A language for information commerce processes[1]

Karl Aberer
Swiss Federal Institute of Technology (EPFL)
DSC-LSIR, 1015 Lausanne, Switzerland
email: karl.aberer@epfl.ch
Tel: +41-21-6934679

Andreas Wombacher
GMD-IPSI
64293 Darmstadt, Germany
email: wombacher@darmstadt.gmd.de
Tel: +49-6151-869820

Abstract

Automatizing information commerce requires languages to represent the typical information commerce processes. Existing languages and standards cover either only very specific types of business models or are too general to capture in a concise way the specific properties of information commerce processes. We introduce a language that is specifically designed for information commerce. It can be directly used for the implementation of the processes and communication required in information commerce. It allows to cover existing business models that are known either from standard proposals or existing information commerce applications on the Internet. The language has a concise logical semantics. In this paper we present the language concepts and an implementation architecture.

Keywords: information commerce, process languages, communication languages, business models

1. Introduction

As modern markets move rapidly onto electronic platforms, ecommerce and ebusiness are becoming key terms in today's economy. Ecommerce addresses the trading of physical goods, such as books, food, computers and appliances. *Information commerce*, i.e. trading *information goods*, like news, software, or reports, is even more attractive over electronic channels, since goods can be distributed through the same infrastructure.

We find nowadays many popular examples of information commerce on the Internet. This ranges from commercial services originating in the old economy, like digital libraries provided by scientific publishers, over new economy applications, like auction market places or information portals, to information exchange communities, like Napster or Gnutella. The *business models* underlying these information commerce applications are numerous and complex.

A limitation of the current situation is that users are overwhelmed by an enormous quantity of unstructured, uncertified data. Organizations, professionals but also private citizens willing to gain a profit from the knowledge of information or to rely important decisions on information demand value-added services like personalization, evaluation, categorization and combination of information. In addition, they require that the origin of the information is trusted and that the information is fresh, in order to ensure that they base their decisions on up-to-date and accurate information.

Models and infrastructures for reliable information commerce, constitute an area of active research and development with many technical challenges, such as delivery mechanisms, copyright issues, tamper-

[1] This work has been partially supported by the European Comission under contract IST-1999-10288 (OPELIX) as part of the Information Technology Society program.

resistance, adequate payment facilities, non-repudiation and information quality. We envisage an infrastructure for information vendors who sell specific pieces of information, information mediators, who buy, recombine and resell information, and information brokers who provide directories of information vendors together with added-value information. We assume that information has an associated value, that requires controlled access in an individualized manner and guarantees concerning the quality and authenticity of the information. In such a setting the different properties associated with an information product and the corresponding interaction between buyers and sellers need to be specified in a highly configurable *business process language*. This is an adequate assumption for many application types, like portal sites, electronic new services, stock market information services, software evaluation services, or directory services.

In this paper we introduce a business process language that has been specifically developed to describe *information commerce processes*. Though there exist a plethora of approaches to specify and analyse business models for electronic commerce, we found that none of them exploits the specific properties of information commerce processes sufficiently. In order to specify information commerce processes in a more concise and modular way we factored out common constituents of all information commerce processes:

- Direct representation of the communication acts occurring in a business process: we view information commerce processes as interdependent communications among the participants in which information goods are exchanged. The communication actions are based on the information good specifications and trading roles, and are the elementary constitutent of the model.
- Rule-based process specification: The process specifications are composed from these communication acts by means of condition-action-rules. From the process specification all the message exchanges that are required in an information commerce process are derived. The process specification defines which actions are *possible* to execute.
- Obligation semantics: by associating with certain combinations of communication acts obligations we can distinguish admissible execution of actions from *obligatory* execution. This allows to make actions interdependent in a way that all participants are driven toward the completion of a business process in order to achieve certain execution goals.
- Modelling of implicit state changes based on properties and relationships of information goods: Since information goods can be partitioned and sold easily in many different forms, frequently the same outcome of a trade can be reached in many different ways. Thus the result of an explicit state change incurred by a communciation action could also have been achieved by alternative actions related to an alternative partitioning of the same or related information goods. These equivalent executions can be specified in the language.

Having such an information commerce language makes it easier to develop, analyze and adapt the specifications of information commerce processes. We will describe an implementation architecture that we use for the ongoing implementation of the language. In addition the language is based on a concise semantics given by a logical model. This provides the possibility for the analysis of specifications given in the language, e.g. with respect to executability or obligations.

In the following Section 2 we give an elaborate analysis of related approaches and approaches that incorporate ideas also we are using. This needs to be done in some detail in order to motivate the necessity of a new kind of process language and in order to position our approach. In Section 3 we introduce a working example that on the one hand is used to motivate why conventional process models, like Petri Nets or Flowcharts are not an adequate means to specify information commerce processes and on the other hand will be used as running example to introduce the language concepts subsequently. Section 4 is the key section of the paper as it introduces in a step-wise manner the language. We give also a short account on the language semantics there. In Section 5 we describe an architecture for the language implementation and different implementation issues. In Section 6 we finally point out a number of issues that need to be explored in the future on specifying and managing information commerce process.

2. Related Work

Mechanisms for the specification of interaction processes in electronic commerce are described in many different contexts, both in standardization and research. We summarize in the following some of the important approaches:

- EDI protocols
- Web standardization
- Crossorganizational information systems
- Secure protocols
- Agent communication languages

We analyze for these approaches what mechanisms for process specification they are based on, in what respects they specifically address information commerce and where their limitations with respect to information commerce process specification lie.

