Modeling and using business collaborations

Giorgio Bruno

Dip.Automatica e Informatica
Politecnico di Torino, Torino, Italy
giorgio.bruno@polito.it

Abstract. Business processes interact with each other by sending and receiving messages; how such interactions take place (i.e. in which order and with which timing constraints) has to be defined in advance with a collaboration model, so the parties can be developed separately in conformity with it. With orchestration languages, however, the role of a collaboration model is that of an interface, so it is the run-time responsibility of the activities of the business process to guarantee that messages are exchanged in proper order and time.
This paper motivates the need of run-time collaboration instances, which, by keeping the state of ongoing collaborations, relieve business processes of the burden of checking the conformity of the actual operations with the collaboration model. Moreover this paper presents a software environment, called bProgress, which we have developed so as to experiment with business processes and collaboration ones.

1 Introduction

At the basic level of e-business, organizations interact by sending and receiving messages, however in order to attain a given common goal, the parties have to exchange a number of messages with appropriate content and order. For this reason it is important to focus the attention on all the messages exchanged for a particular purpose rather than on single interactions. The term collaboration (or conversation) is used to refer to the flow of messages taking place between two parties for a given purpose; it is a particular view on the whole set of the messages exchanged in a multi-party system.

The parties involved in a collaboration are often referred to as services or business processes or simply software applications. Usually one party takes the role of the "provider", the other that of the "requester".

A well-known example of collaboration is the purchasing of goods or services, which can be described as follows: organization A's purchasing service (the requester) sends a request for quote (rfQ) to organization B's selling service (the provider), which responds with a quote; if the purchasing service accepts the quote, it will then send an order to the selling service. This example will be used throughout this paper.

Although, from an external point of view, a collaboration is a flow of messages (such as the flow consisting of an rfQ, a quote, and an order), it is perceived by each

party as an ordered flow of message-oriented activities. For the selling service the above-mentioned collaboration implies that first it has to receive an rfQ, then it has to send a quote, and finally it might receive an order.

The actual flow of messages can vary from case to case. In fact the selling service might not reply or the quote might not be accepted and in that case the order won't be sent. Therefore, in one case the collaboration might consist of only an rfQ message, while in another case it might be made of all the three messages (rfQ, quote and order).

Anyway an actual collaboration can take place only because the parties in advance have agreed on how to put it into practice. Such an agreement is based on an abstract representation of the collaboration, i.e. on a collaboration model.

There is growing interest in modeling collaborations as processes as a process denotes an ordered flow of activities. Moreover the parties involved in collaborations can be modeled as processes as well since they, too, consist of ordered flows of activities.

Although similar in structure, the processes used to model collaborations are different from those related to services or applications; in this paper the former will be called collaboration processes, the latter business processes.

The activities in a business process fall into three major categories: manual activities, automatic activities and control-flow ones. Manual activities require human intervention and they are usually performed through a web-based user interface, so their implementation is outside the scope of the business process. Automatic activities and control-flow ones, instead, are in charge of the process (either directly or indirectly) as will be shown later on. For example the selling business process might consist of the following activities: receiving an rfQ, preparing a quote, sending it to the requester, receiving an order, and processing the order; the activity of preparing a quote and that of processing the order probably require some human intervention (hence they are manual), while the others could automatically be performed by the system.

A collaboration process, instead, is meant to be a contract between the parties, therefore it is mainly made up of message-oriented activities, whose purpose is to show which messages have to be sent or received.

As an example WSCL [1] presents collaboration processes as UML activity diagrams in which activities fall into four categories: ReceiveSend, Receive, Send, and SendReceive. Messages carry XML documents which are defined in appropriate schemas, and the actual protocols are defined using WSDL. The collaboration is presented from the point of view of the provider, the requester's one being easily obtainable if sending activities are turned into receiving ones and vice versa.

