Meta-Patterns for Electronic Commerce Transactions based on FLBC

Hans Weigand and Willem-Jan van den Heuvel

Infolab, Tilburg University

PO Box 90153

Tilburg, The Netherlands

email: H.Weigand@kub.nl, wjheuvel@kub.nl

September 30, 1998

Abstract

Due to the highly communicative character of electronic commerce transactions, open-edi representation languages such as FLBC, take the speech act (operator) as their basic building block. The advantage of this approach is that the 'deep structure' of electronic commerce transactions is addressed rather than the form. In this paper, we try to reveal higher-level units of speech acts which are materialized in so called meta-analysis patterns. We define various levels, from speech acts to scenario's. Once patterns have been identified, they can be stored in an FLBC component library and be (re-)used effectively by business partners to speed up open-edi transactions.

keywords:electronic commerce, patterns, open edi, Language/Action perspective

1 Introduction

The community that is using the world-wide web steadily grows, and has an estimated 20-40 million users (Bell and Gemmell, 1996). This trend offers (new) businesses the Porterian possibility to penetrate new markets and expand their activities by engaging in the electronic commerce. Electronic Commerce is defined by Zwass as "the sharing of business information, maintaining business relationships, and conducting business transactions by means of telecommunications networks" [37]. The scope of Electronic Commerce is large, but in this article we concentrate on the business-to-business transactions As such, it can be interpreted as being a special form of open edi, a successor of EDI. Open edi can be defined as 'electronic data interchange among autonomous parties using public standards and aiming towards interoperability over time, business sectors, information technology systems and data types'.

Traditional EDI, or electronic data interchange, is not suitable for the new type of electronic commerce transactions because, though there exist some standards with regard to the syntax of the messages (such as EDIFACT and the ANSI X.12 standard), additional agreements between the participants have to be made. Open edi on the other side, is directed towards short-term relationships by which EC transactions are characterized [19]. Traditional EDI is characterized by the combination of 'closed trading relationships' and high start-up costs stemming from detailed trading partner negotiations. In Open edi, these start-up costs are supposed to be much lower. One way of achieving that is by industry-side and/or cross-sectoral standards. The problem with standards like these is they are typically inflexible and conservative. An alternative way could be to support the negotiation process itself electronically.

Languages, such as the Formal Language for Business Communication (FLBC) [17, 18] and the Language for Electronic Commerce [5], provide some means to formally describe messages in a more expressive and flexible way than before (EDI). These languages not only support the representation of the syntax of a message, but also the semantics. The linguistic notion of the speech act has been chosen as the basic element of representation in both languages mentioned in the above.

Following this lead, we propose to extend such languages with the notion of meta-patterns, e.g. typical sequences of speech acts within the context of electronic commerce transactions. These meta-patterns can be seen as a means to order the basic elements of a conversation, e.g. the speech acts, and to make the mutual relationships between speech acts explicit. The meta-patterns we introduce in this paper are closely related to methods explored in the Language/Action perspective, such as DEMO [7] and Action-Workflow [21].

The remainder of this paper is organized as follows. In section 2, we describe the formal language for business communication as an example of a representation language for open-edi. In section 3, we introduce the notion of design patterns, and the concept of meta-analysis patterns we have derived from this concept. In section 4, we build up a layered model of an (electronic commerce) transaction. Each layer has its own (set of) pattern(s). Section 5 describes the way the patterns can be applied in the form of an FLBC component library which is currently built in the context of a European ESPRIT project. The syntax of our specification language is given in the Appendix.

2 Formal Language for Business Communication

The Formal Language for Business Communication has been introduced in order to provide a representation that offers more expressiveness and flexibility than the conventional EDI standards, such as EDIFACT and ANSI X.12. The basic idea dates back to a visionary article by Kimbrough and Lee in 1986 [16].

FLBC represents a message as a sequence of speech acts, typically assertions and declarations, that form the basis of potential reasoning procedures. FLBC claims to express the syntax as well as the semantics of a message.

FLBC uses the following elements to describe the speech act: the speaker and the hearer, the illocutionary force, the content and the context. FLBC-II, that is defined in [18], distinguishes between three illocutionary points, to be an assertion, a request and a query. These three atomic speech acts are used to represent a variety of message types, such as appointments (assertions), staff action messages (directives) and read/review/comment messages (directives). The context wherein the communication takes place, is represented by means of either the message-ID to which is responded, the time when the message was sent, the machine-ID from which the message was sent and/or the persons to which the message is cc-ed. The machine-ID can be interpreted as being a means to establish the identity of the sender. In this way, the agent is anchored to a 'real-world' person (see section 4). In this paper, we will not focus on the (representation of the) content of the message, e.g. the proposition. A linguistically motivated knowledge representation such as Functional Grammar [31] can be instrumental for this purpose.

Kimbrough and Moore hypothesize that FLBC can be used to formally represent messages in order to perform some inferencing. In [18], they indicate four main reasons why inferencing is important. Firstly, it can be used to check the pre-condition(s) of the message: only when they are met, the message can be send to the addressee. Secondly, FLBC provides a means to represent the message in an unambiguous manner, adding semantics in the form of speech acts (more in particular, the illocutionary point). Standards such as X12, they quote, consist of transaction sets that express more than one illocutionary force, thereby opening the possibility for multiple interpretations. Thirdly, recording messages at the system level, that is looking at the semantic content/effect of the outstanding and incoming messages, provides an effective way to perform inferences, e.g., about all outstanding requests. Lastly, they note that logging incoming and outgoing messages at the 'application level', provides an adequate basis for many useful derivations.

