Automated VAlidatioN of Trust and Security of Service-oriented ARchitectures

FP7-ICT-2007-1, Project No. 216471

www.avantssar.eu

Deliverable D6.2.1

State-of-the-Art on Specification Languages for Service-Oriented Architectures

Abstract

This deliverable provides an overview of state-of-the-art specification languages for Service-Oriented Architectures (SOA) and their implementation via Web Services. This knowledge is helpful for defining both AVANTSSAR languages: ASLan, the formal language, and ISSL, the Industrially-Suited Specification Language, devoted to specifying trust and security properties of services, their policies and their composition. The deliverable is conceptually organized according to the W3C service protocol stack so as to emphasize the different service architecture layers. For each of them, we describe their corresponding languages and evaluate whether their properties could be of interest in the scope of AVANTSSAR.

Deliverable details

Deliverable version: *v1.1*
Date of delivery: *30.06.2008*
Classification: *public*
Editors: *all*

Person-months required: *6*
Due on: *30.06.2008*
Total pages: *67*

Project details

Start date: *January 01, 2008*
Duration: *36 months*
Project Coordinator: *Luca Viganò*
Partners: UNIVR, ETH Zurich, INRIA, UPS-IRIT, UGDIST, IBM, OpenTrust, IEAT, SAP, SIEMENS

Contents

List of Figures

1 Introduction

This deliverable provides an overview of established specification languages for Service-Oriented Architectures (SOA) and their implementation via Web Services.

The AVANTSSAR workpackage 6 is concerned with dissemination and industry migration. One of its aims is the migration of AVANTSSAR research results to industrial development environments (e.g., Eclipse, NetBeans). This survey will help us to acquire some knowledge on the state-of-the art of standard languages that could be of interest for the specification of the AVANTSSAR languages: ASLan (the formal language) and ISSL (the industrially-suited language), in the sense that both languages will probably consist of extensions to standard languages.

ASLan will be a formal language for specifying services, their trust and security properties, their associated policies, and their composition into service architectures.

ISSL, the industrially-suited specification language, on top of the formal language (ASLan), will provide means to specify business processes and their interaction. It should describe the coordination necessary to achieve a given goal, including notions to describe both the logic and the stateful information needed for such a coordination.

The SOA properties, such as loose coupling, interoperability, reusability, etc., provide a good computer system architectural style for creating and using business processes, packaged as services, throughout their lifecycle. As a starting point of the survey, we begin with a short history of services, to emphasize the main characteristics of service orientation: their heterogeneous, distributed, and dynamic nature. Then we propose some basic definitions and concepts of the service architecture, as the design of the AVANTSSAR languages needs to take into account the existing languages for service description and their orchestration.

We structure our document relying on the W3C service protocol stack, so as to distinguish the different service architecture layers. We present the corresponding languages and evaluate whether they could be of some interest for AVANTSSAR.

Because the term "service" has many different meanings and is so widely used in different areas of computer science, it is important to describe what "service" means in the context of the AVANTSSAR project. We start from the very general and easy-to-read definition and then proceed towards the more specialized one, as proposed by the normalization consortium.

The Merriam-Webster online dictionary contains 46 entries for the term "service" and emphasizes its relation to the words "slave" or "servant". In its

predominant meaning, "service" thus describes "the occupation or function of serving." This definition fits well within the computer science area, as the main purpose of machines and computers is to help and serve humans and society. This could be a reason why the term "service" is so widely used within computer science.

The Organization for the Advancement of Structured Information Standards (OASIS) defines a service as "a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description" [15].

Nowadays, SOA is typically implemented by Web Services, such that services are made accessible via Web interfaces using XML. The World Wide Web Consortium (W3C) [21] defines a Web Service as "a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format. Other systems interact with the Web Service in a manner prescribed by its description using SOAP messages" [23].

It has to be noted that a service-oriented architecture is not tied to a specific technology. It may be implemented using a wide range of technologies, including SOAP, RPC, DCOM or Web Services. A SOA can be implemented using one or more of these protocols. It might, for example, leverage file system mechanisms to communicate data using a defined interface specification between processes conforming to the SOA concept. The key is independent services with defined interfaces that can be called to perform their tasks in a standard way, without the service having foreknowledge of the calling application, and without the application having or needing knowledge of how the service actually performs its tasks.

Service-oriented applications are heterogeneous: the various individual components may be built using different technologies and run in different environments. Nevertheless, both the components and their requirements may interact, and in some cases even interfere with each other. Still, most currently existing security solutions are limited to protecting applications within a single security context.

An important point to mention for the AVANTSSAR project is that service-oriented architectures are distributed systems, i.e., the functionalities and resources are distributed over several machines or processes communicating via messages. The message passing nature of interactions of Web Services and of other security-sensitive services increases their vulnerability: even assuming that the cryptographic primitives work correctly (that is, the system cannot be attacked exploiting weaknesses of cryptographic keys or of encryption/decryption algorithms), it allows for attacks based on interception, mod-

ification, and replay of messages. However, future service-oriented software architectures need to consider security among multiple/heterogeneous security contexts: for instance, between a requester and a service there might be multiple intermediaries that must be able to read or even modify the contents of a message, or the same service might be used in different environments (e.g. mobile vs. stationary). Thus, a notion of static end-to-end security is not applicable here, as in the case of simpler IP-communications.

When considering two of the main characteristics of service orientation, their heterogeneous and distributed nature, the complexity of reasoning about trust and security aspects of services and service-oriented architectures becomes evident.

The AVANTSSAR technology will provide the ability to formally model and automatically reason about (Web) services, their composition, their required security properties and associated policies, both at network and application level.

2 History and context of Services

2.1 History of Services and Remote Procedure Calls (RPC, DCOM, CORBA)

Looking back at the history of software engineering, the ideas of service-oriented programming and service-oriented architectures (SOA) rely on distributed computing technologies. The concept is to build upon object-oriented and component-based programming by strongly decoupling application components from each other. The term "Remote Procedure Call" (RPC) goes back to at least 1976, when it was described in [65].

RPC is an obvious paradigm for implementing a client-server model of distributed computing. In the object-oriented context, RPC may also be referred to as "Remote Method Invocation" (RMI).

An RPC is initiated by the *client* sending a request message to a known remote *server* in order to execute a specified procedure using input parameters. The client typically suspends itself waiting for a response, and then continues along with its process (synchronous mode). There are many implementations, resulting in a variety of different (and usually incompatible) RPC protocols.

The first service-oriented architecture emerged with the use of DCOM [27] and Object Request Brokers (ORBs) based on the CORBA [26] specification. DCOM is the acronym for the Distributed Component Object Model, an extension of the Component Object Model (COM). It was introduced in

1996 and is designed to be used across multiple network transports, including Internet protocols such as HTTP. It runs on Microsoft platform.

CORBA is the acronym for Common Object Request Broker Architecture. It was developed by the Object Management Group (OMG), an international, open membership, non-profit computer industry consortium [48]. The Object Request Broker (ORB) is a middleware that uses the CORBA specification. The ORB handles all of the details involved in routing a request from client to object, and getting the response back. A CORBA-based program is language and platform-neutral.

These protocols are the pillars of distributed computing technologies. They can be seen as the ancestors of current Service Oriented Architecture protocols, like SOAP [23].

2.2 Basic terms and concepts

We will continue exploring the main features of service orientation. The main goal of the service-oriented architecture paradigm is reducing development costs by strong re-usability of software components. To this end, services are encapsulated and loosely coupled. The client–service relationship is eased by giving service providers means to advertise their business functionalities. It obviously also has to give clients means to search for service providers matching their business needs. The basic concepts of the SOA paradigm features three main actors:

- a **service provider** that offers access to some business functionalities,
- a **client** that access some business functionalities and
- the **service registry** that makes the glue between clients and service providers. Service providers can register their business functionalities while clients can browse the registry listings.

The service architecture relies on message exchange between these main actors.

As mentioned in the AVANTSSAR proposal, to meet frequently changing requirements and business needs, (e.g., in a federation of enterprises), components are replaced by services that are distributed over the network (e.g. the Internet), composed statically or at run-time in a demand-driven and flexible way.

Atomic services provide functionalities for the business logic. They can also provide the basic building blocks of security functionalities. The description language for atomic services must allow one to express the mechanisms

used for service identification, service invocation, and message transmission, including the formats used. Web Services and their description languages offer a good starting point.

Each service may rely on the existence and availability of other (possibly, dynamically retrieved) services to perform its computation; moreover, this includes dynamic adaptation and explicit combination of applicable policies, which determine the actions executed and the messages exchanged. For example, a service granting the access to a resource of a business partner may use a local authentication service, trusted by both partners, to assess the identity of a client and rely on authorization services on both ends that combine their policies to decide whether to grant the access or not.

AVANTSSAR languages need to provide appropriate language elements, such as channels, principals, identities (referring to subjects and resources), obligations and trust. They have to be accompanied with means to specify the composition, in terms of service orchestration, choreography, invocation, message passing, etc.

We have started our investigations by examining existing service description languages, the features they offer, and their limitations for expressing interaction and composition in the security domain. As Web Services can be used to implement a service-oriented architecture, the study of some Web Service protocol stacks was useful to distinguish the different service architecture layers for the purpose of new language definitions. The AVANTSSAR languages have to be sufficiently expressive to interact with most of these service layers.

2.3 Document Structure

Based on the W3C Web Service architecture targeted for integrating interacting applications [25], Figure 1 [33] gives an example of a service architecture stack.

Let us first give an overview of the different layers that could be of interest within the AVANTSSAR scope:

- **Transport and Messaging Layer**: describes the various transport protocols that can be utilized by Web Services.
- **Web Services Description Language (WSDL):** describes the static interface of a Web Service. It defines the message set and characteristics of end points.
- **Reliable Messaging and Security layer**: guarantees the delivery of information exchanged between participants and provides security

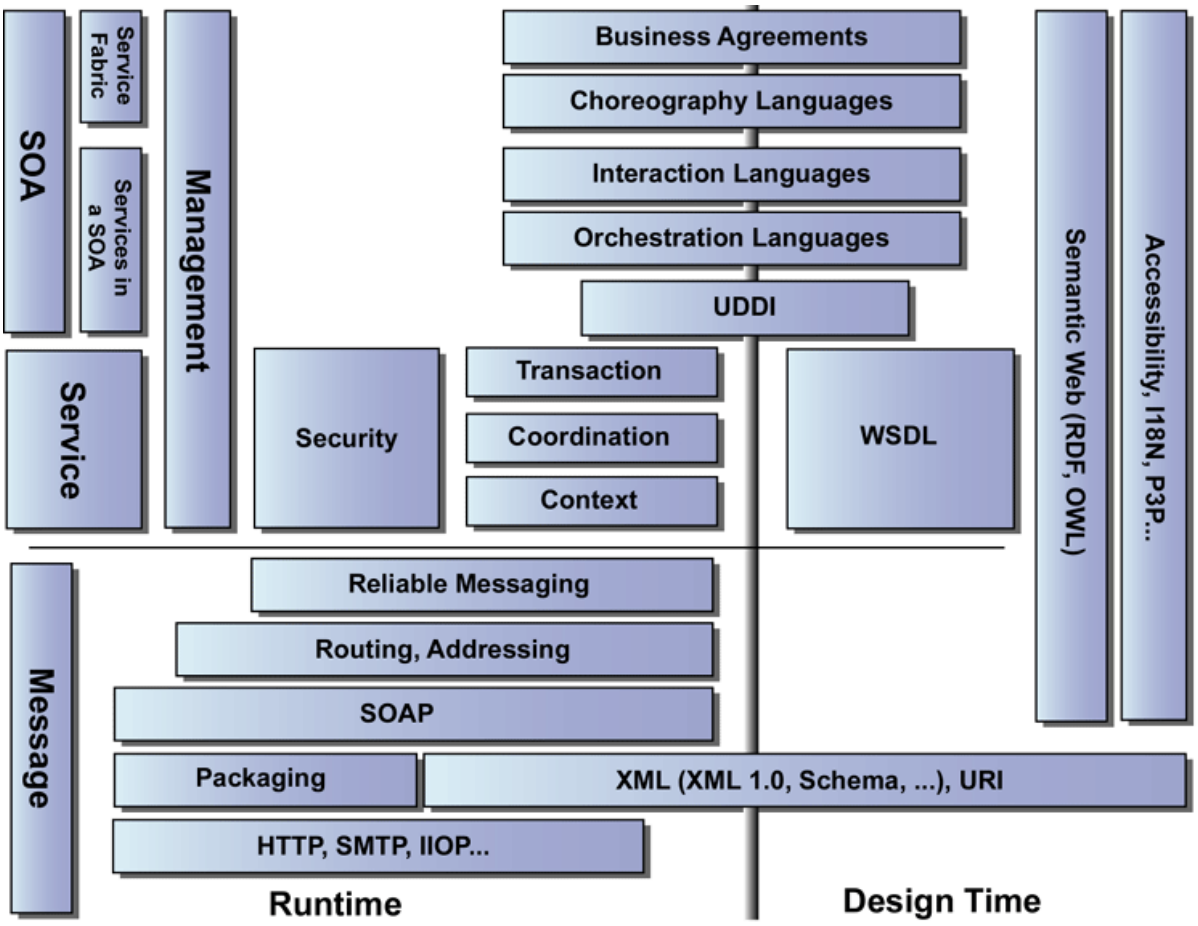

Figure 1: Web Service Protocol Stack

functionalities to cope with the security needs of the participants.

