

# Interoperable transactions in business models - a structured approach

H. Weigand, E. Verharen,  
Infolab, Tilburg University, Tilburg, The Netherlands  
and F. Dignum  
Fac. of Maths. & Comp. Sc., Eindhoven University of Technology,  
Eindhoven, The Netherlands

**Abstract.** Recent database research has given much attention to the specification of "flexible" transactions that can be used in interoperable systems. Starting from a quite different angle, Business Process Modelling has approached the area of communication modelling as well (the Language/Action perspective). The main goal of this paper is to provide some useful structuring mechanisms for interoperable transactions based on the Language/Action perspective. The paper thus tries to build a bridge between two rather separated worlds: the research on interoperable transactions on the one hand, and the research on business process models on the other. Extended deontic logic provides the material for this bridge. To better structure the specification, a distinction is proposed between the transaction level, the task level, and the contract.

**Keywords:** cooperative information systems, business process models, communication, interoperable transactions, deontic logic

## 1. Introduction

The field of Information Systems Engineering no longer concerns itself only with the development of 'isolated' Information Systems (IS). With the advent of high-speed local and global network communications, more and more applications, like EDI, Workflow management and collaborative computing, are being developed that access different independent resources inside and outside the organization. Integration of the various resources might not be possible for technical or organizational reasons, hence the growing reliance on *interaction* between systems. Behaviour of such systems, called Information and Communication Systems, or Cooperative Information Systems (CIS), is described by means of interoperable transactions defined as a logical unit of work, involving different autonomous agents. Interoperable systems cooperate by means of message passing; a specification of an interoperable process consists essentially of a set of communicating agents, the possible message types provided by each agent to support its role, and constraints on the synchronisation of messages.

A notorious problem with interoperable transactions, is that isolation cannot be maintained because of the long life-span. This means that certain accomplishments can become invalid later. For example, a flight reservation has been made, but the airline

company cancels the flight. In the ConTracts model ([Wächter & Reuter,1992]), exit invariants can be specified that have to be maintained by the system for the duration of the ConTract. If they are violated, specific conflict resolution methods are invoked, such as compensation. In Interactions ([Nodine et al.,1994]), the problem is known as weak conflicts. It is recognised that weak conflicts do not fit well into the block structure of a specification language. They are treated more like exceptions and specified by means of event-condition-action rules. The "action" part is always an abort, that is, backtracking (using compensation) to the point where an alternative can be tried. Backtracking can be a very rude way, as [Nodine et al.,1994] admits, but it seems to be impossible to specify a satisfactory general failure handling mechanism. In general, a weak conflict requires a rescheduling of the task components that preserves as much of the original schedule as possible, optimises costs to be made by compensations etc.

Recent years have shown a growing awareness that linguistic theories are relevant for IS design in general and CIS in particular. The so-called Language/Action approach gives content to a new generation of Business Process Models [Teufel & Teufel,1995]. Examples are DEMO [Dietz,1994] and Action Workflow [Medina-Mora et al.,1992]. Underlying these models, the concepts of *obligation* and *authorization* play an important role. An obligation is the result of a commitment to perform a certain act and authorizations restrain or allow the commitment to and operation of an act (including doing other communicative acts). The logic to reason about obligations and related concepts is called *deontic* logic [Wieringa et al.,1989]. The application of deontic logic to communication modelling was first suggested in [Weigand,1993] and worked out in [Dignum & Weigand,1995].

The "weak conflicts" mentioned above usually originate from a violation of an obligation. For that reason, our approach insists that obligations and authorizations of service providers and receivers are modelled. In this way it is possible to reason over any uncooperative action rather than classify them as a true failure (leading to unnecessary rollback or recovery actions of transactions).

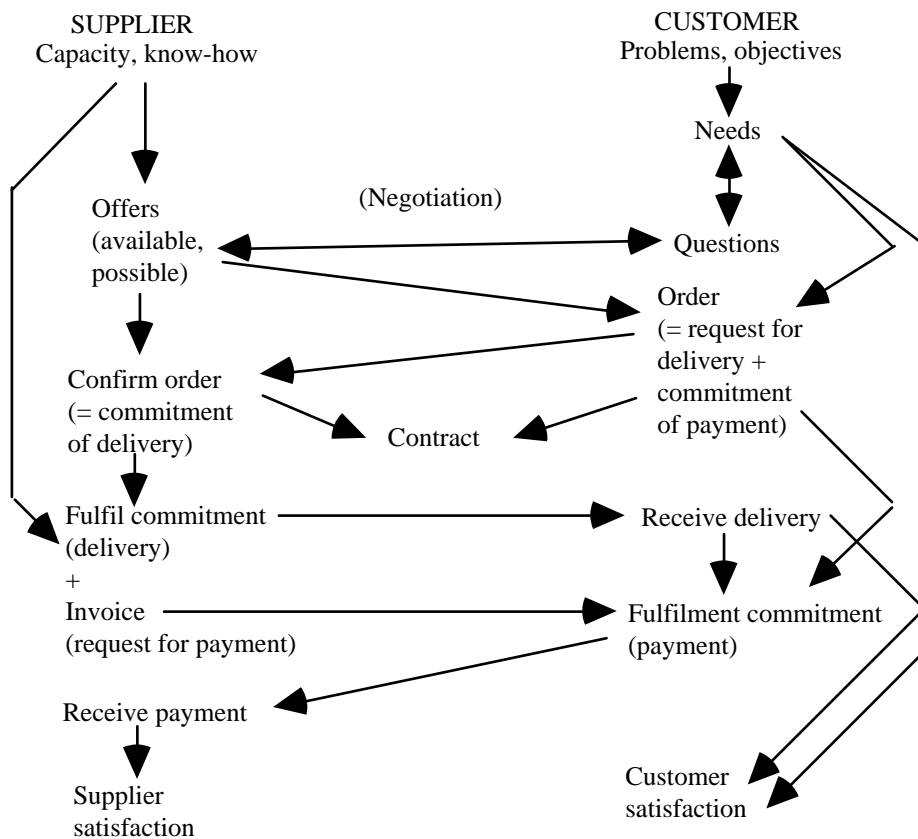
(Extended) deontic logic is a convenient tool for expressing the semantics of interoperable transactions, but does not impose any structure on the specification. Interoperable transaction specifications tend to become rather complicated, because of all possible exceptions that must be envisaged and the absence of one central control agency. This not only causes methodological problems in the design phase, but also reduces evolution and reusability of specifications.

