Towards a Contract-based Interoperation Model

Félix Oscar Fernández Peña and Steven Willmott,

Instituto Superior Politécnico José Antonio Echeverría
Havana, Cuba
felix@ceis.cujae.edu.cu

Universitat Politècnica de Catalunya (UPC),
Barcelona, Spain
steve@lsi.upc.edu

4th September, 2007

Abstract.

Web Services-based solutions for interoperating processes are considered to be one of the most promising technologies for achieving truly interoperable functioning in open environments. In the last three years, the specification in particular of agreements between resource / service providers and consumers, as well as protocols for their negotiation have been proposed as a possible solution for managing the resulting computing systems. In this report, the state of the art in the area of contract-based web service applications is closely studied, identifying current limitations and possibilities. On the basis of this analysis, a general model for contract specification, negotiation, agreement, execution and management is introduced. Such a model has broad applicability both in electronic business integration and distributed knowledge management systems for decision support.

Initial work presented here was completed in September 2005 and is published here as background for the European Commission funded project IST CONTRACT http://www.ist-contract.org/.
# Towards a Contract-based Interoperation Model

## Index

1. Introduction ..................................................................................................................... 1
2. State of the Art ................................................................................................................ 2
   2.1. Introduction .............................................................................................................. 2
   2.2. Contracting. Basic Overview ................................................................................... 2
   2.3. Relevant Previous Research Results ........................................................................ 3
      2.3.1. Contract-based software design ........................................................................ 3
      2.3.2. Electronic Business XML (ebXML) ................................................................. 6
      2.3.3. Social Commitment .......................................................................................... 8
   2.4. Contract Specification .............................................................................................. 9
      2.4.1. Web Services Choreography Description Language (WS-CDL) ........... 10
      2.4.2. Web Service Level Agreement Language Specification (WSLA) ....... 15
      2.4.3. Web Services Agreement Specification (WS-Agreement) ....................... 21
   2.5. Automatic Negotiation of Contracts ...................................................................... 26
      2.5.1. WSAN proposal .............................................................................................. 27
      2.5.2. Agent Research on Negotiation Protocols ...................................................... 29
   2.6. Comparison between proposals analysed ................................................................ 32
3. General Model Proposal ............................................................................................... 34
   3.1 Introduction ............................................................................................................. 34
   3.2. General Model Foundations ................................................................................... 34
   3.3. General Model Views ............................................................................................ 37
      3.3.1. Actors and Components View ....................................................................... 37
      3.3.2. Contract View ................................................................................................. 40
      3.3.3. Contract States View ....................................................................................... 41
      3.3.4. Communication View ..................................................................................... 43
   3.4. Semantic Extension of the Contract Definition ..................................................... 46
   3.5 Benefits of the proposed model .............................................................................. 46
4. Conclusions ................................................................................................................... 48
5. References ..................................................................................................................... 49
1. Introduction

Inter-communication is crucial for improving business profitability in today’s global and competitive world. Web services have been considered as a possible solution for integrating loosely applications in open environments due to their ability for establishing dynamic bindings against their interfaces. Research in this area is expanding rapidly and many specifications are emerging for the integration of web services.

However, as well as for workflow management systems, models are needed to control commitment and agreements between services - especially in open environments where no single party controls all services. This need to explicitly declare and control of the commitments of the parties has produced a significant amount of interest in “contracting”. Some language specifications are in progress in order to explicitly express the interrelation of processes; such is the case of the languages presented in section 2.

A rigid agreement between parties in specific scenarios is the first step in the evolution of mechanisms for integrating web service-based applications. This is what is done with current web service platform implementations, even when the definition of the agreements is not usually made explicit but is implicit in the system documentation.

The explicit declaration of the relations between parties constitutes a step further towards the formalization of contract-based applications, improving software maintenance and quality control. This report aims to define a sufficiently general model to achieve, not just adequate system documentations, but also facilitate the automatic negotiation of system structure for specific domain conditions. The main objective of this research is to assist the automatic adaptability of applications in changing environments through the establishment of a contract-based model.

As result, the report:

1. includes the study of existing proposals of languages for the contract specification and negotiation, noting the most relevant aspects,
2. presents a model which integrates these perspectives in order to allow contract specification, monitoring, and management in web service based distributed applications, and,
3. describes how to make practical use of the model through the implementation of a contract-based framework for negotiating and managing applications. This paper is divided up as follows: in section 2, the state of the art is discussed; section 3 presents the definition of the model and its implementation through the proposed framework. Section 4 concludes the paper and gives an overview of future work.
2. State of the Art

2.1. Introduction

Advances in communication and transportation technologies, combined with free-market ideology, have given goods, services, and capital unprecedented mobility. Today’s globalization demands flexible interoperation between parties across the world, concluding that mechanisms for dynamic negotiation and management of exchange processes are really necessary.

Meanwhile, the global network of the Internet is increasingly used for the application to application communication. In this context, the programmatic interfaces available are referred to as Web Services [W3C05]. Web service-based applications documentation for proper interoperability has been identified as a compelling issue, and in recent years has received attention from different groups from areas of research such as Grid Computing and Web Services choreography.

There are a large number of specifications (many XML-based with and without a semantic flavour [OWLS05] [SUMO05] [XMSG00] [BPEL05] [ebXML05] [Grosof01] [Grosof03]) for modelling the message and process flows across a set of services, but there is a gap between these proposals and the mechanisms for discovering / integrating web services functionalities (UDDI and semantic matchmaking, which look for how to find desired web services) in order to provide an integrated compound service. Further, the integration over the Web Service Description Language (WSDL) [Christensen01], demanded by W3C, is still an open issue.

Finally, these current approaches do not state how to explicitly declare and control the commitment and obligation of parties. This issue is especially important in open environments since no single party controls all services. An alternative means to approach Web Services interaction is therefore to borrow from common human processes for collaboration that are based on an agreement between multiple parties. Such approaches are the focus of this report. In this section, different proposals are analysed, extracting to the most important features of each of them.

2.2. Contracting. Basic Overview

The development of distributed applications using web services today is focused on the integration of multi-party services and automatic negotiation policies for doing so. Standards and initiatives such as ebXML [ebXML05] and the Service Oriented Architecture are gaining increasing acceptance [Keller02].

Keller et al stated that one of the very important elements of a Dynamic e-Business (DeB) is an electronic contract [Keller02]. Other authors have agreed that in an e-service environment, contracts are important for attaining business process interoperability and enforcing their proper enactment [Chiu03]. The basic idea of contract definitions is the
explicit declaration of the terms under which web services interact. These specifications include rewards, obligations, and penalties to be applied if successful operation is not achieved. The system capabilities for automatic contract management and negotiation are proportional to the descriptive capabilities of languages for describing the semantics of contracts across domain boundaries of large distributed systems.

One fundamental characteristic of contract-based applications in open environments is their flexibility, allowing the adaptation of the system behaviour in order to adjust the system behaviour to changing environments. In order to exploit this flexibility it is important to guarantee:

1. The specification of the context in which the contract is executed, and the explicit declaration of context relationships against other contexts.
2. The specification of the actors and their roles in the system (to allow to multiple parties in the scenario to play the role of provider or consumer).
3. The specification of the actions of the involved parties, considering their execution inside a black box (corresponding activities may change without disturbing the system global functioning).
4. The metrics specification for evaluating the QoS associated to the execution of corresponding actions.
5. The specification of mechanisms for measuring metrics (that is to say, the establishment of linkage between metric definitions and actions for measuring them).

2.3. Relevant Previous Research Results

The idea of the explicitly declaration of the existing relationships between parties behaving as providers or consumers was proposed by Meyer to object oriented applications design. We consider that there are some elements to note from the Meyer's proposal (1988) –see section 2.3.1-, even though its field of application is for tightly coupled applications and not for open environments.

The evolution of Electronic Data Interchange (EDI) into what is called Electronic Business XML (ebXML) is analysed in section 2.3.2,

Meanwhile, results developed in the agent and multi-agent systems research community are also potentially applicable to the improvement of web service interoperability. Using agent research community results is proposed in papers such as [Willmott04] and [Paurobally05b]. In the section 2.3.3, social commitment theory for agent communication is analyzed in order to draw out formal specifications that may clarify the appropriate interaction protocols between multiple processes in open environments.

2.3.1. Contract-based software design

Contract-based software design is a Meyer et al. proposal for improving object oriented software design [Meyer97]. This approach for designing by contracts was published for the first time in 1988, and applied in the design of the programming language Eiffel [Eiffel05]. In Meyer’s proposal, contract definition is intended for the specification of
actions starting with some *preconditions*, delivering results as defined in the *postconditions*. This is expressed in the following way:

{Preconditions} Operation {Postconditions}

Any kind of expression like this is considered an assertion –the “what to do”- . An assertion represents an expression involving some entities of the software, and stating a property that these entities must satisfy at certain stages of software execution. A typical assertion might express that a certain integer has a positive value or that a certain reference is not void. An assertion is a conceptual tool enabling software developers to construct correct systems and to document why they are correct [Meyer97].

The first use of assertions is the semantic specification of routines: to express what a routine is supposed to do. Tasks performed by a routine are specified by two assertions: a *precondition* `<require>` and a *postcondition* `<ensure>`. Meyer considers that assertions are a way to compose software’s specifications. From a business point of view, contracts between client and supplier modules, a benefit for one is an obligation for the other one. Producing effective and reliable software is therefore seen as “to draw up the contract representing the best possible compromise in all applicable client-supplier communications” [Meyer97].

