A Conversation-oriented language for B2B integration based on Semantic Web Services

Juan Miguel Gomez
Digital Enterprise Research Institute (DERI)
National University of Ireland, Galway
Galway, Ireland
juan.gomez@deri.org

Armin Haller
Digital Enterprise Research Institute (DERI)
National University of Ireland, Galway
Galway, Ireland
armin.haller@deri.org

Christoph Bussler
Digital Enterprise Research Institute (DERI)
National University of Ireland, Galway
Galway, Ireland
chris.bussler@deri.org

ABSTRACT

Establishing conversations in a B2B environment has significantly eased since the advent of standards such as RosettaNet and ebXML. These standardisation efforts have maintained some flexibility in defining interactions among business partners to allow companies with different internal processes to comply with them. However, the standards are syntactic, rather than semantic. Constraints on the interactions are currently represented informally, if at all. If two business partners want to communicate they still have to find out if the overall constraint set is fulfilled by the respective business partner. To define this global interaction in the context of a conversation we introduce a formally described language, called L3. This language grounds on well-defined execution semantics and incorporates a history of the conversation. In our approach, we show how the language allows business partners to define their constraints on the interaction from a global perspective, how certain properties of the conversation may be formally checked, such as deadlock-freedom and how it builds on an architecture based on Semantic Web Services.

Keywords

Semantic Web Services, B2B conversation, Semantic Web

1. INTRODUCTION

The Web is changing from a mere repository of information to a new vehicle for business transactions and information exchange. Large organisations are increasingly relying on Web Services technologies for large-scale software development and sharing or exposing of services within an organisation. Web Services are loosely coupled software components published, located and invoked across the web. The growing number of Web Services available on the Web raises a new and challenging problem, the interaction among heterogeneous services. Precisely in B2B [5] scenarios, this becomes increasingly important since, if two business partners want to communicate they have to find out which is the role they are playing, what is the format of the messages exchanged, and what is more important which are the temporal and order constraints for the sequencing of the messages.

The Semantic Web is about adding machine-understandable and machine-processable metadata to Web resources through its key-enabling technology: ontologies. Ontologies [9] are a formal, explicit and shared specification of a conceptualisation [32]. The breakthrough of adding semantics to Web Services leads to the Semantic Web Services paradigm. Several Semantic Web Service initiatives such as OWL-S [33], which offers semantic markup based on OWL [8] or the newly established WSMO [27] have recently gained momentum. WSMO builds on a clearly defined conceptual model based on goals, mediators, Web Services and ontologies.

However, in the B2B context the problem of describing interactions remains since there are no specific semantics to describe a conversation among several Web Services. Current technology prevents the ideal situation in which a B2B conversation can be specified in a formal way, where desired properties may be checked in advance, fulfilling a set of basic B2B requirements and having a provided language and architecture for such interaction.

In this paper, we present a novel approach to address those challenges and solve the interaction problem of Semantic Web Services with formal semantics. Our contribution is an overall solution based on a complete language, called L3. L3 has a conceptual model, a syntax, formal execution semantics, an architecture and implementation. The unambiguous execution semantics of L3 based on Petri Nets [25] allow to check deadlock-freedom and human-understandable interpretation. In order to thrust the language design, we follow a simple but effective methodology. Firstly, we introduce a simple B2B scenario. We extract requirements for our language (Section 2.2), comparing them with existing standards (Section 3) and current technology. Finally, we show how our L3 language (Section 4) solves the interaction problem. Conclusions and related Work are discussed in Section 5.

2. REQUIREMENTS FOR A B2B CONVERSATIONAL LANGUAGE

2.1 A simple B2B scenario

In the following a simple B2B [5] scenario is introduced. This scenario is based on a real-world problem and it is useful to extract a set of requirements for modelling B2B interactions.

In our scenario, a customer sends a request for quotes (RFQ) to a supplier. The supplier answers with the quote to the customer. If the result is satisfactory enough for its business logic, the customer proceeds to order the goods. The supplier then verifies with its warehouse if it is possible to provide the goods to the customer by checking the availability of the goods. Subsequently, the customer order is confirmed and the payment is made. Once the payment has finished, the supplier forwards the order to the warehouse and the customer gets all the shipment details. Alternatively, it could be the case that the customer payment is made after delivery, but we will consider only the former possibility. Finally, the customer confirms the delivery of the shipment. Once the warehouse has informed the supplier about this acknowledgement of delivery, the business process is concluded. The scenario is shown in Figure 2 by means of a UML sequence diagram [2]. In this figure, the emphasis is set on the order and sequence of the message exchange.

We will describe in detail each interaction and consider the requirements that would be necessary to model the conversation properly. Firstly, it is quite clear that we need a partner view (*Requirement 1: Partner view*) because we always have a sender and a receiver of the interaction. This partner view must clearly define the role each participant is playing in the conversation. The sequence in plain text is as follows.