A small comment is due on the second point: it is very common in practice that a certain message has several illocutionary points. For example, an order is a directive in the first place, but typically includes the commitment to pay the price. The idea that messages have a single and explicit illocutionary point was also incorporated in the Cooperator system of Winograd and Flores [35], and was one of the main sources of criticism on this system from a CSCW point of view [6]. FLBC aiming at interorganizational conversations is not completely comparable with the Coordinator. Nevertheless, it seems wise to take the lessons learned there into account. For that reason, we would like to weaken the one message- one illocution constraint. The goal of unambiguous semantics is something we want to maintain where appropriate, but this can also be achieved by explicitly stating the effects of the messages in terms of obligations, authorizations etc., as has been described in for instance [29], or simply by defining an FLBC message as a *set* of speech acts. Kimbrough and Moore do use the terms "message" and "speech act" as near synonyms. However, a distinction must be made between the form or means (what Searle calls the utterance act) and the essence or goal (what Searle calls the illocutionary act). In this way, it is possible that one message expresses several illocutionary acts. As we will see later, there may be good reasons for combining speech acts for speakers engaged in a symmetric conversation.

3 Meta-Analysis Patterns

In '92, Johnson [15] proposed to (informally) describe how to use frameworks, and their design by means of design patterns. Design patterns are borrowed from the field of architecture [1]. According to Alexander 'a pattern describes a problem over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution millions times over, without ever doing the same thing twice'.

Gamma [10] was one of the first to apply the notion of patterns to the discipline of software design. He defines design patterns as 'descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context' (ibid). At a higher level, Buschmann [4] discerns architectural patterns, that express the subsystems and components and their mutual relationships.

Patterns are usually described in a Context-Problem-Solution manner [4]. After the context of the problem has been explained, the problem itself is stated together with the solution to the problem. In order to use these patterns, the analyst has to translate them into the right context.

In this paper, we propose to apply the notion of patterns to the analysis of (electronic commerce) domains resulting in four subsequent levels of meta analysis patterns. Our patterns are partly based on linguistic theories, like the speech act theory (Austin, Habermas and Searle), and the semantic theory of narrative structures, and partly on information system design.

We explicitly use the term *meta*-pattern, because the patterns we introduce in this paper do not describe general representations of a problem domain, but denote general communication patterns which are domain-independent, and which can be found back in all analysis patterns that describe the environment of discourse. An analysis pattern can be defined as a 'group of concepts that represent a common construction in business modelling' [9].

The meta-patterns we introduce in this paper are intended both for *analysis* and for *reusability*. Meta-patterns are a useful analysis tool since they focus on the 'deep structures' of electronic business transactions rather than the form. Meta-patterns, once stored in an FLBC component library, can also be reused effectively from one occasion to another.

4 Patterns in Electronic Commerce

In this section, we apply the notion of patterns to electronic commerce transactions. We distinguish between four levels of (communicational) analysis meta-patterns (see Fig. 1) from low-level speech acts to high-level scenario's. *Transactions* are units composed of speech acts, for example, a request/commit. Transactions can be grouped in *workflow loops*. A *contract* represents a reciprocal relationship and typically consists of two workflow loops. Finally, a set of related contracts is called a *scenario*, an instance of a use case.

4.1 Speech Acts

Representation languages such as the Formal Language for Business Communication (FLBC - see section 2) and methods based on the Language/Action

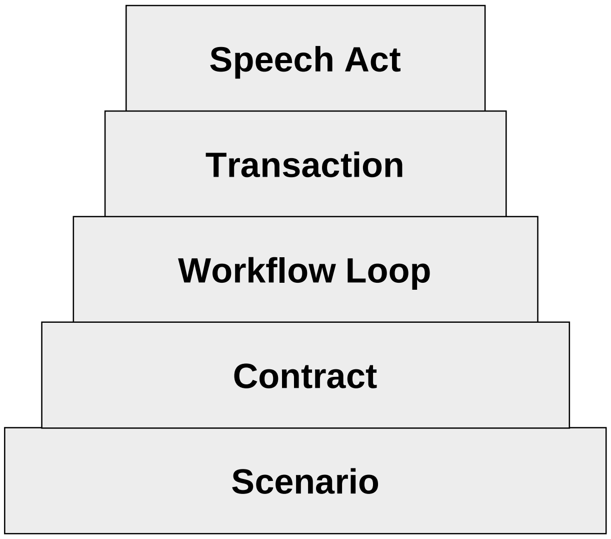

Figure 1: Levels of Meta-Anlysis Patterns

Perspective assume that the speech act is the most elementary unit within the communication between subjects.

Speech act theory was founded by Austin, who developed the 'language as an action theory'. A speech act focuses on what people are 'doing in saying something' [2]. A speech act can be defined as an utterance that in itself is constituted of a performative act, such as requesting and promising.

According to Searle [25], speech acts are constituted of three parts: the propositional contents, the illocutionary point and the illocutionary force. He distinguishes between five different illocutionary points: assertives, directives, commissives, expressives and declaratives. This taxonomy defines what the speaker can do on the basis of an utterance, with a propositional content.

FLBC-II uses only the assertions and directives, leaving out commissives,

```
msg(pers(cust1), pers(suppl3), request, delivery-product, msg627)
```

Figure 2: Example of a FLBC Message

$$\begin{array}{l}\underline{\textbf{MsgType}}\ request-product(\underline{\textbf{sender}}(\$p1),\underline{\textbf{receiver}}(\$p2),\\ product(\$x), date(\$d)) == \\ \quad (\underline{\textbf{request}}, delivery-product(\$x,\$d))\end{array}$$

Figure 3: Example of a MessageType declaration

expressives and declarations. However, they can be added when needed, since the language is not closed. Commissives are used to commit speakers to a future course of action. The expressive point expresses the subjective attitude of the speaker towards the state of affairs. Declarations are used to change the state of the world according to the proposition uttered.

An example derived from [18] is shown in figure 2, where 'delivery-product' is the propositional content, to be refined. It represents a message with id 'msg627' from 'cust1' to 'suppl3' with illocutionary force 'request'. For the use of meta-patterns, we want to generalize from an individual message to a message *type*, where the object instances are replaced by object types. For example see fig 3 (we omit the details of the product parameters $x and $d).

Note that the message type definition uses the same format as the message instance except for the message id. We have used a $-sign to indicate the parameters. In this way, it is possible to develop a list of message types for a certain domain. This list could include requests for delivery, requests for quotes, acceptance etc. When a message type is called with certain parameters, an instance is created with a generated message id.

