E. Verharen, H. Weigand and F. Dignum. Integrated semantics for information and communication systems. In R. Meersman and L. Mark, editors, *Database Applications Semantics*, pages 500--525. Chapman & Hall, 1997.

Integrated Semantics for Information and Communication Systems

Hans Weigand [*] Egon Verharen [†] Frank Dignum [‡]

February 10, 1995

Abstract

Traditionally, Data Semantics dealt with the static and dynamic integrity rules of database systems. The database system was supposed to integrate different viewpoints on the Universe of Discourse of the organization. Nowadays, databases are often decentralized and its integrating function has been taken over by the network. To account for this evolution, we use the term Information and Communication System instead of Information System, and we argue that Data Semantics should move its focus to the semantics of communication. We present a formal language based on Dynamic Logic in which the semantics of objects, actions, but also speech acts can be described. In addition, we describe a lexicon in which the conceptual meaning of the terms of communication can be defined. Together, these models provide an integrated semantics for Information and Communication systems. These can be interorganizational, as in EDI applications, or intraorganizational, as in Workflow Management.

Keywords: data semantics, application architecture, communication model, lexicon, EDI.

[*] Tilburg University, Infolab, P.O.box 90153, 5000 LE Tilburg, The Netherlands, tel.+31-13-662806, email: h.weigand@kub.nl, fax: +3113663069

[†] Tilburg University, Infolab, P.O.box 90153, 5000 LE Tilburg, The Netherlands, email: egon@kub.nl

[‡] Eindhoven University of Technology, Dept. of Mathematics and Computer Science, P.O.box 513, 5600 MB Eindhoven, The Netherlands, tel.+31-40-473705, email: dignum@win.tue.nl

1 Introduction

The traditional view of databases, also underlying the first IFIP working conferences on Data Semantics ten years ago, considered an information system as one central database and a set of users accessing the database through application programs or directly via an SQL interface. The data were supposed to represent a shared model of the Universe of Discourse (UoD), and data semantics was the way this UoD was captured in the form of conceptual models and integrity constraints.

In the mean time, a number of technological developments has challenged this viewpoint. The most far-reaching development, in our view, has to do with communication. A database is no longer an island. Databases are connected to each other and have to be accessible using electronic networks and EDI. In the technological infrastructure of organizations, the database is no longer in the center (or bottom, if you prefer) and does no longer perform its integrating function. Data semantics is deeply affected by this shift in at least two ways:

- A database is no longer a store of objects (data + operations), but must be able to communicate as well. Thus data semantics should include semantics of the communicative behavior. This is an extension of the "classical" semantics that was concerned with static and dynamic aspects only (e.g. [ISO84]).
- In the traditional view, data semantics concentrated on the semantics of the data in the database; it could hardly be otherwise. In the communication perspective, however, the contents of the database are less interesting than the interfaces between different systems. For the organization, it is of utmost concern that the various systems inside and outside can cooperate, and that the semantics of these data is well-defined. The systems themselves can be of various types (heterogeneous) and the management is often decentralized, thus causing their separate semantics to be less relevant.

What are the consequences of these changes? Not all of them can be overseen yet today, but we mention a few:

- the focal object of data semantics is no longer the fact, or proposition, but the message. Data semantics has become both database semantics and data interface semantics, with an emphasis on the latter.
- since interfaces often have to reconcile different conflicting viewpoints, and have to be established by different, sometimes autonomous, parties, and because they are therefore more difficult to adapt, there is an increasing need for standards. This is evident on the lower levels of communication, for which many standards have been developed already, but also applies to the semantic aspect (OSI's 7th layer).
- as far as database semantics is concerned, there is the growing importance of application architectures. Although such architectures could have been useful in a centralized environment as well, to reduce development and maintenance costs, the need is even more urgent in decentralized systems. An example of an application architecture is IBM's Financial Application Architecture - FAA which contains, among others, a Data Model, a Function Model and a Workflow Model. The Data Model contains generic business data objects (e.g. "customer"). Individual applications are developed on top of/drawing on the resources of the architecture. As far as data semantics is concerned, this involves a shift from the semantics of a particular UoD to "generic semantics".

In this paper, we introduce an integrated semantics for information and communication systems. In section 2, we present a taxonomy of information and communication systems in which the traditional setting appears as a special case. In section 3, data semantics is introduced as consisting of four levels: the illocution, the action, the object and the predicate. In the same section, we briefly indicate how predicates are defined and maintained in a Lexicon Management System. Section 4 presents a formal logic for capturing data semantics on the level of illocution, action and object. The language is an extension of Dynamic Deontic Logic.

2 Information and Communication Systems

Databases are used as a memory device, but in business practice they usually have an equally important function of sharing data, i.e. communication. In the traditional view, there is one company database accessed by many users (e.g. [Date90]). These users have different access rights, so some do provide data while others read them, or provide other data. The fact that the users do not communicate directly (by a workflow application e.g.), but indirectly through the database, asynchronously, does not take away that they do communicate.

It is considered important that the database contains the one and only correct version of the data, hence reducing redundancy and possible conflicts resulting from that redundancy. As a consequence, the responsibility of the database is a company concern, typically delegated to a central person or department. Integrity constraints are one of the tools that seek to prevent erroneous or fraudulent data input.

When two companies connect their databases, as in EDI, there are two databases, but no central one. The interface describes the possible messages, and the authorizations. However, a message never updates the receiver's database directly, since each company wants to retain responsibility for its own data.

The best way to generalize over these and other uses of databases, is by starting from the communication structures in the domain. These communication structures can be modelled by communication models such as DEMO ([DieW92] [VerW94]). Such a model describes the communicating agents, also called subjects, and the communication lines between them. The communication lines are specified on a conceptual level only. They do not prescribe whether the communication is synchronous or not, and which technical intermediaries are used. Each communication line can be viewed also as an authorization, e.g. the authorization to place an order or ask for payment. In section 4, we will distinguish different kinds of validity claims. For example, when an agent is not *authorized* to place an order, he can still try and make an appeal on *charity*. In that case, it is up to the receiver whether he wants to reply or not.