**Class Invariants** [Meyer97] Class invariants express global properties of the instances of a class, which must be preserved by all routines. Such properties will make up the class invariant, capturing the deeper semantic properties and integrity constraints characterizing a class. A class invariant is such an assertion, expressing general consistency constraints that apply to every class instance as a whole; this is different from preconditions and postconditions, which characterize individual routines.

**Applications of using assertions** [Meyer97]:

- Help in writing correct software.
- Documentation aid.
- Support for testing, debugging and quality assurance.
- Support for software fault tolerance.

Note: The last two assume the ability to monitor assertions at run time.

**Design by Contract**, is considered by its definer as a conceptual tool for analysis, design, implementation and documentation, helping people to build software in which reliability is built-in, rather than achieved or attempted after the fact through debugging; in mills terms, enabling developers to build correct programs and know it. Some characteristics of Design by Contracts are [Meyer97]:

1. It proposes to declare pre/post-conditions and to do not check them inside routines, gaining clarity and reducing redundancy.
2. It is concerned about software-to-software communication, not software-to-human or software-to-outside-world communication.
3. Assertions express correctness conditions. If some condition is violated it means that there is a bug in the software.

For Meyer, the definition of a contract should fulfil these key concepts [Meyer97]:

- Assertions are boolean expressions expressing the semantic properties of classes and reintroducing the axioms and preconditions of the corresponding abstract data types.
- Assertions are used in preconditions (requirements under which routines are applicable), postconditions (properties guaranteed on routine exit) and class invariants (properties that characterize class instances over their lifetime).
- A precondition and a post-condition associated with a routine describe a contract between the class and its clients. The contract is only binding on the routine inasmuch as calls observe the precondition; the routine then guarantees the post-condition on return. The notion of contracting provides a powerful metaphor for the construction of correct software.
- The invariant of a class expresses the semantic constraints on instances of the class. The invariant is implicitly added to the precondition and the post-condition of every exported routine of the class.
- A class describes one possible representation of an abstract data type; the correspondence between the two is expressed by the abstraction function, which is usually partial. The inverse relation is in general not a function.
- An implementation invariant, part of the class invariant, expresses the correctness of variant, used to ascertain termination.

Some unexpected and undesired event will sooner or later occur whilst systems are executing. This is known in software engineering as an exception. So, preconditions and postconditions are not enough. It is generally also necessary to define possible penalties to apply when processing exceptions. Meanwhile, when an exception occurs during the execution of a routine, there are two legitimate responses:

R1. Retrying: attempt to change the conditions that led to the exception and to execute the routine again from the start.
R2. Failure (also known as organized panic): clean up the environment, terminate the call and report failure to the caller.

Concluding, Meyer identified some characteristics about exceptions [Meyer97]:

- Exception handling is a mechanism for dealing with unexpected run-time conditions.
- A failure is the impossibility, for a routine execution, to fulfil the contract.
- A routine gets an exception as a result of the failure of a routine which it has called, of an assertion violation, or of an abnormal condition signalled by the hardware or operating system.
- It is also possible for a software system to trigger a “developer exception” explicitly.
• A routine will deal with an exception by either Retry or Organized Panic. Retry re-executes the body; Organized Panic causes a routine failure and sends an exception to the caller.
• The formal role of an exception handler, not ending with a retry, is to restore the invariant. The formal role of a branch, ending with a retry, is to restore the invariant and the precondition so that the routine body can try again to achieve its contract.

Meyer’s proposal is designed for the management of under production object-oriented software systems; it does not intend to serve as a solution for the automatic change of systems functioning. Nevertheless, the authors of this paper consider that its principles are valuable for the definition of a general framework for the appropriate deployment of distributed systems in open environments.

2.3.2. Electronic Business XML (ebXML)
In September 1999, the United Nation’s Center for Trade Facilitation and Electronic Business (UN/CEFACT) and the Organization for the Advancement of Structured Information Standards (OASIS) began a 15 to 18-month project to standardize the global exchange of business information. The result of this project is Electronic Business XML (ebXML). ebXML provides a standard infrastructure for global electronic business that enables medium to large businesses to exchange business information [Deitel01].

This standard aims to provide a standards-based business process foundation for promoting the automation and predictable exchange of business collaboration definitions using XML [ebXML05]. ebXML Collaboration Protocol Profile and Agreement (CPPA) defines an XML Schema for controlling the agreement between business involved parties. Currently, ebXML is an industry-standard set of specifications for collaborative B2B-based Web services [ebXML05].

The wide reach of two international organizations, coupled with the open nature in which this standard is being developed, ensures a strong user foundation for a single, XML-based, global business framework. The Global Commerce Initiative, a coalition of 40 major businesses, including Kraft Foods and Home Depot, adopted ebXML as an integral part of its e-commerce framework [Deitel01]. The next figure illustrates the structure of the ebXML architecture.
The ebXML standard addresses the demands of e-business collaborations on Web services by going beyond Service Description and Service Publication and Discovery by providing a framework for establishing the business context and addressing issues such as the following [ebXML05]:

- What business process is this Web service interaction part of?
- What are the roles of the various parties involved?
- What are the XML documents exchanged for in the business interactions?
- Who are the parties involved?
- What are the environmental requirements of this business collaboration (in order to succeed)?
- Do negotiation patterns exist for collaborating parties, after Service Discovery?

ebXML is expected to impact systems designed for allowing the interoperation of multiple partners due to its broad support. ebXML focuses on controlling transactions (platforms for run time execution management) over existing partner profiles and agreements (defined mostly in design time). This characteristic may restrict ebXML's usefulness in truly dynamic environments where automatic negotiation of agreements between corresponding parties is expected in order to fulfill client requirements.

For instance, the Sun One Integration Server [Sun05a] (a business interoperation platform based on ebXML and presented by Sun Microsystems in 2003) includes an agreement editor for proposing, accepting and signing contracts in a semi-automatic way. As noted in [Sun05b], the contract treatment flexibility is limited (the automatic interaction for establishing agreements between partners is out of the scope of the tool and not considered by the ebXML specification).
2.3.3. Social Commitment

Colombetti et al have been working in the field of social commitment for the agent community [Colombetti00]. The formal specification of a logical model based on social commitment for agent communication was published in 2003 [Verdicchio03].

Verdicchio and Colombetti formalised the required agent actions for interacting during the life cycle of agreements. Their work is based in an extension of the CTL* temporal logic, which was called CTL±. Further, Verdicchio and Colombetti, formally defined the conditions under which a commitment is fulfilled or violated, and how CTL± can be used to define the semantics of an Agent Communication Language (ACL).

Verdicchio and Colombetti defined a commitment as a first order formula considered as Comm(e,x,y,u), meaning that event e has brought about a commitment for agent x, relative to agent y, to the truth of u. In the same way, a pre-commitment formula is defined as Prec(e,x,y,u) [Verdicchio03].

From [Verdicchio03], it is possible to deduce six different states for a commitment; they are: **pre-committed, committed, active, pending, fulfilled and violated**. A commitment is considered as **violated** when its content is false whilst been an **active** contract. To know if the analysed contract is active or not, the **Reichenbach’s point of speech** and **Reichenbach’s point of event** are used in the formalisation of these six previous mentioned states [Verdicchio03]. A commitment is fulfilled when the commitment **finalize** without been **violated** ([Verdicchio03] includes the complete reference of the formal definition of commitment between multiple parties).

Verdicchio and Colombetti stated as the fundamental unit of agent communication the **exchange of a message**. A message is sent by an agent (the sender) and received by another agent (the receiver). The message structure includes a message type indicator and a message body [Verdicchio03]. It is important to note that this could be paired to the XML Messaging specification [XMSG00], but in the Verdicchio and Colombetti proposal each type indicator is associated with a performative of the ACL (a truly well defined semantic describes each message).

In [Verdicchio03], it is proposed to see commitments as instances of a commitment class. Instance variables contain:

- A reference to the commitment-inducing event (e.g. a message exchange).
- Two references to agents.
- An abstract representation of a CL sentence.

The commitment manipulation actions are regarded as methods of the commitment class with the formal specification given by axioms proposed in [Verdicchio03]. This way, a communicative act is defined by:

- The general form of the class of messages by which the act is performed.
• The relevant contextual conditions for a successful execution of the communicative act.
• The effect of a successful execution of the communicative act, expressed in terms of commitment – manipulation actions.

Something missing in contract specification languages is the formal specification of their foundations. It is considered by the authors of this paper that it is important to take into account the formal definitions of Verdicchio and Colombetti for evaluating contract-based frameworks against the social commitment theory (the importance of developing mechanisms for validating frameworks is described in [Abrahams04]). Meanwhile, using semantically well defined messages could facilitate the study of interaction protocols working in dynamic environments, bringing as result the improvement of communication mechanisms. The definition of a contract schema should take into account the formalised agreement conforming elements.