4.2 Transaction

Typically, speech acts go in pairs, for example, a request followed by a commit. In the linguistic literature, it has been argued, *contra Searle*, that the speech act is not the basic unit of communication, but the message pair is (see e.g. [30] for a linguistic, and [13] for a language-philosophic discussion). So often (but not always) speech acts have no meaning on their own, but only as part of a bigger unit, which we call a transaction. For example, the request itself does not create an obligation as long as the Addressee has not agreed with the validity of the request.

We define a transaction as the smallest possible sequence of actions (speech acts) that has an effect in the social world of the participants, in other words an obligation, an authorization or an accomplishment [33]. In this sense, the transaction can be interpreted in a more abstract meta-analysis pattern, because it focuses at the (semantic), mainly deontic, effect of one or more speech acts. Deontic logic is the modal logic theory that deals with notions of obligations and permissions and that has been applied in law as well as in computer science [22]. Deontic consequences of (a sequence) of speech act play an important role during the representation of the electronic commerce transaction, because together they define the mutual rights and duties of the two parties, i.e. the *implications* of a message.

A transaction can be represented by means of a set of communicating subjects, communicative actions, constraints on the sequence of these actions and the goal and exit states. [29]. Note that our use of the word "transaction"

$\underline{\textbf{TransType}}\ request - product(\underline{\textbf{speaker}}(\$p1), \underline{\textbf{addressse}}(\$p2),$
$product(\$x),$
$date(\$d)) ==$
$\quad ([request - product(\$p1, \$p2, \$x, \$d, msg1),$
$\quad accept - request(\$p2, \$p1, \$x, \$d, msg2)], [\underline{\textbf{before}}(msg1, msg2)])$

Figure 4: Example of a TransType declaration

bears (intentionally) some resemblance to the classical database transaction, in particular, with regards to the atomicity property, and to more relaxed versions of these kind of transactions [8], but a major difference is the single-agent perspective in these theories versus the fundamental multi-agent perspective in our communicative approach.

To describe transactions in FLBC, we have to add a new construct 'trans' that takes a set of participants, a set of messages, a set of temporal constraints, and a transaction id. At type level, a transaction pattern can be defined as shown in figure 4.

As can be seen in this example, the combination of the request of the customer to deliver a product, and the promise of the supplier to actually deliver it, constitutes a deontic effect, e.g. an obligation to the supplier to deliver the product, and an obligation of the customer to pay the agreed price.

The *semantics* of the transaction consists of the set of possible message sequences (trace semantics). A transaction is valid if the temporal constraints are consistent, that is, if there is at least one possible trace. The deontic effects semantics are described at a higher level.

A particular kind of transaction is the factagenic and the actagenic conversation, each constituted of at least two speech acts.

4.2.1 Actagenic and Factagenic Conversation

DEMO, a Language/Action-based method to model business processes [7], distinguishes between an actagenic and a factagenic conversation. During the actagenic conversation an actor (for instance a customer) requests something from another actor (such as a supplier), which the latter can reject or accept. This leads to a commitment, or obligation for the supplier to keep his or her promise. The factagenic conversation starts after the executor (e.g. the actor that has commitment himself to perform a certain action during the actagenic conversation), has created the desired state of affairs, and is constituted of a declaration of the executor that (s)he is finished, and with an acceptance by or rejection of the initiator. In this way, the fact gets established, as both parties have committed themselves to it.

The delivery example 'request-product' above is an illustration of an actagenic conversation. Apart from request and accept (related to some action of the executor, in the example, the delivery), the transaction can be extended with messages such as 'reject' and 'counter'.

We take the actagenic and factagenic conversation as very basic meta patterns of communication at the transaction level. Each of them is constituted of at least two speech acts, unless the context makes one of them implicit. Being oriented at obligations and facts in the social world, respectively, they are central to all kinds of organizational communication. This does not mean that there are no more conversation types to be distinguished. We will see some of them later on.

4.3 Workflow

The next level that we distinguish is called "workflow" in accordance with the use of this term in the Action Workflow approach of [21]. The workflow can follow the model of the basic conversation of action, as defined by Winograd and Flores. It is assumed in the Business Process Modelling approaches based on the Language/Action Perspective (DEMO, Action Workflow) that the business processes are composed of workflow loops. The basic principles underlying this approach are:

- Actions are performed by subjects and *for* subjects. An action specification is not complete without the beneficiary role;
- Actions do have an effect in the object world, but to count as fact in the social world, the action must be reported and accepted by the initiator (usually the beneficiary). So the action specification is not complete without an evaluative communication afterwards;
- Both the request for action and the acceptance of a fact require a give-and-take, the active involvement of both parties.

Admittedly, these principles are not uncontroversial. Although we largely agree with them, our approach in this paper is liberal in the sense that we only claim that the workflow loops of the Language/Action Perspective constitute basic

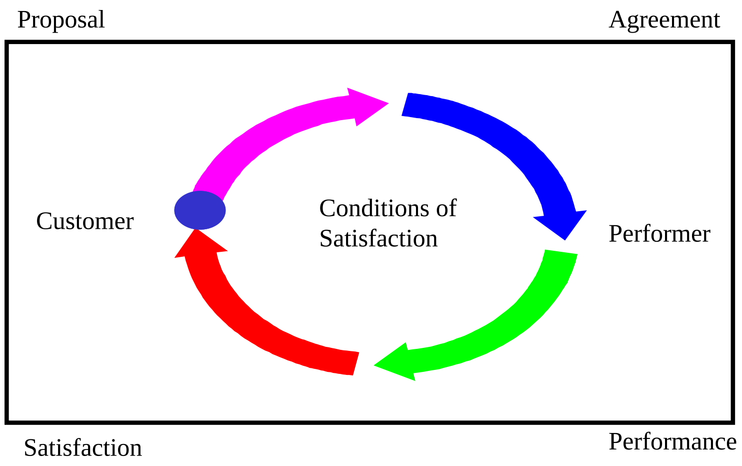

Figure 5: Workflow Loop

meta-patterns, but more may be identified later.

