A B2B message-exchange pattern based formal approach for Semantic Web services

Juan Miguel Gomez, Sung-Kook Han, Christoph Bussler
Digital Enterprise Research Institute (DERI), Ireland. National Univesity of Ireland, Galway.

Abstract: Current Business-to-Business interactions rely on complex exchange of messages. Web Services technologies provide a basis to establish the Web as the ubiquitous technical platform for B2B applications. However, there is a need of formalism for specifying behaviour and verifying certain properties such as well-formedness or deadlock freedom. In this paper, we present a message-exchange pattern based language and its formal semantics and discuss the benefits obtained by such formalization.

Key words: Semantic Web Services, B2B, process algebra

1. INTRODUCTION

This method of producing a text is one we all intuitively understand. After al The Internet is going trough several major changes. Recently, it has become a new vehicle for business transactions and information exchange rather than just a repository of information. Companies are challenged to publish and share services on the web. Furthermore, integration of services through different companies would foster the development of Business-to-Business (B2B) [Bussler, 03] interactions by sharing costs and reusability.

As the technology associated to Business-to-Business (B2B) interactions gains momentum, the need for a formal approach to message exchange becomes increasingly important. The concept of web services has recently come up into the arena. Web services provide standard protocols for discovering, invoking, describing and composing services. Current web service technology, based on SOAP, WSDL and UDDI, has a very basic and non-automated interaction model. Composition of simple services into complex ones represents a natural evolution of the technology. In this context, some business processes proposals are in the playground, from which, at the moment, BPEL4WS [Thatte et al, 03] seems to be ahead.

However, the problem is that none of these current approaches is formalized and present the real semantics of the B2B interaction. Deadlock-detection or well-formedness of the interactions may not be accomplished and they lack the reliability and efficiency that is expected from a business context point of view. For example, this deficiency could lead to a massive failure of the system in case of a deadlock, which reverts into a loss of profit, bad-mouthing and complaints and a negative image that technical departments are usually appalled by and try to avoid at any price.

Hence, in this paper, we propose a new approach that, to our knowledge has never been attempted. We take as a reference the W3C WSDL message exchange patterns and present a formal approach for B2B interactions. This approach is based on a pat-ter-based algebra built on well- founded process algebra semantics. Our model allows properties such as well-formedness and deadlock freedom to be checked. Given that we are trying to provide very well-defined semantics for Web Services technologies, our effort falls into the category of Semantic Web Services, which aim to provide a conceptual and formal model for Web Services.

This paper is structured as follows. In section 2, we present a general introduction to W3C WSDL message exchange patterns. Section 3 describes and defines, firstly the syntax and informal description of our algebra, and then its formal semantics. In section 4, the advantages of our approach for modeling B2B interactions are discussed and a simple B2B use case is modeled. Finally, we present future and related work in section 5.

2. MESSAGE EXCHANGE PATTERNS

The W3C Architecture describes a set of related technologies that exchange messages between senders and receivers. The architecture formalizes the exchange of messages into "patterns". These basic patterns can be combined to more complex patterns that build up a whole conversation. For example a bid process can be modelled as a number of incoming messages (representing the single bids), followed by a single out message signaling one of the buyers the acceptance of a bid. Describing these patterns and the specific combinations of those will enable reusability and a common understanding between the parties involved. In that way patterns form a contract between provider and requester enabling their communication:

To describe those patterns we will use a simplified version of the scheme proposed in [Hohpe & Woolf, 2003]:

– Name of the pattern
– Short Description / Sketch**:** One or two sentences possibly accompanied by a diagram to grasp the essence of the pattern
– Description: The actual description of the pattern.
– Example: A simplified example illustrating the pattern.

First, we focus on patterns describing one participant of the conversation called Input Output Patterns (IOP), in Section 2.1. Section 2.2 describes how these patterns can take both participants of the conversation account.

2.1 Input Output Patterns (IOP)

An Input Output Pattern (IOP) is a sequence of one or more messages related, in the case of W3C WSDL patterns [Gudgin et al., 2003] to a Web Service. Actually, each pattern is a combination of the following elements: an input message received by a web service and output message sent and a fault handling mechanism. The IOP specifies the sequence and the number of messages including their direction as either a specific number or a variable. The IOP patterns are as follows:

P1- Pattern 1: In-Only Pattern

– Name: In-Only
– AKA: One-way-in
– Short Description: An endpoint or node receives a message
– Description: This pattern consists of exactly one message as follows: A message with the direction 'in' received from some node N.
– Example: The receipt of any kind of Notification (e.g. a shipment notice of a retailer that an ordered product left its premises); or a Receipt of a Request for Quotation (that is not directly followed by an offer)

P2- Pattern 2: In-Out Pattern

– Name: In-Out
– AKA: Request-Response
– Short Description: An endpoint or node receives a message and then sends a message handling any fault.