EDI protocols: Web-based EDI is gaining substantial momentum with the growing importance of the Web as a B2B ecommerce infrastructure. Many initiatives are currently under way for defining - or redefining - EDI standards for the Web architecture using XML as the underlying data exchange language. Examples are ebXML [EBXML], RosettaNet [ROSETTA], or the Internet Open Trading Protocol [IOTP], just to name a few. These standards are based on predefined instances of data and process models that capture common, standard business processes for electronic commerce. In this way they provide elaborate but static specifications of ecommerce processes as well as their corresponding domain models and message formats. The data models are expressed as XML DTD's or XML schemas. The process models are usually described either informally or are given as conceptual process descriptions, like flowcharts. From the viewpoint of modelling information commerce processes these standards are normally oriented toward the trading of physical goods and do not specifically support information commerce. They capture existing established business models, and due to the lack of a modelling mechanism they do not allow to specify new business models as they occur frequently in information commerce. Recently, in [JP01] a domain model is proposed to provide a standardized desciption of information commerce processes. We view these standards and models mostly as a source of examples for business processes and domain models from which we can generate examples to test the expressivity of the language we develop.

Web standardization: In the context of Web standardization a number of standards have evolved that address different aspects of information commerce. The ones closest to our work are the Micropayment Markup language [MICRO] and the Information and Content Exchange Protocol ICE [ICE]. The Micropayment markup language is in fact a generic mechanism to implement information commerce on the Web. It has, however, a very limited business model which is limited to a pay-per-view model and the use of micropayments. An interesting aspect of the micropayment markup language is the possibility to relate the scope of the validity of a payment to the structure of a Website. It allows to access whole parts of a Website by acquiring access to some entry point. This is a very restricted form of a mechanism, we propose as one of the key elements of our language and that allows to distinguish the direct acquisition of rights for information, by the explicit exchange of a message stating this fact, from the indirect acquisition of rights implied by the contractual context. The information and content exchange standard ICE is a protocol that supports the exchange of content, like catalog data or news items, in B2B applications. It provides a negotiation protocol to determine the delivery modalities and an exchange protocol to perform the information exchange itself. In this respect this standard shares similarity with our language as we also support both a negotiation and delivery mechanism. In addition to its protocols ICE is based on a communication infrastructure for the Web, that we adopt for the implementation of our language. The business processes supported by these two standards are test cases for the expressibility of any generic language for information commerce.

Crossorganizational information systems: Crossorganizational information systems are addressed in standardization initiatives [CORBA], middleware products [BEA], and research projects [GAHL00, KWA99, XFLOW, MGTMWBL98]. In crossorganizational information systems existing autonomous information systems of different companies are directly connected in order to support crossorganizational business processes. In order to deal with the autonomity of the involved information systems, contracts are specified, that specify a common view of the shared data and processes for the crossorganizational business process. The models used to specify contracts are standard conceptual process model, like flowcharts [XFLOW, BEA] or Petri-Nets [MGTMWBL98]. The business language for information commerce that we will propose can be viewed as an example of such a contract language that is used to interconnect content management systems of different companies for the purpose of conducting information commerce.

Secure protocols: Many protocols have been proposed for electronic commerce and information commerce that guarantee secure and fair exchanges. Netbill [NETBILL, TYGAR99] is a protocol performing information good delivery and payment atomically. In [CHTY97] an atomic protocol is proposed that supports anonymity of the consumer. In [ASW98] an optimistic fair exchange protocol for exchanging electronic goods is proposed that requires third-party involvement only for conflict resolution. The verification of the execution guarantees of these information commerce protocols is either model-based or logic-based and is a challenging task. A different approach is taken with DigiBox [INTER], where digital content is strongly protected using special security infrastructures, even when it is resold. This infrastructure supports thus superdistribution [C96]. Recently a language for expressing license rights for DigiBox containers has been proposed [GWW01]. The protocols are in general too restricted in order to allow the modelling of information commerce processes in general. We view these protocols rather as potential atomic building blocks to implement higher-level information commerce processes. The BAKO protocol [GO00] has a slightly different view of providing execution guarantees. It introduces a well defined notion of obligation, that makes clear who is at each point of time responsible for the correct continuation of a business process. Thus, in case of non-compliance it becomes possible to resolve disputes by involving external authorities. We will adopt this idea of assigning responsibility by establishing obligations in our approach.

Agent communication languages: Agent communication languages, like KQML [FFMM98] focus on the problem of communication among autonomous software agents. We can view information commerce from this perspective. These languages correlate the agents internal states with the messages that are intended to change these states, called performatives. They also specify the interaction languages and interaction protocols that can be used to establish agent communciation. There exist approaches to use agent communication languages in order to model electronic commerce business processes, like FLBC [KIMB, WV98]. The semantics of agent communication languages can be given through logical models, that extend standard predicate logic with concepts to capture dynamics of processes (dynamic logic), modalities, like obligation (deontic logic) [MWD98, LF94] and intention (illocutionary logic) [WVD95]. The language that we will propose belongs to this class of approaches, but focuses in contrast to the existing proposals specifically on the characteristics of information commerce processes.

3. An illustrating example

In order to exhibit the specific characteristics of information commerce processes that need to be considered for business process specification languages we introduce a simple information commerce scenario. We use a standard method for process specification in order to describe the processes in that scenario, namely colored Petri-Nets [J92]. Comparable models are used in many approaches for describing business processes [AALST99]. Using such a model will allow us later to highlight some of the main differences between a standard process specification method and the specification language we propose and to exhibit some of the benefits of our language.

In our scenario a portal site on the Web presents to its visitors the following offer: "*All visitors can register and afterwards receive upon request news on a specific company*". The corresponding process can be specified by a colored Petri Net as shown in Figure 1. We use colored Petri Nets rather than workflow models based on simple Petri Nets to capture the information flow more precisely. A major difference to standard business processes for which workflow models are designed is, that we have to describe the parallel processing of different information items in the business interactions. For example, in our scenario news about different companies can be requested by the same customer and since these requests are not independent we have to capture this within one process instance.