- **Context, Coordination and Transaction Layer**: interoperable protocols that support distributed transactions across platform boundaries.
- **Registry (UDDI)**: enables the publishing of a Web Service, as well as its discovery from service requesters using sophisticated searching mechanisms.
- **Business Process Language layer (Orchestration)**: describes the execution logic of service-based applications by defining their control flows and prescribing the rules for consistently managing their data.
- **Choreography Language layer**: describes collaborations of participants by defining from a global viewpoint their common and complementary observable behavior, where information exchange occurs, when the jointly agreed ordering rules are satisfied.

In the following sections, we will focus on these different language layers to evaluate them and decide whether their properties should be taken into account for the AVANTSSAR language specifications.

3 Transport and Messaging Layer

Transport and messaging layers are the foundation layers of the service-oriented architecture. For our ASLan/ISSL language specifications, we need to consider the message exchange behavior among participating Web Services.

3.1 TCP, UDP / HTTP, SMTP

Transport protocols are required to facilitate message delivery. The Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP) belong to the core protocol stack of the Internet. TCP, as a session-based, reliable and in-order delivery transport protocol, is suitable for applications like file transfer, e-mail, exchanges between Web servers and clients. UDP is a fast and efficient protocol that is stateless and unreliable; datagrams (short messages) may arrive out of order, duplicated or missed without notice. Unlike TCP, UDP is compatible with broadcasting (sending to all on local network) and multicasting (sending to all subscribers).

The most common communication protocols are the Hypertext Transfer Protocol (HTTP) (or its secured variant HTTPS) and the Simple Mail Transfer Protocol (SMTP), thus by sending XML requests, and getting XML responses over the transport protocol. HTTP is a request/response standard between a client (an end-user) and a server (a Web site). HTTP is not constrained to using TCP/IP and its supporting layers, it only presumes a reliable transport. SMTP is a relatively, text-based protocol. It is the de facto standard for e-mail transmissions across the Internet.

Evaluation

The AVANTSSAR languages should be able to handle the message delivery over any transport and communication protocols. Besides these mechanisms, we will now focus on the way a message is formatted and delivered, independently from an operating system, programming language, or platform.

3.2 REST (Representational State Transfer)

The Representational State Transfer (REST) [32] is a network architecture paradigm relying on standard transport protocols like HTTP, without the use of an additional messaging layer. A service call is handled via its URI. HTTP provides the standard operations (Get, Post, Put, Delete) as procedure calls.

Security is handled as for a standard Web application (using SSL, sessions, cookies, etc). REST is a client-server, stateless, cacheable and layered network paradigm. The World Wide Web is the key example of a REST design.

3.3 XML-RPC

XML-RPC [66] is a Remote Procedure Call protocol that uses XML to encode the messages and HTTP to handle them.

A network node (the client) sends a request message to another node (the server), which sends a response message to the client.

XML-RPC is a precursor to SOAP. It is sometimes preferred to SOAP because of its simplicity, minimalism, and ease of use.

3.4 SOAP (Simple Object Access Protocol)

The Simple Object Access Protocol (SOAP) [23] defines a Remote Procedure Call using an XML messaging protocol for basic service interoperability. SOAP once stood for 'Simple Object Access Protocol', but this acronym was

dropped with the version 1.2 of the standard, as it is not simple anymore and it is not only used to access objects. It is a messaging framework for transferring information between an initial SOAP sender, optionally some intermediate receivers and an ultimate SOAP receiver.

This protocol is non-proprietary (it became a W3C Recommendation in 2003) and platform and language independent. It can be run over a simple transport protocol (e.g., HTTP or SMTP). There are examples of the usage of SOAP services over the transport protocols in [57]. For instance, one can also explore sending and receiving service-oriented requests over the Simple Mail Transfer Protocol (SMTP). In fact the nature of the service-oriented architecture enables one to expose services over any protocol, even beyond those described in official bindings such as TCP, named pipes, UDP and custom transport protocols.

A SOAP message is an XML document containing the following elements:

- A required Envelope element identifying the XML document as a SOAP message.
- An optional Header element containing header information.
- A required Body element containing call and/or response information.
- An optional Fault element providing information about errors that occurred while processing the message.

The required SOAP Envelope element is the root element of a SOAP message. It defines the XML document as a SOAP message.

The optional SOAP Header element contains application specific information (like authentication, payment, etc) about the SOAP message. As we will see in 5 there are several standardized SOAP-Header extensions like WS-Reliability, WS-Security, etc. The SOAP actor attribute may be used to address the Header element to a particular endpoint, as a SOAP message may travel from a sender to a receiver by passing different intermediaries along the message path.

The required SOAP Body element contains the SOAP message intended for the endpoint of the message.

Evaluation

SOAP, XML-RPC and REST are commonly used in the service-oriented architecture scope. They should be taken into account for our ASLan/ISSL specifications, as the use of different communication protocols could lead

to compatibility and security problems. To ensure integrity and privacy of messages during the transfer, some security and trust requirements need to be fulfilled. This could be achieved at the transport level or in the message itself.

By examining the protocols mentioned in this section, we have seen that Web Services communications can be supported by various transport protocols. After the definition of the transport and messaging layer, we will now focus on the service description that guides the interaction between computer systems.

4 Web Service Description Language (WSDL)

The Web Service Description Language (WSDL) [22] is responsible for describing Web Services, especially the interface a Web Service exposes to other applications. Means for expressing service interfaces are at the core of all service models, and WSDL provides very flexible, highly-extensible, and well-designed methods for doing this. As most other Web Service standards, WSDL is based on XML. WSDL documents contain all information required for the use of a Web Service, including data types, message patterns, method descriptions, and service location. As a consequence, programming frameworks that are based on Web Services - such as Windows Communication Foundation (WCF) [12] - provide tools that consume a WSDL document and dynamically create the proxy code necessary for the use of a Web Service. In WSDL, Web Services are expressed as collections of endpoints that exchange messages. WSDL also contains information of how these messages are mapped to a concrete network protocol - a so-called binding - so that these messages can be exchanged in an interoperable fashion.

In brief, WSDL includes the definition of the following parameters:

- Data structures: data types required to interact with a Web Service are specified using XML Schema Definition (XSD) [62]. XSD is a W3C recommendation for describing the structure and content of an XML document.
- Operations: operations offered by a Web Service are specified in terms of input and output data.
- Service endpoint reference: URI of a Web Service, i.e., its location on the Internet.

The data types used in the messages of a service are described using XSD that supports a highly-flexible type system. The WSDL specification specifi-

cally mentions that XSD can be replaced with a different type system. XML and XSD are the pillars of SOAP, WSDL and WS-* document messaging requirements.

The specified types are used in WSDL messages, which are combined to a Web Service operation.

A Web Service operation consists of a list of messages, defining the operation's input and output messages. Similarly, a set of operations together with a binding and a network address, specified by a URI, comprises an endpoint (or a so-called port). A Web Service is then a collection of such related endpoints.

Evaluation

The AVANTSSAR language specifications should rely on WSDL because it introduces a common grammar for describing services.

We regard constraints and policies as part of a service's description. Constraints can specify, for instance, the amount of resources a service is allowed to consume in a certain time period; policies typically address security and privacy issues. We try to capture constraints and policies as extra-functional aspect of service descriptions.

It was already remarked above that WSDL is a core language tailored to describe Web Services based on an abstract model of what the service offers. Its standardization is being conducted by the Web Services Description Working Group [24]. When describing how to access a service, it is also important to publish the policies that determine which security mechanisms the requester must apply. This is an open issue today. In the recent WSDL-2.0 (January 2006) the issue is left out of scope, the only syntactical means is to use "secure-channel features", which are simply internationalized URLs (IRIs), without any semantics attached to them.

Within AVANTSSAR, we will focus our attention on the policies attached to atomic services as well as on other composed services. We will see that other Web Service standards such as WS-MetadataExchange or WS-Policy, for instance, could be used to express and exchange extra-functional information associated with a service.

In the following, we discuss some of these WS-* specifications that we envision to be important in the scope of the AVANTSSAR project. Their purpose is to facilitate common message requirements between different systems. At a certain point in time, it became mandatory to establish some kind of standardization when, for instance, one could send username and password credentials within some custom SOAP headers.

The WS-* specifications support and integrate various security models, mechanisms, and technologies in a way that enables a variety of systems to securely interoperate in a platform and language-neutral manner. Each specification addresses specific parts of the security framework. The specifications are flexible and extensible. The architecture essentially supports the secure exchange of SOAP messages, as described in the following.

5 Reliable Messaging and Security Layer

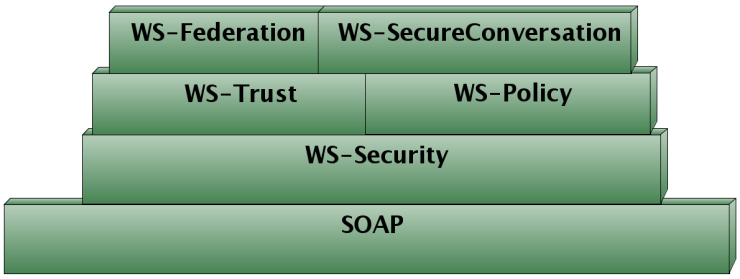

Figure 2: Web Service Security Stack

Given the security issues associated with open networked environments, it is just impossible to deploy business Web Services without a reliable and robust framework for secure end-to-end communication Web Service applications.

Consequently, several WS-* security related specifications have been standardized by the OASIS Open Consortium. Their stack is illustrated in Figure 2.

By design, the specifications are flexible and extensible to support many types of security information and models. However, different implementations need some guidelines to restrict the variety of options and choices in order to inter-operate. A security profile is defined by the Web Services Interoperability organization to meet this requirement.

Using the specifications does not guarantee absolute security of Web Services. In fact, careless combination or unsuitable use of the security options may lead to insecure systems.

5.1 WS-Security

WS-Security defines a SOAP Security Header format containing security-related information. A SOAP message may include multiple security headers. Each header is targeted at a specific SOAP actor/role that may be either the ultimate recipient of the message or an intermediary. Security headers may encapsulate one or many elements of the following types:

- Security tokens: they express claims that the service requester might have regarding the message's security. Such claims may concern identity, public key/subject binding, authorization etc... Currently, WS-Security profiles define token formats that represent User Name, X.509 certificate, Kerberos ticket, SAML assertion and Rights Expression Language licence.

- Signatures: they are built on top of XML Signature standard. They protect SOAP messages from unauthorized modification. Signatures are also used by message producer to demonstrate knowledge of a key. Signatures may contain references to public keys defined in security tokens (such as X.509 certificates). The specification allows multiple signatures to be included in a security header. They may reference body parts and headers of the SOAP message, including security tokens. In fact, Security tokens might be signed to prevent forgery. This is particularly useful for distributed applications where messages transit through multiple processing intermediaries.

- Encryption elements: their format and processing model are based on the XML encryption standard. Thus, they enable encryption of any combination of headers and body SOAP blocks. The specification allows encryption of data using a public key or an encrypted symmetric key.

- Timestamps: they allow the recipient of SOAP message to determine the freshness of the security header by providing means to express their creation and expiration dates.

By providing a common syntax and a flexible processing model for security headers, this specification accommodates a large variety of security models and encryption technologies. Moreover, incorporating security features in the application level ensures end-to-end security.

Evaluation

WS-Security foundation provides the basis for expressing security properties at message level. Thus, validating applicative security and trust requirements will not be possible without equivalent mechanisms in the AVANTSSAR languages. Moreover, the availability of these features in ASLan and ISSL is essential for supporting the WS specifications discussed below.

5.2 WS-Trust

This specification provides a framework built on WS-Security for managing security tokens. In the WS-Trust trust model, a requester examines the policy associated with a Web Service to identify the claims it needs. If the policy statements require security tokens that the requester does not possess, WS-Trust specifies a way of obtaining them: contacting a Web Service referred to as Security Token Server (STS). A STS may also be used to renew, cancel and validate security tokens.

WS-Trust defines abstract formats of the messages used to manage security tokens. To each usage pattern corresponds a specific binding providing concrete semantics to the general security token requests and responses.

For complex scenarios, WS-Trust describes flexible mechanisms for trust establishment. In fact, different STS may get involved to broker, exchange or delegate security tokens issuance. A general model for negotiation/challenge extensions is specified to support multi-messages exchanges for security tokens management.

The flexibility and extensibility of the specification allows interfacing with a large number of security models, including legacy protocols. In fact, increasing interoperability between trust domains is one of the purposes of this standard.

Evaluation

WS-Trust provides essential mechanisms for trust establishment. For their complexity and criticality, WS-Trust communication protocols require special attention. In fact, they cannot be ignored by the AVANTSSAR project, particularly when used in preparative phases for critical business transactions. Supporting WS-Security should make AVANTSSAR languages expressive enough to allow the analysis of WS-Trust tokens.

5.3 WS-SecureConversation

When a Web Service provider and a requester engage in a long-running and multiple-message communication, the use of the basic security model defined by WS-Security becomes unsuitable. In fact, repetitive signature and encryption of a large number of XML blocks is computationally expensive and degrades the security of the keys. WS-SecureConversation reuses the concept of session keys (defined by SSL/TLS) to enhance the model.

It works in conjunction with WS-Security, WS-Trust and WS-Policy to allow sharing of security contexts. A security context is represented by a Security Context Token (STC) defined in WS-Security Header. It corresponds to a shared secret used to sign or encrypt messages. It can be created by an STS, an active communicating party, or after negotiation. WS-SecureConversation reuses WS-Trust bindings to deal with STC issuance, amending, renewal and canceling. An STC can either contain a shared secret or imply it. In fact, this specification allows key derivation to have an even better protection against key analysis.

Evaluation

In the context of AVANTSSAR project, one could focus on validating security of the Context tokens and their management during a long running transaction. Modeling WS-SecureConversation scenarios should reuse features of ASLan and ISSL for expressing WS-Security tokens.