The main goal of the present paper is to provide some useful structuring mechanisms for interoperable transactions based on a Language/Action perspective to communication. Loosely speaking, the paper tries to build a bridge between two hitherto separated worlds: the research on interoperable (extended) transactions on the one hand, and the research on business process models on the other. Extended deontic logic provides the material for this bridge.

In section 2, we start with a business process model proposed by [Goldkuhl,1995] that will be used as leading example in the rest of the paper. To illustrate the feasibility of deontic logic, we use it to formalise the semantics of this model. In section 3, a three-level architecture will be introduced for structuring the specification. A distinction is made between task level and transaction level. Moreover, failure handling is dealt with separately by a contingency plan and a contract, respectively. In section 4, this framework is discussed with a view to reusability and dynamic evolution. We conclude with a brief comparison with related works.

## 2. Business Process Example

Business process is a central concept in the field of Business Process Reengineering. A business process can be defined as a series of coherent activities (value chain) that creates a result with some value for an external or internal customer. In his paper on communicative action, [Goldkuhl,1995] discusses the well-known framework by Michael Porter, and notes some theoretical weaknesses. As an alternative, he describes a communicative action model of business relations and business processes (figure 1).



**Fig.1.** A communicative action model of business relations and business processes (after [Goldkuhl,1995], fig.5)

What distinguishes this model from other BPR models is (a) the focus on interactivity, rather than input-output transformations, and (b), its *generic communicative logic*. For instance, once an order is defined as a mutual obligation (to

deliver and to pay, respectively), the fulfilment is a logical consequence, as is the (undesirable) possibility of non-fulfilment (not pictured yet). That the obligation must be mutual can be inferred from the very nature of a business transaction. It is also quite natural that mutual commitments can only be established after a stage of negotiation. The communicative logic suggests a distinction of four stages in the interchange process between supplier and customer:

1. inquiry and negotiation stage
2. contractual stage
3. fulfilment stage
4. satisfaction stage

It should be noted that no specific messages or actions for the satisfaction stage are given in the model. This is perhaps because satisfaction is often implicit, and only dissatisfaction leads to communicative acts, such as appeals to warrant.

The communicative logic behind the model of Goldkuhl can be made even more precise by formalising the communicative acts (messages) in deontic logic. Deontic logic is an extension of dynamic logic with two operators,  $Obl(i, \alpha)$  and  $Aut(i, \alpha)$ , for *obliged* and *authorized*, respectively. Besides  $Obl$  and  $Aut$ , we include one more primitive operator in the deontic specification language. This is  $Acc$ , for accomplishment.

- The semantics of an authorized request to do  $\alpha$  is that  $Obl(i, \alpha)$  holds (as a postcondition) provided that  $Aut(i, \alpha)$  (as a precondition). In other words, an authorization is a *conditional* obligation.
- $Obl(i, \alpha)$  means that  $i$  not doing  $\alpha$  leads to *violation*. An obligation is weaker than a logical necessity; violation does not lead to inconsistency. Independent rules specify what such a violation implies, e.g. a sanction. These rules are usually expressed in terms of other  $Obl$  and  $Aut$  formulas.
- $Acc(\alpha)$  means that action  $\alpha$  has been executed, and hence the obligation to do  $\alpha$  is fulfilled.

The following gives some examples related to Fig.1 now written in Deontic Logic (where  $[\alpha]q$  means that after action  $\alpha$ ,  $q$  holds,  $DIR$  (directive) stands for the action request):

$$[give-quotation(j, i, g, p)] \\ Aut(i, DIR(i, j, deliver(j, i, g, p)))$$

*If a company gives a quotation for a certain price (p) the client is authorized to order the product (g) for that price. (This could be a meaning definition for give-quotation).*

$$Aut(i, DIR(i, j, deliver(j, i, g, p))) \emptyset \\ [DIR(i, j, deliver(j, i, g, p))] (Obl(j, deliver(j, i, g, p)))$$

$$[deliver(j, i, g, p)] Aut(j, DIR(j, i, pay(i, j, p)))$$

*If a customer is authorized to order a product for a certain price (i.e. a quotation has been given for that price) then the company is obliged to deliver the product after the customer has ordered it. But after delivery of the product, the company is authorized to order the customer to pay for it.*

$$\text{Aut}(j, \text{DIR}(j, i, \text{pay}(i, j, p))) \text{ } \emptyset$$

$$[\text{DIR}(j, i, \text{pay}(i, j, p))] \text{Obl}(i, \text{pay}(i, j, p))$$

*If an order has been delivered (and authority acquired to request payment) a request for payment induces an obligation for the customer to pay.*

Note that the obligation to pay is made conditional: it becomes effective only when the supplier requests for it. This is of course one way of doing it; the obligation can also be instantiated directly when the order is given, and even precede the delivery. These different ways of working can be distinguished in the logic when needed.