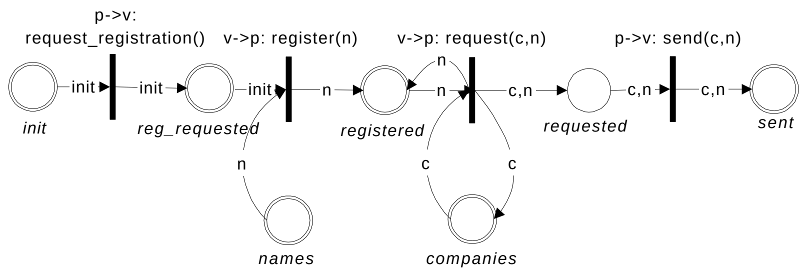

Figure 1: Scenario 1

We assume that the initial states *names* and *companies* contain tokens for all potential user and company names. Starting from the *init* state that contains an init token after being requested by the portal a visitor *n* can register and moves to state *registered*. From there he can request an item of company *c*. After an item has been requested the portal can send the item. The types of the token variables *n* and *c* are declared as *string*. The execution of this process has some additional constraints. Since the portal has promised the visitor that after registration news on companies that are requested are also delivered, a process state where a token *c* is in state *requested* is not a valid termination state of the process. Thus we indicate all states which may contain tokens in a terminal state by a double circle. In addition, as the process describes an interaction process among the portal and the visitor, it is necessary to indicate the message exchanges that take place with each transition. We specify this by denoting for each transition the message name, direction and parameters.

Now let us assume that after a while the portal notices that visitors register rarely. So it decides to add an alternative way of accessing the company news service. Users can immediately request news, but are requested then to register before the delivery. This requires a substantial extension of the process description, which is shown in Figure 2.

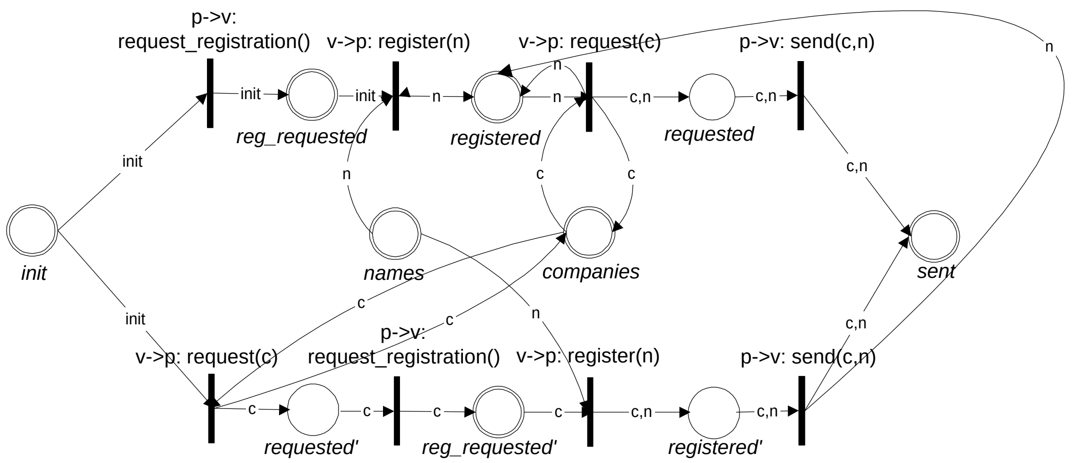

Figure 2: Scenario 2

In this modified process an alternative registration procedure is specified. Note that the state *reg_requested'* can be terminal, if the user decides not to register. Also one can see that after this sequence is processed the token *n* is given to the ordinary *registered* state such that afterwards requests can be handled in the normal way. Note that the *registered'* state in the alternative registration sequence can not be terminal since in this state the portal is obliged to deliver the requested information, since the visitor has registered. Nevertheless, this state may not be necessarily reached, since visitor may refuse to register. It is not under the control of the portal whether it will have to deliver the information, thus it is a *conditional obligation*. We will see later how we will be able to identify and analyse such forms of obligation in our business process language.

One can also see from this example that a fairly simple change in the model, which consists essentially in the reordering of some steps, requires a substantial extension of the process description due to the duplication of states and transitions, though the information goods and the messages have not changed.

After a while, business is evolving well and the portal decides to improve its service by providing monthly summaries on selected topics. Each user should receive those summaries that are related to the requests he has made throughout the month. The new process is depicted in Figure 3.