While business processes are intended to be operational, collaboration processes are meant to specify the behavior, in terms of the messages to be exchanged, the business processes have to conform with.

In fact with orchestration languages, such as BPEL [2] and BPML [3], business processes are executable, so they exist at run-time in the form of business process instances, just as types exist in the form of variables. Collaboration processes, instead, do not have real-time counterparts, since they are interpreted as interfaces, which are defined in a choreography providing the global view of all the interactions in the system. Special languages, called choreography languages, such as WSCI [4] and

WS-CDL [5], have been defined for that purpose. As an interface a collaboration process is to be implemented by a provider and used by a requester.

However, since it is extremely difficult, in general, to statically prove the conformity of a business process to the collaboration process it has to implement or use, it is the responsibility of the activities of the business process to guarantee proper behavior at run-time. As an example assume that an rfQ has a deadline after which the quote won't be taken into account by the requester; then the burden of deciding whether it is worth sending the quote or not is entirely on the provider. If there were a collaboration instance (i.e. a run-time representative of the collaboration process, which knows the state of the collaboration) that decision could automatically be taken by it, so the activities of the provider's business process would be simplified. A collaboration instance can be thought of as a channel through which a party sends and receives messages belonging to a given collaboration.

The purpose of this paper is to motivate the need of run-time collaboration instances, which, by keeping the state of ongoing collaborations, relieve business processes of the burden of checking the conformity of the actual operations with the collaboration model. In addition collaboration processes can be made more flexible with the introduction of parameters, which can be tailored to actual collaborations; such parameters need to be kept in collaboration instances.

Moreover this paper presents a software environment, called bProgress, which we have developed so as to experiment with business processes and collaboration ones; it consists of modeling tools and of services for implementing business processes and collaborations.

This paper is organized as follows. Section 2 describes how collaboration processes are modeled with bProgress, while section 3 shows how business processes use collaboration instances. Then the bProgress environment is illustrated in section 4 and finally a comparison with related work is presented in section 5.

2. Collaboration processes

Basically a collaboration process consists of message-oriented activities placed within a control structure providing for sequential, alternative, and parallel paths as well as for timeout-related paths. In bProgress processes are represented as UML activity diagrams.

The model shown in Fig. 1 is the selling collaboration process introduced in the previous section.

A message-oriented activity is a receiving activity (such as receiveRfQ) if it is connected to an input signal, it is a sending activity (such as sendQuote) if it is connected to an output signal. For simplicity we consider asynchronous messages only. Messages are represented by signals, and each message has a name and a type (not shown in Fig. 1). The types of the messages are defined in an XML schema associated with the collaboration process. Each message-oriented activity has a deadline (such as t1), which appears in the label of the outgoing timeout transition. When a deadline has been reached, a timeout exception occurs and the activity fails.

Unlabelled transitions represent precedence contraints; timeout transitions (i.e. those labelled with the keyword timeout) show the effects of timeouts; in this example timeouts end the collaboration. The structure of a collaboration process can get much more complicated if alternative paths or parallel ones are needed.

Collaborations can have parameters; in this case the three deadlines are parameters.

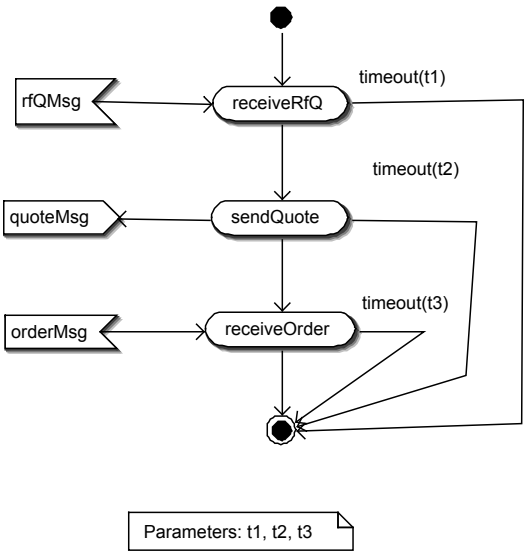

Fig. 1. The selling collaboration process

Parameters have to be agreed upon by the parties before an actual collaboration is made effective; so there is the need of a preliminary phase in which parameters are set, this phase being itself a special case of collaboration. If the preliminary phase is successful, the collaboration is started.