1. Customer sends a RFQ message to the supplier.
2. Supplier replies with a quote. It may be the case that the supplier wants to send multiple quotes and at the same time i.e. in a concurrent way (*Requirement 2: Concurrency*)
3. Customer sends a message for ordering goods to the supplier.
4. Supplier sends a message to the warehouse to check if the goods are available and the shipment is possible.
5. Warehouse sends a message indicating the availability of the goods to the supplier.
6. Supplier sends a message to the customer confirming his order. The supplier may want to trace back the whole conversation. Therefore, the history of the conversation i.e. the explicit representation of how the conversation evolved through time must be stored. (*Requirement 3: History*)
7. Customer sends a message to the supplier paying the order.
8. Supplier sends a message to the warehouse ordering the shipment. The warehouse has already sent a message to the customer directly informing about the delivery of the goods. The supplier is then waiting for a confirmation from the warehouse and the warehouse is waiting for a message from the customer. This leads to a deadlock situation. A deadlock occurs when the participants of the conversation have reached a state from which it is impossible for any of them to proceed. It is advisable to be able to check deadlock-freedom and avoid by any means deadlock situations. (*Requirement 4: Deadlock-freedom*)
9. Warehouse sends a message to the customer informing about the delivery of the goods.
10. Customer sends a message to the supplier confirming the delivery has happened.
11. The warehouse sends a message to the supplier confirming the delivery was successful.

Finally, the correct modeling of the conversation depends on finding a suitable to check the consistency and coherence of the model. If the specification of the conversation is written in a logical language, it is feasible to derive consequences out of the specification. Using inference, properties of the specification that were not explicitly stated can be proven. In this way, invisible behaviour and properties of the system can be predicted and tested without having to be implemented. For this formal execution semantics i.e. a formal method to model execution semantics are needed (*Requirement 5: Formal execution semantics*)

2.2 Requirements for modeling a B2B interaction

2.2.1 Requirement 1: Partner view

A partner is considered one of the entities involved i.e. a participant of the interaction. A partner plays a specific role in a conversation i.e. it can be the Sender or the Receiver of a specific interaction. This distinction is made clear in the design of communication protocols where the perspective of the behaviour modelled becomes an important issue. As described in [18], the Sender is the initiator of an interaction and the Receiver is the passive subject of that communication. This relationship is symmetric in the sense that the Receiver may turn into Sender. For our approach, this means that two partner have to agree which role they are playing in the interaction and, as explained in section 3, agree with the specification of such interaction.

2.2.2 Requirement 2: Concurrency

The reasons for concurrent execution in computing systems are twofold. In a distributed system, concurrent execution is caused by the fact that several systems are active. They evolve independently, communicate with each other in order to exchange date or synchronise. Hence, concurrency is inherited to distributed systems and can not be avoided.

Centralised systems can use concurrency for their benefit. For example, event-ready systems such as industrial surveillance, real-time or simulation systems which must handle sporadic incoming events, such as alerts or user generated. In these cases, designing or solving the problem with a concurrent approach is intuitive and simple.

In our particular case, concurrency appears when several messages have to be sent at the same time. From a functional viewpoint, this means that several processes or software programs are operating at the same time.

Modern operating systems describe two forms of concurrency. Heavyweight concurrency defines processes as programs that usually execute in separate address spaces on a computer system. They can execute concurrently and the processing power of the system is assigned to the processes following different scheduling policies and priorities. Lightweight concurrency defines threads as entities inside a single process which make concurrency possible. Here, the processing power available is split up among threads. Both forms are appropriate to implement concurrency.

2.2.3 Requirement 3: History

In B2B [5], we found that a history is an explicit representation of past sate changes of the objects in a system. Those changes may be related to the time, date and user who caused it. When an error of inconsistency occurs in a system, the history that led to the failure point becomes of immediate interest. History is somehow a subcomponent of monitoring. A history query for one object returns the list of all state changes of this object. For example, the history of an event may retrieve a list of all the state changes that the event went through.

There are several types of history with different implications. One type of history is based on versions of the objects only. This type of history only records the versions of objects, but not the changes that occurred between two versions. Another type of history records only specific states of the objects e.g. the initial and end states. Finally, an ultimate type of history is the complete history. This type of history collects the full set of all state changes that occur in the complete system. There is not state change anywhere that is not recorded. This type of history allows the complete replay of what happened at any point in time. No matter what the situation is that requires history, the history can provide whatever is recorded because it is complete.

In principle, this type of history is the most useful since it is difficult to know which history details are relevant and to what extent. Nevertheless, it is also the most hard-to-implement and cost-ineffective since all state changes have to be recorded without exceptions and the resulting set of history data has to be stored and managed.

