A Pervasive Service Conversations Framework for Ubiquitous Computing

Juan Miguel Gómez-Berbís, Ricardo Colomo-Palacios, Ángel García-Crespo
Departamento de Informática
Universidad Carlos III de Madrid, Spain.
{juanmiguel.gomez, ricardo.colomo, angel.garcia}@uc3m.es

Christoph Bussler
San Jose, California, USA
chbussler@bea.com

ABSTRACT

Pervasive Service Conversations have become a major research challenge for Ubiquitous Computing. They can leverage the ability of companies to streamline business activities, directly or through marketplaces as sellers and buyers, with their trading partners. This paper presents a Pervasive Service Conversation framework based on a real-world scenario which represents a B2B Pervasive Service Conversation, where formal semantics are required to ensure a specific conversation and a common understanding among business partners. The research provides a complete framework based on a conceptual model, BNF syntax, execution semantics, architecture and implementation, which enable unambiguous Pervasive Service Conversations for Ubiquitous Computing, with a focus on B2B transactions.

Keywords: Pervasive Services, Ubiquitous Computing, Service Conversation Framework, SOA.
1 INTRODUCTION

Pervasive Service Conversations have evolved into a *sine qua non* requirement for Ubiquitous Computing, being the new vehicle for business transactions and information exchange. Particularly now when large organizations share or expose services, encapsulating several business processes into a Pervasive Service or enabling the creation of a new value-added Pervasive service by means of the conversation of several Pervasive Services are vital breakthroughs of this trend in technology.

More concretely, the Business-to-Business (B2B) domain [4] raises a challenge. If two business partners want to communicate, they have to determine which role they are playing, what is the format of the messages exchanged, and what is more important, what are the temporal and order constraints for the sequencing of the messages. Similar handicaps appear in multi-agents systems communications [10], where formal languages are needed for fluent communication between heterogeneous agents. The key for successful eCommerce is the ability of business partners to follow a conversation in a meaningful and formal way. A conversation is a behavioral specification of a business transaction among business partners.

Pervasive Service Conversations may become one of the cornerstones of this paradigm. Pervasive Services are loosely coupled software components published, located and invoked via Ubiquitous Computing. Organizations expose their services via Pervasive Services which are self-described and autonomous without specifying the interactions among them or specifying them in an informal way. Hence, conversations among Pervasive Services lack of formal semantics is what prevents the ideal situation of common understanding among the business partners and ensuring specific properties of a conversation.

The research problem addressed in the current paper is how to properly enact Pervasive Service conversations by means of a framework deemed appropriate to encompass different behavioral models and semantics. The goal of this research is to propose an approach to enact conversations (and more in detail, a B2B conversation) via Pervasive Services using a framework based on a conceptual model, BNF syntax, execution semantics, architecture and implementation.

The remainder of this paper is structured as follows. Section 2 introduces a common real-world problem scenario which illustrates the problem faced. Subsequently, a set of requirements is extracted to overcome the caveats of the real-world scenario. Section 3 presents the model proposed by this work, the Pervasive Services Conversation Framework (PSCF), which provides a conceptual model, a particular BNF syntax, Petri Net based Execution Semantics, architecture and implementation. This section also provides a description of the core languages, harnessing the potential of the languages’ well-grounded formal execution semantics, architecture and proof-of-concept implementation. Finally, Section 4 concludes the paper and discusses the related work.

2 MOTIVATING SCENARIO

The goal of B2B integration systems [4] is to facilitate inter-organizational business information exchange. This enables efficient implementation of business processes that span multiple independent organizations. Consequently, business partners with potentially different technical capabilities must agree upon a conversation, that is, a certain behavioral specification of how to communicate.

The problem being dealt with in this work is the enactment of this conversation and its actual realization. It is crucial for business partners to succeed in describing this conversation, and current technology limitations do not permit the development of an efficient solution to this problem. To illustrate the problem being addressed, the next section presents a simple B2B scenario. This scenario is based on a real-world problem, and it is useful to extract a set of requirements for modeling B2B conversations. The next section provides a brief overview of the scenario and consequently defines the resulting requirements.