The model in Fig. 1 shows that, after the collaboration has been started, the provider is ready to receive an rfQ message (rfQMsg) sent by the requester; if that message does not arrive before instant t1, a timeout will occur and the collaboration will end. After the rfQ message has been received, the provider has to reply with a quote message; if it doesn't so before instant t2, a timeout will occur and the collaboration will end. After the quote message has been sent, the provider is ready to accept the order message but, if it is not received before instant t3, a timeout will occur and the collaboration will end.

The three deadines, t1, t2 and t3, are parameters, so they can be tailored to the actual collaboration.

From an operational point of view, a collaboration process requires two representatives, a requester collaboration process instance, rcpi, on the sender's side and a provider collaboration process instance, pcpi, on the provider's side. A pcpi operates in conjunction with a provider business process instance (pbpi), whilst an rcpi operates in conjunction with a requester business process instance (rbpi).

Activating a new collaboration requires a standard protocol at the end of which parameters have been agreed upon and both the rcpi and the pcpi have been started (and they know of each other).

From the point of view of the provider the selling collaboration takes place as follows. After the collaboration has been activated, the provider (i.e. the pcpi) can receive an rfQ message; when it receives it, it will pass it to the business process (i.e. the pbpi). Then the provider can get a quote message from the pbpi; when it gets it, it will send it to the requester (i.e. the rcpi). Finally the provider is able to receive an order message; if it receives it, it will pass it to the pbpi. The purpose of a receiving activity is to pass the related business process instance the message received from the requester, whilst the purpose of a sending activity is to send the requester the message got from the business process instance.

The requester interprets the same collaboration process by turning sending activities into receiving ones and vice versa.

Collaboration process instances accomplish several tasks: they perform the actual sending and receiving of messages and maintain the state of the collaboration (by keeping local states aligned with each other) as well as the history of the messages exchanged. They prevent a party from sending a message in the wrong order or at the wrong time; for example, if the provider tries to send a quote after deadline t2 has expired, its collaboration process instance will reject it. In addition, their presence automatically guarantees the correlation of messages to business process instances, thus relieving the process activities of that burden.

3. Using collaborations

Collaborations in bProgress are managed by business processes and this section illustrates how this is done.

A business process is made up of a number of activities, which fall into three major categories (exception handling is not considered, for simplicity): manual activities, automatic activities and control-flow ones.

Manual activities require human intervention and they are usually performed through a web-based user interface, so their implementation is outside the scope of the business process. Automatic activities and control-flow ones, instead, are in charge of the process, in the sense that the process will use software components (called process-support components) able to perform such activities. In bProgress we can associate actions (i.e. blocks of code) with the model's activities so as to give a complete description of the business process; a code generation tool of the bProgress environment is able to produce process-support components incorporating those actions within suitable contexts.

There are two business processes interested in managing selling collaborations, the purchasing business process and the selling business process; the former is shown in Fig. 2, the latter in Fig. 4.

The purchasing business process operates as follows. The first activity, as it consists in preparing an rfQ, is a manual activity to be performed by a purchaser. A manual activity is connected to the role required. For example activity prepareRfQ

requires a purchaser; this means that it has to be performed by a person playing that role. The next activity, sendRfQ, has to send the rfQ to all the suppliers invited, i.e. the suppliers that have been chosen in the previous activity. Quotes will be received, one at a time, by receiveQuote. When all the quotes expected have been received, or the deadline of receiveQuote has expired, manual activity selectQuote can be performed. If a quote has been accepted, an order to the supplier will be sent by activity sendOrder. The deadline of receiveQuote is a parameter, so it can be tailored to the specific purchasing process instance.