The workflow loop starts with a proposal, a request from the customer (or initiator) or an offer from the performer (or executor). In the second phase, the customer and the performer come to an agreement. After the executor has executed the promised action, he states/declares that (s)he is finished to the initiator. In the last phase, the satisfaction phase, the initiator can declare to the performer that the transaction was (un)successful.

The workflow loop is comparable to the transaction concept of DEMO. In DEMO a transaction is composed of three phases: the order phase, the execution phase and the result phase [23]. During the order phase, the actors involved come to an agreement regarding a future action, that is performed in the execution phase. The result phase terminates the transaction with an (reasoned about) agreement about the result of the execution phase. The order phase and result phase are consequently described by the actagenic and factagenic conversation.

The sequence in which DEMO transactions (we would say workflow loops) take place, is represented in the transaction process model. This model, which is not included in the Conversation of Action theory, consists of three levels: the success layer, the discussion-and-failure layer and the discourse layer. These layers describe subsequently, the resulting facts from a successful transaction, the validity-based discussion based on the theory of Habermas, and lastly, the background conditions of the conversation. By means of the communication framework, we are able to give a formal representation of the conversational transaction, thereby opening the door for claim-based reasoning.

A simplified example of a workflow loop is shown in figure 6.

The *semantics* of the workflow loop has been described in [33] by means of a Petri-Net that specifies the deontic effects of all transactions. The deontic effects are formalized in Dynamic Deontic Logic. In [29], a specification language for (so-called)contracts (CoLa) is spelled out. The translation of CoLa to an FLBC-like language can be done in the same way as we extended FLBC to transactions.

As can be seen in figure 6, the workflow is defined in terms of the mutual obligations between the initiator and the executor (in this case, only one) that are the consequence of the agreement between the two parties, and the way transactions yield or accomplish (or invalidate) an obligation. Since transac-

$\underline{\textbf{WflType}}\ delivery - product(\underline{\textbf{initiator}}(\$x), \underline{\textbf{executor}}(\$y),$
$product(\$p),$
$date(\$d)) ==$
$[\underline{\textbf{obl}}(\underline{\textbf{in}}(request - product(\$x, \$y, \$p, \$d)),\ \underline{\textbf{goal}}(deliver - product(\$y, \$x, \$p)),$
$\underline{\textbf{exit}}(cancel(request - product)))]$

Figure 6: Example of a WorkflowType declaration

tions are related to deontic states, there is an implicit ordering. For example, the cancel(request-product) cannot be performed before request-product. The example shows only one deontic state type, "obl" (obligation). In [29], a few more types are distinguished, including "failure", "accomplishment" and "authorization".

The workflow loop described here corresponds to what Winograd [34] has called the Conversation for Action. In addition, Winograd distinguishes conversations for clarification, possibilities and orientation. Each of them is supposed to have its own regularities of structure, but this structure remains implicit in all his publications. We will come back on the other conversation types in the next section. At this point, it suffices to remark that for each of these different conversation types we can identify a range of possible meta-patterns. Of particular importance for Electronic Commerce is the Conversation for possibilities that characterizes the negotiation phase of the business process.

The elaboration of these different conversation types is beyond the scope of this article, but we expect that this will lead to many interesting meta-patterns to be discovered.

4.4 Contract

The transaction models that we have just discussed give a rather biased perspective on the transaction. The analyst must either choose the viewpoint of the initiator or that of the executor of the transaction (in our case, the customer or the supplier). We follow Goldkuhl who claims that a business transaction must be interpreted as being an 'interchange process between a supplier and a customer' and that it 'involves the creation and sustainment of business relations' [11]). This view is elaborated in Goldkuhl's Business Action Theory (BAT), see fig. 7.

According to Taylor [27], all conversations have as their background a pattern of exchange. Not only at the level of communication (as we already noted above), but also at the object level. Reciprocity is a fundamental principle of human society, as Hubert Mauss already observed one century ago in his famous treatise on the gift. In commercial transactions, one object is usually a product and the exchange object consists of money. This is an example of what Taylor calls a symmetrical type of exchange (p.211). In this type of exchange the actors involved in the conversation have a common interest in a particular object. All organization is composed of a complicated set of exchanges, involving a balance between what March and Simon called inducements and contributions (*ibid*). In [33] we use the term 'contract' for the specification of such a symmetric agreement.

A contract involves at least two parties, but in practice may involve several (trusted) third parties. In commerce, the most obvious ones are the bank (for

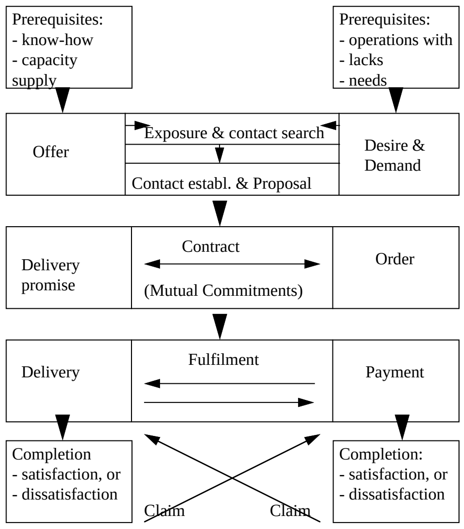

Figure 7: Business Action Theory, adapted from (Goldkuhl, 1996)

the money transfer) and the transporter (for the product transfer). An example of a third party in electronic commerce transaction is the 'EDI network provider' [24]. These parties can be thought of as being some kind of proxy or broker, through which all communication flows.

The reciprocal approach does not lead to two individual representations of the same business transaction, but rather to two patterns that interleave. Figure 9 shows an example of a reciprocal transaction pattern. The customer requests a certain product. The supplier on the other hand, requests money for it in return. Both transaction patterns are coupled by means of an agreement on the terms of exchange. This agreement, that describes the mutual obligations and authorizations is called the contract in BAT. The contract is established in the contract phase that precedes the fulfillment.