5.4 WS-Policy

The WS-Policy is a policy expression language for describing the capabilities and requirements of a Web Service, i.e. representing whether and how a message must be secured, whether and how a message must be delivered reliably or whether the request must follow a transaction flow. Such requirements are translated into 'machine-readable' policy expressions that are usually provided by the web service developer for the client component to automatically apply the requirements.

Basically, WS-Policy is a simple language that defines four elements (*Policy*, *All*, *ExactlyOne*, *PolicyReferences*) and two attributes (*Optional*, *Ignorable*) that suffice to express generic policy expression by combining individual assertions. The policy assertions syntax are outside the scope of WS-Policy specifications. Thus, WS-Policy can be viewed as a meta policy composition language that can express any kind of requirements as long as the

policy-aware clients (Web Services endpoints and relays) are capable of understanding the specific syntax of the unitary assertions.

An individual *policy assertion* expresses one requirement, behavior or capability related to messaging (how the message must be built), security (how the message must be secured through authentication or encryption), reliability (how to ensure that the message has been sent/received) and transaction (what transaction flow must be followed to ensure transaction commit). Examples of policy assertion languages that can be used are WS-SecurityPolicy (cf. subsection 5.5), WS-ReliableMessaging (cf. subsection 5.10). Other languages can be defined using assertion extensions.

Policy assertions are grouped within the WS-Policy XML element *Policy*, which constitutes a *policy expression*. A policy expression can be directly inserted into a WSDL file so as to make the WSDL a policy requirement self-describing interface.

Evaluation

As a standard container language for expressing Web Service security requirements, WS-Policy semantics are to be taken into account in AVANTSSAR languages, notably ISSL. WS-Policy is widely supported by Web Service solution editors, especially since it has become a W3C recommendation on September 2007 in its last version.

5.5 WS-SecurityPolicy

The WS-SecurityPolicy specifications defines a set of security policy assertions for use with the WS-Policy (cf. subsection 5.4) framework with respect to security features provided in WSS SOAP Message Security (cf. subsection 5.1), WS-Trust (cf. subsection 5.2) and WS-SecureConversation (cf. subsection 5.3).

The policy assertions defined in WS-SecurityPolicy fall into the following main categories:

- Protection assertions identify the parts of the message that is to be protected and the level of protection in terms of confidentiality (elements to be encrypted) and integrity (elements to be signed).
- Token assertions specify the type of tokens to use to protect or bind tokens and claims to the message. Such token can be typically a Kerberos token or a SAML token.

- Security Binding assertions identify the type of security binding being used to secure an exchange of messages. A security binding is a set of properties that together provide enough information to secure a given message exchange. A binding can define for instance how tokens are bound to messages, cryptographic algorithms and key transport mechanisms, or the content and ordering of elements in the WS-Security header.

- The properties are defined with regard to 3 kinds of security binding assertions:

  - transport binding when the required security property is applicable to the transport protocol instead of the Web Services message level.
  - symmetric binding when the initiator and the recipient of the message must use the same properties (for instance the same encryption token or signature token).
  - asymmetric binding when message protection is to be provided by means based on asymmetric key (Public Key) technology.

- WSS and Trust assertions express how certain specific aspects of the WSS and WS-Trust are to be applied.

Besides, WS-SecurityPolicy defines some ancillary specifications in order to provide some richer semantic for the security policy model, like Security Binding Properties which are default values or conditions for a security binding and that can be used by a binding in a manner similar to how variables are used in code

Evaluation

WS-SecurityPolicy provides the semantic for expressing a large panel of message level security policy assertions. Such assertions should be able to be translated in the ISSL language at least, although not all elements are to be considered since some of them are closely related to security implementation (Kerberos token properties, symmetric and asymmetric keys properties, etc.) and may not be relevant for higher level formal validation using ASLan.

One should note that WS-Policy (5.5) and its related policy assertion languages such as WS-SecurityPolicy express only partial aspects of a business process security requirements, which are the message and communication

level security requirements. The AVANTSSAR languages will have to address other security policy aspects such as application level authorization policies.

5.6 XACML

XACML (eXtensible Access Control Markup Language) is a "declarative" XML-based access control policy language used to describe the access control restrictions to actions on objects. Usually, access control models involve a *subject* (that is, either a user, a user on behalf of another user, a service, or a service on behalf of a user) making some access request and the system either authorizes this access request or denies it. XACML also defines a processing model, which describes how to operationally interpret the policies. XACML defines both an access control policy language (to express the access control conditions) and a canonical XML language to communicate with with a Policy Decision Point (PDP), to send to the PDP decision requests and obtain decision responses. This canonical form or language is called the XACML "context".

The current version of XACML, Version 2.0, was released by OASIS in February 2005. Version 3.0 is in preparation at the time of preparation of this document (June 2008). It is chartered to add generic attribute categories for the evaluation context and policy delegation profile (administrative policy profile). There are already some prototypical implementations of XACMLv3.0.

What can be specified in XACML. While traditional access control systems were based on the *identity* of the party requesting a resource, XACML adopts the principle of basing authorization decisions on characteristics or *attributes* of the subject, including, if required, its identity. The most common used attribute in access control is the subject's role [30, 31] within the application. The main problem with RBAC is mostly that a system has different role models for different applications. Thus, XACML supports attribute-based access control. In a *multi-domain, multi-lateral security environment* like SOAs on the Internet, this approach is much more effective because often the requester and the resource belong to different domains and are subject to different rules governing identities, authentication procedures and mechanisms, and permissions. While the identity of the requester may be meaningless for the owner of the resource, the attributes may be often translated or matched to corresponding ones. The meaning of an attribute may be a granted capability for a service, an identity, an asserted pseudonym,

or a non-identifying characteristic of a user (e.g., a degree, the job title, being over 18 years, the nationality, or any combination of them).

Similarly, basing authorization on attributes of the resource/service requested provides flexibility and scalability that is important in the context of large distributed open systems.

Not in scope of XACML is the process of providing *credentials* to subjects and verifying them, that is, the questions who is entitled to grant which credentials to whom, which credentials a local administration domain will accept, or even which formats, messages, or technology should be used for expressing the attributes, transporting the credentials and verifying the integrity and ownership of them. In this context, credentials are verifiable assertions that the requester possesses the claimed attributes. One possibility is to use attribute certificates, which are digitally signed assertions about the credential owner by a credential issuer. The attribute certificate would then contain the attributes that specify access control information associated with the certificate holder (e.g., age, citizenship, credit status, group membership, role, security clearance). Another approach, better integrated with XACML, is SAML.

XACML defines a simple yet very efficient and flexible class structure. Roughly, a Policy is composed of a Target, a set of Rules and an optional set of Obligations associated with a request. The Target element specifies the conditions that the requesting Subject, Resource and Action must meet for the Policy (or, in the proper context the PolicySet or the Rule) to be applicable. It provides an efficient method to indexing and looking up the applicable policy set or rules to a given request. To evaluate the rules of a Policy, the structure of the Rule is used: Target, an Effect and Conditions. As mentioned above, the Target describes if the Rule is applicable or not. The Conditions test the relevant attributes and decide if the Effect is applicable. All the outcomes are combined together, and yield an Effect of Permit, Deny or Indeterminate (an error condition, or ambiguity, which will be resolved higher in the hierarchy).

More formally, the three top-level XACML elements are: Rule, Policy and PolicySet. The Rule element contains a Boolean expression that can be evaluated on its own, but that is not intended to be accessed in isolation by a PDP. Instead, the Policy element, which contains a set of Rule elements and a pointer to an algorithm for combining the results of their evaluation, is the basic unit of policy used by the PDP, and so the basis of an authorization decision.

The PolicySet element contains a set of Policy or other PolicySet elements and a specified procedure for combining the results of their evaluations. It is the standard means for combining separate policies into a single

combined policy.

Semantics of XACML. The study of authorization by logical means was initiated by Abadi et al. [4]. For related more recent work see [7, 8, 28, 9, 36], and for a survey see [3].

There is work on providing semantics to XACML via Z (see [52]) and via Haskell (see [39] (for XACML v1.1), and [56] (for delegation in XACML). It is not surprising that the semantics of obligations in not formalized, as their intended meaning is really out of scope of XACML and only provided by the PEP implementation.

Data-flow. Roughly, the usage of XACML is the following: a subject wants to access a resource, whose access is controlled by a Policy Enforcement Point (PEP). When the subject makes the resource request, the PEP constructs an XACML request containing action, resource and other relevant informations, and sends it to the responsible Policy Decision Point (PDP) for making the authorization decision. The PDP will gather the applicable policies and determine the authorization decision. The authorization decision, response will be expressed using the XACML response language and deliver the decision to the PEP, which can then permit or deny access to the requester.

In more detail, the model operates by the following steps [14].

1. The Policy Access Points (PAPs) write policies and policy sets and make them available to the PDP. These policies or policy sets represent the complete policy for a specified target.

2. The access requester sends a request for access to the PEP.

3. The PEP sends the request for access to the context handler in its native request format, optionally including attributes of the subjects, resource, action and environment.

4. The context handler constructs an XACML request context and sends it to the PDP.

5. The PDP requests any additional subject, resource, action and environment attributes from the context handler.

6. The context handler requests the attributes from a Policy Information Point (PIP).

7. The PIP obtains the requested attributes.

8. The PIP returns the requested attributes to the context handler.
9. Optionally, the context handler includes the resource in the context.
10. The context handler sends the requested attributes and (optionally) the resource to the PDP. The PDP evaluates the policy.
11. The PDP returns the response context (including the authorization decision) to the context handler.
12. The context handler translates the response context to the native response format of the PEP. The context handler returns the response to the PEP.
13. The PEP fulfills the obligations.
14. If access is permitted, then the PEP permits access to the resource; otherwise, it denies access.

Evaluation

Besides the fact that the language is the new standard in the context of access control for Web Services and SOA, XACML has several aspects that will be important for AVANTSSAR, including the attribute-orientation, which makes XACML suitable to working in a federated multi-party SOA, and the possibility for distributed composition of policies (combining all appropriate policies from different places and consistently evaluation the result).

On the other hand, some maybe for our purposes the language is not powerful enough. First, the query/respond allows only Permit, Deny, or Indeterminate, but does not provide hints, negotiation mechanisms, or logic support. Second, trust management is out of scope (whose credentials with which type of assertions do I accept?). Third, the meaning of obligations is not explicit and the way that obligations are implemented (at the PEP) may be not optimal (better in many cases would be in a monitor process, independent of the access control. Last, it does not provide neither hierarchical attributes (just list of attributes) nor purposes of actions, which could be very convenient for our purposes.

5.7 WS-Federation

The WS-Federation specification [16] defines mechanisms to support federation between different security realms, i.e. the authorized access for principals of one realm to resources managed by another. The WS-Federation framework builds on the specification for WS-Security and WS-Trust.

Specifically, WS-Federation relies on the Security Token Service (STS) model defined by WS-Trust, and a protocol (involving Request Security Token and RST Response messages) for handling such tokens, which contain information described by WS-SecurityPolicy [40]. The STS is used to broker an establishment of a trust relationship between resource providers / relying parties and other service providers. The goal is to simplify the development of federated services by reusing the WS-Trust STS model and protocol. Different federation services can be developed as variations of the base STS.

Processing in WS-Federation is kept independent of the security token format and the type of token being transmitted. WS-Federation defines a metadata model and a document format describing how services can be discovered and combined, as well as their access policies. For example, this may mean supplying WS-Addressing endpoint references (EPRs) to participants.

Types of services:

- **Authorization** can be viewed as a decision brokering service. Interoperability of services requires a common model for interacting with authorization services. This includes two STS extensions: the passing of additional context about a token request, and a *Claims Dialect* mechanism for expressing common claims requires to process requests.

- **Authentication Type**: a set of URIs is defined for specifying the wst:AuthenticationType parameter in RST and RSTR messages.

- **Attribute Services**: WS-Federation defines a model for accessing attribute services which may be needed to establish a federation context, e.g., information for advanced functionality or personalized user experience.

- **Pseudonym Services** allow principals to have different aliases in different realms or for different resources. They provide different kinds of identity mappings, e.g., with pseudonyms established per login or per service. In combination with the attribute services, they allow information to be provided about a requestor identified by a pseudonym, if the requestor has authorized this.

- **Privacy**: extensions to WS-Trust syntax are defined to express both privacy requirements of a requester and mechanisms used by a STS for issuing a token. This may include, e.g., identification of sensitive claims in a token that must be protected by encryption.

Evaluation

WS-Federation provides high-level features based on WS-Security and WS-Trust. While the latter will be reflected in the language as mentioned above, specific language features which directly reflect WS-Federation concepts are not anticipated at this point.

5.8 SAML

The Security Assertion Markup Language (SAML) is an XML standard for assertions regarding identity, attributes and entitlements of a subject [17]. It allows to exchange authentication and authorization data between security domains, that is, between an identity provider (a producer of assertions) and a service provider (a consumer of assertions). The primary example for the use of SAML is is the Web Browser Single Sign-On (SSO) problem.

The service provider relies on a SAML assertion from the identity provider about the principal to make an access control decision. This setup requires the existence of local authorization services from an identity provider, however, SAML provides a level of abstraction, since it does not specify how they are implemented. SAML is a dialect of XML, uses the XML signature and encryption facilities, and relies on HTTP, specifying the use of SOAP.

Generally, a SAML assertion is a statement made at a given time by an issuer regarding a subject provided that certain conditions hold. SAML assertions can contain three types of statements: authentication, attribute and authorization decision statements. Each of these corresponds to a type of query which forms part of a SAML request-response protocol. The structure of an assertion is described by a SAML profile, which may be defined dependent on the desired application. At the implementation level, SAML messages admit bindings to several standard message types and protocols. [41].