2.2.4 Requirement 4: Deadlock-freedom

The need of synchronisation in a conversation leads to difficulties to be considered in the design of an interaction-based system. Even worse problems in terms of synchronisation lead to a deadlock situation. A deadlock is a situation in which a set of processes are in a state from which it is impossible for any of them to proceed due to mutual dependencies of their resources. For example, this situation arises if some processes are part of a circular chain, and each process is holding resources that are requested by the next process in the chain.

It is difficult to avoid such situation. There are several techniques to deal with systems which model reactive-behaviour i.e. parallel or distributed systems that communicate with each other. For example, in model-checking techniques, a desired property is expressed in a suitable temporal or dynamic logic and a system is modelled as a transition system. By means of an efficient search procedure, it is checked if the property holds in the transition system. The checking is completely automatic and the result indicates if the model satisfies the property or a counterexample in which the model fails to satisfy it. One of these properties to be checked is deadlock-freedom. The main drawback of model checking is the state explosion problem. It occurs in the systems with an enormous number of global states, making it very difficult to check all possible states of the system.

2.2.5 Requirement 5: Formal execution semantics

Semantics provide meaning to computer programs. This meaning enables reasoning about such programs, based on the mathematical properties of the applied semantics. Reasoning is the process of drawing conclusions from facts [1]. In 1970, three main styles of formal reasoning about computer systems, focused on giving semantics (meaning) to programs were defined:

- Operational semantics: A computer program is modelled as an execution of an abstract machine. A state of such a machine is defined as an evaluation of variables. Simple program instructions represent transitions between states [21].
- Denotational semantics: Computer programs are just represented as a function between the input and the output [29].
- Axiomatic semantics: Programs are proved to be correct using proof methods. Central notations are program assertions, proof triples consisting of precondition, program statement, post-conditions and invariants [6].

For execution semantics, we follow the first approach, also called operational semantics. Execution semantics describe how the program evolves and behaves, but they are more efficient when described by a formal method. A formal method is used to describe mathematically and reason about a computer system. The advantage of using a formal method to model the execution semantics is that, if the specification is written in a logical language that could use inference, it is feasible to derive consequences out of the specification. Using this inference feature, properties of the specification that were not explicitly stated can be proven. In this way, invisible behaviour and properties of the system can be predicted and tested without having to be implemented, for instance, deadlocks or errors. Another advantage is verification of properties. Two well established approaches to verification with formal methods are model checking and theorem proving. These formal methods are used to analyse a system for desired properties.

Model checking is a technique that relies on building a finite model of a system and checking that a desired property holds in that model. In other words, an exhaustive state space search check is performed which is guaranteed to finish, given that the model is finite.

Given this restriction, model checking must devise algorithms and data structures to handle large search spaces. This technique has been widely used in protocol verification and recently, it is intended to be used for analysing specification of software systems.

Fundamentally, there are two general approaches to model checking. First one is called temporal model checking. In this approach, specifications are expressed in Temporal Logic [26] and systems are modelled as finite state transition systems. The procedure used is to check if a given finite state transition system is a model for the specification. The second approach requires an automaton behaviour for the specification. Then, the system is also modelled as an automaton and both are confronted to determine if the system behaviour conforms to the specification.

In contrast to theorem proving, model checking is automated and fast, it checks partial specifications, even if the system is just partially designed and produces counterexamples i.e. situations in which the model does not comply with the specifications, what can be used in debugging.

The main drawback of model checking is state explosion. There is a number of methods to alleviate this problem, such as appropriate reduction or abstraction techniques [7], which basically allow checking an almost unlimited number of states.

Theorem proving is used when system properties are specified in a certain mathematical logic. This logic defines a set of axioms and a set of inference rules. Theorem proving is the process of finding a proof of a property from the axioms of the system. This technique is being increasingly used in mechanical verification of safety-critical properties of software designs. Theorem proving can deal with infinite states if it relies on techniques such as structural induction to prove over infinite domains.

Formal execution semantics are also used as a prescription during the implementation of a system, where it is of the utmost importance that the specification is human-understandable. Otherwise the situation could arise that the specification is perfect, several properties has been checked and verified, but since the developer does not understand the specification correctly, the implementation does not follow the specification and the system does not behave correctly.

Hence, the use of formal execution semantics is twofold: automatically check the system properties and help the developers to understand the specification.

3. THE PROMISED LAND: ROSETTANET AND EBXML

A B2B protocol standard is in general with regards to a B2B interaction, the description of the message formats exchanged, their bindings to transport protocols and the sequencing of the messages, the security to be provided and many other properties [3]. The focus of our language is particularly on one of this aspects, the sequencing of messages. Different B2B protocol standards also focus on different aspects and not all cover every aspect. Hence for our overview we will only cover those standards which support modelling of the sequencing and of messages. The exchange sequence definition defines when a conversation is one-way (notification) and when some acknowledgment is required, a so called two-way (conversation). It also defines any retry logic for sending messages, i.e. when to retry and how often. The process definition defines the business event behaviour, i.e. the order of the exchange of several messages. It is between business partners only and does not define the business logic within a trading partner [3].