In this paper, we will use the term 'contract' in a wider sense of the interleaving of two workflow loops. As we have noted in the previous section, several types of workflow loops can be distinguished: besides conversation for action, we have conversations for possibilities, for orientation, and for clarification. These correspond roughly to the different phases in BAT. The contract underlying the fulfillment stage is a combination of two conversations for action. The contract (a better term in this case might be 'negotiation protocol') underlying the contracting stage is a combination of two conversations for possibilities. In the first stage (exposure), we typically find intertwined conversations for orientation. Conversations for clarification are a bit special. In the particular sense of dealing with breakdowns concerning the conditions of satisfaction, such conversations will occur typically in the last (completion) stage; however, they may also interrupt other conversations and whereas the symmetry is expected for the other conversation types, it is not so in this case.

So we can distinguish different types of contracts (corresponding to different BAT stages), and for each type we can develop different meta-patterns. An example of a meta-pattern in the realm of 'contracting contract' is the Contract Net protocol [36]. In this paper, we will limit ourselves to the 'fulfillment contract'. Figure 8 gives a formal representation of such a contract at the type level.

Note that the order in which the different workflow loops take place (defined by the message sequence), as defined by the temporal constraints, is dependent on the trade procedure that is agreed upon (cf. [3] and [19]). For instance con-

ContractType *Deliver_Product* (**customer**($\$x$),
supplier($\$y$), *product*($\$p$), *date*($\$d$)) ==
[$request - product(\$x, \$y, \$p, \$d)$, $request - money(\$y, \$x, \$p)$],
[**before**(*request_product.request_product*, *request_money.accept_request*),
before(*request_product.accept_request*, *request_money.accept_request*])

Figure 8: Example of a ContractType declaration

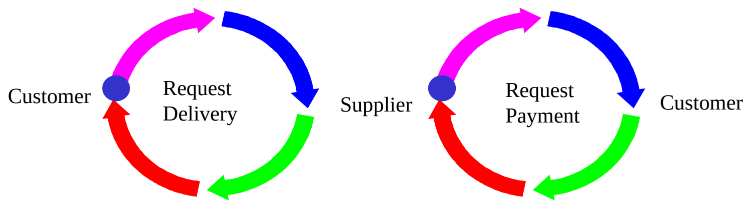

Figure 9: Contract

cerning the way of payment, the supplier can demand receipt of payment before sending the goods or, alternatively, only after delivery of the goods. In other words, the two workflow loops that constitute the contract can be intertwined in different ways, leading to a different balance of risks. The standard alternatives mostly used in international trade lead to different contract meta-patterns. These contract meta-patterns may or may not include the trusted third parties. See [26], among others, for a discussion of the formal properties of delegation and the resulting different levels of obligation.

As we said earlier, a particular message may represent more than one illocutionary act. We think this is particularly relevant in reciprocal interactions. For example, in negotiations one party may want to make an offer but only if the other party complies with a request, and vice versa. This deadlock situation can be broken by bundling the request and the offer in one message; the other party then has the possibility to accept the whole (or not, of course), thereby progressing both workflows in the same pace. There may be other solutions to avoid or repair deadlocks, but this problem, stemming from the reciprocal character of business conversations, is an argument in favor of the differentiation between message and speech act.

4.5 Scenario and Context

Language/Action-based methods focus on conversation patterns, such as the basic conversation for action pattern [35]. However, we take over the hypothesis of Taylor that the representation of the conversation, i.e the inter-action, must be translated into a text to be understood. As Taylor puts it [28], the context of a conversation is defined by 'identities of the speaker and hearer, physical and other incidental circumstances of time and place, the object of the conversational exchange, and the probable intentions of the speaker'.

In other words, in order to be interpreted the conversation has to be placed in a certain context. As has been argued in [14], an important element of the context of an electronic commerce transaction is the *identification* of the communicating actors, as well as their actions. By means of the identification speech act, the speaker tries to set up a communicative relationship with the hearer. Identification in Electronic Commerce typically requires a Domain Administrator that provides identities to new members and can be asked to check the identity of an agent.

To conceptualize the *structure* of the text, Taylor draws on previous work in the field of semiotics, in particular, Greimas [12]. The minimal story element or narrative function is composed of a beginning, a development and an end. It is typical for stories that the final state is the inverse of the initial state. For commercial transactions, this applies easily: in the beginning, the supplier has goods and the customer has money, in the end, the customer has the goods and the supplier the money. Since texts (stories) do show a structure, we can again distinguish meta-patterns. Admittedly, these patterns are very high-level. When compared to the 'conventional patterns' that are used in the area of software development, we could compare these patterns with the architectural patterns as discussed in section 3.

We have found the concepts of context, text and story to be useful, but where Taylor tends to identify these concepts, we think a distinction between context and story is necessary. There is always a context, and this context can be more or less explicit. When an actor starts a conversation, a communication domain is created which serves as context for the messages exchanged. In EDI, a distinction is made sometimes between content-messages and context-messages. Orders, offers and the like are content-message. A proposal or acceptance to comply to Dutch Trade Law is a message that updates the context: in this case, it "grounds" the obligations and authorizations created by the content-messages. During the development of the conversation, the subjects may make use of stored communication patterns by instantiating them in the current domain. The story (or scenario, as we prefer to call it) is the highest level of communication pattern that can be used, but it is not excluded that the subjects do not adhere to one single large pattern, but make use of different smaller patterns in sequence.

In open edi [17], business partners may engage in short-term on-the-spot relationships. In that case, they may want to use a meta-pattern for a complete text, starting with identification and followed by a fixed pattern of exchanges. This is very fast way of working through a business transaction. Alternatively, they may want to enter a relationship first and then negotiate the terms of the (fulfillment) contract ad hoc. The latter approach is of course much more flexible.