The automation of a communication model assigns a database to each agent (assuming here that the communication is automated totally, which need not be the case of course). Each of these databases falls under the responsibility of the respective agent. The original communication structure can then be replaced by communication between the databases, and because of the one-to-one correspondence, this is a direct mapping. The two different settings described above, centralized database and EDI, take on the following structure:

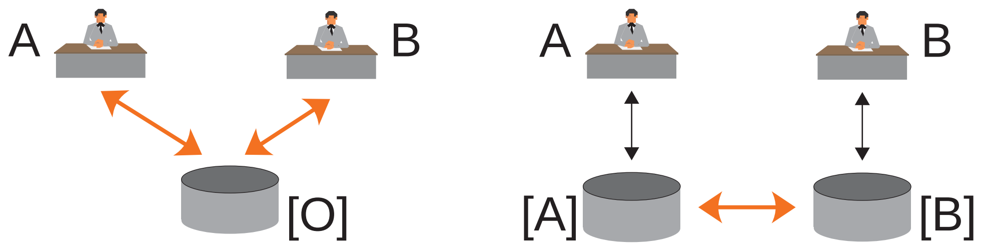

Figure 1: Central Type and Interoperable Type databases

Although we have now introduced system components to the communication structures, the model is still abstract in the sense that a "database" can still be implemented in different ways. It can be in one DBMS, or in one Distributed DBMS, with or without replication etc. These can be varied to achieve the best possible performance. At the conceptual level, databases are considered as distinct from each other when the agents responsible for them differ, because this is what is of primary interest to the organization.

Besides the Central Type and the Interoperable Type, we can distinguish a few more types. The most important to mention are the *Group Type* and the *Mediator (or Client/Server) Type*.

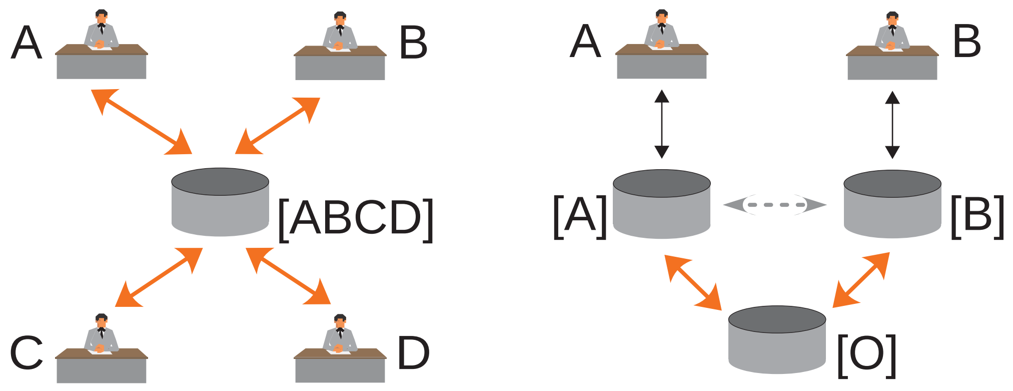

Figure 2: Group Type and Mediator Type databases

The Group Type is special because in this case, the responsibility of the database is not with one agent, but with a group. An example is a co-authoring tool. Note the difference with the Centralized Type, where the data are in one database as well, but the responsibility is with a third agent. The Mediator Type can be viewed as a combination of the Interoperable and Centralized type. The agents do have responsibility for some data, but there is also a common portion that is assigned to a third party. Two variants can be distinguished, depending on whether the clients can communicate directly with each other or not. Note that when there is no server, we come back to the Interoperable type again. Thus, the Interoperable type is applicable to EDI settings, but also to PC networks in one organization.

The four basic types of ICSs can be combined to account for more complex structures. For example, the Mediator Type can be leveled, with one server database for each department, and one global server database for the company.

Both the Group Type and the Central Type have only one database. In these cases, the interfaces between database and human agent should be designed carefully, with proper attention to the authorizations. On the other hand, there are several databases in the Mediator Type and the Interoperable Type. In these cases, the interfaces between the databases are of particular importance. The interfaces with the human agents, the database owners, are less relevant because they are responsible for the database themselves.

In the following, we will use the term "agent" for a human user as well as for a database. The database agent is to be distinguished from the objects stored in the database. The database agent communicates with other agents and executes actions on its objects on the basis of the incoming messages.

3 Data Semantics

In section 2, we have distinguished different types of ICS's, all of which can be viewed as implementations of communication processes. To describe the semantics of these communication processes, we need to specify the meaning of the messages. Once the message types are properly specified, the relevant behaviour of the databases follows more or less automatically (akin to the method specification of objects in OO).

3.1 Decomposition of the message

Definition 1 A message has the following structure. First, it consists of an *illocution* and a *proposition*. A proposition, in turn, consists of an *action* involving one or more *objects*. Illocution type, action type and object type draw on a given set of *predicates*.

We will discuss the different components one by one, starting with the object. Following the tradition in Semantic Data Models and OO modelling, an object is an instantiation of an object type. An object type has a name (a predicate) and one or more attributes. The attributes range over data types or refer to other objects. Object types are ordered in a isa-hierarchy; attributes are inherited. In some OO models, it is allowed to model relations between object types separately. An example of an object specification is given in fig.3. The format roughly follows the traditional OO format.

```
type Order
attributes
          order_no: num[10]
          part: SET_OF Order_Part
methods
          create
          destroy
```

Figure 3: object specification

A proposition consists of an action and one or more objects. The action can be "to deliver" and the object can be a certain "part". Action types can be specified by means of pre- and postconditions and triggers. An example of an action specification is given in fig.4; the format is based on [Wei91] and has been used in a large database project.

```
action pay(m,p,c)
parameters
          m: Company
          p: price
          c: Customer
description
          the Customer pays the price of the Order
preconditions
          account(c)=x
postconditions
          account(c)=x+p
```

Figure 4: action specification

In OO modelling, actions are assigned to objects in the form of methods. We support inheritance of actions, in the sense that any action specified for a certain object class applies also to its subclasses (perhaps in a refined form).