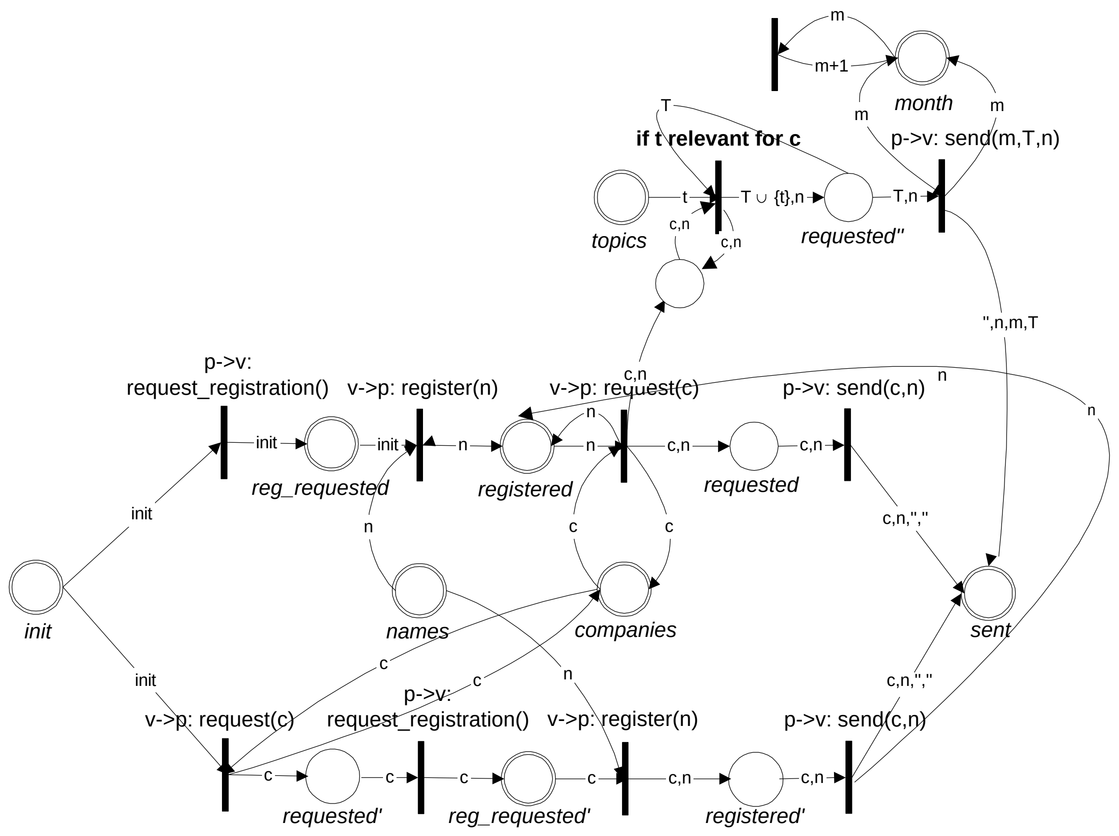

Figure 3: Scenario 3

In this extended process specification a new kind of transitions occurs. Up to now every transition was directly related to a message exchange, we call this an *explicit state change*. In this process some of the transitions do not have this property. The transition from state *topics* to *requested* corresponds to an *implicit state change* that results from the agreement among the portal and the visitor, but is not reflected in their communication. The fact that a requested company is relevant for a topic is sufficient to imply that the state of the topic changes to *requested*, without that this is explicitely communicated.

This example should have made clear that traditional methods of explicit process specification that are frequently used in EDI standard specifications or workflow systems have some difficulties with modelling the information commerce interaction processes for several reasons. We observed that

- Since the parallel processing of multiple information goods within one interaction is required, the resulting specifications are not necessarily very intuitive and easy to understand. This is in contrast to process specifications as they are used e.g. in workflow systems, where the process describes usually a single case, or an EDI order protocol, where the processing of a single order is treated.
- Small changes in the order of processing lead to a duplication of states and transitions in the model. This is unavoidable since each different processing order leads to a different history and thus to different intermediate states and potential final states.
- No formal criteria exist to determine valid termination states, in which obligations that have occurred are satisfied. These execution constraints need to be derived from the informal knowledge of the semantics of the business process.
- Though in most cases state transitions relate to communication actions, due to implicit state changes this is not always the case. The determination of the communication actions that take place is thus not supported by the model and needs to be furnished additionally.

In particular the last two points indicate that different aspects of consistency of the specification are under the responsibility of the developer and his understanding of the semantics of the interaction process. The model itself does not support crucial concepts such as obligation and communication. So we propose to take a different approach. We do not explicitely specify a process model and add specifications for termination properties and communication actions a posteriori. Rather we put the focus directly on the relevant concepts of information commerce processes and derive from that, among others, process specifications. To that extent, we introduce in the next section a language that is specifically designed to capture the commonalities of information commerce processes within the language and allows the user of the language to focus on the specific properties of his proprietary business process.

4. Overview of the language

The business offer language that we introduce in this section has been developed by following a number of key principles that we recognized as essential for the specification of information commerce processes.

1. The main primitive of the language is the action of exchanging an information good. We subsume under the notion of information good all goods that are exchanged electronically in an information commerce process, including payments and certificates.
2. In order to specify such an exchange action we need to provide the description of the information good itself as well as the description of the provider and receiver of the information good.
3. To express the business logic we have to be able to constrain the execution of actions by conditions on the information good, the process state and time. This allows to express which actions are *valid* within the agreement in a certain process state.
4. Since the provider and receiver are autonomous we require a minimal mechanism in order to establish an agreement on an information good exchange action. At least we have to give both parties the opportunity to express their agreement to the exchange. This allows to identify which actions become *obligatory* in a certain process state.
5. We explicitely differentiate state changes that result directly from communication actions from those that result from implicit state changes. Thus the model allows to determine the possible communications from the specification.

We introduce now step by step the constructs of the language that have been developed in order to realize those principles.

Information goods and roles

We assume that an information good is generally characterized by a set of attributes, its *metadata* description. We do not further constrain the types of the attributes, since those are domain dependent. So generically an information good is specified by an expression of the form

```
G(p1: T1, …, pn: Tn)
```

Examples of types for information good parameters are all standard primitive data types, but also domain specific types used to give properties of information goods. Examples of domain-specific types would be URL for Web page addresses, SQL-QUERY for database access, XML-DOC for structured documents, or US$ for payments. Without elaborating on details of such types, it is important to observe that each type also provides certain functions and relations which can be used to express conditions on the execution of the business process. For example a relation DOC1 included-in DOC2 on type XML-DOC could be used in order to determine whether a certain document has already been delivered as part of another document.

Similarly we have to specify who are the participants in the process. We can define different roles that may be parametrized.

```
R(p1: T1, …, pn: Tn)
```

As before we do not restrict what data types are used as parameters. A potential type could be ACCOUNT-TYPE containing the classification of a customer as A, B or C account customer. For business process specifications where the business partners are determined, the roles need not to be parametrized. In such a case the role specification is simply an enumeration of the participants, which we denote in the following as.

```
roles: R1, …, Rn;
```

Having defined the roles we can give the full specification of the information goods by associating with them the direction of exchange, for delivering the good. We denote this as

```
G(p1: T1, …, pn: Tn): R1 -> R2
```

The declaration of all information goods is then a finite set of good declarations of this form, denoted as

```
goods: G1, …,Gn;
```

This completes the specification of information goods and roles and thus covers point 2 of our design principles. For illustration purposes we specify goods and roles for the example from Section 3.

```
roles:
     Visitor,
     Portal;
goods:
     company(c: NAME): Portal -> Visitor,
     summary(t: {TOPIC}, m: MONTH): Portal -> Visitor,
     registration(n: NAME): Visitor -> Portal;
```

Actions and States

As stated in point 1 of our design principles the basic action related to an information good

```
G(p1: T1, …, pn: Tn): R1 -> R2
```

is its delivery, which we denote as

```
deliver(R1, R2, G(p1: T1, …, pn: Tn))
```

However, prior to delivery it is necessary to determine for which information goods and which concrete parameters **`p1, …, pn`** this action has to be performed. This requires that both parties involved in an exchange have the capability to express their requirements and reach consensus on a certain delivery action, as stated in our design principle 4. To support this we introduce two *negotiation* actions

```
promise(R1, R2, G(p1: T1, …, pn: Tn))
```

where **`R1`** tells **`R2`** that it is willing to deliver the good with the specified parameters if it is requested by **`R2`**, and vice versa

```
request(R2, R1, G(p1: T1, …, pn: Tn))
```

where **`R2`** tells **`R1`** that it is interested in the delivery of the specified information good. As opposed to the delivery action, where all parameters **`p1, …, pn`** have to have assigned concrete values, in the **`promise`** and **`request`** actions parameters may be left purposely free to express that the sender of the corresponding message does not care about its value.

