Modelling the ICE standard with a formal language for information commerce*

Andreas Wombacher[1], Karl Aberer[2]

[1] GMD-IPSI, Integrated Publication and Information Systems Institute, 64293 Darmstadt, Germany, wombach@darmstadt.gmd.de
[2] EPFL, Swiss Federal Institute of Technology , 1015 Lausanne, Switzerland, karl.aberer@epfl.ch

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. We demonstrate the use of the language by applying it to an important standard for specifying information commerce processes, the ICE Information and Content Exchange protocol [ICE1]. By doing so we also illustrate the benefit of using formal specifications for information commerce processes allowing to capture informal specifications, like ICE, in a concise way.

1 Introduction

As modern markets move rapidly onto electronic platforms, ecommerce and ebusiness are becoming key terms in todays 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.

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

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

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 news services, stock market information services, software evaluation services, or directory services.

In [AW01] we introduced a business process language that has been specifically developed to model information commerce scenarios. The language allows to compose processes in a flexible way by means of condition action-rules. In order to model contractual constraints, we introduced a concept of obligations that can be derived from process specification. This allows to specify which actions are obligatory to be executed in a given process state. We have shown already, that our approach is sufficient to model simple business processes.

Within this paper, we apply this business process language to model the comparably complex ICE protocol. The Information and Content Exchange (ICE) protocol [ICE1, ICE2] is a standardized model to describe the exchange of content, like catalog data or news items, in closed user group 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 the ICE protocol allows to capture some important concepts of an information commerce system, namely the negotiation and flexible specification of information exchange processes and their execution.

Our motivation to model ICE within a formal business process language is twofold. On the one hand, it serves for us as a test case for the applicability of our language. Being able to represent a complex standard specification in our formal language demonstrates its expressibility and generality. On the other hand, providing a formal specification for ICE is also useful in itself. In particular, we explicate the definition of the processes and execution constraints that are given in the ICE standard specification only informally. In that way we can step forward to a more formal specification of the standard, provide a more concise and compact description of the ICE processes and lay the foundation for the eventual verification of process properties. For example, we will model all the informal obligations that are committed in the ICE negotiation protocol, and have to be satisfied in the ICE information exchange protocol. In addition, from the specification the required message types and possible message exchanges of the ICE protocol can be derived.

In the following Section 2 we briefly mention the business process language for information commerce, that we originally proposed in [AW01] and that we use to model the ICE protocol. Section 3 is the key section and contains a brief overview of the ICE protocol and the description of the formal model for ICE. In Section 4 we give an analysis of related approaches to modelling electronic commerce processes. We conclude with a description of future work in section 5.

2 Description of the Business Process Language

We give a short introduction on the business process language that we will use in the following by means of an example. More details and motivation can be found in [AW01]. The example describes the registration of a customer requesting information from a information vendors web site. First the roles of the involved parties are listed: customer and vendor. Then the exchanged goods are specified: registration and base_info. The goods are parametrized, i.e. by assigning an url to base_info or name and email address to the registration. For each good the provider (owner) and receiver (user) is specified. For each good three standard actions can be performed: request by the user, promise and deliver by the owner. Request and promise are used to agree on the product parameters. As soon as a product is requested and promised with the same parameters and *obligation* on the side of the owner occurs to perform the deliver action, with all parameters bound to concrete values.

The performance of the actions can be further constrained by rules which establish a business workflow. The rule conditions can refer to actions performed on specific goods and lead to an explicit state change (reflected by predicates requested, promised, delivered). For example, one rule expresses that the user is allowed to request the base_info information after he has delivered the registration information. In addition, implicit state changes that are not the direct consequence of executing one of the actions, i.e. that occur without exchanging a message between the trading partners, can be specified by substitution rules. For example, the substitution rule expresses that if a registration is received by the vendor it automatically implies a promise from his side to deliver the information. In case this information has been requested earlier an obligation occurs. This is a way to express conditioned obligations.

roles: customer, vendor;
goods: registration(name: STRING, email:STRING): cust → vend;
base_info(url:STRING): vend → cust;
rules:
→ deliver(cust, vend, registration(n, e));
delivered(cust, vend, registration(n,e)) → request(cust, vend, base_info(url));
...
substitutions:
delivered(cust, vend, registration(n,e),t) ⇒ promised(vend, cust, base_info(url));
...