2.1 Problem Scenario

In the current scenario, a customer wants to buy a particular product from a supplier. To achieve this goal, two different behaviors interoperate: the behavior of the customer and the behavior of the supplier. The behavior of the customer follows three logical steps. Firstly, he will request the price of the product by means of a Request for Quotes (RFQ). A RFQ is a business term for the request of a customer about the price of a particular product. Secondly, the customer will make a decision. If the price of the product suits his expectations, then he proceeds to order the product. Otherwise, he does not proceed to order the product. Finally, the customer makes the payment for the product and receives a delivery confirmation.

From the perspective of the supplier, there are also three steps. Firstly, the supplier receives a RFQ and replies with a quote to the customer. Secondly, the supplier then verifies if it is feasible to provide the products to the customer by checking the availability of the products. If this is the case, the customer order is confirmed and the payment of the product is expected. Finally, the payment is realized and the supplier confirms the delivery. Once the supplier confirms the delivery, the business process is concluded.

In the next section, a more detailed storyboard of this scenario is considered, focusing on the different steps of the conversation. Each step is enumerated and described, extracting the requirements considered of interest.
2.2 Discussion of Requirements

There are several requirements to be taken into account in the scenario. A graphical representation of the storyboard is shown in Figure 1. The legend of the figure is as follows. The squares represent the participants of the conversation, the customer and the supplier. The line under the squares represents a timeline which implies the chronological order of the different interactions, that is, the interactions that happen at the beginning of the timeline happen before the ones at the end. The arrows represent an interaction from a sender to a receiver, following the direction indicated by the head of the arrow. The label of the arrow represents the name of the interaction. For instance, if a Request for Quotes (RFQ) arrow is headed from the customer to the supplier, this means that the customer sends an RFQ and the supplier receives an RFQ.

![Figure 1. Storyboard of requirements.](image)

A partner is considered one of the entities involved, in other words, a participant of the conversation. A partner plays a specific role in a conversation, 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 behavior modeled becomes an important issue, as described in [11].

Requirement 1: 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 synchronize. In this 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.

Requirement 2: History

In B2B [4], a history is defined as an explicit representation of past state 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 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.

Requirement 3: Dead-lock freedom

The need for synchronization in a conversation leads to difficulties to be considered in the design of an interaction-based system. The problems with the most significant negative effect in terms of synchronization 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.

Requirement 4: 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]. Execution semantics describe how the program evolves and behaves, but they are more efficient when described by a formal method. A formal method must be 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.

3 A PERVASIVE SERVICE CONVERSATION FRAMEWORK FOR UBIQUITOUS B2B TRANSACTIONS

This section defines, describes and specifies a Pervasive Service Conversation Framework (PSCF). PSCF encompasses pervasive service conversations where multiple items can be sent. PSCF provides a conceptual model, BNF syntax, execution semantics, architecture and implementation.

3.1 PSCF Conceptual Model

The conceptual model of PSCF is described in the UML diagram depicted in Figure 2. This conceptual model has seven classes. A SendReceive class represents the communication fact, i.e., a type of transmission which sends a ListOfTObjects and receives a ListOfTObjects. A SendReceive transmission has an id property, an identifier. A ListOfTObjects represents certain number of TObjects, a TObject being the actual element which is being transmitted. Partner 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.

![Conceptual model of PSCF](image)

**Figure 2.** Conceptual model of PSCF.

### 3.1.1 PSCF BNF Syntax

PSCF is described as an EBNF (Extended Backus Nauer form) [14] syntax. This syntax is implemented with SableCC [9], a lexical analyzer and parser creator. Following the actual conceptual model of PSCF, a *SendReceive* consists of an identifier, and it is related to a Source, multiple transmission objects *TObject* (more than two, in principle) and a *Target*. The *History* of a *SendReceive* might also be required. Here only the tokens and production rules of PSCF are outlined.