It results from the above description that business processes cannot be described separately from the business entities they are related to; so it is necessary to explain the nature of such inter-relations before the details of the activities are presented.

Business entities are related to each other and such relationships are usually presented in an entity-relationship (or class-relationship) model, such as the data model of the purchasing organization shown in Fig. 3.

From this model it is evident that an rfQ is connected to the suppliers invited (i.e the organizations invited to the purchasing auction), as well as to the quotes received.

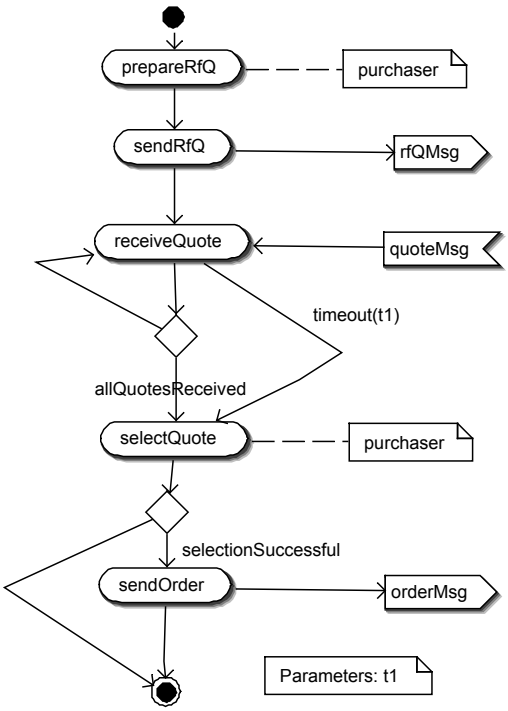

Fig. 2. The purchasing business process

In bProgress business process instances are business objects, too, and they are represented by an entity called BPI (business process instance). Likewise, actual collaborations are business objects and they are represented by entity RCPI (requester collaboration process instance) if they refer to a requester-side collaboration, or by entity PCPI (provider collaboration process instance) if they refer to a provider-side collaboration. A BPI can be involved in multiple RCPIs or PCPIs; however a given RCPI (or PCPI) is connected to just one BPI (the one involved in the collaboration).

Of course a process instance is related to the corresponding process model but such connections belong to a different level and are outside the scope of this discussion.

In general in bProgress a business process is meant to take care of a given business entity and, by extension, of those associated with it, as in the case of the purchasing process which manages the lifecycle of an rfQ. For this reason a business process instance (BPI) can be connected to any business object implementing interface WFItem (workflow item); entity RfQ implements that interface, so an rfQ is process-driven (in other words it is a controlled business object).

In general, the actions associated with automatic activities and control-flow ones operate on the business objects through a set of classes providing an object/relational mapping.