We use the term 'scenario', instead of its linguistic counterpart 'story' since this term is more familiar in Electronic Commerce practice. A scenario is comparable to a use case, that is used within the field of software engineering to denote a sequence of transactions performed by an actor in dialogue with an information system. The scenario need not be limited to the communication between two parties. An example of a scenario based on the Post-Payment case of [19] is given in Figure 4.5. This example has been worked out in [32].

This scenario contains not only a Shipper and Consignee (Seller and Buyer), but also a Sea-Carrier and two banks. Each of the (agency) relations between these parties is a contract. But there may also be dependencies over contracts, for example, when the constraint is formulated that the Consignee does not do the payment before he has evidence that the Sea-Carrier has accepted the shipping order. So, in the same way as a contract consists of several workflow loops and constraints between them, so does the scenario consist of several contracts and temporal constraints between them (or between the transactions

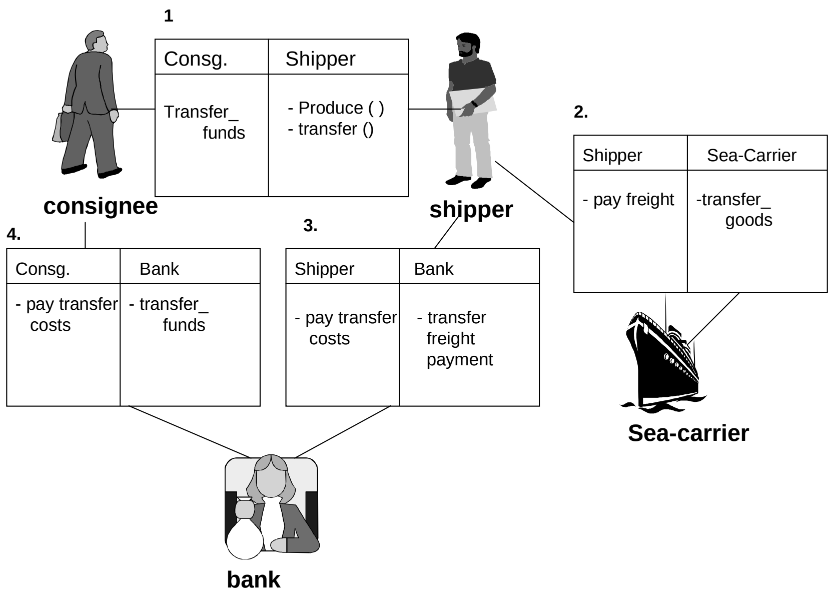

Figure 10: The Post-Payment Scenario

in them).

The BAT framework can be viewed as a prototypical scenario meta-pattern, but taking more detail into account, different business processes can be distinguished. In [20], a case study is presented performed at a Swedish manufacturer in which different 'variant' business processes were identified: 'standard stock customer', 'special production customer', 'whole trading customer', 'long-term agreement customer' and several more. The processes differ in the production process used and the involvement of subcontractors and intermediaries.

Our definition of a scenario is still preliminary, since we have to work out more case studies before its structure can be fixed and before we can identify important meta-patterns. As a starting-point, we define a scenario to consist of three parts. The central part is the set of contracts (or lower-level units) and temporal constraints between them. The Post-Payment scenario can be represented in this way. Secondly, the scenario specifies the conditions on the context. These conditions must be checked when the scenario is instantiated in a given context. In the case of the Post-Payment scenario, these conditions specify, for instance, that the Shipper has an obligation to deliver and the Consignee has an obligation to receive the goods. The third part of the scenario describes the goals of the participants. The scenario can usually unfold in different ways, but there is typically one successful way and desired outcome. The progress of the process can be measured against the fulfillment of the scenario goals.

5 Applying meta-patterns: the FLBC component library

In this paper, we have distinguished different abstraction levels and have claimed that at each level, meta-patterns can be developed that can be reused from one application to another. In this section, we will briefly indicate how this reuse can be implemented.

In [18] a prototype FLBC system is described written in Prolog and with a graphical interface by means of which subjects can choose from a set of possible message types and actions. In this way, the illocution of the message is already given, and the user only has to type in the information required for that specific message type.

In the implementation that we envisage, the message types are stored in an

FLBC component library, and they can be used in the same way as in the FLBC system described above. Such a component library may be provided as an EC service by a Trusted Third Party. However, this component library contains not only message types, but also higher level patterns, such as transaction patterns, workflow loop patterns and contract patterns, up to scenario patterns. When a subject selects a transaction pattern, the subsequent choice of message types will be restricted to the ones defined in that transaction. For the receiver, the choice of reply messages is restricted in the same way.

The same applies to higher-level components, but the use of these is increasingly a matter of negotiation itself rather than a choice of one subject. For example, during the contracting stage of the business process, the parties may discuss and reach agreement on the kind of scenario that they want to adopt in the fulfillment stage. Before the contracting stage, the parties may discuss the kind of negotiation protocol (contract, in our sense) that they want to follow. For this reason, it is necessary that the FLBC component library is transparent in the sense that the patterns are identified objects that can be referenced in the communication.

Once the parties involved have chosen for a certain scenario or contract, the Trusted Third Party may monitor the progress and provide information about the current state. In this way, the parties have mutual knowledge about what has been reached so far and what is on the road ahead.

6 Conclusions and future research

In this paper, we have argued that FLBC components can be used profitably for the rapid development of open-edi protocols. This rapid development is needed in order to support rapid project-based partnerships.

We have argued that there is a need for larger structural components than just messages, and have identified several layers of meta-patterns, starting from speech acts to transactions, contracts, workflow loops and scenarios. Depending on the application at hand, users may want to use complete 'architectural' patterns at scenario level, or build up a scenario themselves using lower-level components. The syntax definition of our pattern language is provided in the Appendix. Note, however, that this definition is at best a starting-point.

In the last section we have briefly sketched a software architecture in which the patterns are made available in the form of an FLBC component library. The layered architecture integrates several existing theories of business process modelling and workflow.

The formal semantics of speech acts as well as the higher-level components is not described in this paper. We refer to our previous work for more details about Deontic Dynamic Logic and the semantics of speech acts, e.g. [33] [29].