**Tokens**

- `sendreceive = 'sendreceive';`
- `partner = 'partner';`
- `tobject = 'tobject';`
- `history = 'history';`

**Productions**

```
PSCF = P.partner+ P.obj_list+ P.sendreceive+ P.history+ ;
```

```
sendreceive = T.sendreceive [name]:id [source]:id [target]:id [tlist]:P.obj_list [rlist]:P.obj_list ;
```

```
partner = T.partner id ;
```

```
obj_list = {single} id | {multiple} lbracket id+ rbracket ;
```

```
history = T.history id ;
```

The production rules determine that the language PSCF must be composed of at least one *Partner*, and two lists of *TObjects* and one *SendReceive* instance declaration. Given the consideration in the conceptual model of an element called *ListOfTObjects*, at the syntax level a new element *obj_list* (corresponding to *ListOfTObjects*) is included. This *obj_list* can have one single identifier *id* (in the case that the list is composed of only one element) or by a set of elements, each of them represented by one identifier. Each element identifier goes between the right and left bracket (specified in the BNF syntax complete listing by *lbracket* and *rbracket*, respectively).

As previously discussed, at least one *Partner* is sufficient for the *SendReceive* to actually be realized if the Partner wants to transmit to itself and be responded to by itself. If there were no *Partners* defined, the *SendReceive* could not take place.

In this particular case, each of the *TObjects* identified at the syntax level becomes an instance of the *TObject* concept (this concept belonging to the metamodel layer) and the instances defined in the model layer belonging to the information layer. Consequently, the set of *Targets* identified at the syntax level becomes instances of the *Target* concept and the instances defined in the model layer belonging to the information layer.

### 3.1.2 PSCF BNF Example

An example of the use of PSCF is a B2B conversation that takes place between two companies, Company A and Company B, in which several products are bought. Company A wants to send a Purchase Order (PO) for each product to Company B. Both companies have agreed previously upon sending a confirmation of the purchase by means of a Purchase Order Acknowledgement (POA) for each product. At a certain point in time, Company A wants to obtain some information from the *SendReceive* ‘t1’. The conversation in PSCF is as follows:

```
partner 'Company A'
partner 'Company B'
tobject 'PO1' 'http://cool.deri.org/PO1'
tobject 'PO_ACK1' 'http://cool.deri.org/POACK1'
tobject 'PO2' 'http://cool.deri.org/PO2'
tobject 'PO_ACK2' 'http://cool.deri.org/POACK2'
sendreceive 't1' 'CompanyA' 'CompanyB' [ 'PO1' 'PO2'] [ 'PO_ACK1'
```
For the PO of the first product, Company A receives a PO_ACK1 and for the second product it obtains a PO_ACK2. Company B is then replying each of the Purchase Orders from Company A. On the other hand, the sending and the replies are happening concurrently, that is, the Purchase Orders are sent at the same time. However, this concurrent aspect is illustrated more clearly in the execution semantics detailed in the section below.

### 3.1.3 PSCF Execution Semantics

This section discusses the observable behavior of the participants of the conversation, **Source** and **Target**. Each participant goes through a number of states during the conversation, as shown in Figure 3, Figure 4, and Figure 5. The states of the participant change each time that a transition takes place.

In PSCF, there are multiple transitions happening in a **SendReceive**. Each transition corresponds to a **TObject** which is sent by the **Source** to the **Target** or when a **TObject** is sent from the **Target** to the **Source**. Hence if there are “n” **TObjects** sent by the **Source** to the **Target** and “m” **TObjects** sent from the **Target** to the **Source**, there will be “n+m” transitions in a PSCF **SendReceive**.

In PSCF, the current framework models the states of the **SendReceive** (and more concretely, the states of the participants) using the places of the Petri Net. Each place represents a state of a participant. A place might represent also the state of two participants (as is the case with the Petri Net describing the conversation from a global perspective).

Eventhough a **SendReceive** is represented by a number of transitions of the Petri Net, each transition occurs when a **TObject** is sent. This means that the sending of several **TObjects** (from the Source to the Target or vice versa) is represented as multiple transitions. Since PSCF introduces concurrency, in other words, multiple **TObjects** are sent at the same time, and the Petri Net transitions are naturally concurrent, when multiple transitions are linked by arcs to two places, this means these transitions happen concurrently, hence there is no need for further clarification.

A token represents a **TObject**. The motion of a token from one state to another represents that a participant of the conversation had a **TObject** in the first state and, after the transition, it still has a copy of the **TObject** for subsequent purposes. However, the particular and precise **TObject** that has been sent is not in the participant anymore.