In the following we will use the ICE protocol in terms of the business process language.

3 Modelling of the ICE protocol

The ICE protocol [ICE1, ICE2] for information and content exchange is a protocol for the communication among content providers (called syndicators) and their users (called subscribers). The ICE protocol is a closed user group protocol, which defines the roles and responsibilities of the participants in the information exchange, the syndicators and subscribers, the message formats and the method

of content exchange, but leaves other dimensions like payment, authentication or metadata for content description open as orthogonal dimensions. The three main phases in the execution of the protocol are

- establishment of the ICE infrastructure
- subscription establishment and management
- data delivery.

The first phase is out of the scope of the protocol itself, but required to set up legal conditions (like copyright conditions and payment) and IT infrastructure (like access conditions) to be able to process the protocol. Based on this contract, the local ICE infrastructures of the syndicator and the subscriber must be configured. After this initial phase, the second phase starts with an interaction between syndicator and subscriber for establishing a subscription. Typically the subscriber first obtains a catalog of offers from the syndicator and then subscribes to a particular offer by proposing it to the syndicator. Alternatively the two parties may engage in a parameter negotiation protocol. After the subscriber subscribes to a particular offer the third phase, the data delivery phase, starts. ICE uses a sequenced package concept. Packages encapsulate contents of arbitrary type. ICE also defines push and pull data transfer models and detailed temporal and quantitative constraints on delivery of packages.

A subscription negotiated in the ICE protocol contains a description of temporal and quantitative constraints for performing the delivery. In particular, the subscription specifies the process of the delivery by using a constraint-based approach. Thus, ICE subscriptions can be seen as an approach to flexibly specifying business processes for content delivery. In the following we will model the ICE specification in terms of the business language mentioned in Section 2, in order to demonstrate the expressiveness of our language and to formalize the semantics of the ICE specification.

3.1 Analysing the expressive power of ICE subscriptions

As mentioned above, a subscription contains the process specification for the content delivery phase within the ICE protocol. In order to model this in terms of the business process language, we first analyse ICE subscriptions in detail. To do so, we have to look at the ice-delivery-rule element, since it contains the constraints with respect to the delivery process. The type definition in XML (DTD) syntax is given below [ICE1].

```
<!ELEMENT ice-delivery-rule (ice-negotiable*) >
<!ATTLIST ice-delivery-rule
        mode        (push | pull)    #REQUIRED
        monthday    NMTOKENS         #IMPLIED
        weekday     NMTOKENS         #IMPLIED
        startdate   CDATA            #IMPLIED
        stopdate    CDATA            #IMPLIED
        starttime   CDATA            #IMPLIED
        duration    CDATA            #IMPLIED
        minfreq     CDATA            #IMPLIED
        maxfreq     CDATA            #IMPLIED
```

mincount	CDATA	#IMPLIED
maxcount	CDATA	#IMPLIED
url	CDATA	#IMPLIED >

We briefly describe the semantics of the different attributes. A detailed description can be found in [ICE1]. The mode attribute is the only required attribute of the ice-delivery-rule element and contains one of the values push and pull representing the delivery type. The attributes monthday and weekday restrict the delivery to a specific set of days within a month or a week. The startdate and stopdate attributes represent the earliest and latest point of time for content delivery. The attribute startdate can be extended by using the starttime attribute. The duration attribute specifies the period of time beginning at starttime during which delivery actions can be performed. In addition to this the minfreq and maxfreq attributes can be used to specify the minimal and maximal amount of time between two content delivery actions. All time parameters are given with seconds as the finest granularity. In addition to the temporal constraints, the mincount and maxcount attributes are quantitative constraints limiting the total number of content delivery actions.

The url attribute specifies the address where to send the update, in case it is not sent to the normal ICE communication endpoint. This allows package delivery to a different location than the other ICE communication. This is not affecting the business process itself and therefore will not be further considered here.

In order to model delivery rules in the business process language we first define a complex data type DEL_RULE that includes the parameters of the ice-delivery-rule element.