2.4. Contract Specification

There are other recent proposals (actually active research projects), for the specification of agreements between web services. They are:

1. The Web Services Choreography Description Language (last updated in 2004), which focuses on the description of the activities to be executed in a multi-party choreography of web services.
2. The Web Service Level Agreement language specification (last updated in 2003), focuses on the description of the Service Level Objectives (SLO) of web services, that is to say, the specification of party’s obligations, metrics for their evaluation and mechanisms for monitoring/managing the system deployment.
3. The Web Services Agreement Specification (last version released in 2004), focuses on the negotiation of agreements between two parties. Some elements from the Web Service Level Agreement language specification, such as the description of web services SLO, apply in this proposal as the fundamental element of descriptions of services as processing units in the whole system.

These three proposals are analysed in the subsequent subsections, noting how they are considered for use in specifying the contractual relations between parties describing the existing agreements among them. In order to have a clear understanding of the capabilities of each of the proposals under analysis, from the Web Services and Grid Computing communities, a practical example was encoded using the specification of each of the languages under revision. Further, a discussion about these proposals and an analysis of the encodings is included.

The example considers the existence of a distributed news system, and particularly the submission process in which a collaborator (newsProvider) sends a news item to get published. This request is received by an administrative service (blackBoard) which processes the news item, and decides whether to publish it or not. The decision is taken according to the type of information contained in the news item.
Towards a Contract-based Interoperation Model

The rules to be encoded in the example are:

- The message requestNewsItemPublication is sent from NewsProvider to BlackBoard as a request message.
- The message informNewsItemAcceptance is sent from BlackBoard to NewsProvider as a response message.

2.4.1. Web Services Choreography Description Language (WS-CDL)

The W3C Choreography Description Language (WS-CDL) [Kavantzas04a] is a standard for describing how services interoperate based on their observable behavior. WS-CDL is a W3C working draft created in 2004 by the Web Services Choreography Working Group. The purpose of WS-CDL is to define multi-party contracts, specifying the observable behavior of web services (WSs) and their clients (usually other WSs) by describing, from a global point of view, the message exchanges between them. WS-CDL choreography descriptions are intended to be used to generate code skeletons for WSs on BPEL abstract processes. WS-CDL intends to extend the Web Services Architecture approved by W3C [Booth04]. This theoretical extension is illustrated in figure 2.

![Figure 2. Proposed Architecture for WS-CDL [Kavantzas04a]](image)

The WS-CDL specification is associated to the description of packages. A package consists of:

1. Name, version, author, and “information types”.
2. Variables to store data taken through the workflow & tokens to refer to them.
3. Participants: name, role.
4. Roles: name, behaviour (name, interface).
5. Relationships between behaviours of different roles.
6. Channels for acting (request and response channels).
7. Choreography, relating all the previous elements as part of a specific package.

Interaction takes place between actors. Each actor plays a specific role and a role can be played by multiple participants. Relationships between different actors are determined by the declaration of channels of communication (request and response channels).

Neither the context nor the relation between multiple packages is explicitly described in WS-CDL. The language is focused on the choreography resulting of the integration of pertinent activities, which are grouped in work units representing the composition of interacting web services (see figure 3 and the meta model presented in figure 4).

![Figure 3. Types of Activities](image)

A choreography is described by the enclosed actions that perform the actual work; its execution can recover from exceptional conditions and provides finalization actions through the definition of:

- One Exception block, which MAY be defined as part of the choreography to recover from exceptional conditions that may occur.
- One Finalizer block, which MAY be defined as part of the choreography to provide the finalization actions.

A failure means that the whole choreography must be rolled back. This is a problem identified by Aiello et al. who propose an extension to the language for restricting the rollback process in order to gain in efficiency [Aiello05].

WS-CDL is focused on the specification of the activities necessary for fulfilling collaboration between web services in environments where the integration of uncoupled processes is required, but it is not intended for the description of contract monitoring and contract negotiation processes.

Channels are the link between WS-CDL choreographies and operations described in WSDL interfaces [Barros05]. This association between WS-CDL and WSDL is relatively restrictive. For instance, in the case of specific domain conditions, it could be necessary to change the execution plan, but as alternative actions are not described in WSDL files, changing one action for another one would be not natural if using the traditional WSDL-based calling mechanisms.
Towards a Contract-based Interoperation Model

Within a *choreography*, two or more related *interactions* MAY be grouped to form a *logical conversation*. The *channel* through which an *interaction* takes place is used to determine whether to enlist the *interaction* with an already initiated *conversation* or to initiate a new one. In this context, it is important to note that asynchronous communication (request/response) between web services is considered often useful for the communication in open environments [Willmott04] [CapeClear05], but it is not clear how to describe it using WS-CDL.

For Kavantzas, et al, the life cycle of collaborations between interacting parties has three states [Kavantzas04a]:

- **Collaboration is established** between parties.
- **Work is performed** within it.
- **Choreography completes** either normally or abnormally.

![Figure 4. WS-CDL meta model in UML [Barros05]](image)

### 2.4.1.1. Example encoding

The XML encoding below shows a *choreography* involving one interaction described in the example presented in section 2.4. This interaction takes place between the NewsProvider service and the BlackBoard service, through the *channel*
Towards a Contract-based Interoperation Model

PublicationChannel, using a request/response message exchange. General rules were specified previously; specific rules for encoding this using WS-CDL are:

- The *interaction* happens on the PublicationChannel which has a token `newsItemID` used as an identity of the *channel*. This identity element is used to identify the business process of the BlackBoard.
- The request message `requestNewsItemPublication` contains the identity of the BlackBoard business process as specified in the *tokenLocator* for the `requestNewsItemPublication` message.
- The response message `informNewsItemAcceptance` contains the identity of the consumer business process as specified in the *tokenLocator* for the `informNewsItemAcceptance` message.
- The AcceptanceChannel is sent as a part of the `requestNewsItemPublication` message from NewsProvider to BlackBoard on PublicationChannel, during the request. The record element populates the AcceptanceChannel at the BlackBoard role.

```xml
<package name="NewsItemProvBlackBoardChoreo" version="1.0" author="Félix Fernández">
  <informationType name="newsItemType" type="news:requestNewsItemPublication"/>
  <informationType name="newsItemAckType" type="news:informNewsItemAcceptance"/>
  <token name="newsItemID" informationType="tns:intType"/>
  <token name="blackBoardRef" informationType="tns:uriType"/>
  <tokenLocator tokenName="tns:newsItemID" informationType="tns:newsItemType" query="/newsItem/newsItemId"/>
  <tokenLocator tokenName="tns:newsItemID" informationType="tns:newsItemAckType" query="/newsItem/newsItemId"/>
  <role name="NewsProvider">
    <behavior name="NewsItemSender" interface="nns:newsItemProv"/>
  </role>
  <role name="BlackBoard">
    <behavior name="NewsItemChecker" interface="bns:newsItemCheckService"/>
  </role>
  <relationship name="newsItemPublication">
    <role type="tns:NewsProvider" behavior="NewsItemSender"/>
    <role type="BlackBoard" behavior="NewsItemChecker"/>
  </relationship>
  <channelType name="PublicationChannel">
    <role type="tns:NewsProvider"/>
    <reference>
      <token type="tns:newsItemProvRef"/>
    </reference>
    <identity>
      <token type="newsItemID"/>
    </identity>
  </channelType>
  <channelType name="BlackBoardChannel">
    <passing channel="PublicationChannel" action="request"/>
    <role type="tns:BlackBoard" behavior="NewsItemChecker"/>
    <reference>
      <token type="tns:blackBoardRef"/>
    </reference>
    <identity>
      <token type="tns:newsItemID"/>
    </identity>
  </channelType>
  <choreography name="NewsItemProvBlackBoardChoreo" root="true">
    <relationship type="NewsItemProviderBlackBoardRelationship"/>
  </choreography>
</package>
```
2.4.1.2. Analysis of WS-CDL

In general, it is important to remark that:

1. The purpose of WS-CDL can be seen as to define multi-party contracts, which describe externally observable behaviour of web services and their clients (usually other web services) by describing the message exchanges between them [Barros05].

2. Channels are the link between WS-CDL choreographies and operations described in WSDL interfaces.

3. A choreography description is a container for a top-level work unit –composed by activities- and an optional exception and finalizer work units.

4. A failure in a choreography means that completed choreographies must be rolled back.

5. WS-CDL recognizes the value of BPEL in the industry and establishes an alignment between them for the description of web services coordinated operations.

Barros et al. presented an interesting critical overview of WS-CDL [Barros05]. Some of the most relevant issues identified by them are:

1. Although WS-CDL appears to borrow terminology from pi-Calculus, the link to this or any other formalism is not clearly established.
2. The mapping between WS-CDL and other elements of the extended Web Services Architecture Stack remains open and conceptual sufficiency in aligning WS-CDL with these standards is arguably limited.
3. The mapping between WS-CDL and WSDL, mandated by the Web Service Choreography group’s requirements, is largely open.
4. Mapping between interactions represented in WS-CDL through channels and the WSDL 2.0 Message Exchange Patterns has yet to be precisely determined.
5. The existing association between WS-CDL and WSDL is arguably too restrictive. A choreography wired to specific WSDL interfaces cannot utilize functionally equivalent services with different WSDL interfaces.
6. It is expected that WS-CDL and WS-BPEL could be complementary; nevertheless how some of the Work Units’ functionalities can be mapped to WS-BPEL remains as an open issue.
7. The explicit support for multi-party interactions and more complicated messaging constraints is missing in WS-CDL -some key requirements emerge in multi-party scenarios, including multiple instances of interactions, atomicity constraints on interactions, and partial synchronization of responses.
8. In terms of messaging quality of service, WS-CDL relies primarily on WS-Reliable-Messaging. The extent of quality of service messaging on which WS-CDL depends is not fully established, and the mapping for reliable messaging remains open. In general, no a priori configurability of WS-CDL specifications for different quality of messaging service is in place.