We acknowledge that the present paper leaves more questions than it answers. However, as far as we know it is the first systematic attempt (in the area of Electronic Commerce) to identify structural components beyond single messages. Although we want to adhere to solid formal semantics of our language proposals, we run into the problem that logic is traditionally focused on single sentences as well. Therefore we have chosen for the approach that we ground our concepts in logic, but abstain from providing something like compositional semantics. This is certainly a topic for long-term future research, as soon as the language itself has grown more stable.

In the near future, we will start implementation of the FLBC component library in the context of a European project on electronic commerce (MEMO). This is one way of verifying the proposal. In parallel, we will work on the analysis of existing trade procedures (in the line of [19]) in order to represent these in the form of patterns that can be included in the FLBC component library. Case studies are an important means for validating the proposal.

References

[1] Ch. Alexander, S. Islawa, M. Silverstein, I. with Jacobson, I. Fiksdahl-King, and S. Angel. *A Pattern Language.* Oxford University Press, 1977.

[2] J.L. Austin. *How to do things with words.* Clarendon Press, 1962.

[3] R.W.H. Bons, R.M. Lee, and C.D. Wrigley. Modeling inter-organizational theories trade procedures using documentary petr nets. In *Proceedings of the Hawaii International Conference on System Sciences*, pages 189–199, January 1995.

[4] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture.* John Wiley and Sons, 1996.

[5] M.A. Covington. Toward a new type of language for electronic commerce. In *Proceedings of the 29th Annual Hawaii International Conference on System Sciences*. IEEE, 1996.

[6] G. De Michelis and M. Grasso. Situating conversations within the language/action perspective: The milan perspective. In *Proc. CSCW '94*, 1994.

[7] J.L.G. Dietz. Business modelling for business redesign. In *Proc. HICSS '94*, pages 723–732. IEEE Press, 1994.

[8] A. Elmagarmid, editor. *Database Transaction Models for Advanced Applications*. Morgan Kaufman, 1992.

[9] M. Fowler. *Analysis Patterns: Reusable Object Models*. Addison-Wesley, 1997.

[10] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Abstraction and Reuse of Object-Oriented Design*. Addison-Wesley, 1995.

[11] G. Goldkuhl. Generic business frameworks and action modelling. In F. Dignum, J. Dietz, E. Verharen, and H. Weigand, editors, *Workshop on Communication Modelling - The Language/Action Perspective*, 1996.

[12] A.J. Greimas. *Du sens*. Paris: Editions du Seuil, 1970.

[13] J. Habermas. *The Theory of Communicative Action, Reason and the Rationalization of Society*, volume 1. Polity Press, 1981.

[14] W-J. van den Heuvel and H. Weigand. Ensuring the validity of electronic commerce communication. In Jan Dietz Frank Dignum, editor, *Communication Modelling - The Language/Action Perspective*, Computer Science Reports. Technical Univ of Eindhoven, 1997.

[15] R.E. Johnson. Documenting frameworks using patterns. In *OOPSLA '92*, pages 63–76, 1992.

[16] S.O. Kimbrough and R.M. Lee. On illocutionary logic as a telecommunications language. In *Proceedings of the International Conference on Information Systems*, San Diego, December 1986.

[17] S.O. Kimbrough and R.M. Lee. On formal aspects of electronic commerce: examples of research issues and challenges. In *Proceedings HICSS'96*. IEEE Computer Society Press, 1996.

[18] S.O. Kimbrough and S.A. Moore. On automated message processing in electronic commerce and work support systems: Speech act. *ACM Transactions on Information Systems (TOIS)*, 1997.

[19] R.M. Lee and R.W.H. Bons. Soft-coded trade procedures for open-edi. *International Journal of Electronic Commerce*, 1(1):27–49, 1996.

[20] M. Lind and G. Goldkuhl. Reconstruction of different business processes - a theory and method driven analysis. In J. Dietz Dignum, F., editor, *Communication Modelling - The Language/Action Perspective*, Computer Science Reports. Technical Univ of Eindhoven, 1997.

[21] R. Medina-Mora, T. Winograd, R. Flores, and F. Flores. The action-workflow approach to workflow management technology. In J. Turner and R. Kraut, editors, *Proc. Of 4th Conf. On Computer Supported Cooperative Work (CSCW '92)*. ACM Press, 1992.

[22] J.-J. Ch. Meyer. Deontic logic: A concise overview. In R. Wieringa J.-J.Ch. Meyer, editor, *Deontic Logic in Computer Science, Normative System Specification*. John Wiley and Sons, Ltd, 1993.

[23] V.E. van Reijwoud and H.B.F. Mulder. Speech act based modelling for workflow management systems: A case study. In F. J. Dietz Dignum, editor, *Communication Modeling - The Language/Action Perspective*, Computer Science Reports. Eindhoven University of Technology, 1997.

[24] M. Roscheisen and T. Winograd. A communication agreement framework for access/action control. Technical report, Stanford University, 1996.

[25] J. Searle. *An essay in the philosophy of language*. Cambridge University Press, 1969.

[26] Y.H. Tan and B. Firozababdi. Modelling the dynamics of contract negitiation and execution. In Jan Dietz Frank Dignum, editor, *Communication Modelling - The Language/Action Perspective*, Computing Science Reports. Technical Univ of Eindhoven, 1997.

[27] J.R. Taylor. *Rethinking the Theory of Organizational Communication*. Ablex Publishing Corporation, 1993.

[28] J.R. Taylor, F. Cooren, N. Giroux, and D. Robichaud. The communicational basis of organization: Between the conversation and the text. *Communication Theory*, 6:1 – 39, 1996.

[29] E. Verharen. *A Language/Action Perspective on the Design of Cooperative Information Agents*. PhD thesis, Tilburg University, 1997.

[30] E. Weigand. *Sprache as Dialog: Sprechakttaxonomie und kommunikative Grammatik*. Niemeyer Verlag, (in german) edition, 1989.

[31] H. Weigand. *Linguistically Motivated Principles of Knowledge-based Systems*. Foris, Dordrecht, 1990.