- Description: This pattern consists of exactly two messages, in order, as follows: A message with the direction ‘in’ received from some node N.
- A message with the direction ‘out’ sent to node N
- Example: The receipt of a Purchase Order (PO) and the reply with a POA (Purchase Order Acknowledgment)

P3 – Pattern 3: Out-Only Pattern

- Name: Out-Only
- AKA: One-way-Out
- Short Description: An endpoint or node sends one message out.
- Description: This pattern consists of exactly one message as follows: A message with the direction ‘out’ sent to some node N
- Example: The sending of any kind of Notification (e.g. a shipment notice of a retailer that an ordered product left its premises); or the broadcast of a Request for Quotation (where the reply is not modelled within this pattern)

P4- Pattern 4: Out-In Pattern

- Name: Out-In
- AKA: Solicit Response
- Short Description: An endpoint or node sends one message out and receives one message in.
- Description: This pattern consists of exactly two messages, in order, as follows: A message with the direction out is sent to some node N and then a response.
- Example: An issue of a Purchase Order (PO) followed by a receipt of a Purchase Order Acknowledgement (POA).

P5- Pattern5: In-Multi-Out Pattern

- Name: In-Multi Out
- AKA: Multicasting
- Short Description: An endpoint or node receives a message and sends zero, one, or more messages out handling any fault.
- Description: This pattern consists of one or more messages. A message with the direction ‘in’ received from some node N and zero, one or more messages with the direction ‘out’ sent to node N.:
- Example: An event message received which triggers a notification message to multiple parties.

P6- Pattern 6: Out-Multi-In Pattern

– Name: Out-Multi In
– AKA: Multicasting
– Short Description: An endpoint or node sends one message out, optionally receives zero, one or more message in and handles any fault.
– Description: This pattern consists of one or more message. A message with the direction ‘out; is sent to some node N and zero, one or more messages with the direction ‘in’ are sent from node N.
– Example: An issue of a Request for Tender followed by an optional receipt from interested parties.

2.2 Message Exchange Patterns (MEP)

A message exchange pattern (MEP) describes both nodes involved in a conversation and introduce the notion of compatibility. This distinction follows also the discussion in [Both & Lewis, 2003], which in essence describe that IOPs make no explicit assumption about the second node of a conversation, whereas MEPs do.

Generally a Message Exchange Pattern can be described as a template for the ex-change of messages between nodes [Haas&Brown, 03]. More precisely a Message Exchange Pattern (MEP) is the combination of two IOPs such that sender and receiver of every message are exactly defined. A conversation is defined here as a specific exchange of messages. Concrete Message exchange patterns are described in Table 2. This table summarizes possible combinations between IOPs.

		Node 2					
		In-Only	In-Out	In-Multi-Out	Out-Only	Out-In	Out-Multi-In
Node 1	In-Only	■			**X**		**X**
	In-Out	░	■			**X**	**X**
	In-Multi-Out	░	░	■	**X**	**X**	**X**
	Out-Only	X	░	X	■		
	Out-In	░	X	X	░	■	
	Out-Multi-In	X	X	X	░	░	■

Table 1: Message Exchange Pattern Matrix

Legend: **X**, the IOP combination is fully compatible between the two nodes

■ not applicable
░ omitted (since duplicated)

By describing MEPs, the analysis is starting to look at both sides of the interaction. If one node is described by an IOP and has a particular behavior, we need to show a complementary behavior (IOP) of a second node in order to describe a whole conversation.

In this section, we discussed various ways of combining message exchange patterns. However, a formal approach is required to check certain properties of the conversation. In this context, next section will provide further steps in that direction.

3. PATTERN-BASED APPROACH

In this section, we describe first the syntax and informal semantics of our pattern-based algebra. Then, we will use the formal semantics of CCS given in [Plotkin,81]. Our goal is to represent a conversation in a formal way as a set of the message-exchange patterns detailed in the previous sections. A conversation is a set of messages exchanged from two processes or entities. By formalizing this exchange, we can analyze and check behavioural properties of the conversation. In the case of a B2B system, this means we can verify deadlock and livelock freedom, behavioural equivalence and some other suitable properties for the reliability of the system. In our model, we will not consider the use of recursive expressions. A conversation can not be infinitely long.

3.1 Informal semantics and syntax

Here comes the first definition of our pattern-based algebra and its informal BNF grammar syntax.