Deontic logic can also be used to reason about violations. This part of the specification is not included in Goldkuhl's picture, but will be part of the business contract as well. This holds also for cancellations, which means a retract of an obligation before it has been fulfilled.

Another thing about which the logic can and must be explicit is the expiration of authorizations and obligations. In Deontic Dynamic Logic, this is accounted for by explicit or implicit frame axioms. For example:

$$\text{Obl}(i, \text{pay}(i, j, p)) \text{ } \emptyset$$

$$[\text{pay}(i, j, p)] \text{aut}(j, \text{DIR}(j, i, \text{pay}(i, j, p)))$$

*After the customer has paid, the company cannot request another payment again.*

Goldkuhl emphasises the importance of the satisfaction stage and the *mutual* satisfaction that must be the goal. In the Deontic Logic, two levels of satisfaction can be distinguished. The basic level of satisfaction is reached as soon as the obligations of both partners have been fulfilled, that is,  $\text{Acc}(\alpha)$  is true for all actions  $\alpha$  in the contract. This is in fact the borderline between fulfilment and satisfaction. However, the contract may also describe authorizations for both partners to make claims in the case of dissatisfaction. The second level of satisfaction, and the real end of the interaction, is when these authorizations have expired as well. We will come back to the mutual satisfaction, and Goldkuhl's deviation from the Action Workflow model ([Medina-Mora et al. 1992]) in this respect, in section 4.

The borderlines between the other stages can be expressed in deontic logic as well. The negotiation stage ends with the establishing of certain *authorizations*.. The contractual stage ends with the establishing of certain (usually, mutual) *obligations*, making up the contract. In the case that the authorizations have been negotiated beforehand (cf. section 4), the contractual stage can be entered right away.

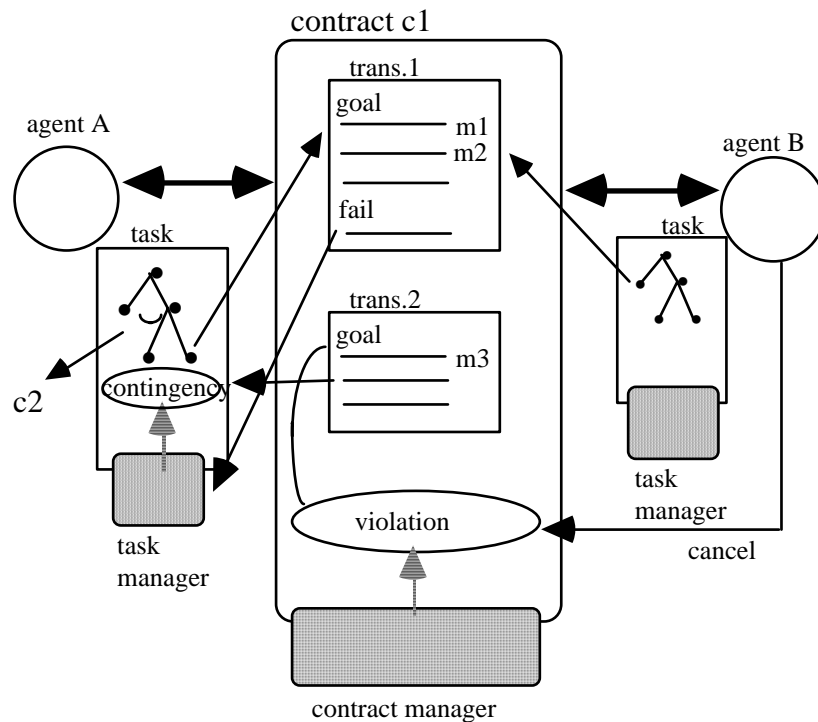
More details about the logic can be found in [Weigand et al.,1995] and [Dignum et al.,1995] where it has been extended with temporal deontic constraints, making it possible to describe for instance deadlines and the request to do some action "as soon as possible". Although deontic formulas describe the exchange between the customer and the company exactly and completely, they lack structure. In [Dignum & Weigand, 1995] we have introduced provisionally a language CoLa (Communication Language) with a more readable syntax that structures the (speech) acts into transactions. In the next section, the CoLa framework will be described in more detail, and we will argue for a distinction between task level and transaction level.

### 3. CoLa Framework

The specification of interoperable transactions needs structure for the sake of complexity reduction and reusability. Besides the notions of messages, transactions and services introduced in our earlier paper ([Dignum & Weigand, 1995]), we distinguish contracts and tasks. We use the general term "agent" for the participants in the interoperable transaction (e.g., the supplier and customer). From an analysis point of view, the agent is a real-world person or company. From a design point of view, the agent can also be a piece of software to which the person or organization has delegated some of his tasks.