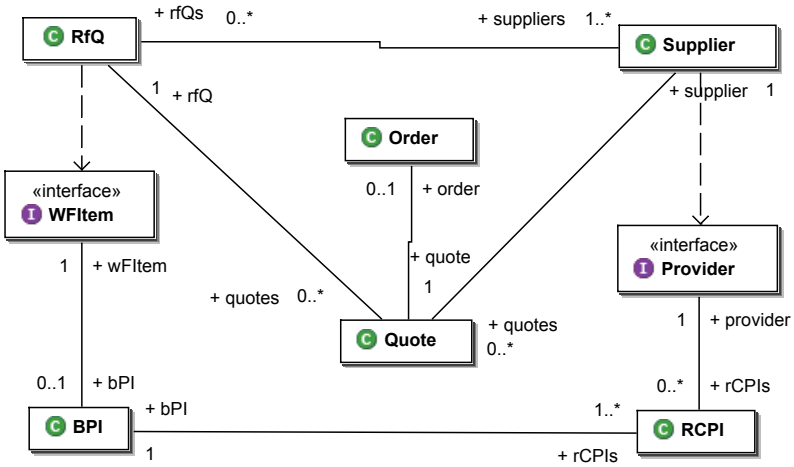

Fig. 3. The business entity model of the purchaser

In fact, while business objects are mapped onto records in a relational database, such records are not acted on directly as it would be too awkward and error-prone. They are, instead, operated on through an object/relational interface consisting of a set of classes, which provide methods for generating, retrieving and modifying the records and also methods for navigating among the records along the relationships defined in the business entity model.

The actions are presented below as blocks of pseudo-java code. Each action must be considered to be placed in a context including a reference (bpi) to the current business process instance and a reference to the controlled business object; the name of the second reference is that of the class of the controlled business object with the initial in lower case (such as rfQ). At run-time those references will be properly initialized by the bProgress process engine (cf. next section). The actions of the activities of the purchasing business process are as follows.

sendRfQ:

```
for Supplier s in rfQ.getSuppliers() {
  RCPI rcpi = new RCPI("SellingCP", s, bpi);
  rcpi.addParameter("t1", rfQ.getT1());   //idem for t2 and t3
  rcpi.start();
  if (rcpi.getState().equals("accepted"))
            rcpi.sendMsg("rfQMsg", rfQ.getMessage());
  else rcpi.remove();
}
```

receiveQuote:

```
Quote q = new Quote(quoteMsg, rfQ);
```

allQuotesReceived:

```
return rfQ.getQuotes().size() == bpi.getRCPIs().size();
```

selectionSuccessful:

```
return rfQ.getQuote("state = 'accepted'") != null;
```

sendOrder:

```
Quote q = rfQ.getQuote("state = 'accepted'");
Order o = new Order(rfQ, q);
for RCPI rcpi in bpi.getRCPIs() {
  if (rcpi.getProvider().equals(q.getSupplier()) {
            rcpi.sendMsg("orderMsg", o); break;
  }}
```

Activity sendRfQ invites suppliers to submit quotes; the suppliers to be invited are those that have been associated with the current rfQ in manual activity prepareRfQ. For each supplier a new collaboration is activated and an rfQ message is sent.

Method getSuppliers is a navigational method, which retrieves all the suppliers (business objects) related to the current rfQ. A supplier implements the Provider interface: this is a synthetic way of indicating that a supplier business object has all the information (e.g. the url of the service) needed to establish a collaboration with the corresponding organization.

For each supplier to be invited a new requester collaboration process instance (rcpi) is generated; the constructor of class RCPI takes as parameters the name of the collaboration process, the provider with which the collaboration has to be established, and the current business process instance. For simplicity we assume that each supplier supports the selling collaboration process shown in Fig. 1. The values of the three parameters are then added to the rcpi; they are taken from the rfQ business object in which they have been defined by the purchaser during activity prepareRfQ. Next the collaboration is started with a synchronuos interaction with the provider. If the collaboration is accepted, the rfQ message will be sent; method sendMsg is provided by class RCPI and takes two parameters, the name of the message and its content (which is provided by method getMessage of class RfQ). Method sendMsg performs all the checks required, in particular it won't send the message if it is late, and in that case it will return a negative result.

Activity receiveQuote is in charge of receiving a quote message; then it transforms the message into a quote business object and connects the latter to the supplier and to the rfQ (this is actually done by the constructor of Quote).

Activity allQuotesReceived returns true if all the quotes expected have been received (i.e. there are as many quotes as the number of collaborations started).

Activity selectionSuccessful returns true if there is a quote whose state is "accepted". The selection is made by manual activity selectQuote; the best quote, if any, has been put into state "accepted", while the others remain in state "received".

Activity sendOrder sends the order to the winner: it searches the collaboration process instances to find the one related to the supplier of the winning quote.