DEL_RULE:	mode: {'push', 'pull'},		monthday: INT,
	weekday: INT,	startdate: DATE,	stopdate: DATE,
	starttime: TIME,	duration: TIME,	minfreq: TIME,
	maxfreq: TIME,	mincount: INT,	maxcount: INT;

We define some functions which are convenient to access the different components of a value of type DEL_RULE. The functions startdate(d) and stopdate(d) extract the startdate and stopdate parameter from the delivery rule value d. The functions mode(d), monthday(d), weekday(d), starttime(d), duration(d), minfreq(d), maxfreq(d), mincount(d) and maxcount(d) are defined similarly. The defined functions return NULL if the parameter is undefined. In order to express temporal conditions we introduce some functions for the domains DATE and TIME. If t is a time value then DAY_OF_MONTH(t) gives the current day of the month and DAY_OF_WEEK(t) gives the current weeekday. DATE(t) gives the date at time t and DAYTIME(t)gives the time of the day at time t. The expression NOW gives the current time. The parameter id is the ice-package counter and the parameter last_delivery the time of delivery of the last package. Now we can formulate a predicate to evaluate whether a package is to be delivered at time NOW as follows.

SAT(d, id, NOW, last_delivery) =
(monthday(d)=NULL or DAY_OF_MONTH(NOW) in monthday(d)) and
(weekday(d)=NULL or DAY_OF_WEEK(NOW) in weekday(d)) and
DATE(NOW) $\geq$ startdate(d) and

(stopdate(d)=NULL or DATE(NOW)<stopdate(d)) and
(starttime(d)=NULL or DATE(NOW)>startdate(d) or
(DATE(NOW)=startdate(d) and DAYTIME(NOW)≥starttime(d))) and
(duration(d)=NULL or ((starttime(d)=NULL and
(DATE(NOW)-startdate(d))*24h+DAYTIME(NOW)<duration(d))) or
(starttime(d)≠NULL and
(DATE(NOW)-startdate(d))*24h+DAYTIME(NOW)<duration(d)+starttime(d))
and (minfreq(d)=NULL or NOW-last_delivery>minfreq(d)) and
(maxfreq(d)=NULL or NOW-last_delivery<maxfreq(d)) and
(maxcount(d)=NULL or id≤maxcount(d))

The start time must be always defined and is therefore not tested for NULL. There exist parameter settings, which lead to unsatisfiable constraints. Those must be detected by the syndicator and he should take care that they are not subscribed.

3.2 Modelling the information goods

Now we model the information items that are exchanged in the ICE protocol. We define the two participating roles (subscriber and syndicator). Then we can identify five types of information that are exchanged. Initially a catalog is sent from the syndicator to the subscriber. The catalog consists of offer elements that are selected by the subscriber and sent to the syndicator. In turn a syndicator may confirm an offer and make it a subscription. After that he sends ice-package elements with the actual content to deliver which is represented by an abstract type CONTENT.

roles: Syndicator, Subscriber;
goods:
catalog(offers:DEL_RULE): Syndicator → Subscriber;
offer(del_rule: DEL_RULE): Syndicator → Subscriber;
offer(del_rule: DEL_RULE): Subscriber → Syndicator;
subscription(subscription-id: ID, d: DEL_RULE): Syndicator → Subscriber
ice-package(package-id: ID, subscription-id: ID, old-state: STRING,
new-state: STRING, cont: CONTENT): Syndicator → Subscriber

A few interesting observations can be made at this point. Though the same information is delivered, e.g. with an offer and a subscription, the semantics of this delivery is a different one. In addition the different information items reference each other through identifiers. The ice-package references both the subscription it is based on as well as the other ice-package which it follows.

3.3 Modelling the delivery phase

Based on these definitions of the goods, we are now modelling push and pull delivery as defined in the ICE protocol. The basic rule for delivering a package is the following. Whenever an ice-package corresponding to a subscription has been promised by the syndicator and requested by the subscriber it has to be delivered. In addition the previous package in the package sequence must have been also delivered. This is expressed by the following rule

rules:
(1) delivered(Syndicator, Subscriber, subscription(sid,d)) and
delivered(Syndicator, Subscriber, ice-package(pid,sid,old1,old,c)), last_delivery)

and promised(Syndicator, Subscriber, ice-package(pid,sid,old,new,c)) and
requested(Subscriber,Syndicator, ice-package(pid,sid,old,new,c)) and
SAT(d, sid, NOW, last_delivery)
→ deliver(Syndicator, Subscriber, ice-package(pid,sid,old,new,c))

For the first package we need a special rule since no previous packages have been delivered

rules:

(2) delivered(Syndicator, Subscriber, subscription(sid-1,d), last_delivery) and
promised(Syndicator,Subscriber, ice-package(pid,sid,'ice_initial',new,c)) and
requested(Subscriber,Syndicator,ice-package(pid,sid,'ice_initial',new,c))
and SAT(d, sid, NOW, last_delivery)
→ deliver(Syndicator, Subscriber, ice-package(pid,sid,'ice_initial',new,c))

The time last_delivery of delivering the subscription is the reference time used to evaluate the SAT predicate. Next we give the rules that lead to the promises of the syndicator for delivering the ice-packages. These are a consequence of the subscription message, thus no more message exchanges are performed to establish those promises. Therefore we model this as an implicit state transition by a substitution rule. The promises are given stepwise with each package for the next one. Promising packages one after the other just in time simplifies the handling of cancellations and changes of subscriptions.

substitutions:

(3) delivered(Syndicator, Subscriber, subscription(sid,d),t) and t≤starttime(d)
⇒ promised(Syndicator, Subscriber, ice-package(1,sid,'ice-initial',new,c))

(4) delivered(Syndicator, Subscriber, subscription(sid,d)) and
delivered(Syndicator, Subscriber, ice-package(pid,sid,old,new,c))
⇒ promised(Syndicator, Subscriber, ice-package(pid+1,sid,new,new',c'))

The difference among push and pull delivery lies in the way of how commitments arise from request issued by the subscriber. In case of a push delivery there exists an implicit agreement that the subscriber accepts any subscription that he receives from the syndicator. We model this by substitution rules such that the sending of the subscription message automatically implies the subscribers requests for the first ice-package to be delivered by the syndicator. In the same way as the promises of the syndicator are established, every time package is requested by the subscriber when the previous one is delivered. This happens without the subscriber sending any requests to the syndicator and gives therefore rise to the following substitution rules.

substitutions:

(5) delivered(Syndicator, Subscriber, subscription(sid,d),t)
and mode(d)='push' and t≤starttime(d)
⇒ requested(Subscriber, Syndicator, ice-package(1,sid,'ice-initial',new,c))

(6) delivered(Syndicator, Subscriber, subscription(sid,d)) and mode(d)='push' and
delivered(Syndicator, Subscriber, ice-package(pid,sid,old,new,c))
⇒ requested(Subscriber, Syndicator, ice-package(pid+1,sid,new,new',c'))

Together with the promises of the syndicator these rules imply that the syndicator is obliged to deliver the packages. This completely specifies the mechanism of push delivery.

For pull delivery request messages are sent by the subscriber for each package. The only limitation on making of requests in the ICE standard is the requirement that a subscription for pull delivery exists. The responsibility for making the

correct requests remains with the subscriber. This is expressed by the following rule for executing requests by the subscriber.

rules:

(7) delivered(Syndicator, Subscriber, subscription(sid,d)) and mode(d)='pull'
→ request(Subscriber, Syndicator, ice-package(pid,sid,old,new,c))

As the syndicator promises automatically the delivery of the packages, each time such a request is issued and the conditions of the first two rules on delivery turn true, the delivery becomes obligatory for the syndicator.

3.4 Subscription establishment and management

The subscription establishment and management phase can be partitioned into different activities

- catalog handling
- negotiation resulting in either end of communication or delivery of a subscription
- renegotiation of already existing subscriptions and potentially changing it
- cancellation of an existing subscription

Within this paper, we treat only the first two points because of space limitations. We start with the catalog handling. In the ICE protocol it is specified, that the subscriber can request a catalog, which has then to be delivered by the syndicator whereby this catalog contains all offers provided by the syndicator. It is important to mention, that no obligations are implied by sending these offers. The syndicator is able to retreat any offer unless he has sent a subscription. The rules related to sending the catalogues are straightforward.

rules:

(8) → request(Subscriber, Syndicator, catalog(o))

(9) requested(Subscriber, Syndicator, catalog(o))
→ deliver(Syndicator, Subscriber, catalog(o))

The ice-get-catalog message does not require any precondition and can be performed anytime. The catalog is delivered after it has been requested, though the syndicator is not obliged to do so. The ICE specification does not state whether the syndicator must deliver the catalog or may deliver it. In case the delivery is obligatory this can be modelled by adding an init clause where the catalog delivery is promised by the syndicator. The negotiation model of the ICE protocol is represented by the following state diagram.