Fig. 2 gives a schema in which the CoLa concepts are related. An agent has a certain task that can be split up into subtasks that the agent tries to fulfil. In doing so the agent initiates the necessary transactions. Transactions are seen as a communication process oriented towards a goal, the possible message types provided by each agent and constraints on the synchronisation of messages, together with a specification of what should happen in the case of a failure. In case of an unrecoverable failure the task manager of the agent should be notified so that he can pursue an alternative.

Transactions are part of a contract describing the communication behaviour between two agents concerning some business relation and process. The contract also specifies what should happen in case of violation of one of the obligations or cancellation by one of the agents (by notifying the Contract Manager), possibly leading to other obligations described by another transaction in the contract and triggering the contingency plan, describing what should happen in order to get the subtask fulfilled, managed by the Task Manager.



**Fig. 2.** Tasks, contracts and transactions

### 3.1. Tasks

A *task* is a meaningful unit of work assigned to an agent. Performing the task often involves communication, that is, using some transactions. However, the task specification and updates thereof do concern the agent in question only, whereas changes in the possible transactions can only be made by consent of the other agent. For that reason, we make a distinction between task and transactions, where the transactions corresponds to the agreements between the two agents and the task draws on this potential for fulfilling an agent's goal.

Tasks can be described in a task language such as TasL [Nodine et al.,1994]. It typically allows the specification of tasks and subtasks, alternatives and temporal constraints. What is crucial in this context, is the way failures are dealt with.

Transactions are prone to many types of failures, including traditional ones like system failures such as system crashes (in the case of interoperable transactions this also includes network failures) and deadlocks caused by concurrent processes. Furthermore a failure occurs when a transaction (as one subtask) is initiated that does not commit (e.g. order product), but also when the transaction does commit first, and the resulting deontic state is violated later (e.g. the company doesn't deliver), or because of cancellation, whereby the other party undoes an achieved effect. These features of transactions directly influence the task specification.

Drawing on previous work in extended transaction models [Elmagarmid,1992], we distinguish the following failure handling methods:

- *fail* - this stops the subtask in a failure state (similar to abort or time out),
- *skip* - this indicates that the subtask is non-vital. The task can be resumed as if the subtask has succeeded,
- *choose* - this prompts for seeking an alternative way to make the subtask run to a successful state (similar to the concept of contingency transaction in DOM [Buchmann et al.,1992]).

A task can be seen as an AND/XOR graph of possible subtasks. The following relations can be defined:

- T1 BEFORE T2 (temporal precedence)
- Ts = AND(T1,T2, ..) (task/subtask relation)
- Ts = XOR (T1, T2, ..) (alternative set)
- Ts = XOR (T1, ..., skip) (non-vital part)

We can use the temporal logic semantics [Ngu et al,1994] for the precedence relations, while the AND/XOR formula is specified as the Goal of the task. The constraint T1 BEFORE T2 is a paraphrase for the TL expression:  $\sim T2 \text{ UNTIL } T1$ .

The execution model is as follows:

- when a task is called, the Task Manager collects the subtasks, puts them into a right order according to the precedence relationships (such an ordering can be stored of course to save computing time). The subtasks are then called one by one. This step continues recursively.

- when a subtask cannot be fulfilled, the next alternative is chosen. If no alternative is available, it means that the subtask fails. It does not mean directly that the parent is aborted, since it is possible that the subtask can still be fulfilled by choosing an alternative in the preceding subtasks. This is tried first (backtracking). Only when no alternative is left, the parent fails.

- the parent also fails when an exit state is reached.

- when an exception occurs , the procedure is basically the same. A next alternative is taken, or, if no alternative is left, backtracking is attempted. To keep the task specification simple, we give the designer the opportunity to specify a contingency plan separately (see below). If no alternatives for backtracking are left, the task fails, but its parent may have alternatives or contingency plans of its own. Every time a task fails, the Task Manager goes up one level in the task hierarchy.

Note that our execution model enforces a "structured approach" in the task specification. We do not allow for arbitrary abort or commit dependencies between subtasks over the boundaries of the parent tasks. This modularity is enforced to keep the specification transparent and maintainable.

A notorious problem with contingencies is that when a subtask is invalidated, later (dependent) subtasks may already have committed, and their result might become obsolete. Whether they have to be retried (and compensated) or not depends on what kinds of results they have produced. We therefore give the designer the opportunity to specify a separate contingency plan. The contingency plan consists basically of a set of *results* (such as supplier list, order-set). Results have an internal object structure, are produced by certain subtasks and can become invalidated by other subtasks. When that occurs, a task can be triggered to repair the damage. This task can make use of the fact that all the essential results obtained so far (and not invalidated) are explicit. For reasons of space, we leave the contingency plan specification for a follow-up paper (Weigand & Ngu, 1995).

We will use the order example to illustrate the task specification (without contingency handling). A task specification consists of a task *name*, a set of *subtasks* (unordered), a set of *temporal constraints*, a *goal* (what makes the task succeed) , and an *exit* (what makes the task fail - abort). Constraints and exit are optional.

```

task order
subtasks
  inquiry
  order-product
  ACCEPT(delivery)
  payment
  cancel-delivery
constraints:
  order-product BEFORE order-delivery;
  inquiry BEFORE order-product;          /*etc
Goal = {delivery}
Exit = {cancel-delivery}
end-task

task inquiry = XOR(get-quotation(abc),
                  get-quotation(xyz))
end-task;

```

- The subtasks are either tasks specified elsewhere or transactions specified in the contract.