SAML provides XML formats for transmitting security information, defines how they work with underlying protocols, and specifies message exchanges for common use cases. In addition, it supports several privacy protection mechanisms (providing means to determine security attributes without revealing identity), and specifies a schema that allows systems to communicate the SAML options they support. SAML is linked to the WS family of standards: SAML assertions are one supported security format for WS-Trust.

Evaluation

The AVANTSSAR languages will provide primitives to specify security assertions. While syntactically different, the languages should provide assertion types that semantically match the main types of assertions provided by SAML.

5.9 WS-Reliability

WS-Reliability is a SOAP-based specification for reliable messaging requirements [47]. Subsequently to its standardization in 2004 (v. 1.1), the WS-Reliable Messaging specification was developed by the same OASIS Technical Committee. WS-Reliability separates reliable messaging issues into a protocol ("wire") aspect which deals with the horizontal contract between sender and receiver (e.g., message headers, choreography) and a quality of service aspect which deals with the vertical contract between service provider and service users. The latter defines a set of abstract operations on messages (such as Deliver, Submit, Respond and Notify). The specification assumes transparency of SOAP intermediaries and support for message integrity (e.g., as in WS-Security).

5.10 WS-ReliableMessaging

WS-ReliableMessaging [18] is an OASIS standard that describes a transport-independent protocol to transfer messages reliably between nodes; it also provides a SOAP binding. The protocol relies on other specifications from the WS-family to identify policies and service endpoints; in particular it can be combined with WS-Security and WS-Policy for secure reliable messaging.

The protocol supports various reliability features, including ordered delivery, duplicate elimination and guaranteed receipt. The standard also discusses security threats (including integrity threats, resource consumption and spoofing); the implementation of countermeasures is supposed to be done using common web service solutions such as Transport Layer Security or SOAP Message Security (WS-SecureConversation/WS-Trust). WS-ReliableMessaging is accompanied by a specification for a domain-specific policy-assertion, to be used together with WS-Policy.

Evaluation

The AVANTSSAR languages should provide different levels of abstraction for communication between principals, from low-level message send/receive to communication with various authentication guarantees. In this respect, it

is desirable to include an abstraction level which closely matches the specifications of WS-ReliableMessaging.

5.11 WS-MetadataExchange

In general, metadata is information that describes other data than the core functional aspects of a service. In Web Services, metadata represent additional information about services that can be essential for the interaction with other endpoints. The WS-MetadataExchange specification [5] defines a format for metadata that is adaptable to future changes, and also defines how metadata can be retrieved from Web Services as a WSTransfer resource. A requester may ask for all available metadata about a certain Web Service or just about a particular type of metadata.

To bootstrap communication with a Web Service, this specification defines three request-response message pairs to retrieve three types of metadata: one retrieves the WS-Policy associated with the receiving endpoint or with a given target namespace, another retrieves either the WSDL associated with the receiving endpoint or with a given target namespace, and a third retrieves the XML Schema with a given target namespace. Together these messages allow efficient, incremental retrieval of a Web Service's metadata.

Evaluation In the context of AVANTSSAR, besides the basic WSDL document, WS-MetadataExchange (together with WS-Transfer) can be used to exchange the extra-functional aspects of a service with other endpoints.

6 Context, Coordination and Transaction-related Protocols

For distributed applications, additional mechanisms are required to ensure *reliability* and *consistency* of the application. Within the context of service-oriented architectures such a mechanism is usually referred to as *coordination*. In particular, the *WS-Coordination* [58] standard provides such a mechanism for Web Services. In combination with the more low level transaction mechanisms, it can ensure reliability and consistency of an application.

The WS-Coordination framework unifies several popular coordination models. It provides three main mechanisms: activation, registration, and coordination of services. An activation service enables an application to create a coordination instance or context. A registration service enables an application to register for coordination protocols. These services do not contain any security measures, and are solely preparation steps towards coordination.

Lower level mechanisms provide support for transactions. The notion of transaction is a fundamental concept in traditional reliable distributed applications. A transaction is considered to be atomic, in the sense that if it is not fully completed, the result is equal to not performing the transaction. In the basic case, a transaction is also indivisible: it is considered to be a single action which cannot be split into multiple phases. Such a mechanism is provided by *WS-AtomicTransaction* [58], which is suitable for transactions of a short time duration. However, some transactions may have a longer duration, and it may be desirable to reveal intermediate data or allow for splitting into multiple phases. Such a divisible transaction mechanism is provided by *WS-BusinessActivity* [58]. We describe these three standards in more detail below.

6.1 WS-Coordination

An application in a service-oriented architecture usually comprises a series of activities. The management of such a series of tasks over time is performed by the coordination mechanism. It provides a context for the activation of services, which allows propagation of an activity to other services.

6.2 WS-AtomicTransaction

The WS-AtomicTransaction standard builds on WS-Coordination, and includes three agreement coordination protocols, which consist of variations of Two-Phase Commit (2PC) protocols. These are:

- Completion
- Volatile 2PC
- Durable 2PC

The first, completion, is a combination of the other two. In the completion protocol, the coordinator begins with volatile 2PC, and then proceeds through durable 2PC.

There are two main differences between WS-AtomicTransaction and a standard two-phase commit protocol [50]: First, the distinction between volatile and durable 2PC participants is specific to WS-AtomicTransaction. It aims to distinguish between participants based on the objects that are expected to managed by them: either volatile (e.g. caches) or durable (e.g. databases). Second, the protocols defined in WS-AtomicTransaction all include an explicit registration procedure. A peculiarity of this registration phase

is that a durable participant can register while volatile participants have completed their registration phase (i.e. are in their preparation phase).

6.3 WS-BusinessActivity

The WS-BusinessActivity standard essentially provides a mechanism for handling exceptions during a series of activities. Contrary to the case of an atomic transaction, intermediate results may already have been released or published, and hence the triggering of external rollback events may be required to undo the partial execution of the business activity. Such rollback events are also referred to as compensations. However, unlike e.g. ACID transactions in a DBMS [37], the compensation behavior is a part of the business logic: it must be explicitly specified. This allows for a looser coupling of the individual services, which is a more appropriate model for service-oriented architectures.

6.4 Evaluation

In the context of AVANTSSAR, there are many security considerations for coordination that are specific instances of more general Web Service security. One security consideration that is particular to coordination is related to availability and guaranteed message delivery. If a malicious party can block certain messages, rollbacks can be trivially forced. Alternatively rollbacks may be prevented from being completed for WS-BusinessActivity, and also for WS-AtomicTransaction in case there is no Expires attribute (which is optional).

Certain aspects of WS-AtomicTransaction (section 6.2) which are not shared with the well-understood standard atomic transaction protocols have to be used and composed prudently. Formal verification techniques developed in AVANTSSAR project can be of great help in assessing these aspects.

7 Registry (publishing /discovery)

Service providers can advertise their services by sending their profiles to registration and discovery services, such as UDDI.

7.1 UDDI (Universal Description Discovery and Integration)

The current version of UDDI, version 3.0.2, was released by OASIS in February 2005 [63, 13]. It provides the infrastructure required to publish and discover services in a systematic way. UDDI specifies an XML-based registry wherein service providers can register their available Web Services and whose content can be browsed by clients. A common usage is the description of every particular service in WSDL and their registering in a UDDI registry. The UDDI data model is an XML schema that describes services, using the following structures:

- businessEntity: represents the provider of Web Services. It contains information about the company, including contact information, industry categories, business identifiers and a list of services provided.
- businessService: represents an individual Web Service provided by the business entity. Its description includes information to bind to the Web Service, its type and taxonomical categories.
- bindingTemplate: represents the technical implementations of the Web Service represented by the business service structure.
- tModel: represents metadata used for more detailed informations about a service.
- publisherAssertion: represents the association between some businessEntity structures according to a specific type of relationship, such as subsidiary or department.
- subscription: is used to subscribe to events about changes of a list of entities.

As a registry is useless without some way to access it, UDDI specifies two interfaces for service consumers and service providers to interact with the registry. Service consumers use the *I*nquiry API to find a service, and service providers use the *P*ublisher API to register a service.

The following example shows how an X Company would register its informations, service description, and online service access information with UDDI:

- Obtain an authentication token from a UDDI operator. Each operator has different terms and conditions for authorizing access to its replica of the registry.

- Use the Publisher API to register business information that would be helpful for searching, i.e., filling the tModel information.
- Use the Inquiry API to test the retrieval of the information, including binding template information.

After the X Company has updated its UDDI entry with the relevant informations, companies can look up contact information in the UDDI registry and obtain the service descriptions and access points for the Web Services that X Company publishes.

A UDDI registry offers a mechanism to manage Web Services, so that they can be discovered and consumed. UDDI can be used to represent information about Web Services in a way such that queries can then be issued to a UDDI Registry at design-time or run-time. Here are some scenarios that are enabled by the combination of the UDDI information model and the UDDI API set:

- Publishing a service in a registry.
- Search for information about a particular service in a UDDI registry.
- Determination of the security and transport protocols supported by a given Web Service.
- Replication of data about a service. For instance, a registry operator want to achieve data replication between sites.
- Transferring custody of data about a service (inter-registry communication).

Evaluation

Besides the discovery of a service, one could also consider the security aspects of the discovery of services: for instance, what about the trust in the case of transferring custody of a service. The security model for a UDDI registry can be characterized by a collection of policies. The security policies and mechanisms in the UDDI specification are related to data management, user identification, user authentication, user authorization, confidentiality of messages and integrity of data. For this purpose UDDI provides a Security Policy API Set.

7.2 WS-Discovery

The WS-Discovery standard [45] provides a mechanism for finding the specific addresses of web-services at run-time. It relies on a multicast protocol to

send out a request for discovery, to which services that match the constraints will respond. When a web-service provider joins the network, it sends an announcement message to the multicast group in order to minimize the need for polling.

Various security threats have been identified for this service [45], including message alteration, replay, and denial of service attacks. To prevent these, the standard *recommends*, but does not enforce, using standard cryptographic techniques such as digital signature schemes and employing timeouts.

Evaluation

Validation techniques for WS-Discovery, in particular, need to address the following two aspects:

1. Interactions between this mechanism and WS-Security and WS-Trust are envisioned in the documentation [45]. Since the documentation is not precise about how the security mechanisms should be implemented (e.g. no particular interfaces are specified), interactions between concrete implementations may result in unexpected security vulnerabilities. Such interactions between WS-Discovery and other security services, indeed, bring up composition concerns. In order to establish the security of the overall system, AVANTSSAR needs to capture the WS-Discovery mechanism, and its composition with other modules.

2. WS-Discovery relies on a multicast protocol. Modeling such communications and the related properties in existing formalisms might require enhancements in the specification language and verification tools.

8 Business Process Language Layer

Beside core Web Service specifications - such as WSDL, SOAP, and UDDI - a range of additional specifications are being developed that enhance the capabilities of Web Services. These extensions of Web Services address different functional areas such as messaging, transactions, resource management, policies, security, or workflows.

The design of the AVANTSSAR languages need to take into account existing languages for service description, policy definition and service orchestration.

Orchestration Languages like WSFL (cf. Section 8.1) and WS-BPEL (cf. Section 8.2) define the sequence and conditions in which the processes within a business are executed.

A business process defines a flow of activities (e.g., to grant a loan, to ship goods for a customer) to be executed by a set of actors to achieve some business goals.

The activity flow specifies the orchestration needed to complete the goal. As described in the [43] paper, for services-based architectures, the activity flow can be implemented in two ways, using traditional methods (such as programming languages e.g., C# or Java) or using a Business Process Definition language.

An interesting book [38] explains the Business Process Modeling concepts in an overview of the different languages that expresses the Business Process. As stated in M.Havey's book, a business process that orchestrates complex system interactions can also be viewed as services that communicate with the processes of other companies according to well-defined contracts. We will focus on languages that fall in the scope of AVANTSSAR, meaning that we will consider the properties that are of interest for the specification of trust and security properties of the services and their associated policies.

Web Services can be combined; the business logic of so-called composite applications can be specified using Business Process languages.

Service-oriented applications as well as their security requirements, are in general not static but rather continuously evolving. Their interaction takes place in highly dynamic environments where the composition of services can be undertaken at runtime. Some security policies are dynamically modified (e.g., for incident handling or in case of emergency), and agents may join or be excluded from a community sharing some security context.

Atomic services provide functionality not only for the business logic, but they also can provide the basic building blocks of security functionality. They cover, for instance, identity providers, time-stamping functionality, provision of credentials, access control rule evaluation (also known as policy decision), and logging or audit services.

Consider the following example: in the Banking Services application scenario, we expect to focus on the following atomic services: authentication and authorization services, non-repudiation services, archive and logging services, services enforcing separation of duty constraints and others establishing secure communication channels, etc. These services and their composition (in order to accomplish more complex security requirement, e.g., accurate auditing trails using non-repudiation, logging, authentication and authorization services) will be validated mainly from a static point of view. Dynamic validation will also be experimented on a few case studies originating from

scenario extensions. For instance, we can easily imagine a scenario extension in which a personal financial advisor service is provided by a financial credit institution to dynamically select from a dynamic set of different banks (all making available a loan origination process service) the best loan offer for a customer.

For the specification of the AVANTSSAR industrially-suited language, we need to study the languages specifying business process behavior based on Web Services.

Although these orchestration languages are suitable to express the composition aspects, they are not specifically oriented towards security, thus we need to provide extensions to describe security requirements and policies. For instance, ASLan and ISSL need to express authorization, obligation, delegation, separation of duty, revocation,etc. within the Business Process language.

8.1 WSFL (Web Services Flow Language)

WSFL is an XML-based language for describing the composition of Web Services (WSs). By design, WSFL can be seen as an extension of the Web Services Description Language (WSDL), which describes what a WS can do, where it can be reached, and how to invoke it. As a consequence, WSFL naturally fits in the WSs computing stack (cf. Figure 1 of the deliverable) where it can be regarded as an additional layer on top of WSDL.