Depending on the perspective of the **Source**, the **Target** or a global view (see Section 3.1.4), three different diagrams may be included. The diagram depicted in Figure 3 shows the PSCF execution semantics from the **Source** perspective. There are four places: the first place represents the state in which **Source** has a set of **TObjects**. The multiple transitions are fired and the second place represents the state reached when **Target** receives the set of **TObjects** (not explicit in the diagram, since it only considers the **Source**) and **Source** is empty (or stores copies of the **TObjects**, as clarified above). The third place represents the state in which **Source** is empty, so it has no **TObjects** and the multiple transitions are fired. The fourth place represents the state reached when **Target** has sent its multiple **TObject** and **Source** has received them.

The diagram depicted in Figure 4 shows the PSCF execution semantics from the **Target** perspective. There are four places: the first place represents the state in which **Target** is empty. The multiple transitions are fired and the second place represents the state reached when **Target** receives the multiple **TObjects** and is not empty anymore. The third place represents the state in which **Target** has the multiple received **TObjects** and the multiple **TObjects** it is intending to reply. Multiple transitions are fired, and the fourth place represents the state reached when **Target** has sent their **TObjects** and **Source** has received them.

---

Ubiquitous Computing and Communication Journal
Figure 4. CPN Tool [17] diagram of a Petri-Net describing PSCF execution semantics from the perspective of the Target.

The diagram depicted in Figure 5 shows the PSCF execution semantics from a global perspective. There are four places: the first place represents the state in which Source and Target are, i.e., Source has a set of TObjects to be sent and Target is empty. The second place represents the state reached when Target receives the set of TObject (explicit in the diagram) and Source is empty. The third place represents the state in which Target has received the set of TObjects and the set of TObjects to which it is intending to reply. Multiple transitions are fired and the fourth place represents the state reached when Target has sent all its set of TObjects and Source has received them.

3.1.4 Fulfillment of Requirements

The PSCF language fulfills all of the requirements outlined in Section 3.1. Firstly, PSCF distinguishes between the behavior of the Source and the one of the Target (“Partner view” requirement). Both behaviors are modeled by formal execution semantics (“Formal semantics” requirement). In PSCF, deadlock-freedom is preserved by means of the execution semantics. This implies that if a two-way, bidirectional conversation is enacted, the states in which participants of the conversation are sending or receiving must be carefully observed (“Deadlock freedom” requirement). PSCF also incorporates a history of the conversation (“History” requirement). Since a set of TObjects can be sent and received at the same time in a SendReceive in a simultaneous manner, it may be concluded that in PSCF there is concurrency (“Concurrency” requirement). These requirements are shown in Table 1.

<table>
<thead>
<tr>
<th>Requirements</th>
<th>PSCF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concurrency</td>
<td>X</td>
</tr>
<tr>
<td>History</td>
<td>X</td>
</tr>
<tr>
<td>Deadlock–freedom</td>
<td>X</td>
</tr>
<tr>
<td>Formal Semantics</td>
<td>X</td>
</tr>
<tr>
<td>Partner view</td>
<td>X</td>
</tr>
<tr>
<td>Arch+Imp</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 1. Analysis of requirements of PSCF.

Once the analysis has been performed of which requirements are fulfilled, conclusions are drawn in the next section regarding the use of PSCF.

3.2 Architecture

The architecture presented in this work is a Service Oriented Architecture (SOA), which defines a software system consisting of a set of collaborating software components with well-defined interfaces that together perform a task. These components may be distributed and executed in different network locations, connected through different communication protocols. These components can be plugged-in and plugged-out from the system.

Furthermore, an important issue is the communication mechanism. This is not limited to the message-passing paradigm between components, but it can be extended to other indirect communication paradigms.

The subsequent section introduces the general architecture and its features.

3.3 General Architecture
This section introduces three views of the architecture. The first view envisages a scenario where two entities involved in a conversation use the general architecture. The second view for the architecture is provided with a layered design. The advantage of layering is the conceptual distinction and functionality between layers. Each layer has a particular and precise functionality. Layers observe a bottom-up dependency relationship. This type of structure implies that the upper layers rely on some functionality of the lower layers. Layered architectures are used in different domains, but particularly in communication systems where each layer implements a different aspect of the information exchange. For example, the bottom layer usually represents the basic communication event at a physical level, e.g., the physical electrical signals of communication.