A proposition is embedded in a message. The illocution of the message first of all defines the illocutionary type. The basic illocutionary types are ASSERT and DIRECT ([Aus62] [Sear69] [SeaV85] [LehL86] [DiWe94]), but many more can be distinguished (cf. [BalB62] [ChaW94]). The illocution also defines the Sender and Receiver of the message. An example of a message is: the assertion that a certain part has been delivered, or the request that a certain part be delivered (from agent A to agent B). In these two cases, the proposition is the same, but because of the different illocutions, the meaning of the two messages is quite different as well. Note that an action occurrence referred to in an assertive message means that the action has taken place, or is taking place, depending on the time of action, and that an action occurrence referred to in a directive message means that the action must be executed.

It is a characteristic of messages that they seldom stand on their own. For example, a request is typically followed by an acknowledgement, commitment or refuse message. Therefore messages can be organized in transactions [NguM94], and they can also be modelled as methods of the agents that support them (in the same way as objects support a number of methods). In this paper, we will not go into these Software Engineering issues (see [Wei93] [DiWe94]), since they do not influence the formal data semantics. An example of a message specification is given in fig.5.

The bottom line of the semantic specification is the predicate. Predicates are used to name object types, attribute types, action types, and illocutionary types. In the traditional

```
message     give_quotation
authorization
            AUT(order(c,o,p,m))(m,c)
parameters
            c: Customer
            o: Order
            p: price
            m: Company
description
            the Company gives a quotation for a
            certain price to the Customer
preconditions
            p NOT NULL
postconditions
            -
```

Figure 5: message specification

view of semantics, these predicates are arbitrary strings. However, as argued at length in [Wei90] the use of these predicates draws heavily on the lexical meaning associated with their natural language counterparts. This observation gains in importance in the communication-oriented view of data semantics. When two parties set up a formal communication, they should not only agree on the syntactic format of the messages, and the formal meaning of the actions, but also on the conceptual meaning. By this we mean an (in context) unambiguous definition of the concept, either in natural language or in a formal language. For example, in the context of car registration, a car can be given the definition "a closed road vehicle with at least three wheels driven by a motor engine". Note that the definition may refer to attributes (e.g. wheels, engine) that might play no role in the object type definition Hence, although object type definition and conceptual definition are related, one can not be reduced to the other. Conceptual meanings are organized in the Lexicon.

3.2 The Lexicon

The Lexicon is the system that stores and manages the terminology of a certain domain. It has two major functions. Firstly, it explicitly represents the mutual understanding about a certain concept of all human agents involved. In the case of a data interface, the human agents involved are the owners of the respective databases. As such, the lexical definition should be understandable to humans, and, to allow comparison with related concepts, preferably make use of a standard description method. For example, in the Financial Application Architecture of IBM, nine basic concepts have been predefined, such as "involved party", "arrangement", "location", "product" and "event". "Involved party" can be specialized, in turn, to "Employee", "Competitor", "Customer" etc. The architecture also contains relationship types between concepts, such as "is employed by".

Note that lexical definitions should not be considered as exhaustive characterizations of a concept. A definition is made relative to a certain context. Within the context, the definitions must differentiate different concepts, and provide a *basis* for mutual understanding.

In [Wei90] a linguistic approach is presented in which the lexical definition as well as the lexical structures are based on linguistic primitives. In such a Lexicon, no formal distinction is made between entities, relationships, actions, attributes, identifiers, in so far all of them have a lexical entry. Following linguistic theory, the concepts are organized in a taxonomy and around prototypes. The higher levels of this taxonomy form a basic ontology, including primitive concepts like "entity", "event", "state", "sign". The lexical entry abstracts from the various word forms and inflections of a certain word, as in normal dictionaries. In the case of complex concepts, such as actions, the lexical entry includes a frame containing the roles of the participants, such as "agent", "recipient", etc. The definition of a concept is in principle simply the (natural language) dictionary entry. However, the definition can be parsed and stored internally in a linguistic representation formalism, which allows formal treatment. Moreover, lexical research has shown that definitions typically make use of a small array of basic structures, such as the "isa" and "partof" relation and "purpose". This allows for even more formalization and hence computational processing, while the definition can still be expressed in natural language.

The second major function of a Lexicon is the input it can provide to the software development process. The more formal structure the Lexicon exhibits, the more can be derived from it in terms of entity types, relationships, actions etc. One research project in Tilburg University investigates to what extent NIAM object models can be derived automatically from a lexical specification.

An example of some lexicon definitions is the following. The italicized noun phrase indicates a taxonomic link to a superconcept. The arguments are given between paratheses.

car: a closed *road vehicle* on more than three wheels driven by a motor engine
model(car): the *name* of a registered car design
registration number(car): a *string* of 6 alphanumeric characters uniquely identifying a car
serial number(car): a *string* of 12 digits uniquely identifying a car of a certain manufacturer
year of production(car): the *year* in which the car has been given a serial number

An example in which the semantic structure of a concept is explicit, is the following. It defines *sell* as a transfer action with three semantic roles (the object has an empty label, ag stands for agent and rec for recipient). It describes pre- and postconditions using predicates from the same Lexicon such as *own*. The incondition says that the action includes a payment action of the recipient.

sell(ag X:human)(P:thing)(rec Y:human)
isa transfer
pre = own(ag X)(P), price(P) = D
post own(ag Y)(P)
inc pay(ag Y)(D)(rec X)

In circumscribing the terminology for a particular application domain, the knowledge engineer might draw on available terminologies for the generic domain. Such generic terminologies might draw in turn on more general dictionaries. Hence, it seems useful to organize the Lexicon as a collection of related sublexicons. A *Lexicon Management System* is a system that can handle multiple lexicons. In this way, it is possible to set up application architectures in a consistent way.

To close this subsection on conceptual meaning, we remark that the conceptual definition is primarily the responsibility of the domain experts. The task of the knowledge engineer is mainly methodological. This holds for both specific and generic terminologies. For that reason, the representation should be as close as possible to the conceptualizations of the domain experts.

3.3 Summary

The concepts and models introduced in this section can be summarized as follows. The *Communication Model* is the starting point. It contains agents and messages. The messages in the Communication Model can be decomposed in elementary communicative actions, also called speech acts. Speech acts make reference to actions, described in the *Action Model*, and objects, described in the *Object Model*. All predicates in all models draw on the restricted vocabulary contained in the *Lexicon*.

