *IBM SYSTEMS JOURNAL*, VOL 44, NO 4, 2005, pp. 887-903.
- [29] H.Meyer, Entwicklung und Realisierung einer Planungskomponente für die Komposition von Diensten, Diplomarbeit, 2005. Hasso-Plattner-Institute, University of Potsdam.
- [30] M.Gajewski, M.Momotko, H.Meyer H.Schuschel M.Weske, Dynamic Failure Recovery of Generated Workflows, *Proc. of 16th International Workshop on Database and Expert Systems Applications (DEXA'05)*, 2005, 982-986.