The diagram can be divided into two parts. The left side is associated with the syndicator, while the right side is associated with the subscriber. The grey circle represents the starting state and the annotations of the arcs depict the message types. The subscriber starts with an offer, which can be accepted by the syndicator resulting in a subscription (final state of negotiation). The syndicator can also respond with a counter offer or can reject the request resulting in a final state. If the syndicator sends a counter offer, the subscriber has similar possibilities. The subscriber now can either send the unchanged offer back to the syndicator to indicate acceptance, response with a further counter offer or reject the offer, resulting in a final state.

It is important to mention, that within this negotiation model no obligations are specified in the ICE protocol, although they might be useful.

The rules for modelling this negotation model are given below.

rules:

(10) → deliver(Subscriber, Syndicator, offer(d))

(11) delivered(Subscriber, Syndicator, offer(d)) and d≠d'
→ deliver(Syndicator, Subscriber, offer(d'))

(12) requested(Subscriber, Syndicator, subscription(sid,d))
and (starttime(d)≠NULL or starttime(d)=NOW)
→ deliver(Syndicator,Subscriber,subscription(sid,d))

substitution:

(13) delivered(Subscriber, Syndicator, offer(d))
⇒ requested(Subscriber, Syndicator, subscription(sid,d))

The first rule specifies that offers can be send by the subscriber, without any former action (as described in section 4.3.2 in [ICE1]). The second rule models the counter offer sent by the syndicator which requires the sending of a previous offer by the subscriber. The fact that the counteroffer must be different is not clearly stated in the ICE specification, but is implied by comments in the specification. The third rule states that only requested subscriptions may be delivered. A subscription is implicitely requested if and only if the subscriber has proposed the corresponding offer, by sending it to the syndicator. This relationship is modelled by the substitution rule. Rejection of offers and subsequent termination of the negotiation needs not to be modelled, as it is simply realized by not continuing the communication.

This model also shows that the constraints for negotiation under the ICE protocol are extremely weak. All the negotiation steps are indicative for the other side, but imply no obligations. For example, offers from the catalog are not binding, offers can be created completely independent of the ones contained in the catalog, and offers sent during negotiation are not binding as well. The only execution constraint is that only those subscriptions are possible that have been explicitely proposed by the subscriber. Obligations, as we have seen, occur however, once a subscription is made. Then the packages have to be delivered according to the specification of the subscription.

3.5 Execution Constraints of the ICE protocol

One of the goals of modelling the ICE protocol with our business process language is to give a formal specification of the constraints on the execution of the ICE protocol. These are given in the ICE specification in an informal manner. This shows that it is feasible to express also a complex standard specification within a formal framework. As a result the specification is given in a more compact form and can provide implementors of the standard with an unambigous specification. In the table below we list some of the important execution constraints of the ICE protocol. We identify the section where the corresponding specification can be found in the ICE specification document [ICE1], cite the textual description of the ICE specification and identify the rules of our formal specification that capture that constraint. The relationship between constraints

and rules is of course not one-to-one. For example, a specification like 'ICE uses a sequenced package model' results in several rules that are needed to express the corresponding semantics formally, i.e. rules (4) and (6). See table for a detailed description of mappings.
We also would like to point out that our specification of the ICE protocol is not complete. The most important part of the specification we left out is the renegotiation of subscriptions, which currently can not be modelled with our approach in a straightforward manner. The corresponding messages of the ICE protocol are ice-change-subscription or ice-cancel. Further, we didn't explicitly modelled the uniqueness of the subscription id (sid) within rule 12 and we assumed, that only one package is transmitted within each content delivery action while the ICE specification allows multiple packages to be included in one delivery operation. We used this approach to reduce the complexity of the handling of the package ids without limiting the functionality of the model.

To summarize, we described in our model the main parts of the ICE protocol. We were in particular treating the negotiation of the delivery processes and the execution of the resulting delivery specification. We showed, that the business process language provides the capabilities to formally describe the semantics of such flexible process specifications, and in particular the execution constraints associated with the process that are given in the ICE specification in an informal manner.