RosettaNet is the first standard we cover supporting among others the above described aspects. RosettaNet is a non-profit organisation that seeks to implement standards for supply-chain transactions on the Internet. The standard forms a common language, to align the processes between supply chain partners on a global basis. The backbone and key concept of the standardisation effort are the Partner Interface Processes (PIPs), which are used to define standard ways of interacting between companies to carry out a specified task. PIPs define business processes between trading partners. For each process, PIPs define the start state, the end state, the participants and their roles, process controls such as authorisation for each participant, documents exchanged and sequence of activities. These PIPs fit into seven clusters, or groups of core business processes. Each cluster is broken down into segments which are cross-enterprise processes involving more than one type of trading partner.

PIPs define the aspects of a business process which are common to the two parties (public processes) [4], but place no constraints on how the internal processes implement these common aspects. A PIP not only defines the flow of the business documents involved in the interaction, but also the format of the messages. As mentioned above the latter is not covered in this paper.

In theory the idea behind RosettaNet is that two partners only have to agree on which PIPs to use, and implement the PIPs according to their specification, when setting up a new partnership. However, as different businesses can have different back-end processes (private processes), flexibility within the standard remains to enable the adoption of it in different scenarios. PIP definitions often make use of generic datatypes and include fields with unbounded cardinalities. Since this is done at a syntactical level, there is no guarantee that two companies implementing the same RosettaNet PIP will ultimately be able to communicate with each other. Hence human interaction is necessary to reconcile the different processes used by two companies which intend to interact via RosettaNet. Developers implement these agreements between two parties when encoding the PIPs.

The second specification also covering the sequencing and process of messages is the Business Process Specification Schema (BPSS) as part of ebXML. ebXML itself is a set of specifications that together enable a modular electronic business framework. Standards in ebXML are covering the specification of a registry, a profile with a Business Service Interface and Business Messages, a Core Library as a set of standard parts incorporated by other ebXML elements and a Collaboration Protocol Agreement (CPA) an agreement between two participants in a business interaction based on two CPPs.

Again, we only deal with the Business Process Specification Schema in here, since it is the ebXMLs standard for sequencing and the process of business documents. It provides a standard framework by which business systems may be configured to support execution of business collaborations. The specification itself is based upon prior UN/CEFACT work (UN/CEFACT Modeling Methodology (UMM)) and consists of the following functional architecture components:

- UML [2] version of the BPSS
- XML version of the BPSS
- Production Rules defining the mapping from the UML to XML version
- Business Signal Definitions

The language is divided into the following key concepts.

- Business Transactions: They represent atomic units of work in a trading arrangement between two business partners, a sender and a receiver. Business Transactions cannot be decomposed into lower level units. They are expected to be enforced by the software managing the transaction, i.e. an ebXML Business Service Interface (BSI). A Business Transaction will always either succeed or fail. An example would be Cancel Purchase Order.
- Business Document Flows: They define the nature of a business transaction, i.e. the passing of Document

Table 1: Fulfilling of requirements by B2B standards

Requirement	RosettaNet	ebXML
Partner view	Yes	Yes
Concurrency	Yes	Yes
History	Yes	No
Deadlock-freedom	No	No
Formal execution semantics	No	No

Envelopes between the requestor and responder, and can either be one-way (notification) or two-way (conversation). Hence there is always a requesting Business Document, and optionally a responding Business Document. The actual business document definition itself is achieved using the ebXML core component specifications, or by some external methodology like RosettaNet.

- Binary Collaboration: The Binary Collaboration is defined to be between two roles: the requestor and the responder. It is expressed as a set of Business Activities between the two roles. Each Business Activity reflects a state in the collaboration. A Collaboration Protocol Agreements (CPA) within the ebXML framework associate itself with a specific Binary Collaboration.

- Choreography: Choreography within BPSS defines the ordering and sequencing of Business Activities within a Binary Collaboration. Choreography is specified in terms of Business States and their transitions. It includes a number of auxiliary kinds of Business States that facilitate the choreographing of Business Activities like: start, completion, fork, join, decision, business transaction activity, business collaboration activity.

Once summarised, we can proceed to analyse which requirements expected in a B2B interaction are being fulfilled. Either RosettaNet and ebXML seem to cover most of them, lacking eventually some of the most important ones, precisely the ones requiring formal semantics. A comparison table is presented in Table 1.