- The goal of the task *order* is *delivery*. This goal can only be reached by first an *inquiry* and a product order.

- In the case of *inquiry* we have used a short-hand notation in which we put the Goal immediately after the task name since there are no constraints.

- The ACCEPT around `delivery` means that this transaction is not initiated by the customer agent himself. Executing the subtask `ACCEPT(delivery)` implies executing that part of the transaction that the customer is supposed to do, in response to the prompt of the supplier.

- A possible contingency part specifies that if the order is cancelled by the other party, the Task Manager has to find an alternative (as if `order-product` had failed). This will involve finding another supplier. Any contractual matters that need to be resolved (for example, collecting a fine) are not specified here, since they are handled by the Contract Manager (see below). However, dependent subtasks, e.g. reservation for the shipping, if any, are cancelled by the Task Manager in accordance with the contingency plan.

In the example, only two alternative suppliers are taken into account. A better approach would be to first collect a list of possible suppliers. The task specification language should be expressive enough to represent this. At the moment, we are working on a pseudo-Prolog language extension.

Goldkuhl explicitly includes in his model the problems and objectives of the customer and the capacity of the supplier. By embedding the transactions in an agent task, the link to the objectives are now explicit, and relationships with other tasks of the customer company can be described. As far as the supplier is concerned, Goldkuhl makes no reference to objectives, only to capacity. The symmetry between the parties the author argues for is not maintained at this point. In our point of view, the supplier has objectives as well, including producing goods and making money. These can be modelled as supplier tasks. On the other hand, the capacity and know-how do play a role, as constraining factors, but so do they at the customer side, in particular, the financial capacity or liquidity. Therefore we argue for a symmetric treatment of supplier and customer *also* as far as objectives and constraints are concerned. The particular task of the supplier can be modelled as follows:

```
task sell
subtasks
  ACCEPT(get-quotation);
  give-quote;
  ACCEPT(order-product)
  delivery;
  ACCEPT(payment)
constraints:
  give-quote BEFORE ACCEPT(order-product);
  ACCEPT(order-product) BEFORE delivery
Goal = {payment}
end-task
```

### 3.2. Transactions

Elementary tasks are either executed by the agent as an internal procedure or involve a transaction with another agent. By transactions we mean all *possible* (authorized) messages and message sequences that can be exchanged between the two communicating agents [Dignum et al.,1995]. Real-world events, such as "product delivered" do not play a role directly, but only via messages such as "assert(product delivered)" by means of which a particular agent communicates a certain fact or

evaluation. Moreover, since the message interface encapsulates the local database actions of the agent, the latter do not play a role either. Provided that there is agreement on the semantics of the messages exchanged, the specification of these messages is the only concern for the designers of an interoperable transaction. To put it sharply, for an agent sending an "order product" request to a supplier, what counts is that the supplier replies by a positive confirmation, and not what it does in its database. This can and must be sufficient to let the transaction succeed, and, if the supplier might turn out to be unreliable, it is the confirmation message that provides ground for further (for example, legal) action. So we may assume that the speech act "assert(product delivered)" implies the material action of delivery, and we could make an axiom of this, but we refrain from stating it in this paper because the material action is not relevant for the communication model.

Each transaction has a set of agents, the messages, the constraints (in propositional temporal logic), and the goal and exit states. A transaction execution leading to a goal state means that the transaction succeeds, whereas an execution leading to an exit state means that the transaction fails. "States" are identified here by message occurrences. Higher-level transactions are aggregations of subtransactions, and can specify temporal constraints on the subtransactions. The temporal constraints (not included in our previous paper) are temporal logic formulae [Ngu et al,1994].

All messages are specified as an illocution function, such as request, and a propositional content (an action or proposition). This is in accordance with the common practice in the Language/Action perspective. The semantics of the illocutions are specified directly in deontic logic [Weigand et al, 1995]. The advantage of using these illocution functions is that otherwise the illocution is hidden in the message name (e.g. req\_quote), so that no generalization is possible over the semantics.

For our example we arrive at the following transactions, among others.

```
transaction quotation
  isa get_authorization(order(Partno, Quantity, Price))
end-transaction
```

```
transaction get_authorization(a:action)
agents:
  c: customer      /* all customers in the network
  s: supplier      /* all suppliers in the network
c can send
  {messages:
    request(authorize(a)) to s
  }
s can send
  {messages:
    authorize(a) to c
    refuse(authorize(a)) to c
  }
constraints
  request(authorize()) BEFORE authorize()
Goal = {authorize(a)};
Exit = {refuse(authorize(a))}
end-transaction;
```

The above states that the transaction quotation is a specialisation of the transaction `get_authorization`. This transaction is defined as a request to get authorization, followed by an authorization or a refusal. Specialization means inheritance of the message set, the constraints, the Goal and the Exit. All these parts can be extended (not overruled) in the specialization. The message `authorize` is predefined and creates an authorization for a certain action. The effect of this sample specification is the same as the `give`-quotation example in section 2, but here we make use of more generic primitives. Note that the constraint here is so self-evident that it is better built-in in the language as an axiom.

```

transaction order-delivery(O: order)
agents:
  s: supplier
  c: customer
s can send
{messages:
  assert(delivered(O)) to c
}
c can send
{messages:
  accept(delivered(O)) to s
}
Goal = {accept(delivered(O))}
end-transaction