Concluding, WS-CDL allows the description of distributed applications in a way understandable by computers. Computers may retrieve information about data flows and communication channels, but some issues reduce its suitability for improving open environments applications 1) there is no formal definition of the language, making it difficult to generalize its use; 2) there are a number of gaps concerning to the integration of WS-CDL into the WS-Architecture, decreasing model’s flexibility; 3) it does not include the description of the application execution context, so it is no possible to keep track of applications evolution using its specification; 4) there are no explicit mechanisms for measuring the QoS; 5) there is no mechanism for negotiating the execution of activities.

2.4.2. Web Service Level Agreement Language Specification (WSLA)

WSLA is a specification developed by IBM for describing agreements between a service provider and a service customer. The specification is built upon the definition of obligations of the involved parties. Primarily, this is the obligation of a service provider to perform a service according to agreed-upon guarantees (such as availability, response time and throughput).

WSLA does not take care of the establishment of the agreement between parties but of the explicit declaration and description of functionalities for enabling their management. Parties could be supporting parties and signatory parties. Signatory parties are the
provider and consumer—two services—, so WSLA allows the specification of contracts between just two parties, supported by other services (supporting parties) that take care of measuring and computing the operations throughput through specific metrics.

The assertions of the service provider are based on a detailed definition of service parameters including the algorithms (how basic metrics should be measured in systems and how they are aggregated into composite metrics and SLA parameters). The definition of the role of third parties (such as Management Service Providers) allows the specification of complex mechanisms for metrics measurement. These multi-party constellations require the definition of the interactions among the parties supervising the WSLA and the declaration of the penalties to be applied in case of deviation and failure to meet the asserted service guarantees.

The deployment of contracted services should be supervised by third party-components or provider/customer services by themselves, using the information provided by WSLA. This is illustrated in figure 5.

The metrics are conceptualised as the definition of any service property value that is measured (1) by a measurement service provider or (2) computed using other metrics and constants. Metrics are the key instrument to describe exactly what SLAParameters mean by specifying how to measure or compute the parameter values.
The metric result values are located directly by a metric URI, or can be computed through the processing of a measurement directive (prescription for the measurer), or a function. If a metric is computed from other metrics, a metric’s description contains a function element detailing how to compute the value. If the metric is defined in the context of a metric macro, a Measurement directive variable can be defined to leave this definition to be filled in by the macro expansion using metric macro definitions / metric macro expansion declarations (these last elements are not explicitly included in the data model presented in this document). For details about the WSLA model see [Ludwig03].

The measurement directives define how parameter values are to be measured. How the measurement is conducted and which information is needed for this purpose strongly depends on the application context. This dependency implies that the complexity of the measurement directives definition increases as more different specific scenarios are considered.

WSLA's mechanism for extending the definition of measurement directives and implementing their processing for each context is based on using any XML-compliant language. The specific tag any have been defined in order to hold any extension to common measurement directive definitions.

A Service Level Objective expresses the commitment to maintain a particular state of the service in a given period of time. The state is defined as an expression on predicates that refers to the SLA Parameters defined in the service definition section of the SLA. An Action Guarantee expresses a commitment to perform a particular activity if given preconditions are met. The precondition set is defined as an expression on predicates that refers to the SLA Parameters defined in the Service Definition section. The next figure illustrates the main WSLA concepts and relationships between them [Ludwig03].
2.4.2.1. Example encoding
The XML encoding below shows the capabilities of WSLA to describe interactions between the NewsProvider and BlackBoard services defined in the use case described in section 2.4. General rules were specified previously; specific rules for encoding this using WSLA are:

- NewsProvider should be declared as the service provider, BlackBoard as the service consumer, and NewsAuditing as the web service in charge of measuring the news items quality.
- The NewsProvider service has the obligation of providing adequate news items in a defined period of time.
- The QoS of providing news items is measured by the metric called distanceAgainstInappropriateTermsMetric.
- The metric distanceAgainstInappropriateTermsMetric is measured by the NewsAuditing service.
- The NewsAuditing service has the obligation of to notify violations on the QoS of providing news items.

```xml
<?xml version="1.0"?>
<!-- Author: Félix Fernández-Peña ffernandez@lsi.upc.edu
Date: May 19, 2005.-->
<SLA xmlns="http://www.ibm.com/wsla"
xmns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ibm.com/wsla WSLA.xsd">
<!-- ... -->
</SLA>
```
Towards a Contract-based Interoperation Model

<Towards a Contract-based Interoperation Model>

<!-- Definition of the Involved Parties, the signatory parties as well as
the supporting ones -->

<!-- Definition of the Involved Parties, the signatory parties as well as
the supporting ones -->

<Parties>

<ServiceProvider name="NewsProvider">

<Contact>

</Contact>

<Action xsi:type="WSDLSOAPOperationDescriptionType"
name="notification"
partyName="NewsAuditing">

<WSDLFile>Notification.wsdl</WSDLFile>

<SOAPBindingName>SOAPNotificationBinding</SOAPBindingName>

<SOAPOperationName>Notify</SOAPOperationName>

</Action>

</ServiceProvider>

<ServiceConsumer name="Blackboard">

<Contact>

</Contact>

<Action xsi:type="WSDLSOAPOperationDescriptionType"
name="notification"
partyName="NewsAuditing">

<WSDLFile>Notification.wsdl</WSDLFile>

<SOAPBindingName>SOAPNotificationBinding</SOAPBindingName>

<SOAPOperationName>Notify</SOAPOperationName>

</Action>

</ServiceConsumer>

<SupportingParty name="NewsAuditing" role="MeasurementService">

<Contact>

</Contact>

<Action xsi:type="WSDLSOAPOperationDescriptionType"
name="requestNewsItemPublication 
partyName="BlackBoard">

<WSDLFile>newsAuditing.wsdl</WSDLFile>

<SOAPBindingName>SOAPRequestNewsItemPublication Binding</SOAPBindingName>

<SOAPOperationName>requestNewsItemPublication </SOAPOperationName>

</Action>

</SupportingParty>

</Parties>

<!-- The definition of the service in terms of the service parameters
and their measurement. -->

<ServiceDefinition name="NewsPublication">

<!-- Schedules could be used like in the sample coming with WSLA Specification, in order to do something
if the BB is overloaded -->

<!-- Schedules could be used like in the sample coming with WSLA Specification, in order to do something
if the BB is overloaded -->

<Operation name="requestNewsItemPublication 
xml:type="WSDLSOAPOperDescType">

<SLAParameter name="distanceAgainstInappropriateTerms"
Towards a Contract-based Interoperation Model

type="integer" unit="">
  <Metric> distanceAgainstInappropriateTermsMetric </Metric>
  <Communication>
    <Source> BlackBoard </Source>
    <Push> NewsAuditing </Push>
  </Communication>
  <SLAParameter>
    <Metric name="distanceAgainstInappropriateTermsMetric" type="integer" unit="">
      <Source> BlackBoard </Source>
      <MeasurementDirective xsi:type="ValueRequest"
        resultType="integer">
        <RequestURI>
          http://ymeasurement.com/ValueRequest/distanceAgainstInappropriateTerms
        </RequestURI>
        </MeasurementDirective>
      </Metric>
      </SLAParameter>

<!-- The obligations of the parties, referring to parameters defined above. -->
  <Obligations>
    <ServiceLevelObjective name="informAppropriateNewsItems">
      <Obliged> NewsProvider </Obliged>
      <Validity>
        <Start> 2001-11-30T14:00:00.000-05:00 </Start>
        <End> 2010-12-31T14:00:00.000-05:00 </End> <--Whatever … //-->
      </Validity>
      <Expression>
        <Predicate xsi:type="Greater">
          <SLAParameter> distanceAgainstInappropriateTerms </SLAParameter>
          <Value> 3 </Value>
        </Predicate>
        <EvaluationEvent> NewValue </EvaluationEvent>
      </Expression>
      <ServiceLevelObjective>
        <ActionGuarantee name="informAppropriateNewsItemsGuarantee">
          <Obliged> NewsAuditing </Obliged>
          <Expression>
            <Predicate xsi:type="Violation">
              <ServiceLevelObjective> informAppropriateNewsItems </ServiceLevelObjective>
            </Predicate>
            <EvaluationEvent> NewValue </EvaluationEvent>
          </Expression>
        </ActionGuarantee>
        <QualifiedAction>
          <Party> BlackBoard </Party>
          <Party> NewsProvider </Party>
          <Action actionName="notification" xsi:type="Notification">
            <NotificationType> Violation </NotificationType>
            <CausingGuarantee> informAppropriateNewsItemsGuarantee </CausingGuarantee>
            <SLAParameter> distanceAgainstInappropriateTerms </SLAParameter>
          </Action>
        </QualifiedAction>
        <ExecutionModality> Always </ExecutionModality>
      </ActionGuarantee>
    </ServiceLevelObjective>
  </Obligations>
</SLA>

20
2.4.2.2. Analysis of WSLA

Some characteristics to note about WSLA are:

1. Contracts have one service provider, one service consumer, and as many third parties as necessary. –no multiple consumer/provider contracts can be defined using WSLA.
2. WSLA was designed for specifying contractual obligations and metrics for evaluating the accomplishment of the obligations.
3. WSLA provides input to the measurement and management system of an organization that checks and manages organization's compliance with a WSLA.
4. WSLA complements service description languages. Service descriptions are input to the design and implementation of the service provider system and the “consumer” application using them.