These actions are defined in a way such that they exactly correspond to message exchanges. Thus *the only possible actions in the model are the communication actions promise, request and delivery* and they are implicitely given by means of the specification of the information goods.

For each of the actions we introduce a state predicate, expressing the fact that the corresponding action has occured. These states are thus given as

```
promised(R1, R2, G(p1: T1, …, pn: Tn))
requested(R2, R1, G(p1: T1, …, pn: Tn))
delivered(R1, R2, G(p1: T1, …, pn: Tn))
```

In addition these state predicates can be parametrized by the time of the occurrence of the action since this is frequently an important criterion for the further flow of actions.

```
promised(R1, R2, G(p1: T1, …, pn: Tn), t)
requested(R2, R1, G(p1: T1, …, pn: Tn), t)
delivered(R1, R2, G(p1: T1, …, pn: Tn), t)
```

If a certain information good is both requested and promised, then its delivery is presumed to become obligatory. These leads to the state transition diagram in Figure 4, according to which the actions can occur

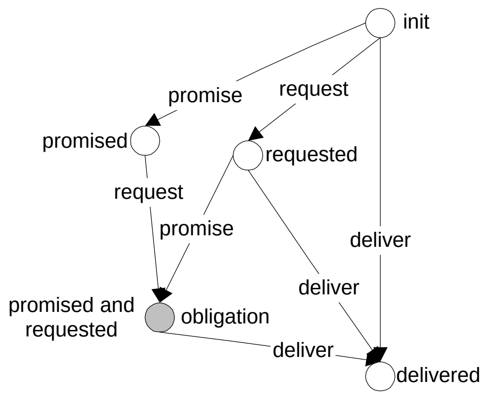

Figure 4: State transitions associated with the actions

This state transition diagram associates with every information good an elementary business process. In this process all information goods are first by default in an initial state *init* and can move by different transition sequences to the final *delivered* state. Note that not for all paths necessarily an obligation occurs. If the information good is parametrized the **`promise`** and **`request`** actions can constrain the possible parameter values without needing to determine them. If the information good is in state *promised and requested* the union of the parameter constraints is taken. The concrete delivery parameters must then satisfy all the constraints that have been specified.

We would like also to point out two important issues this agreement mechanism is not intended to address. First, it does not substitute negotiations prior to the execution of the information commerce process. It is insofar limited as it allows only to negotiate on the execution of the delivery actions and the corresponding parameterization. The information goods are specified as part of the business process. No new information goods can, for example, be added and no structural changes to the control flow, that will discussed subsequently, are possible. Second, the introduction of an obligation concept does not imply any specific way of how to treat non-compliance with an obligation. Possible reactions could range from taking notice of the violation and deriving from that a reputation of the business partner to taking legal actions and using the execution history as proof for the breach of a contractual obligation.

Rules

Up to now we can specify which actions are possible and which obligations occur when certain actions have been executed. In general it is necessary to also create dependencies among the delivery of different goods. We have seen this in the example of Section 3 where the portal was only willing to deliver company news to visitors that have delivered a registration. Therefore the execution of actions can be constrained as follows.

```
condition -> [action]
```

where the **`condition`** expression is a Boolean combination on the state predicates *promised*, *requested* and *delivered*, on relations that are defined on the parameter domains of the information goods and on the temporal parameters that occur in the state predicates. In this way the execution of one action can be made a precondition for the execution of another action.

We introduce some definitions and short-hand notations to deal with temporal specifications in conditions and finite repetitions of actions which are helpful to make specifications more readable and expressive. We write **`START`** for the starting time of the business process, and **`NOW`** for the time of evaluation of the predicate. Furthermore we define

`promised(R1, R2, G(p1: T1, …, pn: Tn), ALREADY)` as

```
promised(R1, R2, G(p1: T1, …, pn: Tn), t) and t<NOW
```

and **promised(R1, R2, G(p1: T1, …, pn: Tn), NOTYET)** as
not promised(R1, R2, G(p1: T1, …, pn: Tn)).

We will write

```
[action(p1, …,pk)]^n
```

for actions that are repeated for a finite number of times with arbitrary parameters. If the parameters are constrained we will write

```
[action(p)]^dom | condition(p, i)
```

for actions that are performed for values satisfying **condition(p, i)** for **i** Î **dom** where **dom** is a finite set. Similarly we may use this notation in the init clause and in rule conditions to specify multiple instances of states as follows

condition(p1, …,pk)^n
(condition holds for n different instances of p1, ... ,pk)
condition(p)^dom | condition(p, i)
(condition holds for each parameter **i** Î **dom**, such that **condition(p, i)**)

We illustrate the use of rules now for scenario 1 of our portal example.

```
rules:
-> [deliver(Visitor, Portal, registration(n))]

delivered(Portal, Visitor, registration(n)), ALREADY)
-> [request(Visitor, Portal, company(c))]

requested(Portal, Visitor, company(c), ALREADY) and
delivered(Portal, Visitor, registration(e)), ALREADY)
-> [deliver(Portal, Visitor, company(c))]
```

Three types of actions and corresponding message exchanges can occur according to this specification. In business processes executing according to these rules never an obligation occurs. On the other hand we have expressed in the informal semantics of the portal example that there exists a promise of the portal to deliver news for registered users. In order to deal with such a situation we allow the specification of initial states of the process that are obtained without executing the corresponding message exchanges that would lead to the state. This makes sense only for the negotiation actions, thus the specification can have the form

```
init:
promised(R1, R2, G1(p1: T1, …, pn: Tn), START)
| condition(p1,…,pn)
```
or
```
init:
requested(R2, R1, G2(p1: T1, …, pn: Tn), START)
| condition(p1,…,pn)
```