Definition 3.1 Our pattern-based algebra sticks to the following BNF-grammar:

$$C ::== N \mid K \mid P_1 + P_2 \mid P_1 \,||\, P_2$$

where:

– C represents a conversation
– N represents the 0 / Nil pattern, which performs no action.
– K represents a simple pattern.
– $P_1 + P_2$ represents a composite pattern. The *add operator* is a constructor to describe a sequential behavior. Given two patterns, P_1 and P_2, P_1 is performed followed by the P_2 pattern.
– $P_1 \,||\, P_2$ represents a composite pattern. It describes two patterns P1 and P2 being performed in parallel.

3.2 Formal semantics

We will somehow base our algebra in the CCS theory and formal semantics. The theory of Calculus Communicating Systems (CCS) was developed by Robin Milner, from 1973 to 1980, culminating with the foundational book [Milner,80]. Milner noticed that concurrent processes have an algebraic structure. For example, once we have built two processes P and Q, a new process is created from combining P and Q sequentially or in parallel. These combinations result in processes whose behaviour depends on P and Q and the used operation to combine them. Identifying ways of combining them with algebraic operators leads to the conclusion that we can create new processes from existing ones following an algebraic structure. These processes also need to communicate and interact with one another, so an inter-process communication theory is also developed. In CCS, a process presents an interface, which de-scribes a collection of communication ports, also called channels. Communication happens trough the ports in an input or output way. However, this interface only gives static information about the process. Behaviour of the process is given by a CCS description.

The formal semantics of CCS are given by Labelled Transition Systems (LTS), explained in [Keller, 90]. CCS expressions can be translated to set of states of a Labelled Transition Systems, whose actions are either input or output actions on communications ports or internal actions. Since we will use in future sections a Labelled Transition System, here comes the definition.

Definition 3.2 A *Labelled Transition System (LTS)* is a triple (Sts, Act ,{ $\xrightarrow{a} | \, a \in Act$ }), where:

Sts is a set of states

Act is a set of actions

$\xrightarrow{a} \subseteq Sts \times Sts$ is a transition relation for every $a \in Act$

When defining the formal model of CCS, some rules capture the informal semantics of CCS constructors and turn them into, first, an elegant and formal syntax, secondly defining its formal semantics. These semantics are taken from the framework for Structural Operational Semantics. [Plotkin,81].

Now we will proceed to define all the elements of our algebra in the context of the CCS formal semantics.

Definition 3.3 The N pattern represents the empty pattern. It is equivalent to the CCS 0 or Nil process, which performs no action i.e. it is a no-op process.

Definition 3.4 The composite pattern P_1+P_2 is defined as follows. Let LTS_1 be the Labelled Transition System corresponding to P_1 and LTS_2 the one corresponding to P_2 as stated in Definition 3.1. Let Sts_{1N} be the final state of LTS_1 and STS_{21} the initial state of LTS_2. P_1+P_2 is represented by the LTS formed by the union of LTS_1 and LTS_2 in a way that the next state for STS_{1N} is STS_{21}.

Definition 3.5 The composite pattern P_1 || P_2 is defined by the CCS *parallel composition operation*. It describes two processes that run in parallel one with the other and may communicate via the communication ports they share and use in complementary fashion. By complementary, we understand that one of the processes uses the port for input and the other for output. In the general theory of CCS this communication is possible, but not compulsory. These processes may proceed independently and have no interaction.

Actually, some of the patterns described in section 2.1 can be expressed by means of our algebra. For example, Pattern 5 could be described with the expression (1), a Pattern 1 pattern and a sequence of Pattern 3. In the expression P_1 represents a Pattern 1 and P_3 a Pattern 3.

$$P_1 + P_3 + P_3 + \ldots + P_3 \quad (1)$$

Likewise, Pattern 6 can be described as one Pattern 3 and a sequence of Pattern 1 as in expression (2).

$$P_3 + P_1 + P_1 + \ldots + P_1 \quad (2)$$

In the next section, we take a closer look to the advantages of this approach and how can we use our algebra.

4. THE ADVANTAGES OF OUR APPROACH

There are several advantages in our approach. First of all, the defined semantics can be used to prove algebraic properties of the constructs such as commutativity or transitivity. These properties may help to produce complex conversations by combining a set of different patterns in different ways. Properties of the overall conversation can be proofed and analyzed.

Model checking is another interesting feature we can apply to know about the well-formedness of our conversation model. As specified in [Clarke et al,96], 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. In our conversation, model checking devises algorithms and data structures to handle large search spaces. This technique has been widely used in hardware and protocol verification and recently, it has been used for analyzing specification of software systems.