Inference rules can be attached to the models in various ways. For example, object specifications can include definitions of derived attributes. We can also have inference rules on the state of a world (the world modelled in the database) and inference rules on the state of an agent (taking into account the belief structure of the agent). We will not consider inference rules in this paper.

4 The formal language

The concepts described in the previous section can formally be represented in the logical language described in the following. Objects and their relations can be described in L_{stat}, the propositions describing actions (the contents of the messages) are expressed in L_{dd}. In L_{ill} we can express the messages themselves.

4.1 The static language L_{stat}

L_{Stat} [WWMD91] is a simple first-order language with variables, constants, function symbols and predicate symbols. Two special predicates are the unary predicate E (existence) and the binary predicate $=$ (equality). Terms and formulas are built in the usual way using $\wedge$, $\vee$, $\neg$, $\Rightarrow$, $\forall$, $\exists$, and punctuation symbols (,), [and]. We use infix notation for $=$. The existence predicate E is used by convention to single out the set of existing objects among the set of possible objects. The following abbreviations are used:

$\forall^E x[\phi(x)] == \forall x[E(x) \Rightarrow \phi(x)]$ and
$\exists^E x[\phi(x)] == \exists x[E(x) \wedge \phi(x)]$.

In the following definition, we presuppose the usual model concept from first-order predicate logic.

A function symbol f of arity $n > 1$ is called *transparent* with respect to a structure M of L_{Stat} if for any constants $c_1, \cdots, c_n$, there is a constant c_0 such that $M \models f(c_1, \cdots, c_n) = c_0$. If f is transparent then if the arguments of a particular application are known (in the sense of having a name), then the result of the application is known. In any expression, function applications to constants can thus be eliminated. In the following, we will only consider languages with transparent function symbols.

Definition 2 (First-order semantics)
For any language L,

1. the *Herbrand universe* U_L of L is the set of constants of L. (Since we consider only languages with transparent function symbols, it is sufficient to consider a Herbrand universe without function symbols.)
2. The *Herbrand base* $\mathcal{B}_L$ over the universe U_L of L is the set of all ground atoms (closed atomic formulas) of L containing no function symbols.
3. A *Herbrand structure* $\mathcal{M}_L$ is a subset $\mathcal{M}_L \subseteq \mathcal{B}_L$. Truth in $\mathcal{M}_L$ is defined for ground atoms as

   $\mathcal{M}_L \models P(c_1, \cdots, c_n)$ *iff* $P(c_1, \cdots, c_n) \in \mathcal{M}_L$ and $E(c_i) \in \mathcal{M}_L$, $i = 1, \cdots, n$.

   For an arbitrary closed ϕ, truth in $\mathcal{M}_L$ is defined in the usual way.
4. $\mathcal{M}_L$ is called a *transparent* Herbrand structure of L if every function symbol of L is transparent with respect to $\mathcal{M}_L$.

Note that the use of transparent Herbrand structures is a technical convenience that could be eliminated. Without transparent Herbrand structures, the construction would be more difficult due to the presence of the equality symbol.

To prepare for the extension to Dynamic Logic in the next section, we present the semantics of L_{Stat} in a modal framework.

Definition 3 (Possible Worlds Semantics)
An *S5 Herbrand-Kripke structure* $\mathcal{PW}_L$ of a language L is a collection of Herbrand structures which are called the *worlds* or *states* of $\mathcal{PW}_L$. Truth of a ground atom in $\mathcal{PW}_L$ is defined as

$\mathcal{PW}_L \models \phi$ *iff* $w \models \phi$ for all $w \in \mathcal{PW}_L$.

Truth of a closed formula ϕ in a single transparent Herbrand structure $w \in \mathcal{PW}_L$ is defined in the usual way. The collection of all Herbrand-Kripke structures of L is called $\mathcal{L}$.

If knowledge is expressed as closed statements about objects of a certain type, then we must be able to talk about types. We follow Sowa ([Sowa84]; see also [Gua92]) in using an explicit *type* predicate to declare the type of a term.

Definition 4 (Typed Logic)
Let T be a finite set of constants not occurring in L_{Stat}. The elements of T are called *type names* and τ is used as metavariable over T. L_{Stat} is extended to the typed language TL_{Stat} as follows.

1. TL_{Stat} contains a special binary predicate *type* and the set T of type names. The only well-formed atomic formulas that can be built with *type* are of the form $type(t,\ \tau)$ for a term t and a type name τ, and the only place where τ can occur is as the second argument of *type*. $type(x,\ \tau)$ is called a *declaration* of x.
2. We introduce the abbreviations
   $\forall x:\tau\ (\phi(x))\ ==\ \forall x(type(x,\ \tau) \Rightarrow\ \phi(x))$ and
   $\exists x:\tau\ (\phi(x))\ ==\ \exists x(type(x,\ \tau) \wedge \phi(x))$.
3. The language TL_{Stat} is the set of all *closed* statements that can be built this way and which have all their variables typed. The inference relation $\vdash$ is defined as usual for first-order logic. We only consider formulas in prenex normal form, i.e. $Q_1x_1 \cdots Q_nx_n(\phi(x_1,\ \cdots,\ x_n))$, where $x_1,\ \cdots,\ x_n$ are all the free variables in ϕ and Q_i are quantifiers. Because all variables are typed, we can write this as
   $Q_1x_1:\ \tau_1 \cdots Q_nx_n:\ \tau_n\ (\phi'(x_1,\ \cdots,\ x_n))$,
   with $\tau_i \in T$.

So far, we have defined a syntax of a first-order language containing some special predicates like *type* and E, and a distinguished set T of constants. We must now give a semantics to the type names. We have a choice of keeping the extension of a type name constant in each world, or varying it. This choice has an intuitive meaning, for compare the types *Person* and *Employee*. Some objects can become employees or cease to be employees without coming into existence or ceasing to be. There is life before being hired by a company, as well as after terminating a contract. On the other hand, there is no kind of object that can become a person without coming into existence, or that can cease to be a person without ceasing to exist. Apparently, being a person is an essential property of objects in the way that being an employee isn't. We call types like *Person natural kinds* and types like *Employee roles.* With this, we have sufficient motivation for the following definition.