[32] H. Weigand, W.-J. van den Heuvel, and F. Dignum. Modeling electronic commerce transactions: A layered approach. In *Proceedings of the Langauge/Action Perspective Workshop, Steningevik, Stockholm*, 1998.

[33] H. Weigand, E. Verharen, and F. Dignum. Dynamic business models as a basis for interoperable transaction design. *Information Systems*, 22(2/3):139–154, 1997.

[34] T. Winograd. A language/action perspective on the design of cooperative work. In I. Greif, editor, *Computer Supported Cooperative Work: A Book of Readings*. Morgan Kaufmann, 1988.

[35] T. Winograd and F. Flores. *Understanding Computers and Cognition*. Addison-Wesley Publishing Company, 1986.

[36] G. Zlotkin and J.S. Rosenschein. Cooperation and conflict resolution via negotiation among autonomous agents in noncooperative domains. *IEEE Transactions SMC*, 21(6), 1991.

[37] V. Zwass. Electronic commerce: Structures and issues. *Int. Journal on Electronic Commerce*, 1(1), 1996.

A Multi-Levelled FLBC-II EBNF Grammar

We have used the Backus-Naur Form to describe the syntax of the multi-levelled FLBC-II.

```
/* Symbols used                                                       */
() grouping
[] one or zero time repitition
{} zero or more time repitition
| or

/* Electronic Commerce Transaction Specification                     */


/* EC Specification                                                   */
ec_specification: BEGIN scenariolist contractlist  workflowlist transactionlist
                speech_actlist END ;

scenariolist    : scenario scenariolist
| scenario


scenario        : SCENARIOTYPE scenarioname ';'
                  scenario body

scenario_body   : '('
                  IDENTIFICATION ident ';'
                  CONTRACTSlist ';'
                  TRANSACTIONSlist ';'
                  TERMINATION
                  ')'

ident           : DOMAIN domainname '(' SUBJECTS {subjects} ','
                  IDENTIFICATION idents ')'

CONTRACTSlist : contracts CONTRACTSlist
| contracts
contracts : CONTRACTS contractspec
contractspec : '(' '['
    {args}
  ')' ']'

args : argname'('arg')' {','argname'('arg')'}
argname : ident
arg : '$'ident
```

```
TRANSACTIONSlist: transactions TRANSACTIONSlist
| transactions
transactions : TRANSACTIONS transactionspec
transactionspec : '(' '['
  {transaction_label}
  ')' ']'
transaction_label: transactionname '(' {arg} ')' {',' transactionname '(' {arg} ')' }
tranactionname : identifier


/* Contract */
contractlist    : contract contractlist
| contract
contract : CONTRACTTYPE contractheader contractbody
contractheader : contractname contractarg
contractarg : CUSTOMER customername ',' SUPPLIER suppliername {args}
customername : identifier
suppliername : identifier

contractbody : '(' {person} workflowloopname contractconstraintsclause ')'
person : PERSON '(' name ')' personclause
personclause    : ',' PERSON '(' name ')'
                |

name : identifier
workflowloopname: identifier
speechactname : identifier

contractconstraintclause : contractconstraints
 |

contractconstraints: workflowloopname'.'transactionname BEFORE|AFTER
                     workflowloopname'.'transactionname
|
workflowloopname BEFORE|AFTER workflowloopname

/* WorkflowLoop */
workflowlist    : workflowloop workflowlist
| workflowloop
workflowloop : WORKFLOWLOOPTYPE workflowloopheader workflowloopbody
workflowloopheader: workflowloopname workflowlooparg
workflowloopname  : identifier
workflowlooparg : INITIATOR initiatorname ',' EXECUTOR executorname argclause
argclause : {',' args}
|
workflowloopbody: '(' {person} clause ')'

clause : clausename ':'oblclause '('agent ',' action_spec ')' |ACC'('agent')'
 IN ':' transaction transactionclause
 deadlineclause
 goalclause
 exitclause
 modifyclause

oblclause       : OBL|AUT

deadlineclause  : DEADLINE ':' obl_deadline
                |

transactionclause: {',' transction}
|

clausenameclause:{'&'clause_name}
|

goalclause : GOAL ':'{action_spec '=>' clause_name clausenameclause}
|

exitclause : EXIT ':' {(cancel '(' action ')' | transaction) '=>'
                  clausename clausenameclause}
|

modifyclause : MODIFIED BY {(action_spec|message)}
|

clausename : identifier
obl_deadline : action_spec | time | action_spec '+' time | now '+' time '<<'
  action_spec '<<' action_spec|time|action_spec'+'time|now'+'time
  | actionspec '!' | action_spec ASAP

actionspec : transaction | action
transaction : transname argclause
transname : identifier
action : actionname argclause
actionname : identifier


/* Transaction */
transactionlist : transaction transactionlist
| transaction
```

```
transaction : TRANSTYPE transheader transbody
transheader : transname transargsclause
transargsclause : {transargs}
|
transargs : SPEAKER speakername ',' ADDRESSEE addresseename ',' argclause

speakername : identifier
addresseename : identifier

transbody : '(' {person} saclause transcclause ')'
transcclause : {trans_constraint}
|
saclause : {speechact}
speechact : speechactname'('{args} ',' mesident ')'
speechactname : identifier
trans_constraint: BEFORE'('mesid')'
mesid : mesident','mesident
mesident : identifier

/* Speech Act */
speech_actlist  : speech_act speech_actlist
| speech_act
speech_act : MSGTYPE msgheader msgbody

msgheader : SENDER sendername ',' RECEIVER receivername argclause
sendername : identifier
receivername : identifier

msgbody : '(' {person} stclause ')'
stclause : {speechact ',' transact}
speechact : ACCEPT | ASSERT | SUGGEST | COMMIT | CLAIM | ASSERT | NOMINATE |
                  RETRACT | FORBID | PERMIT | CONFIRM | COMMAND | REQUEST |
                  PROMISE | REJECT
transact : transname
%%
```