One of the most desirable properties to be checked is deadlock-freedom. Even if our system is well-formed, it can contain the possibility of deadlock. [Roscoe,98] describes deadlock as the situation in which there exist several unanswered attempts to communicate from process to process. In this situation, each process is blocked, waiting patiently for the other process to unblock it.

In our case, we will prove that deadlock detection is possible following the approach described in [Brand et al, 83]. First, we will refer to Definition 3.2 to express the interaction as a Labeled Transition System, composed by states and actions. Then we will use a common representation for the connection of our processes. Each pair of processes involved in the conversation is connected by a full-duplex, error-free, FIFO channel. Queues and delays are not represented in the model. There are no assumptions about the time a transition can occur, which stresses the asynchronous nature of our system. Finally, we formally define:

Definition 4.1 .A global state is a pair <Sts,C>, where Sts is a N-tuple, where N is a finite natural number, of states $S_1 ... S_N$ (where S_i represents the current state of the process) and C is a N^2-tuple $(C_{11} ... C_{1N}, C_{21} ... C_{NN})$ where each C_{ij} is a sequence of messages. Actually, C_{ij} represents the content of the channel from process *i* to process *j*.

Definition 4.2 .A *stable N-tuple* is a reachable global state where all the channels are empty and there is no transmission from any state.

Hence, a deadlock can be defined as the situation in which a stable N-tuple is reached. Deadlock freedom subsequently depends on the avoidance of such stable. For this it is necessary the detection of stable N-tuples. A “tree-algorithm” as the one showed in [Brand et al, 83] must be used. A “tree protocol algorithm” evaluates all possible global states and identifies every stable N-tuples. This can only be possible because the finite number of states of the model.

Behavioral equivalence as understood in the CCS concurrency theory is also an interesting property. We may like to verify if our pattern-based conversation is similar to another. A first attempt would take us back to Definition 3.1. Viewing our model as a Labeled Transition System, which models processes in terms of states and transitions, we could be tempted to argue the trace equivalence of several patterns. However, as explained in [Milner, 80], trace equivalence is not suitable for testing behavioral equivalence, but strong bisimulation is. Nevertheless, the analysis of how to apply strong or weak bisimulation to our approach is far beyond the scope of this paper.

5. CONCLUSIONS AND RELATED WORK

In this paper, we presented a message exchange pattern-based algebra for modeling interactions. The formal semantics of the algebra is expressed in terms of the Calculus of Communicating Systems by providing a direct correspondence between our algebra operators and a CCS constructor. Hence, any conversation, understood as a message exchange sequence, can be validated by means of formal methods techniques such as model-checking. Interesting properties from the conversation can be verified, like well-formedness or deadlock freedom.

In the context of Semantic Web Services and B2B interactions, the need of modeling conversations as a set of interaction patterns is gaining momentum. A conversation can be described as “the set of acceptable message exchanges and the order in which they should occur” [Benatallah et al, 03].

Some proposals like the Web Services Conversation Language (WSCL) deal with defining business payload in the public interface, typically using XML and XML schemas. Yet defining only which documents are expected by a web-service and which are returned in response, is not enough, it is also necessary to define the *order* in which these documents are exchanged. The notion of conversation in WSCL allows to specifying the order by diving them into interactions, transitions and, finally, conversations. The Web Services Choreography Interface (WSCI) [Arkin et al, 03] is a W3C initiative to describe the flow of messages exchanged by web services. WSCI addresses choreography from two levels: first, it builds on top of WSDL capabilities and then it defines a model, which allows the composition of two or more interface definitions. In The Business Process Execution Languages (BPEL4WS) [Curbera et al,03], Abstract Processes use process descriptions that specify the mutually visible message exchange behaviors of each of the parties involved in an interaction.

Finally, WS-CDL [Kavantzas et al, 04] appears as the ultimate initiative of the W3C to represent choreographies. In a nutshell, WS-CDL is a declarative, XML based language for defining the complementary and observable behavior of a set of collaborating web services. The observable behaviors are defined from a global point of view and not from the point of view of a particular partner.

However, none of these alternatives take into account formalization. Formalization of web services composition based on a Petri-net algebra is presented in [Hamadi et al, 04]. Formalization of web services choreographies, concretely WSCI, are tackled by the interesting contributions of [Brogi et al,0 4] by using CCS [Milner,80]. An interesting link with some other process algebras such as CSP [Hoare, 85] or Pi-Calculus [Sangiorgi, 04] could be appealing research in the direction of mobility or axiomatization of behavioral equivalences.