Definition 5 (Type and Role Semantics)

1. We assume that T is partitioned into the sets $\mathcal{K}$ and $\mathcal{R}$. The constants in $\mathcal{K}$ will be called *natural kind names* and those in $\mathcal{R}$ *natural role names.* Metavariable over $\mathcal{K}$ is k and over $\mathcal{R}$ is r. τ is still our general metavariable over $\mathcal{K} \cup \mathcal{R}$.
2. A *typed structure* $\mathcal{PW}_T L_{Stat}$ of TL_{Stat} consists of a structure $\mathcal{PW}_{L_{Stat}}$ of the untyped version L_{Stat} of TL_{Stat}, and assignments
   $\parallel . \parallel_w$: $\mathcal{K} \to \mathcal{P}(U_{L_{Stat}})$ for each world $w \in \mathcal{PW}_{L_{Stat}}$, and $\parallel . \parallel_w$: $\mathcal{R} \to \mathcal{P}(U_{L_{Stat}} \cap \parallel E \parallel_w)$ for each world $w \in \mathcal{PW}_{L_{Stat}}$,
   where $U_{L_{Stat}}$ is the universe of $\mathcal{PW}_{L_{Stat}}$, and we must have:
   - for each $r \in \mathcal{R}$ there are worlds $w,\ w' \in \mathcal{PW}_{TL_{Stat}}$ with $w\ \neq\ w'$ and $\parallel r \parallel_w \neq \parallel r \parallel_{w'}$,
   - for each $k \in \mathcal{K}$ and all $w,\ w' \in \mathcal{PW}_{TL_{Stat}}$ with $w\ \neq\ w'$, we have $\parallel k \parallel_w = \parallel k \parallel_{w'}$, and
   - $U_{L_{Stat}}\ =\ \cup_{\tau \in \mathcal{K}} \parallel \tau \parallel$.

   The elements of $\parallel \tau \parallel$ are called the possible *instances* of τ.
3. Truth for formulas $type(c,\ \tau)$ in $w \in \mathcal{PW}_{TL_{Stat}}$ is defined by

$w \models type(c,\ \tau)\, iff\, c \in \parallel \tau \parallel_w$.

Truth in $\mathcal{PW}_{TL_{Stat}}$ is defined as usual.

We drop the index TL_{Stat} from $\mathcal{PW}_{TL_{Stat}}$ when the language is clear or can be presupposed to be clear.

4.2 The dynamic deontic language L_{dd}

We start by defining a language of actions L_{act} (cf. [WMW89]).

Definition 6 (Actions)
The language L_{act} of actions is given by the following BNF:

$$\alpha ::- \ \underline{a}|\alpha_1 \cup \alpha_2|\alpha_1 \& \alpha_2|\overline{\alpha}|\mathbf{any}|\mathbf{fail}$$

$\underline{a}$ stands for an atomic action. The meaning of $\alpha_1 \cup \alpha_2$ is a choice between α_1 and α_2. $\alpha_1 \& \alpha_2$ stands for the parallel execution of α_1 and α_2. The expression $\overline{\alpha}$ stands for the non-performance of the action α. The **any** action is a universal or "don't care which" action. Finally the **fail** action is the action that always fails (deadlock). This action does not lead to a next state.

The language L_{act} can be used to describe actions within dynamic deontic logic. The language of dynamic deontic logic (L_{dd}) is given in the following definition.

Definition 7 (Dynamic Deontic Logic)
The language L_{dd} of dynamic deontic logic is given by the following BNF:

$$\Phi ::- \ \phi|\Phi \vee \Psi|\Phi \wedge \Psi|\neg\Phi|[\alpha]\Phi|B(\Phi)|I(\alpha)$$

Where ϕ is a first order logic formula from L_{Stat} and α an element of L_{act}. The language L_{Stat} is supposed to contain a special predicate $Violation$.

The intuitive meaning of $[\alpha]\Phi$ is that after the execution of α Φ necessarily holds. The semantics of actions is that they are functions on $\mathcal{PW}_L$. A formal semantics is given in [Mey88]. B stands for "belief" and I for "intend"; these modalties are used below for the representation of the sincerity conditions of speech acts.

The following *objective modalities* are introduced by definition:

Definition 8 (Objective modalities)

- $\mathbf{POS}(\alpha)$ == $\neg[\alpha]\mathbf{false}$ ("α can possibly happen"),
- $\mathbf{NEC}(\alpha)$ == $[\bar{\alpha}]\mathbf{false}$ ("α necessarily happens"),
- $\mathbf{DIS}(\alpha)$ == $\neg\mathbf{NEC}(\alpha)$ ("α is discretionary, may not happen"), and
- $\mathbf{IMP}(\alpha)$ == $\neg\mathbf{POS}(\alpha)$ ("α can impossibly happen").

These are called *objective* modalities to distinguish them from the deontic ones which are introduced below. Both are dynamic in that they apply to actions, but where objective modalities concern what objectively *can* happen, deontic ones concern what is *admissible.* We have

$\mathbf{NEC}(\alpha) \Leftrightarrow \neg\mathbf{POS}(\bar{\alpha})$,
$\mathbf{DIS}(\alpha) \Leftrightarrow \mathbf{POS}(\bar{\alpha})$, and
$\mathbf{IMP}(\alpha) \Leftrightarrow [\alpha]\mathbf{false}$.

A simple example is

$$\forall p, b\ (Person(p) \wedge Book(b) \Rightarrow \mathbf{POS}(borrow(p,\ b))),$$

which can be paraphrased as "It is possible for a person to borrow a book" (of course, other entities, like institutions, may also be able to borrow books).

The deontic operators are defined by the following abbreviations (cf. [Mey88]; [WMW89]):

Definition 9 (Deontic Modalities)

$$\begin{array}{ll} O(\alpha) = [\overline{\alpha}]Violation & \text{"}\ \alpha\ is\ obligatory\text{"} \\ F(\alpha) = [\alpha]Violation & \text{"}\ \alpha\ is\ forbidden\text{"} \\ \quad = O(\overline{\alpha}) & \\ P(\alpha) = \neg[\alpha]Violation & \text{"}\ \alpha\ is\ permitted\text{"} \\ \quad = \neg F(\alpha) & \end{array}$$

So, the action α is obliged if not doing α leads to a violation state, α is forbidden if doing α leads to a violation state and α is permitted if it is not forbidden to do α. In practice, we usually distinguish between different violations, depending on the action performed. This means that the $Violation$ predicate is indexed with the action and/or the actor. When deontic constraints are specified, this means the constraint *can* be violated. In such a case, we usually also want to express what the consequences are in terms of sanctions or remedial actions. These can be specified in our logic taking the $Violation$ predicate as precondition.