```

This example shows a typical transaction, in which the two partners express agreement about a situation by means of two messages. In most cases, the `accept` is essential, since delivery is more than saying the order is delivered, it is also more than just putting the containers in front of the entrance. Only when the customer accepts the goods as such (a speech act), the delivery succeeds.

```

transaction payment(s:supplier,$:amount)
agents:
  c: customer
  b: bank
c can send
{messages:
  request(transfer_money(s,$)) to b
  request(increase_credit) to b
}
b can send
{messages:
  commit(transfer_money(s,$)) to c
  assert(credit(c,$)) to c
  refuse(transfer_money(s,$), "not enough credit") to c
}
constraints
  refuse(transfer_money(s,$), "not enough credit")
  BEFORE request(increase_credit)
  request(transfer_money())
  BEFORE commit(transfer_money)
Goal = {commit(transfer_money(s,$))}
end-transaction

```

Although the transactions often take the form of a request followed by a commit, or an assert followed by an accept, sometimes intermediary messages are allowed, as with `increase_credit` in the bank payment example, and sometimes the response is superfluous because the request/assert was already authorized. In the method DEMO ([Dietz,1994]), communication is modelled exclusively by means of so-called actogeneous and factogeneous conversations, creating an obligation and a fact respectively. Each of these conversations is required to have at least two messages (request/commit, assert/accept). A limited set of additional messages, to be used in the negotiation, such as refuse, and counter offer, are allowed as well. Although this is not essential for our method, DEMO could be integrated into it as far as the transaction modelling is concerned. DEMO complements this model with an Information Model and an Action model. The Information Model, describing the object structure of the message content, can be easily combined with our transactions as well, but the Action Model is, in our method, divided over the Task specification and the Contract.

### 3.3. Contracts

Conceptually, in our framework, contracts specify obligations between different parties about services provided to each other. If a particular service is not being fulfilled, it is possible to reason over this violation and take a remedial action without forcing the whole task to abort. In this way, the process is more reactive to failures.

From a technical point of view, a contract is nothing but a protocol binding different parties to their commitments by explicitly specifying the type of services agreed upon, the obligations and the failure recovery methods. From an organizational point of view, a contract between interoperable systems stemming from different organisations also has the purpose of laying down some agreement. By grounding it in other contracts or business law, it may have legal status.

A contract is based on a set of transactions, where a transaction is a set of messages and constraints between them, or, alternatively, an aggregation of reified transaction types. A contract describes the authorized communication behaviour among providers and receivers of services. If the provider does not adhere to the obligation, it is the job of the Contract Manager to impose the violation policies. It would complicate the task specifications and lower the reusability if this communication is included in the task. As stated before the Task Manager should only be responsible in ensuring that a task is brought to its goal, not how violation of commitments are being dealt with.

The reader may have noted that we sometimes use the term "contract" for the specification of the transactions plus the way failures are dealt with, and sometimes for the latter part only, separated from the transactions. This is to avoid inventing yet another term. In the following, the restricted meaning is intended.

A contract is specified as a set of deontic clauses (called 'services' in [Dignum & Weigand, 1995]). The status of an interaction of the two partners can be described in terms of obligations, authorizations and accomplishments. Logically speaking, this is a big deontic logic formula. This formula can be brought into a conjunctive normal form. Each clause in this normal form is a disjunction of literals. The contract describes the deontic clauses and their dynamics, that is, a clause is created by one or more transactions, and is removed by other transactions. So the state of the interaction can be represented by a *set* of clauses, as in a Petri Net, where a state is identified by a set of

token placements. Due to lack of space, the formal aspects of the contract will be worked out in another paper ([Weigand& Ngu, 1995]). We restrict ourselves to a few intuitive examples, related to order-product:

```

clause S1: Obl(a,deliver)
in
  order product
goal
  delivery          =>    S3
exit
  VIOLATION        =>    S4
  cancel-order product =>    S2
  cancel-delivery   =>    S4
modified by
  change-conditions
end {S1}

clause S2: Obl(t,pay($100));
in
  cancel-order product
goal
  fine-payment
end{S2}

clause S3: Acc(a,order)
in
  delivery
end {S3}

clause S4: Obl(a,pay($1000))
in
  cancel-delivery
end{S4}

clause S5: Aut(t,warrant)
in
  delivery
goal
  date(D1)
end{S5}

```

The clause is identified by a number (unique within the contract) and may have a name. Its content is specified as a deontic formula. The "in" part sums up the transactions that lead to this state, provided they have been closed successfully. The goal and exit transactions have the effect of moving to another deontic clause. The old clause is no longer valid. The difference between goal and exit is that one involves the fulfilment of the obligation whereas the other involves a violation or cancellation. Violation means the deadline has passed without fulfilment of the obligation (in this example leading to the same state as cancel-delivery). Note that in S4 we only record the incoming transactions, and not the fact that violation of S1 also leads to this clause. Finally, the contract also allows the specification of update transactions that do not invalidate the deontic content but only modify certain parameters (e.g. a change of the delivery date).