4 Related work

Mechanisms for the informal and formal specification of interaction processes in electronic commerce are described in many different contexts, both in standardization and research. In the following, we give a brief overview of approaches more or less related to the business process Language. In the context of **Web standardization** the Micropayment Markup language [MICRO] and the Information and Content Exchange (ICE) Protocol [ICE1, ICE2] are the ones closest to our work describing information commerce, but less general in defining possible business processes. Many **secure protocols** have been proposed for electronic commerce and information commerce that guarantee secure and fair exchanges. They consider electronic delivery, like Netbill [NETBILL, TYGAR99] or DigiBox [INTER, C96], or right management and granting fair exchange of goods [ASW98, GWW01]. These approaches are in general too restricted in order to allow the modelling of information commerce processes in general. **Agent communication languages**, like KQML [FFMM98] focus on the problem of communication among autonomous software agents. These languages correlate the agents internal states with the messages that are intended to change these states and specify the interaction languages and 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, WH98].

ICE - Rule Mapping

Rule number	Section in [ICE1]	Textual description in the ICE specification
2, 3	5.1.4	When it (the subscriber) first starts a new subscription, the Subscriber starts in state ICE-INITIAL.
5, 6	5.4	If a subscription has a delivery policy method of type push, the Syndicator must initiate the delivery of the packages containing one or more ice-package elements. When a Syndicator sends a package to the Subscriber, the Syndicator MUST provide the expected state of the subscription before and after the package is processed.
4, 6	5.1.2	ICE forces a Syndicator (and a Subscriber) to view the package stream as a strictly ordered sequence of packages. This means that packages cannot be processed out of order, and all intermediate packages must be processed.
7	5.3	If a subscription has a delivery policy method of type pull, the Subscriber must initiate the delivery of the packages with the ice-get-package request.
1	5.3	When a Subscriber requests a package from the Syndicator, the Subscriber MUST provide the state of the subscription and subscription identifier,.
8	4.3.2	Subscribers can use ice-get-catalog to obtain the list of subscription offers for which they are eligible.
9	4.3.2	Return response (to an ice-get-catalog) is an ice-catalog.
10, 13	4.4, 4.5.2	A Subscriber uses the ice-offer request to establish a subscription. Typically, a Subscriber will use ice-get-catalog to get a catalog, take one of the ice- offer structures from that catalog, and send it back to the Syndicator in a request. However, the Subscriber is free to create an ice-offer structure in any implementation-defined manner it wants. For example, a Syndicator might e-mail an ice-offer to a Subscriber, who could then feed it into their ICE tool and begin the protocol processing here. AND: Negotiation begins with the Subscriber making an ice-offer request to the Syndicator.
11	4.5.2	The Syndicator indicates a counter-proposal by rejecting the ice-offer ., and including a counter ice-offer in the response.
12	4.5.2	The Syndicator accepts the offer including an ice-subscription response.
10	4.5.3	If the Subscriber receives a counter proposal (that is another offer instead of a subscription), the Subscriber MAY try another ice-offer, either with the contents of the counter proposal received from the Syndicator, or with some other mixture of parameters. The method of choosing what parameters to alter is a quality of implementation issue. AND: If the Subscriber receives a Sorry response (that is no further action performed, neither a subscription nor a counter offer), the Subscriber MAY try again with some other ice-offer, although the Syndicator has (unhelpfully) not given any clues as to what to try.
13	4.4	A Subscriber uses the ice-offer request to establish a subscription.

5 Future work

The next steps are to provide for the business process language a formal semantics in terms of dynamic deontic logic and to complete the implementation of the system, which is based on the architecture described in [WKA01] and focuses the support of a light-weight infrastructure.

6 References

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

[AW01] K. Aberer, A. Wombacher: A language for information commerce processes, Technical Report EPFL, 2001.

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

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

[GWW01] C. Gnther, S. Weeks, A. Wright: Models and Languages for Digital Rights, Proceedings of the 34 th HICSS, 2001.

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

[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 34 th HICSS, 2001.

Web references

[ICE1] W3C note of the ICE protocol. www.w3.org/TR/1998/NOTE-ice-19981026

[ICE2] Information and Content Exchange Protocol home page. www.icestandard.org/

[INTER] Intertrust home page. www.intertrust.com

[MICRO] Micropayment Markup Language home page. www.w3c.org/ecommerce

[NETBILL] NetBill home page. www.netbill.com

[OPELIX] OPELIX home page. www.opelix.org