WSLA defines a way for the explicit specification of **Service Level Objectives**. The specification of third-parties in WSLA is an important contribution to the explicit description of contract-based applications in open environments (advocating for using the corresponding third parties for monitoring specific actions). The definition of guarantees and metrics for measuring guarantees fulfilment is flexible, been able to specify not only typical notions of quality of service (QoS) like time restrictions but even others with complex semantic definition.

Concluding, WSLA enables the description of distributed applications and describes how to measure the QoS whilst contracts stay active. Furthermore, WSLA allows the definition of schedules of activity execution and the extension of the semantic description of the model, increasing the flexibility of the descriptions.

Nevertheless, some issues remain as limitations 1) there is no formal definition of the language, making it difficult to generalise WSLA; 2) WSLA does not include the description of the application execution context, so it is not possible to keep track of applications “evolution” using its specification; 3) there is no mechanism for the execution of activities on negotiating the contract.

2.4.3. Web Services Agreement Specification (WS-Agreement)

WS-Agreement is a specification under development. The latest draft of version 1.1 was released in May, 2004 [Andrieux04]. The general structure of agreements in the WS-Agreement specification consists of the description of the context in which the agreement is established, the service itself and guarantee terms. In WS-Agreement, guarantee terms and Service Level Objectives are not differentiated. A qualifying condition property allows a user to establish the preconditions to take into account.

The WS-Agreement specification is less focused on the description of the related activities that should be choreographed but on the definition of the commitments (how to evaluate the fulfilment, no matter how service objectives are achieved) and penalties (what to do if commitments are not satisfied). A relevant aspect in the specification of
agreements using WS-Agreement is that a WS-Agreement document starts with a clear
definition of the context in which the contractual relation takes place. That is to say that
the language demands the specification of:

- The agreement’s name.
- The agreement’s termination time: time in which its execution should fulfil.
- and the agreement’s relations with:
  - The context (related agreements and their relationship type), and
  - Agreement’s terms.

The Guarantee terms express promises and penalties. WS-Agreement is the only
specification of those under study that includes the explicit declaration of penalties. WSLA allows to penalties to be applied, but they are not explicitly differentiated against
any other type of actions. Each guarantee term consists of [Andrieux04]:

- Service scope.
- Variables {name / metric / reference}: aliases to concepts understood in the
  context of the agreement or to parts of it, used in qualifying conditions and
  service level objectives.
- Qualifying conditions.
- Service Level Objectives.
- Business value list.

WS-Agreement defines agreement templates to specify agreement types in the specific
domain of an application. An agreement template is simply an agreement (definition of
its name, termination item, context, and terms), and the specification of a set of
CreationConstraints, that defines the conditions to be fulfilled as preconditions of the
contract agreement between involved parties.

The explicit specification of the context (related agreements) is important for the
management of automatic-negotiated contracts between multiple parties in open
environments. On the other hand, the explicit declaration of time-based constrains such as
the termination time of the agreement to execute is crucial information for contract
monitoring execution.

By default, the initiator of the agreement creation request is also the consumer of the
service that the agreement is bound to, and the agreement provider is the service provider.
Every agreement created will have this role mapping, unless specified otherwise by the
template(s) or the agreement offer itself.

The next figure illustrates the main WS-Agreement concepts and relations between them.
Figure 6. Meta model of the WS-Agreement proposal.
Towards a Contract-based Interoperation Model

Once a WS-Agreement exists between the service provider and the service consumer, the contract enters the management layer, where contract monitoring takes place until the contract is discontinued. The WS-Agreement specification establishes that a contract in the management layer may be in one of three different states:

- **Not determined state.** No activity regarding contract guarantees has happened yet or is currently happening.
- **Fulfilled state.** Currently, all the guarantees specified in the contract are fulfilled.
- **Violated state.** Currently, some guarantees specified in the contract are violated.

The WS-Agreement specification states as well, that these contract states are interchangeable, meaning that any contract in a fulfilled state could enter in a violated state if violations of guarantees occur. This monitoring of the contract state, whilst managing the contract, takes place until one of the parties decides to cancel the contract.

2.4.3.1. Example encoding

The example below shows the capabilities of WS-Agreement for describing interactions between NewsProvider and BlackBoard services, in the use case described in section 2.4. General rules were specified previously; specific rules for encoding this using WSLA are:

- This agreement is related to previous agreements, and its validity is fixed in time.
- The InformNewsItem method of the web service NewsAuditing measures the distance against inappropriate terms of submitted news items.
- The value measured by the InformNewsItem method of the web service NewsAuditing is considered a metric to be stored in the variable distanceAgainstInappropriateTerm.
- A qualifying condition establishing that distanceAgainstInappropriateTerms should be greater than three is defined.
- A penalty is defined. This penalty implies that if the previous defined qualifying condition is violated, the service provider is not allowed to deliver more news items (news items are rejected if coming from this provider).

```xml
<wsag:Agreement>
  <wsag:Name> NewsItemProvBlackBoardChoreo </wsag:Name>
  <wsag:AgreementContext>
    <wsag:AgreementInitiator>newsProvider</wsag:AgreementInitiator>
    <wsag:AgreementProvider>BlackBoard</wsag:AgreementProvider>
    <wsag:AgreementInitiatorIsServiceConsumer>true</wsag:AgreementInitiatorIsServiceConsumer>
    <wsag:TerminationTime>2010-12-31T14:00:00.000-05:00</wsag:TerminationTime>
  </wsag:AgreementContext>
  <wsag:RelatedAgreements>
    <wsag:RelatedAgreement wsag:RelationshipType="wsag:dependency">
      <wsag:RelatedAgreementEPR>
      </wsag:RelatedAgreementEPR>
    </wsag:RelatedAgreement>
  </wsag:RelatedAgreements>
</wsag:Agreement>
```
Towards a Contract-based Interoperation Model

<xsd:any/>  <!--Allows us to extend the explicit information about the agreement context/-->  
<wsg:AgreementContext>

<wsg:Terms>
<wsg:ServiceDescriptionTerm wsg:Name="NewsAuditingServiceDescription" wsg:ServiceName="NewsAuditing">

<xsd:any/>

<!--This XML block could be used to express that NewsAuditing has an requestNewsItemPublication action implementation in order to send the distanceAgainstInappropriateTerms to BlackBoard and NewsProvider/-->  
<wsg:Location>

<wsg:Variable wsg:Name="distanceAgainstInappropriateTerms" wsg:Metric=""/>

<wsg:ValueExpr>

</wsag:Variable>

<!--Guarantees specify both promises and penalties-->
</wsag:Terms>
</wsag:Agreement>

25
2.4.3.2. Analysis of WS-Agreement

WS-Agreement allows us to describe contracts between two parties, assuming default roles of consumer/provider. By default, the contracting initiator is considered to be the contract’s consumer and the one offering the contract template is considered the provider. Some issues related to WS-Agreement include:

1. The specification of penalties is not as flexible as in WSLA because WS-Agreement does not allow the extension of the definition of penalties using custom languages as WSLA allows to.
2. The specification of penalties does not include the definition of proper “finalizer” operations in case of fault.
3. WS-Agreement does not include the specification of third parties working in the management of contracts (Working with third parties is a necessity in specific environments, especially when existing distributed systems are going to be extended).
4. WS-Agreement does not include the declaration of schedules for the execution of the activities of the web services.
5. No metrics are defined in order to support flexible monitoring implementations over the web services choreographies.

Concluding, WS-Agreement allows the description of distributed applications in a way understandable by computers. WS-Agreement meta-model is focused in the description of the guarantee terms and the definition of how to negotiate agreements over contract rules, within the boundaries a well defined context. It is true as well, that the WSLA mechanism for measuring the quality of service using metrics has been integrated in the WS-Agreement proposal. The resulting model is then supposed to make possible the automatic negotiation of services and the management of distributed applications using contract specifications for making able computers to “understand” data flows and rules to be applied to the inter-process communication.

Nevertheless, some issues, related to the absence of any formal definition of the language, remain as limitations, and these are: 1) it is difficult to know how to describe domain-specific terms and conditions 2) WS-Agreement does not allow the description of multi-party contractual relationships, as well as the roles played by third-parties.

2.5. Automatic Negotiation of Contracts

The automatic negotiation of contracts constitutes a revolutionary step forward in the development of distributed applications in open environments. In the field of web services, there is just only one proposal for describing how to negotiate the agreement between different parties. This approach is the Web Service Agreement Negotiation (WSAN), an upper layer, created over the WS-Agreement specification, as introduced by Andrieux et al [Andrieux05].

Nevertheless, it is important to note that it has been argued that many other protocols can be mapped to the web services worlds. In this way, the results of the web service
research community for negotiation were analysed and compared with other proposals. This is the case of the *contract-net proposal*, developed by the agent research community.