The contract does not only specify the "success line" of the interaction, but especially the exceptions. In some cases, it is possible to return to the success line. For example, the supplier might have the obligation to offer an alternative product (in change-conditions). If this is acceptable, clause S1 remains valid (this can be handled by the Contract Manager). If the supplier does not offer an alternative product, or when it is not acceptable, it is no longer possible to reach the final state, and the interaction fails. Such a failure has to be handled by the Task Manager (see previous section).

Note that this is done only when the contract has no solutions to offer anymore. During the execution of the interaction, other obligations may be instantiated. For example, the obligation to pay a fine or the authorization to warrant. These obligations can be represented explicitly in our framework. However, the further processing of these "side-effects" should not interfere with the task execution. To achieve this independence, we suppose that execution of the interactions takes place in an agent architecture. When the execution of the interaction leads to an obligation of the customer to pay a fine, this obligation is automatically put on the agent's agenda. Since by definition the agent recurrently checks its agenda and executes its tasks, the fine will also be paid in due time. On the other hand, if it is the supplier that has a liability, it is up to the agent to see to it that the payment is actually made.

#### 4. Contracting

Contracts, in the general sense of agreements between commercial partners, can be defined on two levels. In Goldkuhl's model, a contract is the result of the negotiation, and boils down to a commitment to deliver and a commitment to pay. Let us call this an order contract. In the previous section, we have defined a contract as a set of services that specify authorized messages (in transactions) and their deontic effects. We assume that such a contract has been set up by the two partners only once and then frequently used. We call this a procedural contract.

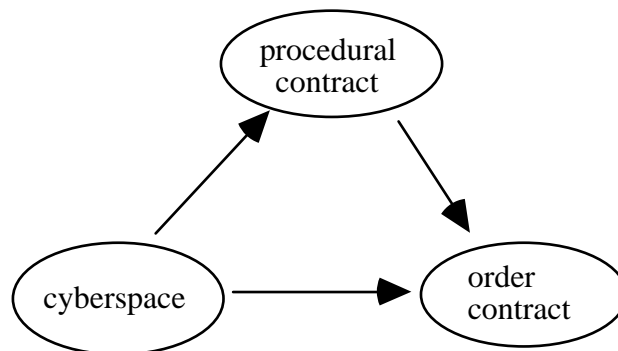


Fig. 3. Contracting in cyberspace

Since "contracts" can be defined on different levels, we should require from any communication modelling approach that it can account for these levels. We want to say a few words about how our approach can do this.

Starting from an empty cyberspace (Fig. 3) in which agents can only communicate, agents must have the possibility to offer services. A service is a publicly accessible interface of the agent, comparable to a method in object-oriented programming. A supplier agent can provide the service of giving an offer. When a customer agent requests for this service, the supplier can refuse or accept. If he accepts, he sends an offer, that is, an authorization for the customer to order a product with the obligation to pay for it. If the customer agent uses this authorization, an order contract in the sense of Goldkuhl has been accomplished. All that is required in this case is a reliable and standardised way of requesting a service, as well as the "umbrella" protection of some international business law that guarantees the legal status of the obligations.

Nowadays it is often considered an economic advantage to have closer business relations with some preferred supplier or customer. In that case, the two parties can make more agreements beforehand, for example about a guaranteed delivery time. Such a contract has to be set up. This can be supported by another service of the agent that offers contracts rather than specific products. If the customer accepts the offer, a mutually agreed contract is accomplished. This can be used then by the customer in the way described in section 3.

The contract can be "symmetric" in the sense of Goldkuhl (section 2), but also establish an asymmetric power relationship, in the same way as an employer can contract a new employee. We then arrive in a situation in which Goldkuhl's critique of Action Workflow - that it is rather one-sided in its customer[initiator] emphasis - is no longer valid. The one-sided Action Workflow loops fit well in an organizational setting with existing power and authorization relationships, whereas the symmetric business process model fits well in the free market context of autonomous negotiating agents. So we feel that both models have their value, depending on the context.

## **5. Comparison with Related Work**

The Interactions model [Nodine et al.,1994] uses "weak constraints" and backtracking when they are violated. Our failure handling uses backtracking as well, but it goes further in two respects. First, we do not abort later (dependent) subtasks immediately, but only when the committed service cannot be maintained (by the Contract Manager). The contract may provide specific alternatives; if these succeed, no abort or backtracking is necessary. Secondly, we have sorted out the complex concept of "compensation" into two more focused notions: compensation of the other party, as specified in the contract, and "compensation", or contingency handling, of the task. For contingency handling, we proposed a separate specification part that monitors the results obtained and invalidated so far. Although exception handling is a complex issue and will remain so, the least we can do is try to manage the complexity. This is the most important but modest goal of the task/contract distinction. We also consider it an advantage that task management can be turned into a local issue, rather than a concern of the (global) multidatabase as in the Interactions model. In this way the global control is kept to a minimum, which makes the specification and implementation much more flexible.