4. L3: A CONVERSATION-ORIENTED LANGUAGE FOR B2B INTERACTIONS

In this section, we will describe our conversation-oriented language. Our language has four main elements. Conceptual model, BNF syntax, execution semantics, architecture and implementation.

4.1 The L3 conceptual model

A conceptual model is a set of concepts and relationships which describes a specific domain. This conceptual model is grounded in a language. In principle, all languages have syntax and semantics. Most programming languages are today defined as having grammars written in some extension of the Backus Naur Form [23]. The precise layout and use of meta-symbols varies. Here we consider an Extended Backus Naur Form (EBNF) typical of the variants used to describe our different languages. A grammar written in this way defines rules of syntax. These rules specify what sequences of symbols are allowed to occur in a legal program and in what ways these symbols may be combined. They do not

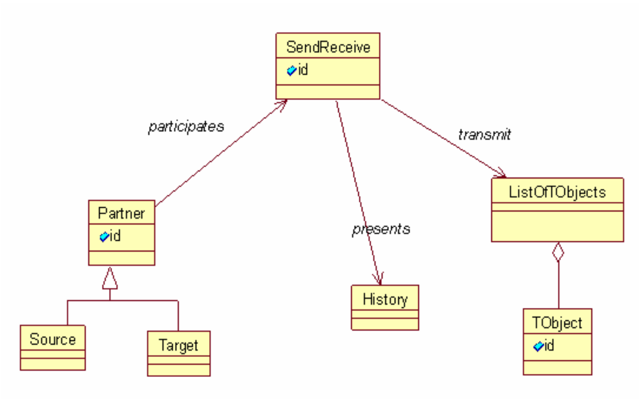

Figure 1: L3 Conceptual model

specify how such sequences are to be interpreted. That requires semantic rules, which attach meaning to the syntactic structures of the grammar. We use BNF as the syntax of our language.

4.1.1 L3 Conceptual model

The conceptual model of L3 is described in the UML [2] diagram depicted in Figure 1. This conceptual model has five main concepts. A *SendReceive* is the fact of communication. A *SendReceive* transmission has a property *id* and transmits a *TObject*. A *Transmission Object*, *TObject*, is the actual element which is being transmitted. In a business context, it can be a Purchase Order (PO) or a Purchase Order Acknowledgement (POA). It also has an *id* property. The *Partner* concept models the entities involved in the communication, represented by *Source*, the origin entity and *Target*, the destination entity. *Partner* has an *id* property and participates in a *SendReceive transmission*. *TObject* has also an *id* property. Finally, a *SendReceive* transmission presents a certain *History* i.e. an explicit representation of the state changes of the transmission.

4.2 Execution Semantics

Execution or operational semantics were first introduced in Section 2.2. A computer program is modelled as an execution of an abstract machine. A state of such a machine is defined as an evaluation of variables. Simple program instructions represent transitions between states. Since we are modelling execution semantics, we benefit from the use of a formal method for this, as explained in section 2.2. We use Petri Net to model our execution semantics.

A Petri Net [24] is an abstract, formal model of information flow. The properties, concepts and techniques of Petri Nets are being developed in a search for natural, simple and powerful methods for describing and analysing the flow of information and control in systems, particularly in systems that may exhibit asynchronous and concurrent activities. The major use of Petri Nets has been the modeling of systems of events in which it is possible for some events to occur concurrently but there are constraints on the concurrence, precedence or frequency of these occurrences. A Petri Net can be represented as a graph. This pictorial representation models the static properties of a system, like a flowchart represents the static properties of a computer program. A

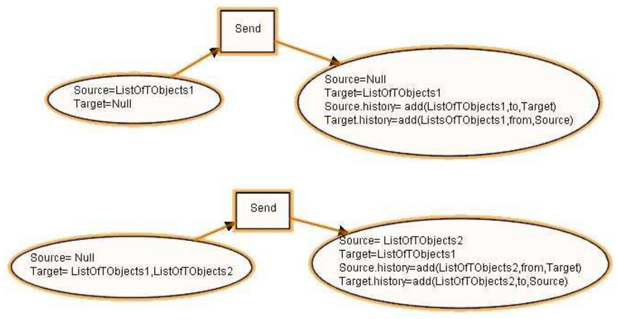
Send
Source=ListOfTObjects1
Target=Null
Source=Null
Target=ListOfTObjects1
Source.history= add(ListOfTObjects1,to,Target)
Target.history=add(ListsOfTObjects1,from,Source)
Send
Source= Null
Target= ListOfTObjects1,ListOfTObjects2
Source= ListOfTObjects2
Target=ListOfTObjects1
Source.history=add(ListOfTObjects2,from,Target)
Target.history=add(ListOfTObjects2,to,Source)

Figure 2: Execution Semantics from a global perspective

Petri net graph is composed by two types of nodes: a place (represented by a circle) and a transition (represented by a bar). These two types of nodes are connected by arcs. The link can only happen from places to transitions and vice versa.