What can be specified in WSFL. Two types of WSs composition can be specified in WSFL: flow models and global models. The former specifies how to use the functionality provided by the collection of composed WSs (this is also known under the names of flow composition, orchestration, and choreography). The latter specifies how the composed WSs interact with each other. Another key feature (for modularity and scalability of specifications) of WSFL is its ability to support recursive composition of WSs, i.e. every WS composition can be seen as a (single) WS and then used as a component in further compositions.

Roughly, a flow model can be seen as the specification of a particular instance of a WS obtained by composing a set of available WSs to handle a certain business process for a company; while a global model is a specification of a generic WS capable of performing a certain business process once instantiated with (bound by) a set of suitable WSs. To better illustrate the difference between flow and global models in WSFL, it is helpful

to consider the following two related scenarios. First, an enterprise wants to implement a business process for processing purchase orders using WSs, e.g. the following processing steps must be performed: collecting orders from customers, checking the credit history of customers, rejecting/accepting orders, processing orders, and shipping goods. As a first step, they should find WSs offered by third parties that can be used to realize the various activities, such as a shipping company, a goods-supplier company, and a bank. Then, they would use a WSFL flow model to define the structure of the business process: WSFL activities model processing steps (e.g. collecting orders) and control and data links specify the control and data flows between these, respectively. Furthermore, for each activity a service provider is identified (by means of locators) which is responsible for the execution of the process step together with the association between activities in flow model and operations offered by the service provider (by using plug links). The second scenario is a variant of the previous one: an enterprise would like to offer a WS that mediates between customers who want to order goods and service provides who produce and deliver goods. As before, the business process to handle purchase orders is described a WSFL flow model where the activities are not bind to particular services providers. Instead, only the role (the kind) of service provider are identified for each activity and a WSFL Service Provider Type interface is added in order to specify which operations are provided and which are required.

Semantics of WSFL. The semantic of WSFL is given in terms of a flow metamodel that describes how WSs are wired together into flows that represent business processes. The metamodel is described in terms of a particular class of labeled, directed, and acyclic graphs.

An activity (i.e. a business task to be performed as a single step within the context of a business process) is represented by a node with several fields. Such fields are related both to the data flow and the control flow. Regarding the data flow, there are several fields for input/output/fault messages which correspond to the input/output/exception parameters of the operation used to implement the activity. Each message can have multiple parts, each one defined by some type system (e.g. XSD).

Control-flow. Activities are connected by control links. A control link is a directed edge that prescribes the order in which activities are to be performed (i.e. it specifies the control flow): first the source activity must be completed before the target can be initiated. A control link is labeled by a transition condition, i.e. a Boolean expression on the messages produced

by some "previous" activities, not necessarily the source activity of the edge labeled by the condition but some other activity which is backward reachable from the source node. Activities produce actual data values for their output messages which are substituted as actual parameters of the formal parameters of transition conditions. Indeed, the target activity can be initiated only if the transition condition evaluates to true. A start activity has an incoming control link labeled by true as transition condition. An end activity has no outcoming control link.

A fork activity has more than one outgoing control link. When an activity A completes, all control links leaving A will be determined and all associated transition conditions will be evaluated in their actual parameters. The target activities of all control links whose transition conditions evaluate to true are exactly those activities which are to be performed next. Those target activities whose conditions evaluate to false are eliminated by a procedure called death-path elimination. Indeed, parallel flows are to be synchronized at a later time. This is done through join activities, i.e. activities with more than one incoming control link. The decision whether a join activity is to be performed or not depends on the so-called join condition, which is a Boolean expression whose formal parameters refer to the transition conditions of the incoming control links of the join activity.

An activity has two more fields. The former is the implementation field specifying which kind of service is needed at run time to actually perform the business task represented by the activity. The latter is the exit condition, the purpose of which is to determine whether or not the execution of the implementation activity completed the business task represented by the activity. The expression can refer to the output message of its associated activity or even to output of any activity that ran before on the control path of the subject activity. The exit condition is evaluated once the operation of the implementing port type terminates. If the exit condition evaluates to true, the activity is considered to be finished and the control flow continues to the next activity(ies); otherwise, the activity is executed again. One motivation to have an exit condition associated to an activity is the capability to distinguish between the situation in which the activity implementation returned successfully and the situation in which this is not the the case, i.e. the business task completed has been interrupted. The other motivation for exit conditions is to introduce a controlled form of loops: an activity is iterated until its exit condition is satisfied. This is an important extension as the metamodel does not support cyclic graphs to void ambiguous situations.

Data-flow. Data links are the second and last kind of directed and labeled edges in the graphs metamodel. A data link specifies that its source activity passes data to the flow engine which in turn has to pass this data to the target activity of the data link. A data link can be specified only if the target of the data link is reachable from the source of the data link through a path of control links. This constraint avoids dead-locks or situations where an activity tries to consume data which are not yet produced. It is not required that data be always passed to an immediate successor of its producer. A map prescribes how a field in a message part of the target's input message of a data link is constructed from a field in the output message's message part of the source of the data link.

Evaluation. The two different notions of composition (flow or global models) in WSFL seem to be particularly interesting. The former allows one to specify a particular instance of a composed WS while the latter provides one with a specification of a generic WS which still needs to be instantiated. We believe these two notions are useful structuring mechanisms for specifications of composed WSs: global models describe generic and reusable WSs while flow models are component designed to satisfy particular needs.

8.2 WS-BPEL, WS-HumanTask and BPEL4People

WS-BPEL 2.0 (Web Services Business Process Execution Language Version 2.0) [46] is an XML-based language for describing Web Service composition in terms of service orchestration. The WS-BPEL standard is based on other WS-* specifications, more precisely the WS-BPEL process model is layered on top of the service model defined by WSDL 1.1 (see Section 4), that is a stateless model of correlated request-response (or solicit-response) interactions or uncorrelated one-way interactions (one-way or notification) and adds support for business transactions. Note that, e.g., the standard WSDL SOAP binding comprises one-way and request-response primitives only, thus providing an even weaker service model. Other bindings may support the full WSDL process model.

As defined in the abstract of BPEL specification [46], WS-BPEL is a language for specifying business process behavior based on Web Services. Processes in WS-BPEL export and import functionality by using Web Service interfaces exclusively. BPEL processes are executed by an execution engine, which publishes BPEL processes through a Web Service interface. Thus, every BPEL-process composed of Web Services is a Web Service itself and can be used as a component of higher-level BPEL-processes.

For a better understanding of the WS-BPEL context it makes sense to have a short look on the history of WS-BPEL. In 2001, both IBM and Microsoft had each defined their own languages for Web Service composition, namely the graph-based WSFL (see Section 8.1) and the calculus-based XLANG (whose Specification is deprecated and isn't available any more on the Web), respectively. To be able to compete against other emerging languages in the context of Web Service composition, IBM and Microsoft combined their languages into a new one: BPEL4WS (BPEL for Web Services) which represents the convergence of the ideas and concepts of XLANG and WSFL specifications. In April 2003, BEA, IBM, Microsoft, SAP and Siebel Systems submitted BPEL4WS 1.1 to OASIS for standardization. OASIS WS-BPEL technical committee voted on 14 September 2004 to change the name of the specification to WS-BPEL 2.0 (instead of BPEL4WS as submitted) to align the name of the specification with other Web Service standard naming conventions (WS-*) and to expose the significant enhancements between BPEL4WS 1.1 and WS-BPEL 2.0. It is common to use BPEL as a shortcut to WS-BPEL 2.0 as it will be used in the remaining of this deliverable.

Vanilla BPEL is designed to specify the behavior of business processes as long as all the activities of processes are Web Services and no human interaction is needed, which is a significant gap for many real-world business processes. In June 2007, Active Endpoints, Adobe, BEA, IBM, Oracle and SAP tried to fill this gap and published WS-HumanTask (Web Services Human Task, Version 1.0) [1] and BPEL4People (WS-BPEL Extension for People, Version 1.0) [2] specifications. These specifications describe how human interaction could be implemented in BPEL processes. BPEL4People is not a new version of BPEL but extends BPEL using WS-HumanTask. Both specifications went into OASIS specification process quite recently.

What can be specified in BPEL. BPEL provides a language for the specification of both, executable and abstract business processes composed of Web Services. Abstract business processes are partially specified processes that are not intended to be executed and, thus, only describe the observable behavior of a process ("behavioral interface"). This interface captures constraints on the ordering of messages to be sent to and received from a service. Using abstract processes, concrete operational details of a service can be hidden from, e.g., a business partner.

An executable process on the other side augments an abstract process with all of the required concrete operational details so that the process can be executed by a BPEL execution engine. This is achieved by defining the execution order of a set of activities, the partners involved in the process, the

messages exchanges between the partners, and reactions to specific events, exceptions or faults. Moreover for handling of exceptions, BPEL introduces a mechanism for defining how individual or composite activities within a unit of work are to be compensated, i.e., undone in an application-defined way. For a short introduction on the concept of compensations see Section 6.3.

The formal semantics of BPEL can be expressed and control flow of BPEL can be analyzed using Petri nets [49].

Semantics of BPEL. BPEL provides a grammar to describe arbitrary complex business processes. From a bird's eye view, a BPEL process defines the interaction between partners. Thus, the definition of partners is the first step in a BPEL process. Partners are the Web Services involved in the execution of a service, i.e., the Web Service invoking the BPEL process (client), or Web Services that are invoked by the BPEL process (service providers). Partners are defined through partner links that represent the interaction of a BPEL process with the client or service providing Web Services, containing roles, and the WSDL ports types associated with a role.

Because business process execution engines usually execute several processes and even instances of processes concurrently, a mechanism relating a response to the corresponding request and therefore to the corresponding process is needed. To this end, BPEL offers a correlation construct to specify the message elements that uniquely identify a process instance via XPath expressions.

On closer inspection, BPEL defines a business process that relates activities. There are two kinds of activities: basic and structured activities. Basic activities describe elementary steps of the process behavior, structured activities are those which encode control flow logic on groups of activities contained within them. Structured activities can be nested and combined in arbitrary ways.

There are only few basic activities that do not affect the control flow:

- assign: copying data from one variable to another, as well as constructing new data using expressions,
- wait: waiting for a specified period of time,
- empty: doing nothing.

Control flow. The other basic and structured activities affect the control flow of a process:

- Basic activities

  - invoke, receive, and reply: communication primitives, which are self-explanatory and are used to communicate along partner links,
  - exit: terminates the entire process instance,
  - throw, rethrow: signal a fault/exception in the execution of the process.

- Structured activities

  - scope: groups activities into blocks to which event-, fault-, and compensation-handlers may be attached,
  - sequence: defines a sequential execution order,
  - if, while, repeatUntil, forEach: encode conditional or repetitive execution,
  - pick: selective event processing. From a mapping from events to activities, only the activity associated with the first occurring event is executed,
  - flow: provides concurrency and synchronization,
  - compensate, compensateScope: start compensation of the effects of already completed activities.

In addition to the structured activities, just like WSFL, BPEL offers control links for biasing the control flow, more precisely they support the definition of precedence, synchronization, and conditional dependencies inside flow-activities. A control link is a directed relationship between two activities and defines, in which order the activities must be completed. Control links may be used in combination with the associated notions of transition condition and/or join conditions.

Data flow. Variables are used in BPEL processes to store, reformat, and transform messages and to store the state of a process instance. Each variable has a type assigned in form of a WSDL message type, an XML Schema element, or an XML Schema simple type.

Usually, there is a dedicated variable for each message a process sends or receives. Variables can be modified by activities, e.g. assign, which can be used to copy data from one variable to another, as well as to construct new data using expressions, and can be used as values for input-variables or containers for output- and fault-variables.

In BPEL, there are no data links like in WSFL 8.1. The data flow must be explicitly modeled by the process designer using variables and activities.

For example, a process designer assigns the result of a Web service invocation to a variable X, modifies the structure and content of the variable X to be compliant to the expected input of the next Web service that is invoked by the process, and uses X as input variable for the next Web Service invocation.

What can be specified in WS-HumanTask and BPEL4People. The specification of WS-HumanTask [1] (or WS-HT for short) states, that a human task is a service, which is "implemented" by people. Such a human task has two interfaces. The first interface exposes the service offered by the task, e.g., a translation service, as Web Service. The second interface allows people to deal with tasks, e.g., to query for tasks waiting for them or to work on these tasks.

The contribution of WS-HT is a definition of human tasks, their properties, and their behavior including timeouts and escalations, and a standardized Web Service interface dealing with, e.g., creation of a human task, annotation of tasks with comments, or notification of a client when a task is completed. An implementation of WS-HT will be some kind of task inbox or task management system, where people or groups of people can select tasks assigned to them, process them, and notify the system when they have finished a task. The task management system will then notify the client waiting for the termination of the human task it had initiated.

WS-HT additionally defines a protocol based on WS-Coordination (see Section 6.1) that allows external applications to talk to a WS-HT compliant task management system. This protocol supports role based interaction of people, provides means of assigning users to a generic human role, and supports scenarios like dual control or the four-eye principal, escalation, and chained execution.

The intention of WS-HT is to define how to interact with task management systems or task inboxes in a standardized way addressing the issue of vendors of business process management systems having invented their own task inboxes with proprietary interfaces making interaction between different systems difficult. The internals of task inboxes is out of scope of WS-HT.

WS-HT itself is independent of BPEL, so it can be used with BPEL as well as with proprietary systems of any vendor. BPEL4People [2] is one aspect of a client for WS-HT with the aim of defining a standardized extension within BPEL for how to interact with WS-HT. The BPEL4People specification defines a people activity as new basic activity using the extension mechanisms provided by the BPEL specification. This new activity enables the specification of human interaction in BPEL: it allows specifying tasks local to a process or use tasks defined outside of the process definition.