The code of the selling business process is similar and is not shown. A short account of its activities is as follows. After the rfQ has been received, manual activity prepareQuote is enabled (and scheduled for an account manager). The quote is then sent. If the order is received, it will be processed otherwise the quote will be closed. The timeouts of the receiving activities are those signalled by the provider collaboration process instance; for this reason the values of timeouts are not specified in their labels.

When a provider accepts a request for a new collaboration, it starts a provider collaboration process instance and it also normally activates the corresponding provider business process instance (and associates the two).

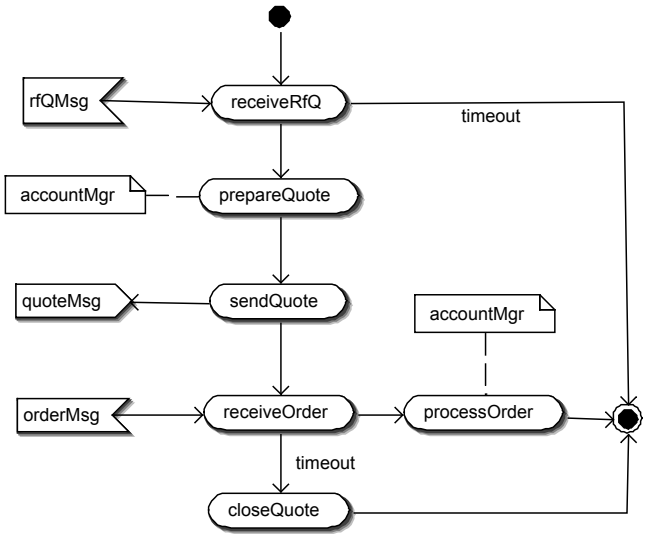

Fig. 4. The selling business process

4. The bProgress environment

We have developed a java environment (based on Eclipse), called bProgress, in order to prototype business processes and collaborations.

The three types of models presented in this paper, i.e. business entity models, business processes and collaboration processes have a graphical UML representation (based on EMF activity diagrams) and also an XML representation. The XML representation can be written directly or can automatically be produced from the XMI version of the UML models.

There are two code generators in bProgress. One takes the XML representation of a business entity model and produces: a) the scripts for generating the corresponding relational tables; b) a set of classes providing an object-oriented interface to those relational tables. The other code generator takes the XML representation of a business process (including the code of automatic activities) and produces the corresponding process-support software component.

Business and collaboration processes are loaded into the database by the process loader. The environment includes a process engine, which manages process instances by interpreting the corresponding business processes. Collaborations are managed by specific bProgress components which use the SOAP protocol and run on top of the JBoss application server.

Transactional aspects in collaborations is of primary importance as shown by the OASIS ebXML [6] and BTP initiatives [7]; and bProgress is meant to support experimentation in that research field. At the moment we address binary transactions

by having receipt signals and acceptance signals automatically managed at the implementation level; current work is being devoted to multi-party transactions for which we envisage a new kind of processes, i.e. coordination processes acting as a bridge between business processes and collaboration processes.

5. Comparison with related work

Business process languages fall into two major categories, orchestration languages and workflow languages, each category emphasizing a particular point of view.

Orchestration languages, such as BPEL [2] and BPML [3] are mainly devoted to supporting interactions between services described in WSDL and ignore manual activities (as first-class constructs); on the contrary, workflow languages such as XPDL [8] support manual activities but are weaker in message processing.

As to the control flow, XPDL has a graph structure where activities (i.e. the nodes) can have multiple incoming/outgoing transitions; although the language has an XML representation, it is difficult to manually write a program made up of nodes and links. Orchestration languages, on the other hand, enforce structured programming through the use of nested blocks.

Collaboration types, in general, are considered to be different from business processes.

WSCI [4] describes collaboration types as interfaces and it is meant to provide a global view (i.e. a choreography) of all the parties involved in a complex interaction. WS-CDL [5] provides more features than WSCI while basically following the same approach.