Apart from the static properties represented by the flowchart, Petri Nets may describe the execution of a system i.e. its dynamic properties. This is achieved through the motion of a token, which are moved by the firing of transitions of the net. A firing is an activation of a transition. A transition can only fire if it is enabled. A transition is enabled if it has a token in all its input places. The token path shows the execution path, and, finally, the dynamic behaviour of the system. Petri nets may be used in a large number of ways. For instance, they can be considered as formal automata and investigated either as automata or as generators of formal languages [19]. There is also a link with the theory of computational complexity [20].

Petri Nets are also a modelling tool. They were devised for use in the modelling of a specific class of problems i.e. discrete-event systems with concurrent events. Particularly, they model events and conditions in systems. At any point in time, a system is in a given state. In this state, some conditions holds and this can trigger the occurrence of certain events. These events may change the state of the system, causing some of the previous conditions not to hold and new conditions to hold.

4.2.1 L3 Execution Semantics

The execution semantics of L3 modelled with Petri Nets are as follows. Firstly, *Source* has initially the instance of a *TObject* ready to send in a state ready to send. Once sent, the *Target* has the transmitted *TObject* and *Source* has the *TObject* in a different state e.g. sent. *Source* can not send any number of instances of *TObject* and can be sent to any number of *Targets*. *Source* maintains a list of *TObjects* to be sent and those that have been sent. So it is possible to inquire *Source* at any time with respect to what happened so far and the same *Partner* can be *Source* and *Target*. Now, a *Target* has another instance of *TObject* ready to send and is in a state ready to send. Once sent, the *Target* is in a state which has the received instance of *TObject* but lacks the sent instance of *TObject*. Then the *Source* is in a state which has the instance of *TObject* sent by the *Target*.

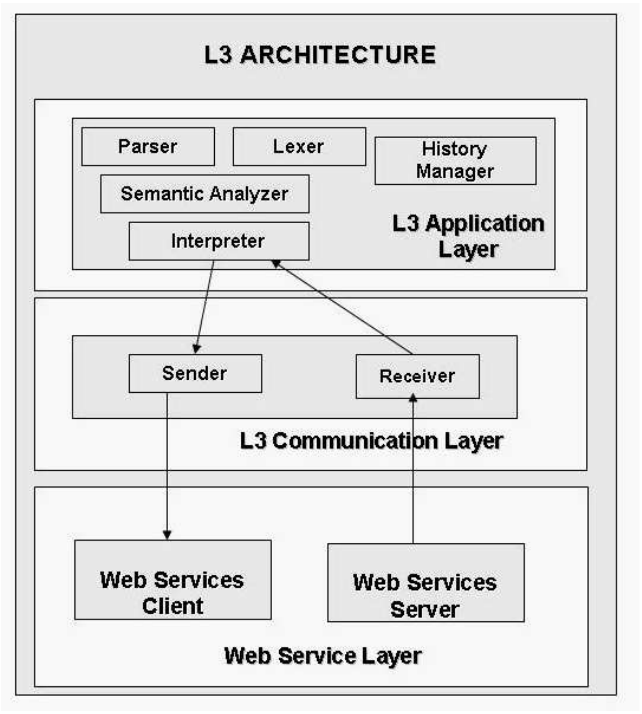

Figure 3: L3 Architecture

4.3 Architecture

4.3.1 L3 architecture

In this section, we introduce the architecture of L3. It is a service-oriented architecture with a layered design. The basic functionality of L3 is situated in the bottom layer, while the more complex functionality is situated in the upper layer. Like in other layered architectures, the purpose of the bottom layer is to provide the access to the Web Services required by the upper layers, hiding the details of how the Web Services are implemented. The layers are abstracted in such a way that any software agent can communicate with any counterpart of the upper layers.

In this context, each layer has a defined function as depicted in Figure 3.

- **Web Service Layer** enables the lowest level communication. It consists of a set of simple Web Services, but it is mainly divided between a Web Service client and a Web Service server. Those Web Services interact with the lower levels of communication and transport at a network stack level.

- **L3 Communication Layer** has two main *boxes* or components, namely, a Sender and a Receiver. The Sender component use a mechanism of the lower layer to send transmission objects *TObjects*. The Receiver component retrieves received *TObjects* by the lower layers.

- **L3 Application Layer** has five components. Its functionality is the following:

  – The Lexer or Lexical Analyser reads an input stream and returns tokens one by one.

- The Parser recognizee valid sentences of the language by analysing the syntax structure of the set of tokens passed to it from the Lexical Analyser.
- The Semantic Analyser specifies the action taken for each instance of the language and it builds up the data model in memory.
- The Interpreter / Execution Environment reads one statement at a time, translates that statement to machine language and executes the machine language statement, then continues with the next statement.
- The History Manager records all the history of the system in a persistence store. Actually the history is an explicit representation of past state changes of the objects in a system, so the History Manager stores all those state changes, so that it can be queried about the History of the system.