Evaluation. With at least most major players in the industry participating in the BPEL open standards process, BPEL can be considered the standard for Web Service orchestration nowadays. The AVANTSSAR languages should rely on BPEL because it introduces a common grammar for describing service orchestration.

The BPEL specification notes, that although BPEL is inherently binding neutral, it is strongly recommended that business process implementations use WS-Security (see Section 5.1) when using a binding where messages might be modified or forged and security is a concern.

8.3 BPMN (Business Process Modeling Notation)

The Business Process Modeling Notation (BPMN) [19] was developed by Business Process Management Initiative (BPMI) [53], and is now being maintained by the Object Management Group (OMG). The current version is 1.1 but the adoption of BPMN 2.0 is scheduled before the end of 2008.

What can be specified in BPMN. BPMN supports both the orchestration of Web service and the execution of human workflow tasks, while enabling the choreography of multiple business processes, thus by providing a Business Process Diagram (BPD). This flow-chart format is designed to be used by the business process analysts. BPMN is constrained to support only the concepts of modeling that are applicable to business processes. It could be a good candidate for the AVANTSSAR Industrially-Suited language because of its simplicity. It is made of simple diagrams with a small set of graphical elements.

Semantics of BPMN. BPMN defines a Business Process Diagram (BPD), based on a flowcharting technique to build graphical models of business process operations. A Business Process Model can be viewed as a network of graphical objects representing activities and the flow controls defining their order of performance.

The semantic correctness of BPMN process models can be checked using a mapping to Petri nets [29].

BPMN also provides a formal mapping to an execution language of Business Process Management Systems (i.e., BPEL4WS).

BPMN is intended to bridge the gap between the format of the initial design of business processes and the format of the languages, such as WS-BPEL, that will execute these business processes. BPMN provides a mapping from the visualization of the business processes (a notation) to the appropri-

ate execution format (a Business Process Management Execution Language) for these business processes.

There are four basic categories of elements: Flow and Connecting objects, Swimlanes and Artifacts.

Control-flow. BPMN has a small set (three) of core elements, which are the Flow Objects.

- Flow Objects:
  - Events: something that happens during the course of a business process. There are three types of Events, based on when they affect the flow:
    - Start.
    - Intermediate.
    - End.
  - Activities: the kind of work that company performs. An Activity can be atomic or compound. The types of Activities are:
    - Task.
    - Sub-Process.
  - Gateways: the control of the divergence and convergence of Sequence Flow. For instance, the forking, merging, and joining of paths. Three possibilities are available:
    - Exclusive Decision / Merge (XOR).
    - Inclusive Decision (OR).
    - Parallel Fork / Join (AND).

  The Flow Objects are connected together in a diagram. There are three Connecting Objects:

- Connecting Objects: connects the flow objects:
  - Sequence Flow: the order that activities are performed within a Process.
  - Message Flow: the flow of messages (ingoing or outgoing) between process participants.
  - Association: the association of data, text, and other Artifacts with flow objects data. An Association can also be used to show the input and output of an activity.

Data-flow. BPMN provides a mechanism to organize activities into separate visual categories in order to illustrate different functional capabilities or responsibilities.

- Swimlanes:
  - Pool: the involved persons of a business process.
  - Lane: the organization and categorization of activities. It partitions a Pool and assigns roles to activities.
- Artifacts: the possibility to add some contextual information into the model/diagram:
  - Data Objects: data needed by an activity. There is also the possibility to notice the actual state of the object. They are connected by an Association.
  - Group: the grouping of objects of a process diagram with no impact on the process flow. It can be used for documentation or analysis purposes.
  - Annotation: the possibility to add notes or comments to improve the readability of a process diagram.

The basic structure of a Process is determined by the Activities, Gateways, and Sequence Flow.

Evaluation. One could argue that besides this easy-to-use modeling notation, there is a need to map the graphical notations into a language that expresses a business process, such as BPEL for instance. We have seen that it is an XML-based language for describing a business process. The BPEL process itself could be represented as a Web Service, realized by a BPEL engine executing the process description. BPMN as a standard set of diagramming conventions, is designed to visualize a rich set of process flow semantics within a process and the communication between independent processes. Since BPEL is currently considered as the most important standard for execution languages, a translation to BPEL is specified in the BPMN standard. Some softwares such as Telelogic's System Architect (www.telelogic.com/popkin) generates BPEL code from BPMN diagrams but it is restricted to a limited subset of BPMN. Intalio (www.italio.com) also proposes to generate executable processes directly from a BPMN diagram. As stated in [55], in BPEL, they miss to consider that business processes generally have some form of human involvement. Some patterns are missing

such as "user task", i.e. Role-Based Distribution, Authorization, Separation of duties,..... which are supported in BPEL4People Creation Patterns. By design there are some limitations on the process topologies (such as Ad-Hoc sub-processes). As BPMN provides a much richer set of modeling constructs [54], there are some BPMN processes that cannot be mapped to BPEL. For example, BPEL misses support for a range of BPMN control flow patterns.

As we will also see in the following section, the OMG has scheduled the adoption of BPMN 2.0, during the third quarter of 2008, with an improved representation of choreography and support for displays tailored to different aspects of modeling and analysis. There are may wishes expressed concerning the BPMN 2.0 specifications, one would be to make the graphical process modeling notation executable; a BPMN 2.0 diagram would specify exact semantics and could be executed on a computer system. The mapping to BPEL wouldn't be needed anymore. We will see with the release of the official specification whether these wishes would be adopted.

While defining the AVANTSSAR ISSL, we could take the advantages of the flexibility and business-friendly notation of BPMN, extended to one or several standard languages, to express processes with their related security goals and requirements.

8.4 BPDM (Business Process Definition Metamodel)

On June 2005, the Business Process Management Initiative (BPMI) and the Object Management Group (OMG) announced the merge of their Business Process Management (BPM) activities. They still focus on Business Process Management, such as refinement and promotion of BPMI's Business Process Modeling Notation (BPMN) as the basis for business modeling and delivery of OMG's Business Process Definition Metamodel (BPDM). They defined a language, vocabulary, and rules. An OMG Adopted Beta Specification was released on July 2007 [20].

What can be specified in BPDM. BPDM is a common model for describing all business processes, independently of notation or methodology. BPDM defines a suite of model elements complete enough to represent both intra- and interenterprise business processes in an human-reading way.

BPDM supports the specification of WS-Choreography. One of its objectives is the unification of orchestration and choreography. For that purpose, it should capture the ordering and conditions for the inputs and outputs flows of a process. It should also share the same ordering elements between choreographies and orchestrations.

BPDM recommends the use of BPMN as the standard notation for processes. As we previously saw, the goal of BPMN is to be recognized as a flexible and business-friendly notation for process orchestration, but some robust foundations are still missing, such as:

- An explicit metamodel and vocabulary.
- A serialization mechanism.
- A rigorous execution semantic.

BPDM provides BPMN with an explicit metamodel that would unify the various business process definition notations, a serialization mechanism for BPMN concepts and a rigorous execution semantic.

By design, BPDM supports many process approaches and notations. For instance, workflow and more general concepts of process, activities, tasks and sub-processes that could be executed by a combination of human and automated participants, conditional execution paths, parallel processes, process flow and data flow, time-based events and conditions, transactions with rollback, roles, responsibilities and collaborations.

Semantics of BPDM. Not only does BPDM provide a common basis for all process oriented models but also integrates rules within processes. It offers the capability to represent and model business processes independently of notation or methodology, using a "meta model". The meta model behind BPDM uses the OMG "Meta Object Facility" (MOF) [44] standard to capture business processes in a very general way and to provide an XML syntax for storing and transferring models between tools and infrastructures.

BPDM relies on formal methods to verify the precision of the semantics and to ensure that the execution and communication of processes are consistent.

Control-flow. In BPDM, the Process inputs and outputs are treated as Messages. The messages are themselves "Steps" in a Process. They can be ordered in time, conditioned, monitored.

To handle the communication between processes, BPDM tries to unify orchestration and choreography using an interaction centric approach. It distinguishes:

- Orchestration: the Execution of a process within an organization; that is, within a pool, with an established sequence flow in a process managed by an authority recognized by all participants. Orchestration

is represented through "Process", which includes the traditional view where sequences of "Activity" are carried out, with branching and synchronization of different threads.

- Choreography: Collaboration across processes; that is, from one pool to another (its peer), with communication via message-passing, and no coordinating authority. The Interactions of collaborating entities are often structured into "Interaction Protocols" to represent the conversation between the parties.

The "Common Behavior Model" allows orchestration and choreography to be treated independently. It enables the sharing of common information about business being modeled. The core elements of the Common Behavior Model cover "Events", "Interactions" and "Process Steps". There are three aspects of the Common Behavior Model which support the dependencies between "Activities" (in orchestration), between "Interactions" of participants (in a choreography), and between the "Interaction" of an orchestrated Process with its environment (a combination of both orchestration and choreography).

- One aspect is an event-oriented view of processes called "Happening". It covers all the elements that involve time; either at a single point or over an extended duration. For instance, a "Timer" 7 days has a particular relationship with the race course Activity.

- The second aspect is "Processing Behavior". It leverages Happenings to represent sequences of generic Steps (applicable to both orchestration and choreography). For instance, the "Universal Behavioral Happening" represents the "Process Behavioral Change" (i.e., Start, Stop, Abort, Error). It is the default type for all process Steps and choreography Steps.

- The third aspect is "'Simple Interaction". It addresses the definition and reuse of protocols. It is applicable to both choreography, and to the interaction of an orchestrated process with its environment.

Data-flow. BPDM provides a consistent approach for composing processes. It includes an extensive "Composition Model" that defines a general concept of relationship. The Composition Model defines a framework defining how all the classes of the metamodel work together and how do they relate to runtime execution. It provides the underlying principles and mechanisms for reuse, (de)composition, interconnection, and instantiation. The Composition

Model also defines composition relationships between entities (e.g., how an Activity is a part of a Process, how an "Interaction Flow" is part of an Interaction Protocol). It also describes the recursive structure of a Composite, e.g. how a Process has "Typed Parts" that are Activities. Some of these may in turn be "Sub-Processes". It also defines relations between entities at the same level of decomposition, e.g., that one Activity comes after another under a certain Process, or that one participant interacts with another in an Interaction Protocol.

An extension to the Composition Model yields the "Course Model", which connects elements in time and becomes the basis for representation of a Process. The Course Model provides a very general coordination mechanism of steps. It includes functionality that is the basis for Gateways in BPMN (Splits and Joins), as well as Sequence Flows.

To handle the processing, BPDM embeds a Process as a grouping of activities. A "Processor Role" represents the actors responsible for the Process. It also considers the notion of "Performer Roles" as a partition of processing responsibilities.

Evaluation. By concentrating on the metamodel separately from the modeling language used to represent it, business modeling experts are able to incorporate the full range of business activities into a metamodel. But models based on BPDM need to be parsed and mapped to a modeling language.

BPDM provides a robust serialization mechanism for the BPMN modeling notation. BPMN 2.0, currently undergoing adoption at OMG, would produce matched versions of BPMN and BPDM.

The support for both orchestration and choreography could be considered for our AVANTSSAR language specification. BPDM facilitates the evolving business boundaries as companies outsource, reorganize, merge and split their operations. For instance, once the commitments of each role are defined (its boundary conditions), it becomes possible to change the realization of the process without changing the commitments.

BPDM is often compared to the existing process interchange format XPDL. The two efforts are similar in that they could be used to exchange business process definitions using XML.

8.5 WPDL (Workflow Process Definition Language), XPDL (XML Process Definition Language)

XML Process Definition Language (XPDL) is standardized by the Workflow Management Coalition (WfMC) since 2002. Formerly working on WPDL

(Workflow Process Definition Language), WfMC soon changed their language name to XPDL when it was decided to use XML for its syntax. Current version is XPDL 2.1 .

XPDL is designed to answer the *Process Definition Interchange* question, which is, according to WfMC, one of the key features a workflow management system must have. Since 2.0, XPDL is even explicitly designed to allow store and exchange of BPMN diagrams. To quote WfMC: "XPDL is the Serialization Format for BPMN". This means there is a one to one matching between a BPMN diagram and its XPDL translation. That includes the graphical part of BPMN: the coordinates of the different elements, of the lines linking those elements, all this is included in XPDL.

What can be specified in XPDL. Just like its counterpart BPMN 1.1, XPDL is a process design format, as opposed to being a process execution language like BPEL. And even if XPDL can sometimes run on workflow engines, it is not its main purpose. In other words, what can be specified in XPDL is strictly what can be specified in BPMN.

Semantics of XPDL, Control-flow and Data-flow. In regard to semantics, what has been said for BPMN can be said for XPDL. With the precision that XPDL language capabilities are defined in a XML schema. This XPDL schema [59] defines classical workflow entities:

- process containers
- process activities
- message flows
- associations
- participants
- ...

And it also makes room, through extensible elements and attributes, for user or vendor specific information. Thus, with XPDL, one can export and import back and forth between workflow editors or workflow engines without losing information (even product-specific).

Evaluation. We have seen that BPMN could be of interest to AVANTSSAR ISSL. In that respect, XPDL, being the textual equivalent to the graphical notations of BPMN, cannot be neglected.

8.6 PSL (Process Specification Language)

PSL is based on substantial research ranging from the situation calculus and enterprise/manufacturing systems modeling. It has been applied in a wide variety of areas such as scheduling, process modeling, planning, simulation, and project management. PSL is project 18629 at the International Organization of Standardization, and part of the work is a Draft International Standard. Given its generality and potentials for capturing many aspects of WSs, it is difficult to make PSL fit precisely in the WSs computing stack in Figure 1 of the deliverable. Certainly, it can be used to model aspects of both single WSs and their composition.