We now give some examples of typical constraints using the static and dynamic language.

Static constraints
These are of the form

$$\boxed{\forall x : \tau\ (\phi(x) \Rightarrow \psi(x)) \quad \phi \text{ is sufficient for } \psi \text{ and } \psi \text{ is necessary for } \phi.}$$

for $\phi(x),\ \psi(x) \in L_{Stat}$. An example is the following:

$$\forall x : Person\ (type(x,\ Manager) \Rightarrow type(x,\ Employee)).$$

This says every manager is an employee, in other words, that Manager is a subclass of Employee. Both are role names and subtypes of Person.

Other types of static constraints are, amongst others, selection restrictions (or domain constraints; e.g. the title of a book is of type string), and analytic constraints (e.g. derived attributes, such as: the fuel consumption of a car manufacturer is the product of the average fuel consumption and the number of cars produced).

Dynamic constraints
The first type is the postcondition. We are only interested in necessary postconditions, not in possible postconditions. An important form of this is

$$\boxed{\forall x : \tau\ ([\alpha(x)]\phi(x)) \quad \alpha \text{ leads necessarily to } \phi.}$$

for $\phi(x) \in L_{Stat}$. An interesting subcase is that of *role-changes.* These are of the form

$$\forall x : k\ ([\alpha(x)]type(x,\ r) \wedge E(x)).$$

For example,

$$\forall^E p : Person\ ([register(p)]type(p,\ Student))$$

Note that we require p to exist, so that existence need not be explicitly mentioned as post-condition. The statement is ill-formed if $Student \notin \mathcal{R}$.

The second type of a dynamic constraint is the precondition. We are interested in necessary as well as sufficient preconditions. For $\phi(x),\ \psi(x) \in L_{Stat}$, these have the form

$\forall x:\tau\ (\phi(x) \Rightarrow [\alpha(x)]\psi(x))$	If $\phi(x)$, then $\alpha(x)$ necessarily leads to $\psi(x)$. $\phi(x)$ is a sufficient precondition for $\alpha(x)$ to lead necessarily to $\psi(x)$.
$\forall x:\tau\ (([\alpha(x)]\psi(x)) \Rightarrow \phi(x))$	If $\alpha(x)$ necessarily leads to $\psi(x)$, then $\phi(x)$. $\phi(x)$ is a necessary precondition for $\alpha(x)$ to lead necessarily to $\psi(x)$.

For example, when the queue of book reservations has n elements, then after reserving a book, it has $n+1$ elements:

$$\forall n : integer\ (queue(n) \Rightarrow [reserve\_book]queue(n+1))$$

When after a reservation the queue has $n+1$ elements, then before the reservation it has n elements:

$$\forall n : integer\ ([reserve\_book]queue(n+1)) \Rightarrow queue(n)$$

A negative form of sufficient preconditions gives us the form of frame axioms:

$\forall x:\tau\ (\neg\phi(x) \Rightarrow [\overline{\alpha(x)}]\neg\phi(x)$.

For example

$$\forall p_1, p_2 : Person\ (\neg married(p_1, p_2) \Rightarrow$$
$$[\overline{marry(p_1, p_2)}]\neg married(p_1, p_2)),$$

which says that marrying is the only way to get married. In other words, the state of being married is invariant under any action but marrying.

Preconditions for objective modalities have the form

$\forall x:\tau\ (\mathbf{POS}(\alpha(x)) \Rightarrow \phi(x))$	$\phi(x)$ is a necessary precondition for $\alpha(x)$ to be possible.
$\forall x:\tau\ (\phi(x) \Rightarrow \mathbf{POS}(\alpha(x)))$	$\phi(x)$ is a sufficient precondition for $\alpha(x)$ to be possible.

Similar schemas exist for the other objective modalities.
For example, someone can only sell something, if he owns it:

$$\forall p : Person,\ t : Thing\ (\mathbf{POS}(sell(p,t)) \Rightarrow own(p,t)),$$

If you are a person, you can borrow a book:

$$\forall p : Person,\ b : Book\ (\mathbf{POS}(borrow(p,b)).$$

These are the preconditions that are often used in database specifications. Usually, we specify only necessary preconditions, and it is assumed that these together are also sufficient. The advantage of specifying only necessary preconditions is that the it is easy to add new preconditions, e.g. when actions are refined downward in an inheritance hierarchy. However, it requires that we assume a kind of "closure" of these necessary constraints at "compile time" to get the sufficient preconditions that make the system run (see [WWMD91]).

Preconditions for deontic modalities have the form

$\forall x:\tau\ (\mathbf{P}(\alpha(x)) \Rightarrow \phi(x))$	α is allowed to occur only if ϕ holds.
$\forall x:\tau\ (\phi(x) \Rightarrow \mathbf{P}(\alpha(x)))$	If ϕ holds, α is allowed to occur.

Similar schemas exist for the other deontic modalities.
For example,

$$\forall p_1, p_2, p_3 : Person,\ t : Thing\ (\mathbf{P}(sell(p_1, t, p_2)) \Rightarrow$$

$$\neg\exists^E p_3 : promised\ (p_1, t, p_3) \wedge \neg p_2 \ = \ p_3,$$

which can be paraphrased as "Someone can sell something only if he has not promised it to someone else." Or, "if someone is a manager, he or she is permitted to park"

$$\forall m\ (type(m,\ Manager) \Rightarrow \mathbf{P}(park(m)).$$

Triggers
A trigger means that action α necessitates action β. In L_{dd}:

$$\forall x : \tau\ ([\alpha(x)]\mathbf{Nec}(\beta(x))))$$

Such a non-deontic causal relation is only possible when the actions are impersonal or executed by the same agent. If $\beta(x)$ is to be executed by another agent, a (directive) message is required. Messages are formally described in the next section.