In our example we would thus add a clause

```
init: promised(Portal, Visitor, company(c), START);
```

which induces the conditional obligation we intend to express. Extending the specification to scenario 2 requires the following rules

```
rules:
-> [deliver(Visitor, Portal, registration(n))]

-> [request(Visitor, Portal, company(c))]

requested(Visitor, Portal, company(c), ALREADY) and
delivered(Visitor, Portal, registration(n), NOTYET)
-> [request(Portal, Visitor, registration(n))]

requested(Portal, Visitor, company(c), ALREADY) and
delivered(Portal, Visitor, registration(e)), ALREADY)
-> [deliver(Portal, Visitor,company(c))]
```

For moving from the first specification to the second only two changes had to be applied. First, the condition on the request action is removed, which is the intended purpose of the change, and second a rule is added that actively requests registration in case non-registered users ask for information. Note that the condition on the delivery action could have been simplified for scenario 1 by exploiting the condition that is made there on the request action for company news. However, in order to make rules more robust against changes all state-based conditions an action that are required should be included explicitely into the rules.

Finally, let us look at the scenario 3. To do so we introduce an initial promise by the portal.

```
init:
promised(Portal, Visitor, summary(top, m),START)^{1,…,12} | m=i
```

The rule on the delivery of the summary is then given by

```
requested(Portal, Visitor, summary(top, m), ALREADY) and
delivered(Portal, Visitor, registration(e)), ALREADY) and
NOW=end_of_month(m)
-> [deliver(Portal, Visitor, summary (top, m))]
```

The problem is here how to reach the `requested` state for the summary, as this information is never explicitely requested by the visitors, but is implied by his other requests for company information. For that purpose we introduce now another category of rules, that differentiate explicit state changes that are directly connected to message exchanges to implicit state changes that emerge from the properties of the agreement. Such a rule has the form

```
substitution: condition => state
```

if a single new state predicate is implied or

```
substitution: condition_1 => state(p)^dom | condition_2(p,i)
```

if we use our shorthand notation to imply a finite set. **`condition`** is defined as for action rules and **`state`** is one state predicate of the three possible types.

Using this rule type we can now specify the implicit request for summaries as follows

```
substitution:
delivered(Visitor, Portal, registration(n)) and
requested(Visitor, Portal, company(c))
   => requested(Visitor, Portal, summary(top))
        | relevant(top, c)
```

This completes the overview of the language concepts. We have seen that the three scenarios could be modelled with ease in the language. The specification we gave for the example process produced implicitely the following:

1. the message exchanges, as they are directly linked to the actions.
2. the process definition by means of the conditions for the execution of actions.
3. the obligations as they are directly derived from the promise and request actions occuring.

In our example only obligations on the portal side occurred. They resulted from initially assumed promises of the portal, explicit requests from the visitor and implicit requests from the visitors resulting from implicit state changes. They are conditional in the sense that they only hold if the visitor satisfies preconditions, i.e. the delivery of a registration.

The specification could be easily extended with further conditions, most notably temporal constraints on the execution of actions. For example the portal site could promise to deliver any news with a limited delay of one hour. This could be expressed as:

```
requested(Portal, Visitor, company(c), t) and NOW < t+1h
delivered(Portal, Visitor, registration(e)), ALREADY)
-> [deliver(Portal, Visitor, company(c))]
```

Thus the execution of the deliver action can only take place within one hour after the request and since it is obligatory the portal has to do so in order not to violate the agreement.

Finally, if we want also be able to express that certain states are reached within the process we provide a language construct that introduces explicit obligations

```
goals: delivered(R1, R2, G(p1: T1, …, pn: Tn), t).
```

Language Semantics

It is beyond the scope of this paper to discuss the semantics of the language in detail. We give here an indiciation of the mapping of specification to a logical model. The logical semantics is given by first order dynamic logic [MWD98], which allows to distinguish actions and states. The roles and information goods are then the function symbols of the logical model and the rules correspond to axioms of the language. For example, a rule of the form

```
condition(p) -> [deliver(r1, r2, g(p)]
```

translates to an axiom

condition(p) and time(t) ® [deliver(r1, r2, g(p)); inc_time()] delivered(r1, r2, g(p), t)

where the action **`inc_time()`** increments the time predicate **`time(t)`** according to the axiom *time(t) ® [inc_time()] time(t+1)*

The time expression **NOW** translates to *time(t)*. Obligations can be modelled by deontic extensions of dynamic logic as described in [WVD95, MWD98].

5. Implementation Architecture

From an architectural viewpoint the business process language allows to introduce a process level as an abstraction layer in information commerce systems which lies above the content management systems, as is shown in Figure 5. This abstraction layer allows to specify and control the processes that are executed when trading information good abstracting from technical details of how the goods are delivered electronically at a content level. Thus it facilitates the specification, negotiation, verification and execution of those processes.

Before starting such an information commerce process the trading partners have to establish a common agreement expressed as a business process specification, as indicated by an agreement level in Figure 5. The mechanism used to achieve such an agreement is out of the scope of the business language itself. It can be performed by any automatic or interactive mechanism that results in a common agreement on a business process specification. However, the agreement itself can be an information good that has been exchanged in the context of another information commerce process.