PSL has a rigorously-developed semantics using first-order logic, and is based on a three-step methodology: identifying intuitions, refining them in mathematical structures, and then defining a logical language for the intuitions. First, intuitions about executing processes arising from applications and existing languages have been overviewed so to serve as informal requirements. Second, each intuition has been mapped to an element of some algebraic structure such as graphs, linear orderings, partial orderings, groups, or vector spaces. Another important part of the second step of the methodology was the writing of axioms and definitions in first order logic to formalize the original intuitions. The third and last step consisted in showing that the structures were isomorphic to the (first-order) semantics of the predicates in the logical language.

What can be specified in PSL. PSL has been designed to facilitate correct and complete exchange of process information in a wide range of application domains. This is so because of the growing use of information technology in virtually any aspects of everyday life. In this context, the interoperability of software applications has become increasingly important. Initially, translation programs were written to enable communication from one specific application to another. As the number of applications has increased and the information has become more complex, it has become much more difficult for software developers to provide translators between every pair of applications that need to exchange information. Standards-based translation mechanisms have simplified integration for some manufacturing software developers by requiring only a single translator to be developed between their respective software product and the interchange standard. By only developing this single translator, the application can inter-operate with a wide variety of other applications that have a similar translator between that standard and their application. In this respect, PSL can be seen as a neutral, standard language for process specification to serve as an interlingua

to integrate multiple process-related applications. This interchange language is unique due to a formal semantic (the ontology) that underlie the language. Because of these explicit and unambiguous definitions, information exchange can be achieved without relying on hidden assumptions or subjective mappings.

Semantics of PSL. The basic concepts of PSL serve to disambiguate many kinds of behavior models for process execution. An activity is a reusable behavior and an occurrence is a run-time execution of an activity. These and any other concept in PSL are formalized by predicates of first-order logic. Any concrete first-order language can be used for this purpose although it is customarily used the Knowledge Interchange Format (KIF). By using the SUBACTIVITY relation, it is possible to specify that the execution of a PSL activity may involve the execution of another.

Control-flow. The SUCCESSOR relation associates occurrences with each other to represent all temporal orderings of run-time execution of activities whether they conform to a behavior specification or not. An oriented tree representation of the SUCCESSOR relation is possible, where nodes are occurrences, edges represent the pairs in the relation, and branches possible execution trances. A behavior specification identifies those parts of the occurrence tree conforming to a certain behavior and it is expressed as first-order constraints on run-time execution using the predicates defined by PSL, usually by means of KIF. PSL supports several forms of constraints on sequences of activity and sub-activity executions by means of relations such as NEXT_SUBOCC to specify the order for sub-activity occurrences of a complex activity, MIN_PRECEDES which is the transitive closure of NEXT_SUBOCC, and ROOT_OCC and LEAF_OCC to indicate which sub-activity occurrences are at the beginning and end of a complex occurrence.

All flow languages support flows that split and come together. To support flow choices, PSL has constructs for representing aspects of the state of the world before and after each activity execution in the occurrence tree. The PSL representation of decisions uses states to express which branches of the occurrence tree are allowed under the specification. The relation HOLDS tells what states of the world is the case after a specific activity execution, i.e. at one point in the occurrence tree. Combining the relation HOLDS with (first-order) disjunction (so to require all combinations of run-time execution order as possibilities), it is easy to specify multiple concurrent flows.

PSL features constraints on how much time can elapse between execution

of activities. It has functions for begin and end time behavior of executions which can be combined with first-order predicates to mandate durations of activities.

Data-flow. One of the main goals of the design of PSL was to allow for interoperability of heterogeneous applications. In this respect, one of the crucial issues is to obtain a meaningful exchange of information between applications as, e.g., different types of applications may associate different meanings with similar or identical terms. To address this point, PSL proposes a formal semantic layer (an ontology) based on KIF, a formal language developed for the exchange of knowledge among disparate computer programs. For example, PSL was successfully used to integrate several manufacturing process applications by using the ontology to define explicitly and unambiguously the concepts intrinsic to manufacturing systems.

In PSL, an ontology is a set of sentence in first-order logic comprising a set of foundational theories and definitions. Hence, a language, a model theory, and a proof theory are defined. The language provides the terminology to talk about activities and their flow; the model theory provides for a rigorous mathematical characterization of the semantics of the terminology; and the proof theory provides axioms for the interpretation of terms in the ontology. Indeed, the challenge is to make explicit the meaning of the terminology for the ontologies that are in people's head as any implicit idea is a possible source of ambiguity.

The axioms and definitions in the PSL ontology are organized in a semantics architecture encompassing a set of core axioms (called the PSL-Core), the core theories, and extensions.

The PSL-Core gives the semantics of the primitives in the PSL ontology corresponding to the fundamental intuitions about activities. It introduces activities, their crucial events for timing (e.g., the beginning or end of an activity), and their temporal relationships (e.g., before or after). The basic notions of the PSL-Core are axiomatized formally as a first-order theory whose axioms capture the basic properties of the PSL ontology.

The terms that have definitions can be grouped into modules, each of which is an extension of PSL-Core. The set of extensions of PSL can be classified in three categories: generic activities and ordering relations (partially ordered activities, non-deterministic activities, complex sequence ordering relations, and junctions), process planning (resource roles, processor actions, resource paths), resources and schedules (durations, activities, temporal ordering relations, reasoning about state, interval activities).

In addition to PSL-Core and its extensions, other sets of axioms may

be required to introduce new primitive concepts; these axioms are grouped into core theories. One of the most important such core theories in PSL is situation calculus. This is powerful enough to prove theorems about PSL-Core and its extensions, such as theorems to characterize the completeness of the set of resource roles, and similar theorems to characterize the structure of partially ordered actions. It is also strong enough for building formal semantic models and proving the soundness of proposed semantic translation schemes.

Evaluation. While PSL may seem too complex and general purpose a language for the description of the behavior of the processes and transactions taking place in the realm of WSs, the key idea of using first-order logic as its semantic foundation is particularly appealing for AVANTSSAR. In fact, using some logical formalisms (and, in particular, first-order logic) to build the semantics would facilitate not only the writings of properties systems should satisfy but also the development of the techniques to analyze specifications by using automated reasoning techniques, which is one of the main objectives of AVANTSSAR. On a more concrete level, some of the notions used in PSL to formalize the behavior of processes may be directly used or suitably specialized to the case of WSs.

8.7 YAWL (Yet Another Workflow Language)

YAWL (Yet Another Workflow Language) [34] is, as the name says, (another) workflow language. Its development started in 2002 and is still evolving thanks to an open source community leaded by Eindhoven University of Technology and Queensland University of Technology. YAWL is not (yet) supported by any standardization committee.

YAWL project originated as the natural complement to *Workflow Patterns* [67] project. *Workflow Patterns* is an initiative started in 1999, also from Eindhoven University of Technology and Queensland University of Technology, trying to list all recurrent problems and proved solutions (i.e. patterns) related to the development of workflow applications. In that respect, they evaluate workflow-related products and standards, judging which of the patterns they support, and to what extent. Seeing that no product or standard could answer all workflow patterns, they decided to create it: YAWL.

Workflow Patterns are divided into categories. These are control-flow patterns, resource patterns, data patterns and exception handling patterns, currently adding up to more than 200 total patterns:

- Control-flow patterns aim at covering all process control aspects of a workflow. They range from, for instance, *sequence pattern* (where "A task in a process is enabled after the completion of a preceding task in the same process.") to all kinds of iteration or advanced branching.
- Resource patterns deal with the *entities capable of doing work*. That includes *authorization* pattern for instance ("the ability to specify the range of privileges that a resource possesses in regard to the execution of a process").
- Data patterns are focusing on the various ways a system can handle data. Whether it supports different scopes for data, whether modifications can be applied to the data before being passed to the next activity, etc.
- Exception handling patterns deal with the various exception cases and their handling.

YAWL is primarily a workflow language, but has also an execution engine, a graphical editor and an analysis and verification tool. The editor handles .ywl (binary) files for graphical representation and can also produce .xml files with only WF process data. The .xml files are used in the execution engine.

What can be specified in YAWL. Basically all identified workflow patterns can be specified in YAWL. To this day, YAWL can noticeably deal with:

- synchronizations
- choices
- multiple instances
- exception handling
- resource allocation policies (role-based or direct)
- etc

Semantics of YAWL. Initially built on top of *Petri Nets*, YAWL has evolved to support patterns that could not be easily (or not at all) handled with *Petri Nets*. YAWL grammar involves tasks ('atomic', 'composite' or 'multiple instance') and join/split connexions between tasks:

Control-flow. All tasks are related to their predecessor(s) with a join condition and to their successor(s) with a split condition. That gives:

- simple split: the task is followed by one, only one and always the same other task
- AND split: activates all outgoing links from this task upon completion
- OR split: activates a number of outgoing links from this task upon completion
- XOR split: activates one outgoing link from this task upon completion
- simple join: the task follows one, only one and always the same other task
- AND join: activates this task when all incoming links have been activated
- OR join: activates this task when one or more incoming links are activated and there is no possibility for other links to be activated if the task continues to wait
- XOR join: activates this task each time an incoming link has been activated

YAWL also defines a 'Cancellation region' item, where all elements within a dotted region are deactivated upon task activation. Workflow designers can thus specify cancellation of single tasks up to whole processes.

Data-flow. YAWL uses XML Schema, XPath and XQuery for data definition and manipulation:

- It provides a set of idiomatic primitive data types and the ability to define user defined data types (XML Schema)
- It supports variables, input/output parameters, and run-time variable transformations (XPath and XQuery)

Evaluation. While not being standardized, YAWL looks like the most complete workflow language to date. The AVANTSSAR language should, if not altogether based on YAWL, at least keep an eye on its own coverage of the various workflow patterns.

9 Choreography Layer

Choreography vs. Orchestration. Previously in this deliverable, we have already discussed orchestration in relation with the Business Process Layer section 8 and BPDM subsection 8.4. The difference to choreography can be illustrated by the origin of the metaphor in the fine arts, namely that a choreography is a plan that every actor follows by, while orchestration is controlled by a conductor.

While orchestration is defined from the point of view of a particular role or process, choreography describes the observable behavior of an entire system from a global point of view, without emphasis on any role. This holds both for the types of exchanged messages and for the control and data flow.

The goal behind choreography is, according to the W3C working group charter, that a choreography defines a kind of contract between the services based on the externally visible behavior. Also, it is intended to generate code skeletons for web services and BPEL process from a choreography.

While orchestration languages are closer to executable processes, whereas choreography languages describe—in a top-down fashion—constraints of the system from which we cannot necessarily infer an implementation. However, this distinction between choreography and orchestration is blurred with the evolution of specification languages.

9.1 WSCI (Web Services Choreography Interface)

The Web Service Choreography Interface (WSCI) was proposed in 2002 by BEA Systems, Intalio, SAP, and Sun and has the status of a note of the W3C since then [60]. Apparently, it never made it to a recommendation so far.

WSCI works in conjunction with WSDL so that one can dynamically find web services and use them in one's business processes—which requires a definition of the possible collaboration and interaction of partners. A WSDL description provides basically static information about a service; WSCI complements the static interface details provided by the WSDL file by describing the choreography of operations. Thus, WSCI describes the observable flow of messages of (stateful) web services. The goal is to enable users to understand how to interact with a service in a meaningful way.

More in detail, the additional informations about the dynamic behavior of web services that a WSCI can provide and that are not covered by a WSDL description are the following.

- Sequence of messages and processes, e.g. the operation of placing an order must occur before the actual payment operation is performed.

- Correlation of messages, e.g. correlating a reply (and subsequent communication) to a request using an order-ID.
- Workunits: what triggers a message exchange/process, when is it finished? This includes also the ability to either execute the entire unit or restore a consistent state prior to execution (compensation).
- Handling of exceptions like a time-out. WSCI also supports catching exceptions, allowing for recoverable operations.
- Defining Contexts in which variables are shared or to which exceptions are related.

There are basic activities like request and response messages or invocation of external services, and structured activities like sequential and parallel processing.

Every party describes, in an own choreography document, its role and sequence of messages subjectively. Thus, a WSCI choreography consists of a set of WSCI documents, one for each participant. On top of this, WSCI allows for the global composition of several interfaces, e.g. composing a booking service with an airline service. This is based on the WSDL port type features, describing sequential and logical dependencies between observable message exchanges. One may thus also understand WSCI as another layer on top of the web services stack as done in [51]: the WSCI description maps work-units of every participant down to WSDL operations. Other features include selectors to abstract from concrete message format, calling subroutines, several control elements for activities such as loops, switches, etc., and constructs for timing.

Semantics and Tools. There have been several approaches to formalize the semantics of WSCI. The difficulty lies—as also for other choreography languages—in the fact that the descriptions are usually far from executable. [69] describes an approach based on hierarchical colored Petri nets. [10] formalize WSCI in CCS and check deadlock-freeness and compatibility of web services, i.e. whether they are able to interoperate. They also consider automatic generation of adaptors for the case that interoperation is not directly possible.

9.2 WS-CDL (Web Services Choreography Description Language)

The Web Services Choreography Description Language (WS-CDL) is a W3C candidate recommendation since 2005 [61]. Many concepts are similar to WSCI, and, once more, the philosophy of WS-CDL is not the definition of an executable description or a new web-service, but to describe the common collaborative observable behavior of a system in terms of abstract business processes. This may also be regarded as a protocol between autonomous parties without a central point of control. When a service does show a behavior violating these constraints, this is considered as an error in the sense that it is not conformant with its WS-CDL description.