4.3 The illocutionary language L_{ill}

In order to model the communication between agents in a distributed system the language L_{dd} has to be extended in a way so that it contains all the elements of illocutionary semantics as introduced in section 3 (cf. [Wei93] [DiWe94]).

(1) We introduce a special class Ag of agents (cf. [Shoh93]. The agents in the distributed system can be humans (as in Workflow Management) or database applications (as in EDI) that have been delegated certain tasks.

(2) the language L_{act} of actions is extended to include speech acts.

(3) actions are parameterized; the first parameter represents the agent of the action

(4) deontic operators O, F and P are indexed with the Superordinator and Subordinator agents. Thus $O_{ij}(\alpha)$ should be read as: "agent i is obliged to agent j to perform α." The violation corresponding to this obligation can be indexed accordingly.

(1) Two relations are defined on the class of agents. One implementing the power relation and the other one implementing the authorization relation between agents. Power and authorization are different validity claims, as we introduced them in section 2.

The power relation is the most primitive relation of the two. There exists a power relation between the agent i and the agent j with respect to action α, if i has the power to order j to perform the action α. For instance, the boss can order his secretary to type a letter for him. Note that he might not have the power to order his secretary to make coffee for him. We assume that the power relation is persistent, that is, it is not changed or finished by the fulfilment or non-fulfilment of the command. The power relation defines a partial ordering on the class of agents for every action α. This ordering is reflexive (self-power) and transitive but not necessarily total.

Notation: if i has power over j with respect to α we write: $j <_\alpha i$. if i has power over j with respect to the truth of ϕ (see below) we write $j <_\phi i$

The second relation between agents is the authorization relation. This relation can be established for a certain time with mutual agreement (under certain restrictions). For instance, I can agree that a company can order me to pay a certain amount of money after they delivered a product. This relation ends after I pay the money. The authorization relation is modelled using a special predicate.

Notation: if i is authorized to do α we write: auth(i ,α).

It seems that the above does not define a relation between two agents, because only one agent appears in the notation. However, the action α will always involve another agent in the context of speech acts, as we will see below.

(2) The language of actions is extended to include the speech acts. A speech act is formalized as an illocutionary point with three parameters: the Speaker, the Addressee, and the content. We distinguish the following basic speech acts:

DIR(i,j,α) – i does a request to j for α
COM(i,j,α) – i commits himself to j to do α
ASS(i,j,p) – i asserts to j proposition p
DECL(i,j,p) – i declares and informs j that p holds from now on

From these basic speech acts we can construct other basic speech acts by using the logical negation of actions. E.g.:

FOR(i,j,α) == DIR(i,j,$\overline{\alpha}$) – i forbids j to do α
PER(i,j,α) == DECL(i,j,$P_{j,i}(\alpha(j))$ – i permits j to do α

For a motivation of this set of basic speech acts, we refer to [DiWe94]. Declaratives can only be used for specific institutionalized speech acts, so the propositional content is usually rather restricted. In practice, a limited number of specific declaratives will be distinguished, such as the "authorization" action that we will introduce later.

Speech acts can be grounded in different ways. Besides power and authorization, we also distinguish charity. A request based on charity does not create an obligation directly, but may urge the Addressee to commit hinmself. Because of the differences between the three validity claims of a speech act, we distinguish three variants, indicated by a subscript c,p or a. So, DIR_a stands for an authorized request, whereas DIR_p stands for an order based on power. Similarly for assertives and declaratives. For commissives, the distinction seems to be not very relevant and we ignore it here.

The language of all acts is now defined in two steps. First we define recursively the set of all speech acts L_{Sact}.

Definition 10 (Speech acts)

- All basic speech acts are elements of L_{Sact}.
- If $\alpha \in L_{Sact}$ then also $IP(i,j,\alpha) \in L_{Sact}$ and $IP(i,j,\overline{\alpha}) \in L_{Sact}$ where $IP \in \{DIR, COM\}$

The following axioms hold for speech acts (for the inference relation on actions used here, see [WMW89]).

Axiom 11 *(Distributivity axioms)*
for IP∈{DIR,COM}:

$$\text{IP}(i,j,\alpha_1)\&\text{IP}(i,j,\alpha_2) = \text{IP}(i,j,\alpha_1\&\alpha_2)$$
$$\text{IP}(i,j,\alpha_1) \cup \text{IP}(i,j,\alpha_2) \to \text{IP}(i,j,\alpha_1 \cup \alpha_2)$$

for IP∈{DECL,ASS}:

$$\text{IP}(i,j,p_1)\&\text{IP}(i,j,p_2) = \text{IP}(i,j,p_1 \wedge p_2)$$
$$\text{IP}(i,j,p_1) \cup \text{IP}(i,j,p_2) \to \text{IP}(i,j,p_1 \vee p_2)$$

The language of actions and speech acts L_{ACT} can now be defined as:

Definition 12

$$L_{ACT} \ = \ L_{act} \cup L_{Sact}$$

The language L_{ill} has the same form as L_{dd} except that the actions α are taken from L_{ACT} instead of L_{act}.

The preparatory conditions (ϕ) and the intended effects (ψ) of a speech act (α) can be modelled through the following schema:

$$\phi \rightarrow [\alpha]\psi$$

which means that if ϕ is true then ψ will hold after α has been performed. The intended effects of the speech acts are described by means of deontic and epistemic operators, while the preparatory conditions refer to either the authorization relation or the power relation. We have the following general preparatory conditions and intended effects for the basic speech acts. Of course, for speech acts mentioning specific actions there might be more conditions and effects.

Axiom 13

$([DIR_p\ (i,j,\ \alpha)]\ O_{ji}(\alpha)) \leftrightarrow j <_\alpha i$
$([DIR_a\ (i,j,\ \alpha)]\ O_{ji}(\alpha)) \leftrightarrow auth(i,DIR(i,j,\alpha))$
$[COM(i,j,\alpha)]\ O_{ij}(\alpha)$
$[DECL_a\ (i,j,\phi)]\ \phi \leftrightarrow auth(i,DECL(i,j,\phi))$
$[DECL_p\ (i,j,\phi)]\ \phi \leftrightarrow j <_\phi i$
$([ASS_a\ (i,j,\phi)]\ B(j,\phi)) \leftrightarrow auth(i,ASS(i,j,\phi))$
$([ASS_p\ (i,j,\phi)]\ B(j,\phi)) \leftrightarrow j <_\phi i$

The last of the above properties expresses the fact that a person can be authorized to assert some facts. If this person asserts such a fact the effect is that the Addressee(s) will believe that fact. Note the difference with the declaration that makes a fact true.