4.4 L3 fulfilment of requirements

When taking into account the requirements enumerated in section 2.2 and the lack of the fulfillment of some of them by the current B2B standards, such as the ones considered in section 3, it becomes a key question to prove if our L3 language fulfills those requirements. Here we will show it step by step. We have outlined such results in Table 2.

- *Requirement 1. Partner view.* L3 is designed to differentiate between Sender and Receiver, which explicitly defines which role the participant is playing in the conversation. In its conceptual model, *Partner* is an explicit concept and its subtypes *Source* and *Target* define Sender and Receiver, respectively.

- *Requirement 2. Concurrency.* The L3 Architecture is loosely-coupled and emphasises the external behaviour description. This architecture is a set of significant decisions about the organisation of the system, the selection of the structural elements and their interfaces by which the system is composed. Together with the behaviour as specified in the collaboration among those elements, the composition of these structural and behavioural elements the L3 architecture decouples these elements, their interfaces, their collaborations and their composition. Each of them can interact independently and can be used in a concurrent way. Regarding the two types of concurrency described in section 2.2, both of them can be used.

- *Requirement 3. History.* In the L3 Architecture, the History Manager stores all changes in the system. It uses persistent storage such as a database or a file system.

- *Requirement 4. Deadlock-freedom.* Checking deadlock-freedom is possible since we have formal execution semantics defined by Petri Nets. In L3, we use the same approach as in [14]. We built in the state space and check the particular states following to a deadlock-situation. This problem always arises when detecting deadlock situations in concurrent systems. There are different possibilities to minimise the impact of the problem. For example, the approach followed in [14] aims to simplify the construction of the whole transition systems and produce short counterexamples for the deadlock. This is achieved by using heuristics in the search, specifically a type of heuristics known as the A* algorithm. In our particular case, we tackle with deadlocks by constructing the whole space of states and checking which particular states are following to a deadlock.

Table 2: L3 fulfillment of requirements

Requirement	L3
Partner view	Yes
Concurrency	Yes
History	Yes
Deadlock-freedom	Yes
Formal execution semantics	Yes

- *Requirement 5. Formal execution semantics.* L3 formal execution semantics are well-defined by means of Petri Nets [25]. Petri Nets were devised for use in the modeling of a specific class of problems i.e. discrete-event systems with concurrent events. Particularly, they model events and conditions in systems. At any point in time, L3 is in a given state. In this state, some conditions holds and this can trigger the occurrence of certain events. These events may change the state of L3, causing some of the previous conditions not to hold and new conditions to hold. Checking conditions and their properties are possible thanks to techniques such as model-checking as described in section 2.2

In this section, we have shown how L3 fulfills all the requirements. Apart from the cornerstone functionalities of L3, it also provides a loosely coupled architecture and implementation.

5. CONCLUSIONS AND RELATED WORK

As the use of Web Services grows, the problem of interaction among them will get more acute. The need for semantics to describe this interaction will leverage B2B interactions. In this paper, we proposed a conversational language for B2B interactions with a clearly specified conceptual model, execution semantics and architecture. The forthcoming of our approach are the use of precise semantics by means of formal methods, which allow us to use model-checking and make feasible to have the specification of some properties proven.

A similar approach was presented in [13], where a CCS-based [22] formal semantics for conversations was discussed. Also in [12], an architecture for conversations based on the Speech Act [30] was proposed. While this paper focuses mainly on Semantic Web Services concerns about B2B interactions, there is a whole plethora of conversation-oriented interactions in many other domains. There are several other approaches to the one presented in this paper. Using agents for taking advantage of the machine-processable metadata provided by the Semantic Web or the Semantic Web Services has been studied previously. In [17], the author points out how the ontology languages of the Semantic Web can lead to more powerful agent-based approaches for using services offered on the Web. The importance of the use of ontologies in agent-to-agent communication has also been highlighted. In [11], the authors outline their experiences of building semantically rich Web Services based on the integration of

OWL-S [33] based Web Services and an agent communication language (it is done to separate the domain-specific and the domain-independent aspect of communication).
A more practical approach is shown in [10]. The authors describe an application where intelligent agents, aided by context information provided by Semantic Web Services, assist their users with different sets of tasks.
Semantic Web Services initiatives such as WSMO [31] intend to present their own choreography model. In WSMO, choreographies are ontology-based and grounded on the Abstract State Machines (ASM) formal model [15]. Other interesting formalisms to model conversations are process algebras such as the Pi-Calculus [28] or the Statecharts formalism [16]. In the future, we will conduct our research in this direction, since they look promising directions to formally specify and model conversations among Semantic Web Services.