Once such an agreement is established the execution of the business process can start. The execution of the business process requires the following functions at the process level

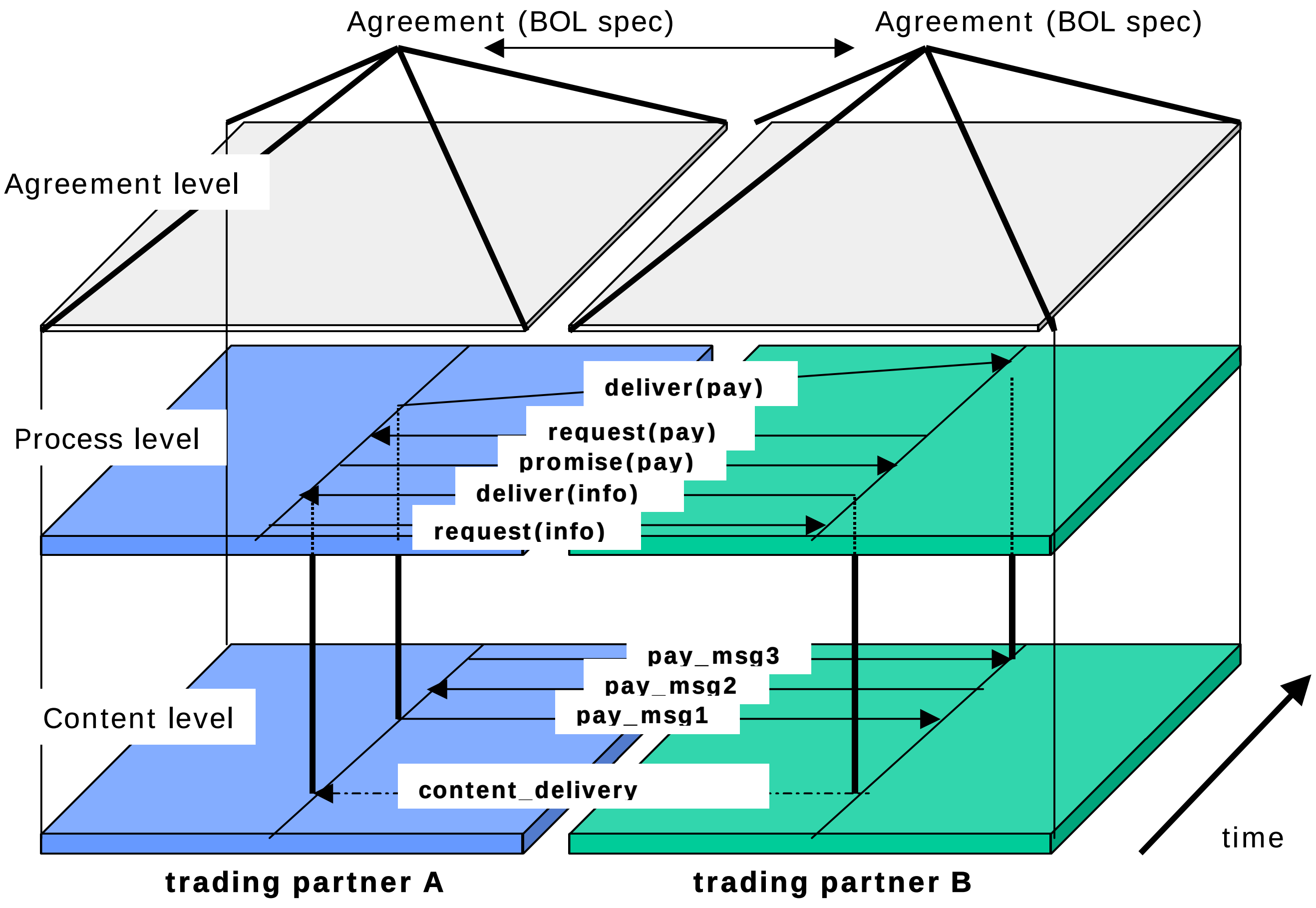

Figure 5: Overview of the architecture levels

- interfaces to the content management applications realizing the information good delivery
- generation and verification of the messages interchanged
- implementation of the data and control flow by the participants in the process

In addition, at any point during execution each participant can analyse the current state of the process execution in order to plan his future actions. Such analyses are specifically important with respect to the current obligations and the reachability of states where the obligations are satisfied. In the following we focus our description on the three basic functions at the process level.

Interfaces to Content Management for information good delivery

As a key concept of the business process language all actions are related to an information good and the generic action is that of delivering an information good. For the execution of a delivery action usually content management systems of both participants are involved. As with information goods we use the notion of content management system in a fairly wide sense, i.e. it could be a payment system, if the information good is a payment. In Figure 5 two cases of a delivery action are depicted, the delivery of an information good and the delivery of a payment. From these examples we can see two different possibilities of how a delivery action is implemented at the content management system level. In the first case, the delivery of an information item, a corresponding delivery operation at the content level is executed. The delivery can be realized by any communication mechanism that transfers the information content. Examples are the sending of an email, the access to a Web server, or the use of a push system [HJ99]. In some, possibly exceptional cases, all the relevant information corresponding to an information good may be transferred directly through messages at the process level, requiring no corresponding activity at the content level. In the second case, a payment, we see that a single delivery action at the process level translates to the execution of a more complex (payment) protocol at the content level. At the process level this protocol is executed as an atomic action.

In order to couple the delivery actions at the process level with the execution of the corresponding operations at content level, the business process language interpreter needs to be furnished with the specifications of the calls to the content management systems APIs that realize the delivery operation for every information good.

Communication at the process level

Every action at the process level results in a message exchange. For every information good and action type a message structure is generated from the business process language specification. This message structure is represented in XML as the standard data exchange format. In order to exchange these messages, in our implementation an XML based request-response protocol is used, that is derived from the ICE communication protocol [WKA01]. This protocol adds communication headers to the messages encoding actions and provides a communication error handling. Received messages are analyzed by parsing the message and creating internal data structures for further processing.

Implementation of Data and Control Flow

All participants in the business process need a runtime component that coordinates the execution of the business process. The runtime components orchestrate the enabling of actions by evaluating their enabling conditions, the tracking of the executed actions and thus of the process state, the generation and receipt of messages, and the calls to the content management systems required for the implementation of the delivery actions. The runtime component provides an interface for the triggering of enabled actions. This interface is either used by an application or directly by a user interface. The implementation of the runtime component can be based in a straightforward manner on a rule interpreter using the business process language specifications.

For the evaluation of enabling conditions the runtime component requires access to the domain specific implementations of the functions and predicates that are associated with the parameter domains of the information goods and roles.