### 2.5.1. WSAN proposal

The next figure shows the *negotiation* and *renegotiation* layer, proposed by WSAN. Three separated but related layers constitute the proposed basis for an agreement-based system. This separation into layers allows using the basic communication layer for a specific application domain, the management layer when needed, and the negotiation layer, over the management layer for autonomic system approaches.

![Diagram](image)

**Figure 7. WS-Agreement Negotiation and Renegotiation Architecture** [Andrieux05]

WSAN specifies the following port types [Andrieux05]:

- A negotiation port type exposes a negotiate operation that the initiator can call in order to negotiate the related agreement. Eventually negotiation leads to the agreement been created (i.e. both parties commit to it). The operation can then be
called again in order to renegotiate the agreement (if permitted by the web service interface implementation).

- A negotiation factory exposes a negotiation service creation operation which takes an agreement endpoint reference (EPR) as a parameter. The operation creates a negotiation service related to that specific agreement. The only way for the initiator to modify the agreement is through the negotiation protocol exposed by the negotiation port type.

The overall negotiation process has the following structure [Andrieux05]:

1. The initiator calls the createAgreementAndNegotiation operation on the specialized Negotiation Factory service.
2. The Factory service returns an EPR to an Agreement and an EPR to a Negotiation.
3. The initiator calls the negotiate operation on the Negotiation service in order to change the current state of the agreement: the terms being negotiated or the commitment status. The Negotiation service either rejects the offer using a non-terminating fault or accepts the offer and updates the state of the Agreement.
4. Step 3 is repeated until one party decides to stop negotiation or both parties commit to the current offer. For example the Negotiation service can send a terminal fault, indicating unwillingness to accept any further message.
5. Eventually both parties commit to an offer and the agreement becomes observed.

In the case of renegotiation of an existing agreement, it is specified that there SHOULD be an additional operation in the specialized Negotiation Factory port type for the initiator to obtain the EPR to a Negotiation service in case it lost the Negotiation EPR obtained when requesting the creation of the agreement (agreement negotiation use case) or if it never requested a Negotiation in the first place (simple agreement creation use case – see figure 8). Therefore, the specialized Factory MAY choose to compose an equivalent of the wsan:createNegotiation operation defined in the wsan:NegotiationFactory.

Process:

1. The steps in the simple agreement creation use case or the steps in the negotiation use case are used here.
2. The initiator calls the createNegotiation operation on the Factory in order to retrieve the EPR to a Negotiation service related to the agreement which EPR was supplied to the operation.
3. Negotiation iterative process: Steps 3-5 of the negotiation use case are performed.

---

1 createAgreementAndNegotiation refers to an specific port of the negotiation service, defined by the specification [Andrieux05].
Towards a Contract-based Interoperation Model

WSAG was designed to be used for managing not only WSs but WSs and any other kind of service provider in a computational environment. So, from bottom to top, in the service model proposed by Andrieux et al., the first layer represents the functional capabilities of the service, no matter what type of service it is.

2.5.2. Agent Research on Negotiation Protocols

FIPA Contract Net Interaction is the proposal from the *Foundations of Intelligent Physical Agents* (FIPA) [FIPA05a]. In FIPA contract net interaction, the initiator is an agent that takes the role of the manager who wishes to have some task performed by one or more other agents (the participants), and further wishes to optimise a function that characterises the task. This characteristic is commonly expressed as its price. The representation of this interaction protocol is given in figure 9, based on extensions to UML. This FIPA protocol is identified by the token fipa-contract-net as the value of the protocol parameter of the ACL message [FIPA05a].
Contract Net protocol has the advantage of been well known and formed by standard messages with an explicit semantic. Table 1 summarizes the description of the composing parts of the FIPA performatives used in the communication.

It is important to note that these performatives do not describe how to negotiate contracts but how to negotiate activities between parties. Nevertheless, if the activity to be negotiated is the “negotiation of a contract”, then FIPA Contract Net Protocol could be seen as a solution for the automatic contract negotiation.

By comparing the WS-Agreement Negotiation proposal and FIPA Contract Net Interaction Protocol (represented in figures 8 and 9, respectively), the authors of this paper concluded that the implementation of both approaches are different but the negotiation algorithm is the same.
Towards a Contract-based Interoperation Model

<table>
<thead>
<tr>
<th>Performatives.</th>
<th>Parameters.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cfp(^2).</td>
<td>-action.</td>
</tr>
<tr>
<td></td>
<td>-referential expression defining a single-parameter proposition which gives the preconditions of the action.</td>
</tr>
<tr>
<td>Refuse.</td>
<td>-action refused.</td>
</tr>
<tr>
<td></td>
<td>-reason for refusing.</td>
</tr>
<tr>
<td>Propose… Responding to cfp.</td>
<td>-action.</td>
</tr>
<tr>
<td></td>
<td>-expression. How to fulfil preconditions.</td>
</tr>
<tr>
<td>Reject-proposal.</td>
<td>-action.</td>
</tr>
<tr>
<td></td>
<td>-expression. Condition expression proposed.</td>
</tr>
<tr>
<td></td>
<td>-expression. Reason for rejecting.</td>
</tr>
<tr>
<td>Accept-proposal.</td>
<td>-action.</td>
</tr>
<tr>
<td></td>
<td>-proposition explicitly declaring the condition expression conforming the agreement.</td>
</tr>
<tr>
<td>Failure.</td>
<td>-action.</td>
</tr>
<tr>
<td></td>
<td>-proposition giving the reason for the failure.</td>
</tr>
<tr>
<td>Inform. This could be:</td>
<td>-Proposition</td>
</tr>
<tr>
<td>Inform-done.</td>
<td></td>
</tr>
<tr>
<td>Inform-result.</td>
<td></td>
</tr>
</tbody>
</table>

Table 1. FIPA Contract Performatives.

Whilst WS-Agreement Negotiation approach proposes that messages are sent against multiple web service endpoints, each of them with specific functionalities, FIPA Contract Net Interaction Protocol proposes to use a unique communication channel (just only one endpoint\(^3\)), and to use performatives to describe the type of action intended in each step of the negotiation procedure.

Considering applying the FIPA Contract net protocol to the contracts negotiation in web services-driven distributed applications implies that:

1. The action of the \(\text{cfp}\) (see table 1) would be “to propose a contract specification”, meaning that the initiator calls for proposals for signing a contract based in specific preconditions. These preconditions are specified in a contract template. A reference to this contract template would be included as a Cfp parameter.

2. If any participant does not accept the preconditions a refuse response is sent back to the initiator, otherwise, a propose response is generated. This response does correspond to the action of “accepting to sign the contract”, and includes the conditions over the contract template. These conditions should be accepted by the initiator in order to sign the contract.

---

\(^2\) cfp means Call For Proposal.

\(^3\) Considering web services jargon.
3. If the initiator accepts the conditions presented by the participant, an accept-proposal is sent, otherwise a reject-proposal is posted.

4. Contract signing is confirmed using the failure, inform-done and inform-result performatives.
   a) If there are errors in the communication a failure message is sent back.
   b) If the contract specification is well formed and valid an inform-done is sent to the initiator (the activity related to the contract signing was successful).
   c) If there is additional information about the activity related to the contract agreement, this information is sent back using the inform-result performatives.

2.6. Comparison between proposals analysed

The authors of this report consider that the idea of contract-based software development, proposed by Meyer [Meyer97] for object oriented programming, is valid as well for designing distributed applications. Nevertheless, distributed applications in open environments require even more complex models for facilitating the monitoring and the enforcement of contract agreement rules, the management of these contracts and the automatic negotiation of services provided and consumed by different parties. The main proposals in the field of web services have been analysed. The next table shows how much they cover needs of the web service development community.

<table>
<thead>
<tr>
<th>Aspects</th>
<th>WS-CDL</th>
<th>WS-SLA</th>
<th>WS-Agreement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi-party contracts</td>
<td>Covered</td>
<td>Partially covered</td>
<td>Covered</td>
</tr>
<tr>
<td>Negotiation mechanism</td>
<td></td>
<td>Partially covered</td>
<td>Covered</td>
</tr>
<tr>
<td>Includes the explicit declaration of the contract’s context.</td>
<td>Partially covered</td>
<td>Covered</td>
<td>Partially covered</td>
</tr>
<tr>
<td>Metrics specification</td>
<td></td>
<td>Covered</td>
<td></td>
</tr>
<tr>
<td>Compatible with WS-Architecture</td>
<td>Intended</td>
<td>Not intended</td>
<td>Not intended</td>
</tr>
<tr>
<td>Choreography-oriented</td>
<td>Covered</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Management-oriented</td>
<td></td>
<td>Covered</td>
<td>Covered</td>
</tr>
<tr>
<td>Negotiation-oriented</td>
<td></td>
<td></td>
<td>Covered</td>
</tr>
<tr>
<td>Formal definition of the model</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2. Extract of the properties of analysed proposals for managing and negotiating contracts

WS-CDL focuses on the description of activities integration in an open environment. Neither WS-CDL, nor WSLA intend to solve the contracting establishment issues (agreement templates are mentioned as something beyond the scope of WSLA specification), but WSLA allows to include a more detailed description of the components conforming an agreement (WSLA definitions of supporting parties and their interactions in the contracts through metrics are an interesting proposal, even when further work is necessary for the truly functional definition of metrics in automatic managed systems).
On the other hand, the WS-Agreement protocol for negotiation is well founded and WSLA is been integrated to the WS-Agreement specification. It is important to note that in order to specify the “conversation” between Web Services, Web Services Conversation Language (WS-CL) exists as an specification of W3C [Banerji02], and that it has been proposed as an element to integrate into WS-Agreement [Paurobally05a].