In [Alonso et al, 1996] a comparison is made between the advanced transaction models and workflow models. It is argued that transaction models are too centered around databases, and that (in many respects) workflow models offer a superset of transaction models. This is in accordance with our approach. However, we feel not at ease with the FlowMark workflow model adopted in that paper that (a) still separates activities from data (cf. speech acts) and (b) specifies flow of control in a mechanistic way (cf. contracting, negotiation)

The paper has also aimed at providing a formal way of specifying business process models such as DEMO, Goldkuhl's BPR and Action Workflow. We have shown how these approaches can be put to use. However, it has also become clear that they have different scopes. For example, DEMO offers a rigid methodology, but only on the transaction level. Goldkuhl's BPR is a generic task model that is particularly aimed at free market exchanges, whereas Action Workflow is a generic task model (and perhaps transaction model) that is more oriented towards an organizational context. All models mentioned do not pay much attention yet to failure handling.

Our results are quite compatible with the ideas on electronic contracting developed by Ronald Lee and his colleagues (eg. [Dewitz, 1991]). In this work, attention has been given to the "grounding" of electronic messages in an "umbrella contract" that describes in particular the legal effect of the speech act. Legal rules are formalized in deontic logic and legal procedures are represented by means of Petri Nets. It appears that our use of Petri Nets (contracts) is more restricted: we don't use it for representing temporal constraints, we identify each place node with a deontic clause, and use it in particular for violation handling.

## References

[Alonso et al, 1996] G. Alonso, D. Agrawal, A. El Abbadi, M. Kamath, R. Gunthor, C. Mohan, "Advanced Transaction Models in Workflow Contexts", In: Proc. of the 12th Int Conf on Data Engineering, New Orleans, Louisiana, March 1996.

[Buchmann et al.,1992] A. Buchmann, M. Tamer Özsu, M. Hornick, D. Georgakopoulos, F. Manola, "A Transaction Model for Active Distributed Object Systems", in: Database Transaction Models for advanced applications, A. Elmagarmid (ed), Morgan-Kaufman, 1992.

[Dewitz, 1991] S.D. Dewitz, "Contracting on a performative network: using information technology as a legal intermediary", in: Collaborative Work, Social Communications and Information Systems, R.K.Stamper et al (eds), North-Holland, 1991.

[Dietz,1994] J.L.G. Dietz, "Business Modelling for Business Redesign", in: Proc. of 27th Hawaii Int.l. conf. on System Sciences, IEEE Computer Society Press, 1994.

[Dignum & Weigand,1995] F. Dignum and H. Weigand, "Communication and Deontic Logic", in: Information Systems, Correctness and Reusability, Proc. of ISCORE-94 Workshop, R. Wieringa and R. Feenstra (eds.), World Scientific, Singapore, 1995.

[Dignum & Weigand, 1995] F. Dignum and H. Weigand, "Modelling Communication between Cooperative Systems", in: K.Lyytinen, J.Ivari, M.Rossi (eds), "Advanced Information Systems Engineering" (LNCS-932), Springer-Verlag, Berlin, 1995, pp140-153. (Proc. of CAISE-95).

[Dignum et al.,1996] F. Dignum, H. Weigand, E. Verharen, "Meeting the deadline: on the formal specification of temporal deontic constraints", Proc. of ISMIS'96.

[Elmagarmid,1992] A. Elmagarmid (ed.), Database Transaction Models for Advanced Applications, Morgan-Kaufman, 1992.

[Goldkuhl,1995] G. Goldkuhl, "Information as Action and Communication", in: The Infological Equation, Essays in honour of B. Langefors, B. Dahlbom (ed.), Gothenburg Studies in Information Systems, Gothenburg Univ, 1995. (also: Linkoping Univ report LiTH-IDA-R-95-09)

[Medina-Mora et al.,1992] R. Medina-Mora, T. Winograd, R. Flores, F. Flores, "The Action Workflow Approach to Workflow Management Technology", in: Proc. of 4th Conf. on Computer Supported Cooperative Work (CSCW'94), ACM, 1994.

[Ngu et al,1994] A. Ngu, R. Meersman and H. Weigand, "Specification and verification of communication for interoperable transactions", in: Int.l. Journal of Intelligent en Cooperative Information Systems (IJICS) 3 (1), p. 47-56, 1994.

[Nodine et al.,1994] M.H. Nodine, N. Nakos, S.Zdonik, "Specifying Flexible Tasks in a Multidatabase", in: Proc. CoopIS-94, March 1994.

[Teufel & Teufel,1995] S. Teufel and B. Teufel, "Bridging Information Technology and Business - Some Modelling Aspects", in: ACM SIGOIS Bulletin 16 (1), p. 13-17, 1995.

[Wächter & Reuter,1992] H. Wächter and A. Reuter, "The ConTract Model", in: Database Transaction Models for advanced applications, A. Elmagarmid (ed), Morgan-Kaufman, 1992.

[Weigand,1993] H. Weigand, "Deontic aspects of communication", in: Proc. of Deontic Logic in Computer Science (DEON'92), J.-J.Ch. Meyer, R. Wieringa (eds.), Wiley, 1993.

[Weigand et al.,1995] H. Weigand, E. Verharen, and F. Dignum, "Integrated Semantics for Information and Communication Systems", in: Proc of IFIP DS-6 Database Semantics, Stone-Mountain, Georgia, USA, 1995.

[Weigand& Ngu, 1995] H. Weigand, A. Ngu, "Flexible specification of interoperable transactions". Working paper, Tilburg University, 1995.

[Wieringa et al.,1989] R.J. Wieringa, J.-J.Ch.Meyer, and H. Weigand, "Specifying dynamic and deontic integrity constraints", in: Data & Knowledge Engineering (4) 2, pp.157-191, 1989.