Very interesting major initiatives in the field of semantic web services such as the Web Service Modelling Ontology (WSMO) [Roman et al., 2004] provide a way to model choreographies by means of Abstract State Machines [Glasser et al, 04]. Together with its reference implementation, the Web Services Modeling Execution Environment (WSMX), they will be a reference point for future achievements in the scope of this paper.

6. ACKNOWLEDGEMENT

This work is funded by the European Commission under the projects DIP, KnowledgeWeb, Ontoweb, SEKT, SWWS, Esperonto, COG and h-TechSight; by Science Foundation Ireland under the DERI-Lion project; and by the Vienna city government under the CoOperate program.

7. REFERENCES

[Alexander, 1979] Alexander C.: *The Timeless Way of Building*, Oxford University Press, 1979.

[Arkin et al. 2002] Arkin, A. et al, Web Service Choreography Interface (WSCI) 1.0.

[Booch et al., 1998] Booch G. Jacobsen I. and Rumbaugh J. : *The Unified Modeling Language Reference Manual*, Addison Wesley, 1998.

[Both & Lewis, 2003] Booth D. and Lewis A. : *Message Exchange Patterns Versus Input/Output Patterns,W3C Disscussion Document*, 2003.

[Brand et al, 83] Brand D., Zafiropulo, P. *On Communicating Finite State Machines.*

[Brogi et al, 04] Brogi, A.Canal, C. Pimentel, E. Vallecillo, A. Formalizing web services choreographies. *Proceedings of the 1st International Workshop on Web Services and Formal methods.*

[Bussler,2003] Bussler,C. B2B Integration. Concepts and architecture. Springer-Verlag,2003.

[Clarke et al, 96] Clarke,E. Wing, J.M. *Formal methods: state of the art and future directions.* Technical report. CMU-CS-96-178. Carnegie Mellon University.

[Chinnici et al., 2003] Chinnici R., Gudgin M., .Moreau J.J, Schlimmer J. and Weerawarana J.: *Web Services Description Language (WSDL)* Version 2.0 Part 1: Core Language, W3C Working Draft, 2003, available at: http://www.w3.org/TR/2003/WD-wsdl20-20031110/

[Glasser et al,04] Glasser,U. Gurevich,Y. Veanu,M. *An Abstract Communication model.*Microsoft Technical report.

[Gudgin et al., 2003] Gudgin M., Lewis A. and Schlimmer J.: *Web Services Description Language (WSDL) Version 2.0 Part 2: Message Patterns*, W3C Working Draft 10 November 2003, available at: http://www.w3.org/TR/2003/WD-wsdl20-patterns-20031110/.

[Haas & Brown, 2003] Haas H. and Brown A.: *Web Services Glossary*, W3C Working Draft, 2003

[Hamadi et al,04] Hamadi R.,Benatallah B.. *A Petri Net based model for Web Service Composition.* Fourteenth Australasian Database Conference (ADC2003).

[Hoare, 85] Hoare, C.A.R. *Communicating sequential processes*. Prentice-Hall. London,1985.

[Hohpe & Woolf, 2003] Hohpe G. and Woolf B.: *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*, Pearson Education, 2003.

[Huth & Ryan, 1999] Huth M. Ryan M.: *Logic in Computer Science: Modelling and reasoning about systems*, Cambridge University Press, Cambridge, 1999.

[Kavantzas et al, 04] Kavantzas, K. Burdett D. Ritzinger G. W3C Working Draft. *Web Services Choreography Description Language*. Version 1.0, 27 April 2004.

[Keller, 76] Keller,R. *Formal verification of parallel programs.* Communications of the ACM,19.pp.371-384

[Milner, 80] Milner,R. *A Calculus of communicating systems.* Number 32 in Lecture Notes in Computer Science.Springer-Verlag,1980

[Plotkin,81] Plotkin,G. *A structural approach to operational semantics.* Report DAIMI FN-19, Computer Science Department, Aarhus University,1981.

[Roscoe, 98] Roscoe,A.W. *A theory and practice of Concurrency*. Prentice-Hall. London,98

[Roman et al., 2004] D. Roman, H. Lausen and U. Keller: *Web Service Modeling Ontology - Standard (WSMO - Standard)*, WSMO Working Draft, 2004, available at: http://www.wsmo.org/2004/d2/v02/20040306/

[Sangiorgi et al, 04] Sangiorgi,D. Walker.D. *The Pi-Calculus: A theory of Mobile Processes.*Cambridge Press.

[Thatte et al,03] Thatte. D. (Ed) Andrews T. Curbera, F., Goland, Y., Klein, J., Leyman, F., Soller, D., Thatte, S., Weerawarana, S., *Business Process Execution Language for Web Services.*