There is no proposal for the choreography, negotiation and management of multi-party services. Nevertheless, as the specifications under study in this research cover part of the domain of specification required, and all of them are XML-based, it is possible to generate a common model by integrating these proposals’ meta-models. In the next section, a first attempt to do so is introduced.
3. General Model Proposal

3.1 Introduction

The role of the agreements is twofold: they stipulate obligations and expectations of the involved parties and they represent the goals to be met by the infrastructure. As a consequence of this latter point, in order to automate run-time adaptation and management of systems and services, agreements should be encoded and integrated in management software platforms [Molina05].

In the previous section, three different agreement encoding proposals have been discussed. In the next subsections, a general model for the negotiation, deployment, monitoring and enforcement of contracts is presented. This model is based on the identified needs (described at the beginning of section 2) and defined taking into account the specifications previously analysed.

3.2. General Model Foundations

The authors of this research consider that a contract-based model should:

1. Provide a clear identification of those actors, roles and components involved in the processes associated to different levels of implementation of a contract-based framework for managing distributed applications in open environments.
2. Provide a clear definition of a contract specification (the content of contract specification).
3. Define the different contract states, through the life cycle of a contract.
4. Model the interaction schema of parties involved in specific processes.

The contract-based framework should be, depending on specific application requirements, capable of:

- Negotiating and agreeing to contracts, describing their expected behaviour based on assertions (preconditions / postconditions / invariants / rewards).
- Controlling involved parties (service providers, service consumers and third parties).
- Carrying out / managing the resulting services been requested / provided in the agreed contract, based on:
  - Metrics for measuring the state of web services execution.
  - Penalties to be applied over parties that do not truly behave as expected.
  - “Finalizer” operations, executed once web services get successful results.

The common overview of negotiating contracts, taking into account the similarities of FIPA Contract Net proposal and WSNA proposal were considered for defining a general contract specification for web service-based distributed applications to be inserted into
Towards a Contract-based Interoperation Model

the Web Services Architecture Stack. The location for this proposal into the Web Services Architecture Stack is illustrated in figure 10.

![Figure 10. Web Services Architecture Stack, including Contract Specification Language (based on [Booth04])](image)

The contracting layer in the abstract level of the model is defined using a contract specification for describing the contract agreement and defining performatives (adapting speech acts from the agent community) for the communication between web services (conversation).

Keller and Ludwig presented the general architecture for the WSLA [Keller03]. This approach resembles the management of the activities execution (see figure 11). Nevertheless, by comparing the different analysed proposals, it is concluded that it is important to consider other three main aspects not explicitly included in [Keller03]:

1. Describing the contract’s context.
2. Controlling multi-party scenarios.
3. Considering the renegotiation processes.
Towards a Contract-based Interoperation Model

Figure 11. General Architecture of Contract-based Web Service Applications [Keller03]

Considering the meta models of each of the third XML-based proposals analysed in the previous section, the resulting meta model considers the merging of WSLA specification into WS-Agreement with slight adjustments and additions in the process. This adjustments include:

- Controlling more detailed information about contracts: author, version, targetNameSpace (as proposed by WS-CDL) and semantic domain-specific information. Explicit semantic information would be an optional extension of the contract specification (the definition of an ontology would work as a thesauri for matching enhancing descriptive capabilities of expression conforming preconditions for the negotiation and renegotiation of contracts.
- To consider that an agreement could be or not be a template agreement, and that an agreement is related to other agreements.
- To consider the specification of Web Service Level Objectives and Guarantees, and the specification of metrics for their evaluation, as proposed by WSLA.
- To define communication performatives in a precisely way, in order to allow the definition of multiple particular mechanisms for negotiation / renegotiation / execution of contracts using the general specification as foundation for the processes integration in the distributed environment.
Towards a Contract-based Interoperation Model

- To allow the renegotiation of contracts. The renegotiation of a contract is requested by any of the consumer or service provider parties involved in the contract-based distributed application.
- To enable the specification of multi-party contracts.

3.3. General Model Views

This model proposal is based on four views of the contract definition for interoperation between multiple parties in open environments. These views are:

1. **Actors and Components View**, describing the roles of the interacting parties.
2. **Contract View**, which instantiates the corresponding structure of the contract.
3. **Communication View** that structures the protocol to be used between parties and how to process messages in the architecture stack.
4. **Contract States (life cycle) View**, illustrating the different possible states of contracts content for each state of the application.

3.3.1. Actors and Components View.

The **Actors and Components View** focuses on the parties that interact in the life cycle of a contract and the resources exchanged among them. This view is shown in the next figure (figure 12).
NOTARY
A notary is a third party web service responsible of the management of contracts. A notary keeps track of (logs) the life cycle of any contract, asks to appropriate third parties to evaluate contract execution, and interacts with “Arbitrator” parties in order to log the penalties imposed to the signatory parties.

ESTIMATOR
A third party specialised in measuring and/or evaluating one or more metrics against one or more services provided by signatory parties.

ARBITRATOR
A third party specialised in playing the role of jury, deciding what to do in correspondence with the result of evaluating service execution.

SIGNATORY PARTY
A signatory party is any web service agreeing to provide / consume some service, as prescribed in the corresponding contract.

SERVICE CONSUMER
A service consumer is a web service agreeing to consume some service(s), as prescribed in the corresponding contract.

SERVICE PROVIDER
A service consumer is a web service agreeing to provide some service(s), as prescribed in the corresponding contract.

CONTRACT
A contract is a statement of intent that regulates behaviour among organizations and individuals [Morciniec01]. Contracts formally specify the behaviour that each contractual party is expected to follow in an ideal world [Dignum04] (the contract view details contract’s structure in the current proposal).

CONTRACT SET
A contract set is the collection of agreed contracts between multiple parties working together in an specific domain.

CONTRACT AGREEMENT
A contract agreement is the “document” that contains the specification of the contract, once there is an agreement between all parties to accept its specification.

CONTRACT AGREEMENT TEMPLATE
A contract agreement template is a contract that is used as starting point to specify other contracts.

CONTRACT AGREEMENT TEMPLATE SET
A contract template set is the collection of contract agreement templates that exists in a specific domain.

ACCOMPLISHMENT EVALUATION
The accomplishment evaluation is the third parties’ report about the evaluation of corresponding metrics, associated to the execution of activities circumscribed to the contract’s scope. The accomplishment evaluation means to determine, by measuring pertinent metrics, the adequacy of activities carried out by the signatory parties.

RELATIONSHIPS
Signatory parties propose GUARANTEES, PENALTIES, and REWARDS, to be included in a contract. Those proposals constrain the contract reach in its specific domain, and they include references to third parties for evaluating the execution of the corresponding actions. Once, there is an agreement between parties, a CONTRACT AGREEMENT is then created. This CONTRACT AGREEMENT MAY extend other CONTRACT AGREEMENTs, that all together function like its foundation. The specification of an agreed contract and its relation with other previous contracts describes the CONTEXT of the distributed application under specification.

---

4 The specification of a contract is the context of this article should be interpreted as the specification of a contract agreement.
Hereafter, the NOTARY keeps watching service execution until CONTRACT EXECUTION is declared as finished. Whenever, an action takes place, the SIGNATORY parties can request for a proper EXECUTION’S EVALUATION. Corresponding THIRD PARTY-ies provide their EXECUTION’S EVALUATION.

If EXECUTION’S EVALUATION is not successful, corresponding penalties are applied over the corresponding parties. After that, the ARBITRATOR informs about the restrictions applied to the SIGNATORY PARTY-ies. SIGNATORY PARTY-ies role is constrained by the EXECUTION’S EVALUATION.

3.3.2. Contract View
The Contract View describes the elements comprising a contract (see figure 13). There is no consensus on the necessary components of a contract. The resulting view, introduced here, is the result of the analysis of existing proposals and identified requirements.

![Figure 13. Contract View](image)

**CONTEXT**
Each contract is created in a specific context. The context characterises the environment in which the contract is created (contract’s name, contract relationships with other contract specifications, among other non-functional characteristics).

**PARTIES**
A contract gets to the active status when specific parties agree to the contract conditions (guarantees, penalties, rewards). These “parties” accept the contract and its implications.

**GUARANTEES**
Guarantees are logical expressions describing the rules that apply when proper enactment of the provider’s services is achieved (these rules are logical expressions that describe the benefits of the contract fulfilment).

**PENALTIES**
Penalties are statements of restrictions to be applied to some party/parties. Penalties are related to faults. When faults rise, pertinent penalties are applied, so the system functioning is updated.
REWARDS
Rewards are statements of gain that a service consumer receives by applying to a specific contract.

3.3.3. Contract States View
Meanwhile, the Contract States View is equivalent to the definition of the life cycle of contracts in the model. This is illustrated in the next figure.

![Diagram of Contract States View]

Figure 14. States of the Contract View\(^5\)

UNDER-NEGOTIATION
In this state, the different parties interact in order to propose conditions that refine the final contract (to be signed by signatory parties\(^6\)). These iterations continue until an agreement between the parties involved is reached.