6. Further work

We have already specified a number of example applications in the language including a portal site for exchanging technical information on object-oriented middleware, which is currently being realized in the context of the OPELIX project [OPELIX], the business model that is underlying the Napster MP3 audio exchange community [NAPSTER], and relevant parts of the ICE protocol [ICE]. It showed that the language allowed to express the business processes underlying each of these applications very directly and in an intuitive manner. Also the expressivity of the language proved to be sufficient. One possible extension relevant for trading multimedia contents with long-lasting deliveries would be to distinguish start and completion of the delivery action, to relate different conditions to each of both. Such an extension of the language can be done in a straightforward manner.

Another application of the language is the modelling of the negotiation process that precedes the execution of an information commerce process as an information commerce process itself. In that case the language would be used to model meta business models in which specifications in the business process language itself are the subject of trading.

The implementation of the system is ongoing. It is based on the architecture that has been described in [WKA01] and focuses on the support of light-weight infrastructures. When using heavy-weight middleware architectures such as WebLogic [BEA], which include rule interpreters, an implementation of the language could provide as a specialized abstraction layer for supporting information commerce based on the integrated rule engines.

Regarding user interface support adminstration interfaces as known for example in workflow management systems can be employed for administration purposes, while for customers on the Web a more Web-like look-and-feel should be achieved. This requires mappings from the process specifications into Web interfaces that present the process states and interaction capabilities in the form of Web documents and interactive links.

References

[AALST99] W.v.d.Aalst: Process-oriented architectures for electronic commerce and inter-organizational workflows, Information Systems No. 8, Vol. 24, pp 639-671, Elsevier, 1999.

[ASW98] N. Asokan, V. Shoup, M. Waidner: Asynchronous Protocols for Optimistic Fair Exchange, Proc. of S&P 98, Oakland, California, 1998.

[C96] B. Cox: Superdistribution, Addison-Wesley, 1996.

[CHTY97] J. Camp, M. Harkavy, J.D. Tygar, B. Yee: Anonymous Atomic Transactions, USENIX, 1997.

[FFMM98] T. W. Finin, R. Fritzson, D. McKay, R. McEntire: KQML As An Agent Communication Language. CIKM 1994: 456-463, 1994.

[GAHL00] P. Grefen, K. Aberer, Y. Hoffner, H. Ludwig: CrossFlow: Cross-organizational workflow management in dynamic virtual enterprises, International Journal of Computer Systems Science & Engineering, Vol. 15, Nr. 5, Sep. 2000, pp 277-290, CRL Publishing, 2000.

[GO00] R. Grimm, P. Ochsenschläger: Elektronische Verträge und ihre verbindliche Aushandlung - Ein formales Modell für verbindliche Telekooperation. Informatik Forsch. Entw. 15(4): 182-192, 2000.

[GWW01] C. Günther, S. Weeks, A. Wright: Models and Languages for Digital Rights, Proceedings of the 34th International Hawaiian Conference on Systen Sciences, 2001.

[HJ99] M. Hauswirth, M. Jazayeri: A Component and Communication Model for Push Systems. ESEC / SIGSOFT FSE 1999: 20-38

[J92] K. Jensen: Coloured Petri Nets, Springer Verlag, Heidelberg, 1992.

[JP01] M. Jazayeri, I. Podnar : A Business and Domain Model for Information Commerce, Proceedings of the 34th International Hawaiian Conference on Systen Sciences, 2001.

[KIMB] S. Kimbrough: Formal Language for Business Communication (FLBC): Sketch of a Basic Theory, forthcoming in *International Journal of Electronic Commerce*.

[KWA99] J. Klingemann, J. Wäsch, K. Aberer: Adaptive Outsourcing in Cross-Organizational Workflows, Proceedings of the the 11th Conference on Advanced Information Systems Engineering (Caise), Heidelberg, Germany, June 14-18, 1999.

[LF94] Y. Labrou, T. W. Finin: A Semantics Approach for KQML - A General Purpose Communication Language for Software Agents. CIKM 1994: 447-455, 1994.

[MGTMWBL98] M. Merz, F. Griffel, M. T. Tu, S. Müller-Wilken, H. Weinreich, M. Boger, W. Lamersdorf: Supporting Electronic Commerce Transactions with Contracting Services. IJCIS 7(4): 249-274, 1998.

[MWD98] J. Meyer, R. Wieringa, F. Dignum: The Role of Deontic Logic in the Specification of Information Systems, in: J. Chomicki, G. Saake (Eds) Logics for Databases and Information Systems, Kluwer Academic Publishers, 1998.

[TYGAR99] J. D. Tygar: Atomicity versus Anonymity: Distributed Transactions for Electronic Commerce. VLDB 1998: 1-12, 1998.

[WH98] H. Weigand, W.J. van de Heuvel: Meta-patterns for Electronic Commerce based on FLBC. Proc. HICSS'98, IEEE Press, 1998.

[WKA01] A. Wombacher, P. Kostaki, K. Aberer, WebXIce: An Infrastructure for Information Commerce on the Web, Proceedings of the 34th International Hawaiian Conference on Systen Sciences, 2001.

[WVD95] H. Weigand, E. Verharen, F. Dignum: Integrated Semantics for Information and Communication Systems.DS-6 1995: 500-525, 1995.

Web references

[BEA] BEA systems home page. www.bea.com
[CORBA] CORBA home page. www.corba.org
[EBXML] EBXML home page. www.ebxml.org
[ICE] Information and Content Exchange Protocol home page. www.ice.org
[INTER] Intertrust home page. www.intertrust.com
[IOTP] IOTP home page. www.oasis-open.org/cover/otp.html
[MICRO] Micropayment Markup Language home page. www.w3c.org/ecommerce
[NAPSTER] Napster home page. www.napster.com
[NETBILL] NetBill home page. www.netbill.com
[OPELIX] OPELIX home page. www.opelix.org
[ROSETTA] RosettaNet home page. www.rosettanet.com
[XFLOW] CrossFlow home page. www.crossflow.org