6. ACKNOWLEDGMENTS

The work is funded by the European Commission under the projects DIP, Knowledge Web, Ontoweb, SEKT and SWWS; by Science Foundation Ireland under the DERI-Lion project; and by the Austrian government under the CoOperate Programm.

7. REFERENCES

[1] J. C. M. Baeten. A brief history of process algebra. Technical report, 2005.

[2] G. Booch, I. Jacobsen, and J. Rumbaugh. *The Unified Modeling Language Reference Manual.* Addison-Wesley, New York, 1998.

[3] C. Bussler. B2B protocal standards and their role in semantic B2B integration engines. *IEEE Data Engineering Bulletin*, 24(1):3–11, 2001.

[4] C. Bussler. The role of semantic web technology in enterprise application integration. *IEEE Data Engineering Bulletin*, 26(4):62–68, 2003.

[5] C. Bussler. *B2B Integration.* Springer, Berlin, 2004.

[6] H. C.A.R. An axiomatic basis for computer programming. *Communications of the ACM*, 12(2):576–580, 2004.

[7] E. Clarke and O. Grumber. Avoiding the state explosion problem in temporal logic model checking. *Annual ACM Symposium on Principles of Distributed Computing*, 12(2):294–303, 1987.

[8] M. Dean and G. Schreiber, editors. *OWL Web Ontology Language Reference.* 2004. W3C Recommendation 10 February 2004.

[9] D. Fensel. *Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce.* Springer, Berlin, 2001.

[10] F. Gandon and N. Sadeh. Semantic web technologies to reconcile privacy and context awareness. *Web Semantics Journal*, 1(2):30–37, 2004.

[11] N. Gibbins, S. Harris, and N. Shadbolt. Agent-based semantic web services. In *Proc. of the 12th International World Wide Web Conference*, May 2003.

[12] J. M. Gomez, G. Alor, O. Olmedo, and C. Bussler. A B2B conversational architecture for semantic web services based on bpim-ws. In *Proc. of the 10th IEEE International Conference on the Engineering of Complex Computer Systems (ICECCS2005)*, 2005.

[13] J. M. Gomez, H. Lausen, S.-K. Han, and C. Bussler. A message-exchange pattern based formal approach for B2B interactions. In *Proc. of the 2st European Semantic Web Conference (ESWS 2005)*, MAY 2005.

[14] S. Gradara, A. Santone, and M. L. Villani. A* for deadlock detection in ccs processes. In *Proc. of Concurrency specification and programming*, 2003.

[15] Y. Gurevich. *Evolving Algebras 1993: Lipari Guide.* Oxford University Press, Oxford, 1993.

[16] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, JUN 1987.

[17] J. Hendler. Agents and the semantic web. *IEEE Intelligent Systems*, 16(2):30–37, 2001.

[18] G. Holzmann. *Design and validation of computer protocols.* Prentice-Hall, London, 1990.

[19] H. J. Petri net languages. Technical report, 1975.

[20] D. Jones, N. Landwebee. Complexity of some problems in petri nets. Technical report, 1976.

[21] J. McCarthy. A mathematical basis for computer programming. *Computer Programming and Formal Systems*, 1(2):30–37, 1963.

[22] R. Milner. *A Calculus of communicating systems.* Springer, Springer, Berlin, 1980.

[23] P. Naur. Revised report on the algorithmic language algol 60. *Communications of the ACM*, 3(5):299–314, 1960.

[24] J. Peterson. Petri nets. *Communications of the ACM Surveys*, 9(3):223–252, JUL 1977.

[25] C. Petri. Kommunikation mit automaten. Technical report, 62.

[26] A. Pnueli. *Applications of temporal logic to the specification and verification of reactive systems: a survey of current trends.* Springer, New York, 1986.

[27] D. Roman, H. Lausen, and U. Keller, editors. *Web Service Modeling Ontology (WSMO).* 2004. WSMO Final Draft D2v1.0. Available from http://www.wsmo.org/2004/d2/v1.0/.

[28] D. Sangiorgi, D. Walker. *The Pi-Calculus: A theory of Mobile Processes.* Cambridge Press, Cambridge-Press.Cambridge, 2004.

[29] D. Scott and C. Strachey. Towards a mathematical semantics for computer languages. In *Proc. Symposium Computers and Automata*, May 1971.

[30] M. Singh. Agent communication languages: Rethinking the principles. *Computer*, 31(12):40–47, 1998.

[31] M. Stollberg, D. Roman, and J. M. Gomez. A mediated approach towards web service choreography. In *Proc. of the workshop on Semantic Web Services held at the 3rd ISWC*, 2004.

[32] G. T. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5:199–220, 1993.

[33] The OWL-S Services Coalition. Owl-s 1.1 beta release. Technical report, 2004.