SIGNED
In this state, the contract has been accepted by all the parties involved but no event has raised any operation (neither executing activities, nor monitoring activities has been requested to operate) upon the accepted conditions of the contract.

DEPLOYED
In this state, the activities execute under the specification of the contract. Each event captured by any party implies the execution of an activity (a simple or composite activity

\(^5\) Note that the messages used in the actual execution of the contract may not follow this pattern.

\(^6\) Signatory party is a term previously defined in the Actors and Components View.
including internal monitoring activities to verify the satisfaction of the contract functional and non functional conditions).

**MONITORED**
The *notary* requests the evaluation of QoS by the corresponding *third party(ies)*. Depending on the results of evaluating the contract execution, the *arbitrator* applies penalties to the *signatory parties*, and informs them of their actual conditions under the execution of the contract. In this state any party may request the re-negotiation of contract conditions or even to finalize the contract (note that it is possible for one service to implement the roles of *notary, evaluator* and *arbitrator*, as well, altogether).

**CONTRACT ENFORCEMENT**
*Deployed* and *monitored* are both contract states that characterise the enforcement of a contract. Any contract may go through these states in a sequential cycle and even it can be in both states at the same time. These states are reached in parallel when any enforced contract gets into the *deployed* state (because any party requests the execution of an activity upon the rules specified in that contract), and in the monitored state (because some monitoring activities are taking place) at the same time.

**FINISHED**
This state is reached when some party requests to cancel the contract or deadline is reached. After that, a last evaluation of contract conditions is carried out in order to penalize corresponding parties, if penalties apply. Then, the contract is stored for future referencing; a contract could be used whilst been in the finished state 1) as a contract template for the definition of other contracts, and 2) for characterising operation on the domain of application in a determined period of time (evaluating changing conditions in the institution and its adaptation to these changes over time).

**RENEGOTIATION**
This state is reached if any signatory party request to re-negotiate the contract. Reaching this state implies that the contract is finalised. After that, the contract specification is taken as the template for getting into an UNDER-NEGOTIATION state.

The proposed overall sequence for the creation of a contract specification between multiple parties is described in the communication view.

“As more semantically rich languages are used to describe the mechanics of interaction, more of the essential information may migrate from the informal semantics to the service description. As this migration occurs, more of the work required to achieve successful interaction can be automated” [Booth04].
3.3.4. Communication View

The Communication View represents how it is proposed to structure the contract manipulation in the processes layer\(^7\) over the web services stack. This view focuses on structuring the way in which messages are sent forth and back through the life cycle of a contract (see figure 15).

![Figure 15. Communication View](image)

**Figure 15. Communication View**
* Metric measures and metric reference values.
** Resulting service execution data.

As the processes layer is located on the top of the Web Services Architecture Stack, the communication protocol specification requires a higher level formalism like [FIPA05b] or more specifically the FIPA ACL language adaptation to web services environments proposed in [Willmott04]. Nevertheless, this paper is not focused on technology decisions but on the model specification in a higher abstract level. This is why the communication view only restricts the messaging schema: an asynchronous interaction mechanisms are to be used in all the different contract states defined in this model.

PARTY.
Any party associated with any actor role identified in the actors and components view. Depending on the actual role of the party some of the asynchronous messages in the diagram are invalid (for instance, an estimator\(^8\) does not participate in the negotiation process).

CONTRACT MANAGEMENT SYSTEM.

---

\(^7\) This proposal locates the contract manipulation in the processes layer of the web services stack (see section 3.2).

\(^8\) Defined in the actors and components view (section 3.3.1).
Towards a Contract-based Interoperation Model

The contract management system instantiates a technology implementation of the model proposed in here, considering the necessary actors (as depicted in the actors and components view), the corresponding contract specification (upon a specific contract language) and the implementation of the pertinent processes, as described below.

NEGOTIATION.
This process depicts the activities involved within the under-negotiation and signed states. These activities include proposing conditions for a contract specification, accepting conditions, and the acknowledgement of a contract relationship with other parties.

DEPLOYMENT.
This process is characterised by the execution of activities for satisfying contracted services.

MONITORING.
The monitoring process comprehends activities for monitoring contract enactment and activities for applying corresponding penalties until contract finalisation.

In the next figure, processes are related to contract components in the contract context. These processes are bound to the previously identified contract states (see contract states view), and performed by the corresponding actors (according to the actors type defined in the actors and components view).
Towards a Contract-based Interoperation Model

**Processes**
The processes were previously defined.

**Contract Context**
The contract context refers to the set of elements that describe the specification of a contract in the environment in which the communication takes place.

As interoperation is the main aim in this context, it is important to explicitly define the content of the messages to be sent back and forth considering the asynchronous schema of communication. Each message includes:

1. a type/performative, expressing the attitude towards the content of the message (the semantic meaning of the intention of sending the message should be well defined),
2. metadata which supports semantic descriptions of the data using ontologies and,
3. the content of the message itself.

![Figure 16. Contract content dataflow related to the execution of contract processes](image)
Messages are classify taking into account the processes to which they are involved. This classification includes:

1. Negotiation messages.
2. Deployment messages.
3. Monitoring messages.

### 3.4. Semantic Extension of the Contract Definition

This model proposes the explicit definition of the semantic of the contract specification. This extension includes the definition of a domain ontology (a business ontology) for creating a shared vocabulary to be instantiated in the specific domain of application. More specifically, this ontology describes:

- Semantically well defined possible roles in the application domain.
- Semantic description of the operations and message types.
- Semantic description of the metric definitions.
- Semantic description of the function definitions.

Once, this ontology is fully instantiated, a complete semantic description of the environment of the application exists, increasing the contract-based application flexibility.

The rules expressing the obligations of the parties will be related to the *business ontology*, as well, bringing a better understanding of the *significance* of these rules. The specification of the metrics uses the semantic description of the business included in the *business ontology*. Their semantic description supports a refined description of:

- How to measure / aggregate the metrics.
- Which party is in charge of measuring / aggregating each metric.
- How metrics can be retrieved.

### 3.5 Benefits of the proposed model

The proposed model is seen by its authors as complementary to the architecture proposed by Keller et al [Keller02] [Keller03], and the contract negotiation protocol proposed by [Andrieux04]. The four model views presented in this paper complements the Keller model because:

1. The actors and components view identifies the necessary roles in a contract-based application, isolating the components that conform to the processes described in [Keller02].
2. The contract view isolates each party's role when parties conform to a contract-based framework for interoperating in open distributed environments.
3. The contract states view identifies the states of a contract through its life cycle evolution, taking into account the research on social commitment (see section
2.3.3) and the description of contract definition, negotiation, deployment, monitoring and enforcement in the consulted bibliography.

4. The communication view describes the inter-process flow, complementing the contract deployment process presented by Keller in [Keller02] [Keller03].

Furthermore, there are several characteristics that differentiate the proposed model against other proposals:

1. This model does not restrict the definition of a contract to relations among two parties like [Keller03] [Andrieux04] [Aiello05].
2. Messages are classified taking into account the processes to which they are associated. This classification allows for matchmaking between different applications through the identification of general patterns of similarities.
3. Using a domain ontology is proposed in order to increase the manageability of context conditions over a shared vocabulary about the semantic description of the information processes and information resources available in the interoperating environment.
4. Conclusions

Contract-based systems are been considered as the key for automatic management of dynamic distributed systems. Many approaches have been proposed but there is a lack of integration between the different research field results. Two different levels for developing contract-based web service applications have been identified with this research. (1) Using contract specifications as the global outlook of workflows composing the whole system, and (2) the dynamic improvement of services, based on the automatic negotiation / renegotiation of contracts between parties.

Current research proposes a general contract-based model for the development of web services-based applications in truly open distributed environment. This model was defined taking into account research results in the fields of grid computing, multi-agent environments, and web services.

The most relevant aspect of making this proposal is not its definition by itself but the proposal of integrating the most important characteristics of each of the related works that were analysed, considering the matchmaking done between the different points of view of the different research communities. Refining the solution, in order to gain truly dynamic interoperation in open environments, some adjustments were proposed. These adjustments include 1) the categorization of message types in order to facilitate the message processing and 2) the definition of a business ontology for improving the explicit semantic description of the application domain.

Since, this is an abstract model, there are two fundamental aspects to work to be carried out in the future work:

- Applying this general model to the environment of a particular application (a prototype already exists for working in the area of knowledge management in small to medium-sized business and institutions).
- Matchmaking this model and the particular application under development to the middleware technology support, spreading the spectrum of technologies to be used in the development of contract-based applications.

The work carried out here provides input for the European Commission funded project IST CONTRACT (http://www.ist-contract.org/).
5. References


Towards a Contract-based Interoperation Model

Descriptions”. MIT Sloan Working Paper No. 4188-01
[OWLS05] OWL-S Project. “OWL-S 1.0 Release”. http://www.daml.org/services/owl-s/1.0/ (Consulted: 06/09/05).
[Sun05a] Sun Microsystems, Inc. “Sun ONE Integration Server” http://docs.sun.com/source/816-6462-10/overview.html (Consulted: 11/12/05)
[Sun05b] Sun Microsystems, Inc. “Sun ONE Integration Server, editing ebXML Agreements” http://docs.sun.com/source/816-6462-10/agreement_editor.html