We believe that collaborations are inherently binary, so, in our opinion, a global picture describing several collaborations provides too much information.

Furthermore we consider collaboration types as abstract processes consisting of message-oriented activities placed within a suitable control structure; deadlines, and hence timeouts, are of primary concern. Parameters are also important as they allow a generic process to be tailored to specific situations.

Collaboration types are shown from the provider's point of view although we envisage a peer-to-peer relationship between the parties. Alternatively it is possible to take a neutral position, as it is done in [9], if the model directly focuses on the message flow rather than on message-oriented activities; however as the direction of messages has to be specified, it is not really different to take one direction, i.e. that of the provider, as the reference one.

Collaboration types are public processes acting as standards the interested parties have to conform to. The notion of public process has been introduced in [10] within workflow languages, as a Petri net showing the interactions between two organizations, one acting as the contractor, the other as the subcontractor. Each organization is responsible of a distinct portion (public subflow) of the public workflow and it can then develop a private workflow which has to comply with the respective public subflow. Rules are given so conformity can automatically be checked.

In bProgress we use run-time collaboration instances, which, by keeping the state of ongoing collaborations, are able to prevent a party from sending/receiving a message in the wrong order or at the wrong time.

In bProgress a business process is definitely a workflow process, so manual activities are first-class activities. The process control structure should be as rich as possible yet amenable to direct writing, so we have selected a number of constructs and represented them with XML tags and UML activity diagrams. Such constructs meet the programming patterns required in workflow applications as shown in [11].

6. Conclusion

This paper has addressed the issue of modeling cross-enterprise collaborations and has presented an approach based on collaboration processes.

Basically a collaboration process consists of message-oriented activities placed within a control structure providing for sequential, alternative, and parallel paths as well as for timeout-related paths.

A collaboration process is a special case of business process and this uniformity entails several advantages both in conceptual and in practical terms.

This paper has also presented the bProgress environment whose tools make the models of business and collaboration processes operational.

References

1. W3C: Web Services Conversation Language Version 1.0. http://www.w3.org/TR/2002/NOTE-wscl10-0020314/ (2002).
2. IBM: Business Process Execution Language for Web Services Version 1.1. http://www-106.ibm.com/developerworks/library/ws-bpel/ (2003).
3. BPMI.org: Business Process Modeling Language. http://www.bpmi.org/ (2002).
4. W3C: Web Services Choreography Interface Version 1.0. http://www.w3.org/TR/2002/NOTE-wsci-20020808/ (2002).
5. W3C: Web Services Choreography Description Language Version 1.0. http://www.w3.org/TR/2004/WD-ws-cdl-10-20040427/ (2004).
6. UN/CEFACT and OASIS: ebXML Business Process Specification Schema Version 1.01. http://www.ebxml.org/specs/ebBPSS.pdf (2001).
7. OASIS: Business Transaction Protocol Version 1.0.9.1. http://www.oasis-open.org/committees/workgroup.php?wg_abbrev=business-transaction (2004).
8. WfMC: Workflow process definition interface - XML Process Definition Language. http://www.wfmc.org/standards/docs/TC-1025_10_xpdl_102502.pdf (2002)
9. Dubray, J-J.: OAGIS implementation using the ebXML CPP, CPA and BPSS specifications Version 1.0. http:// xml.coverpages.org/OAGI-ebXML-WhitePaper-103.pdf (2001).
10. van der Aalst, W.M.P.: Inheritance of interorganizational workflows: how to agree to disagree without loosing control? Information Technology and Management Journal (2002), 195-231.
11. Russell, N., Hofstede, A.H.M. ter, Edmond, D., van der Aalst, W.M.P.:Workflow data patterns. QUT Technical report, FIT-TR-2004-01, Queensland University of Technology, Brisbane (2004).