WS-CDL is related to formal methods, in particular π-calculus by which some concepts have been inspired. However, the designers have not defined a formal semantics so far and many questions remain open, in particular the mapping of constructs to lower levels of the stack, in particular the link to WSDL and BPEL descriptions [6].

A WS-CDL description consists of a static and dynamic part. The static part describes the communicating entities and their communication medium:

- All interactions are between roles.
- The declaration of participants partitions the set of roles. A participant is a set of roles that will be played by the same physical entity and thus defines the scope over which variables may be shared between roles.
- A channel is always between two roles (i.e. no multi-cast), where interaction is divided into two phases, request and response. Related to that, one can define aliases to parts of messages in order to abstract from the concrete XML-presentation of messages.

The dynamic part describes the observable behavior in terms of a choreography (and possibly subchoreographies) consisting of the following three types of activities:

- Basic Activities are interactions, i.e. request, response, and request-respond exchanges, invoking a sub-activity, and assigning new values to variables.
- Workunits define conditional and repeated execution of activities.
- Structural Activities define sequential and parallel execution of activities, as well as form of (external) choice.

Besides that, the dynamic part may also contain blocks for exceptions and finalization of activities.

Semantics and Tools. As it is the case for WSCI, the tool support for WS-CDL is currently limited. Since descriptions are lengthy and hardly human-readable, this is a particular disadvantage. There is an Eclipse plug-in `pi4soa` [64] which provides a basic support with a graphical editor for WS-CDL choreographies and the generation of BPEL specifications. [42] discusses more generally the generation of BPEL choreographies from WS-CDL choreographies.

As said before, WS-CDL is, at least partially, related to process calculi and a semantics may be defined in these terms, although this has not been done in the official documents and the large parts of the semantics remain unclear. [68, 70] formalize simplified versions of WS-CDL, describing the roles from a local view-point, introducing the concept of a *dominant* role. They use the SPIN model-checker to check CDL specifications for properties like deadlock freeness. [35] directly interprets WS-CDL as a transition system where the state is essentially a stack of currently executing (sub-)choreographies.

9.3 Evaluation

With respect to orchestration languages, choreography languages focus on the global view of a system and are more abstract. There are thus several questions that can be interesting from formal methods point of view, e.g. considering choreography as a layer on the WS-stack. However, these issues appear to be mainly interesting from a theoretical point of view, for most practical problems BPEL/orchestration languages appear sufficient. Also, a major weakness of WSCI and WS-CDL is that the integration of web services into business process execution cannot be modeled in full detail [11]. So at least for the design of an industrially suited specification language, choreography issues should not be a top priority. For foundational research on composition, choreography however may still be interesting, at least raising important questions.

References

[1] A. Agrawal et al. Web Services Human Task (WS-HumanTask), Version 1.0. `http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel4people/WS-HumanTask_v1.pdf`, June 2007.

[2] A. Agrawal et al. WS-BPEL Extension for People (BPEL4PEOPLE), Version 1.0. `http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel4people/BPEL4People_v1.pdf`, June 2007.

[3] Martín Abadi. Logic in access control. In *LICS*, pages 228–. IEEE Computer Society, 2003.

[4] Martin Abadi, Michael Burrows, Butler Lampson, and Gordon Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, 1993.

[5] Keith Ballinger, Don Box, Francisco Curbera, Srinivas Davanum, Don Ferguson, Steve Graham, Canyang K. Liu, Frank Leymann, Brad Lovering, Anthony Nadalin, Mark Nottingham, David Orchard, Claus von Riegen, Jeffery Schlimmer, Igor Sedukhin, John Shewchuk, Bill Smith, Greg Truty, Sanjiva Weerawarana, and Prasad Yendluri. Web services metadata exchange (ws-metadataexchange). `http://msdn.microsoft.com/ws/2004/09/ws-metadataexchange/`, 2004.

[6] A. Barros, M. Dumas, and P Oaks. A Critical Overview of the Web Service Choreography Description Language. *BPTrends Newsletter*, 3, 2005.

[7] Lujo Bauer, Scott Garriss, and Michael K. Reiter. Distributed proving in access-control systems. In *IEEE Symposium on Security and Privacy*, pages 81–95, 2005.

[8] Lujo Bauer, Scott Garriss, and Michael K. Reiter. Efficient proving for practical distributed access-control systems. In *ESORICS*, pages 19–37, 2007.

[9] C. Braghin, D. Gorla, and V. Sassone. A distributed calculus for role-based access control. In *Proceedings of 17th IEEE Computer Security Foundations Workshop, CSFW'04*, pages 48–60. IEEE Press, 2004.

[10] Antonio Brogi, Carlos Canal, Ernesto Pimentel, and Antonio Vallecillo. Formalizing Web Service Choreographies. *ENTCS*, 105:73–94, 2004.

[11] A. Bucchiarone and S. Gnesi. A Survey on Service Composition Languages and Models. In *WsMaTe*, 2006.

[12] Windows Communication Foundation. WCF. `http://msdn2.microsoft.com/en-us/netframework/aa663324.aspx`.

[13] Oasis Consortium. Universal Description, Discovery, and Integration specification. `http://uddi.org/pubs/uddi-v3.0.2-20041019.pdf`.

[14] OASIS Consortium. eXtensible Access Control Markup Language (xacml) version 2.0, oasis standard. `http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf`, 2005.

[15] OASIS Consortium. Oasis: Reference model for Service Oriented Architecture 1.0. `http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.html`, 2006.

[16] OASIS Consortium. Web Services Federation Language (WS-Federation) V1.1, December 2006.

[17] OASIS Consortium. Security Assertion Markup Language V2.0 Technical Overview. `http://wiki.oasis-open.org/security/Saml2TechOverview`, March 2008.

[18] OASIS Consortium. Web Services Reliable Messaging (WS-ReliableMessaging) V1.2. `http://docs.oasis-open.org/ws-rx/wsrm/200702`, February 2008.

[19] OMG Consortium. Business Process Modeling Notation (BPMN). `http://www.bpmn.organdhttp://www.omg.org/spec/BPMN/1.1/PDF`, 2008.

[20] OMG Consortium. Business Process Definition Metamodel. `http://www.omg.org/spec/BPDM/1.0/Beta1/`, July 2007.

[21] The World Wide Web Consortium. The world wide web consortium. `http://www.w3.org`.

[22] The World Wide Web Consortium. Web Services Description Language (WSDL) version 2.0. `http://www.w3.org/TR/2006/WD-wsdl20-rdf-20060327/`, 2006.

[23] The World Wide Web Consortium. Simple Object Access Protocol 1.2. `http://www.w3.org/TR/soap12-part1`, Apr 2007.

[24] The World Wide Web Consortium. Web Services Description Working Group. http://www.w3.org/2002/ws/desc/, June 07.

[25] The World Wide Web Consortium. Web Services Choreography Description Language version 1.0. http://www.w3.org/TR/ws-cdl-10/, Nov 2005.

[26] Common Object Request Broker Architecture (CORBA). http://www.corba.org/, 1992.

[27] Microsoft Corporation and Digital Equipment Corporation. Component Object Model Technologies (COM). http://www.microsoft.com/com/default.mspx.

[28] Jason Crampton, George Loizou, and Greg O'Shea. A logic of access control. *Comput. J.*, 44(1):54–66, 2001.

[29] Remco M. Dijkman, Marlon Dumas, and Chun Ouyang. Formal semantics and analysis of bpmn process models. Technical report, Queensland University of Technology, 2007. http://eprints.qut.edu.au/archive/00007115/01/7115.pdf.

[30] D. Ferraiolo and R. Kuhn. Role-based access controls. In *15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992. http://citeseer.ist.psu.edu/ferraiolo92rolebased.html.

[31] David F. Ferraiolo, John F. Barkley, and D. Richard Kuhn. A role-based access control model and reference implementation within a corporate intranet. *ACM Transactions on Information and System Security (TISSEC)*, 2(1):34–64, February 1999. http://www.acm.org:80/pubs/citations/journals/tissec/1999-2-1/p34-ferraiolo/.

[32] Roy T. Fielding. Architectural Styles and the Design of Network-based Software Architectures, 2000.

[33] The Center for XML and Web Services Technologies. the web services protocol universe. http://www.qcc.cuny.edu/xmlcenter/protocolstack.htm.

[34] The YAWL Foundation. YAWL. http://www.yawl-system.com.

[35] Lars-Åke Fredlund. Implementing WS-CDL. In *proceedings of JSWEB 2006*, 2006.

[36] Deepak Garg and Martín Abadi. A modal deconstruction of access control logics. In Roberto M. Amadio, editor, *FoSSaCS*, volume 4962 of *Lecture Notes in Computer Science*, pages 216–230. Springer, 2008.

[37] J. Gray. The transaction concept: Virtues and limitations, September 1981.

[38] Michael Havey, August 2005.

[39] Polar Humenn. The Formal Semantics of XACML. `http://lists.oasis-open.org/archives/xacml/200310/pdf00000.pdf`.

[40] IBM and Microsoft Corporation. Understanding WS-Federation. `http://msdn.microsoft.com/en-us/library/bb498017.aspx`, May 2007.

[41] Harold Lockhart. Demistifying SAML. `http://dev2dev.bea.com/pub/a/2005/11/saml.html`, September 2005.

[42] J. Mendling and M. Hafner. From Inter-Organizational Workflows to Process Execution: Generating BPEL from WS-CDL. In *MIOS workshop, OTM 2005 Workshops, LNCS 3762*. Springer, 2005.

[43] L. Bastida Merino and G. Benguria Elguezabal. Business Process Definition Languages versus Traditional Methods Towards Interoperability, Jan 2005.

[44] Meta-Object Facility. MOF. `http://www.omg.org/mof/`.

[45] Microsoft and All. Web services dynamic discovery (WS-Discovery. `http://specs.xmlsoap.org/ws/2005/04/discovery/`, April 2005.

[46] Oasis Consortium. Web Services Business Process Execution Language Version 2.0. `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf`, 11 April, 2007.

[47] OASIS Web Services Reliable Messaging TC. WS-Reliability 1.1. `http://docs.oasis-open.org/wsrm/ws-reliability/v1.1/wsrm-ws_reliability-1.1-spec-os.pdf`, November 2004.

[48] The Object Management Group. OMG. `http://www.omg.org`.

[49] Chun Ouyang, Eric Verbeek, Wil M. P. van der Aalst, Stephan Breutel, Marlon Dumas, and Arthur H. M. ter Hofstede. Formal semantics and analysis of control flow in WS-BPEL. *Sci. Comput. Program.*, 67(2-3):162–198, 2007.

[50] N. Goodman P.A. Bernstein, V. Hadzilacos. Concurrency control and recovery in database systems, 1987.

[51] Chris Peltz. Web Services Orchestration, 2003. HP white paper.

[52] D. J. Power, M. A. Slaymaker, and A. C. Simpson. On formalising and normalising role-based access control systems. *Accepted for publication in The Computer Journal*, 2007.

[53] Business Process Modeling Initiative. BPMI. `http://www.bpmi.org`, 2000.

[54] Jan Recker and Jan Mendling. On the translation between BPMN and BPEL: Conceptual mismatch between process modeling languages. `citeseer.ist.psu.edu/recker06translation.html`, Jan 2006.

[55] Nick Russell and Wil M.P. van der Aalst. Evaluation of the bpel4people and ws-humantask extensions to ws-bpel 2.0 using the workflow resource patterns, 2007.

[56] Frank Siebenlist. Modeling delegation of rights in a simplified xacml with haskell. `http://www-unix.mcs.anl.gov/~franks/haskell/XacmlDelegationHaskell0.html`.

[57] Simple Object Access Protocol v1.2. `http://www.w3.org/TR/2003/REC-soap12-part0-20030624/`, 2003.

[58] Joint specification by IBM, Microsoft, and BEA. Web services transactions specifications. `http://www.ibm.com/developerworks/library/ws-transpec/`, August 2002.

[59] the Workflow Management Coalition. XPDL Schema. `http://www.wfmc.org/standards/docs/TC-1025_bpmnxpdl_24.xsd`, October 2005.

[60] The World Wide Web Consortium. Web Service Choreography Interface, 2002. Version 1.0, `http://www.w3.org/TR/wsci/`.

[61] The World Wide Web Consortium. Web Services Choreography Description Language, 2005. Version 1.0, Candidate Recommendation, `http://www.w3.org/TR/ws-cdl-10/`.

[62] The World Wide Web Consortium. XML Schema Definition (XSD). `http://www.w3.org/XML/Schema`, March 2005.

[63] UDDI. Introduction to UDDI: Important Features and Functional Concepts. `http://uddi.org/pubs/uddi-tech-wp.pdf`, 2004.

[64] pi4soa. `http://www.pi4.org/`.

[65] J.E. White. RPC definition in the RFC707. `http://tools.ietf.org/html/rfc707`, 1975.

[66] Dave Winner. XML-RPC specifications. `http://www.xmlrpc.com/spec`, 1999.

[67] The Workflow Patterns initiative. Workflow Patterns. `http://www.workflowpatterns.com`.

[68] Zhao Xiangpeng, Yang Hongli, and Qiu Zongyan. Towards the Formal Model and Verification of Web Service Choreography Description Language. In *Web Services and Formal Methods, LNCS 4184*. Springer, 2006.

[69] YanPing Yang, QingPing Tan, and Yong Xiao. Verifying Web Services Composition Based on Hierarchical Colored Petri Nets. In *Workshop on Interoperability of heterogeneous information systems*. ACM, 2005.

[70] Qiu Zongyan, Zhao Xiangpeng, Cai Chao, and Yang Hongli. Towards the Theoretical Foundation of Choreography. In *International World Wide Web Conference*. ACM, 2007.