Since the architecture presented represents a communication system, a layered approach for this view has been used here. In Figure 6, the layered general architecture developed is shown. The Pervasive Services and the Communication Layer are the ones taking care of basic functionality, while the more complex functionality is situated in the upper layer. The purpose of the Pervasive Services layer is to provide the access to Pervasive Services technology required for the communication to happen, hiding the details of how the Pervasive Services are implemented. The Communication Layer abstracts from the lower layer and its use is twofold. On the one hand, it provides the upper layer, the Application Layer, with what has been received from the other communication entity. On the other hand, it is provided with what must be sent to the other communication entity.

In this view, there are a number of components in each layer. The current section first introduces the three layers, and then discusses each component and its functionality. The three layers are as follows:

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

- **The Communication Layer** aims to abstract all the communication among the Pervasive Services and the upper layers. It consists of two main “boxes” or components, namely, a Sender and a Receiver. The Sender component uses a mechanism of the lower layer to send transmission objects TObECTs. The Receiver component retrieves received TObECTs by the lower layers.

- **The Application Layer** aims at capturing and interpreting the user intention at the language level. It recognizes valid sentences of the language, processes them and interprets them in terms of which language is being used. Finally, it dictates to the lower layer what must be communicated. It also does the same in the other direction, understanding what the other communication entity has transmitted and displaying it to the user. The following components are in the Application Layer:
  - The Lexer or Lexical Analyzer reads an input stream and returns tokens one by one.
  - The Parser recognizes valid sentences of the language by analyzing the syntax structure of the set of tokens passed to it from the Lexical Analyzer.
  - The Semantic Analyzer 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 (when applicable in L3, L4 and L5) records all the history of the system in a persistent 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. In terms of the conceptual model, all elements of such a conceptual model.
The Concurrency Manager deals with concurrency where applicable (L4 and L5). It handles the concurrency in conversations using threads.

Finally, the last view of the architecture is a broad perspective of the components and the interaction among those components. This view is used in the diagram to relate the execution semantics of the languages and their architectures. The general architecture is shown in Figure 7.

![General Architecture](image)

**Figure 7.** General Architecture.

This section above has presented three views of the architecture.

4 RELATED WORK AND CONCLUSIONS

The WWW has become a platform for the support of a wide variety of transactions. B2B has recently become one of the most widespread marketing vehicles, since it generates an enormous transactional volume and it has surpassed traditional marketing mechanisms. Within this environment, Pervasive Services have become a standard that leads to the simplification of the successful integration of applications. Nevertheless, despite the generalization of Pervasive Services and the ever-growing relevance of semantics applied to modeling B2B conversations, no solutions are currently available which are able to fulfill all of the requirements proposed in this work. Pervasive Service Conversation Framework (PSCF) is a framework that allows leveraging its different components, providing a set of features that, being considered as a whole, mean a novel, promising, and effective solution to the ubiquitous transactions problem in B2B environments. Other proposals such as WS-Coordination [5] and WS-Transaction [6] provide a framework for the enforcement of transactional conversations properties, but they do not provide any consistent conversation model.

One of the most important challenges for B2B is the interaction with internal and external applications and data [12] and advances in this area will facilitate the automatic discovery and invocation of relevant services [3]. In response to this challenge several efforts have focused on effectively supporting cross-organization application integration[16]. For instance, BPEL4WS standard [8] proposes a model for composing Web services as well as a way to define the conversations that a particular service supports [3]. Another related area of interest is the extension of traditional transaction techniques to provide reliable and dependable execution of integrated services. Within this area, two proposals should be outlined. The first of them is OASIS Business Transaction Protocol (BTP) [15], and the second one is WSCL [2]. WSCL builds upon WSDL [7] to describe valid interactions that a service can support, but focuses only on which are the acceptable messages and the order in which they should occur. Another approach to the problem of managing Web services’ dialogues is the inclusion of a middleware to monitor and control client transactions according to transactional capabilities of provider services as proposed in [13]. However, the current work provides formal execution semantics, also called operational semantics, as an added value face to these approaches.

Several lines of work emerge from this paper. The authors’ effort is currently focused on the development of a customized version of the environment for several particular applications in both organizational and cross-organizational environments. This solution will be aimed to be implemented in Supply Chain Management and Customer Relationship Management.

5 REFERENCES