The axioms describe the effects of power and authorization speech acts, but not of those based on charity. This is correct, although we might add some politeness rules that say that a message is always replied. such rules can be built on the sincerity conditions.

The *sincerity conditions* of the speech acts refer to the mental states of the agents. In a formal context we assume that an agent is always sincere and thus we have:

Axiom 14

$[DIR(i,j,\alpha)]\ I(i,\alpha)$ – any DIR speech acts expresses that i intends α to happen
$[DECL(i,j,\phi)]\ I(i,\phi)$ – any DECL speech acts expresses that i intends to bring about ϕ (by the speech act)
$[ASS(i,j,\phi)]\ B(i,\phi)$ – any ASS speech act expresses that i believes ϕ

So the effect of a DIR_c is at least that the agent knows about the subjects intention, and this can trigger him to commit himself, or to send a refusal message.

To illustrate how the speech acts work, we specify how obligations can be laid upon another agent in three different ways:

(1) by means of an authorized speech act:

$[DIR_a\ (i,j,\ \alpha)]O_{ji}(\alpha)$

This works only when the subject is properly authorized.

(2) by means of a DIR_c , followed by a COM from the other party:

$[DIR_c(i,j,\alpha)][\text{COM}(j,i,\alpha)]O_{ji}(\alpha)$

No authorization is needed, the other party commits himself.

(3) by means of a DIR_p and an existing power frame:

$j <_\alpha i \rightarrow [DIR_p\ (\ i,j,\ \alpha)]O_{ji}(\alpha)$

In that case, the obligation arises independent of the commitment of the other party.

4.4 The dynamics of authorization

If the subject is not authorized, it can not issue a DIR_a speech act successfully. In that case, it can try to attain an authorization first. This can be done by means of a DIR_c(i,j,DECL(j,i,auth(i,DIR(..)))), that is, a request for authorization of the other party. If the other party complies to the request and grants the authorization, the subject gets authorized from that time on. This example shows that a dynamic distributed system should not only formalize authorized behavior itself, but also the creation of authorizations, and, for that matter, the deletion. The basic assumption underlying our formalization is that authorizations can only be made and retracted by an act of the other party. In this way, the autonomy of the agents is ensured. Because the establishment of authorizations is an important and frequently occurring speech act we introduce the following notation:

AUT(i,j,α) == $DECL_a$(i,j,auth(j,α))

So, AUT(i,j,α) means that i gives authorization to j to do α. Of course, this speech act is only successful if i is authorized to give this authorization. For that reason, we have to presuppose the following axiom:

Axiom 15

1. *auth(i,AUT(i,j,DIR_a(j,i,α(i))))*
2. *auth(i,AUT(i,j,ASS_a(j,i,p)))*

that says that each agent is authorized to authorize other parties as far as actions and beliefs of the agent himself are concerned. This is irrespective of whether the granting of authorizations is forbidden by for example a higher power. If that would be the case, the authorization would still be successfull, although the agent might be punished for it.

Authorizations may refer to any action: material actions, communicative actions, and also to deontic speech acts. An example of such an "indirect" authorization is the following:

auth(i,DIR_a(i,j,AUT(j,i,α)))

which says that i is authorized to direct j to authorize him action α. So i might be not authorized yet, but he has the possibility to obtain an authorization if he wants. From this example it is clear that quite precise agreements can be made. Such agreements may also concern the retracting of authorizations.

The ability to retract an authorization should be left to the subject of the authorization. If i has granted j an authorization, it is only j who can retract the authorization. For this purpose, we introduce a new declarative RTR:

RTR(i,j,α) == $DECL_a$(i,j,$\neg$auth(i,α))

The preparatory condition of RTR is that the authorization does exist. By axiom, every agent is authorized to retract authorizations given to him. If an agent has first granted an authorization, and then wants to retract it, he must ask the other party to do so. Of course, the agents may have made appointments. For example, the agent who grants the authorization may ensure himself of the authorization to request the retracting. The effect is that he can have the authorization retracted whenever he wants.

auth(j,α) $\wedge$ auth(i,DIR_a (i,j,RTR(j,i,α)))

An important question with respect to authorization is whether an authorization can be passed on. We refer to [DiWe94] for some preliminary remarks on delegation.

5 Conclusion

In this paper, we have described an integrated semantics for modern Information and Communication Systems. We characterized several types of ICSs, all of which are used today in different application settings, such as Business Computing, EDI, Groupware and Workflow Management.

The conceptual framework we introduced contains at least an Object Model, an Action Model, a Communication Model and a Lexicon. The semantics of the models has been described in one formal language, L_{ill} which is an extension of Dynamic Deontic Logic.

The models themselves are not very different from those found in other current approaches. In OO design, one usually has Object Models, Data Flow Diagrams (for the actions) and Functional Models (that overlap with the Communication Model). In [ByS92] a specification method is described for interoperable transactions that uses the language LOTOS (based on process algebra) for the communication between systems and TM (based on typed lambda calculus) for the description of the local database. One difference between our framework and these approaches is that our formal semantics are given in one language L_{ill} instead of two or more. Admittedly, the description of the semantics is not complete yet, but our approach avoids the problems of the integration of different models as experienced sometimes in OO modelling.

However, the main difference between our approach and the above-mentioned, in our view, is that our approach gives more proper weight to the peculiar problems occurring in Information and Communication System design. As explained in the introduction, these problems are not the formal problems in the first place, but (1) problems of conflicting conceptual frameworks, (2) the establishing of authorizations, and (3) the use of standards and application architectures. Therefore, we require that the predicates are well-defined in a semantic Lexicon, which can be used also for the specification of generic models. The Lexicon is not found in the other approaches. Secondly, our Communication Model highlights who is responsible for a certain database, and it also contains primitives for assigning and retracting authorizations dynamically. Although not worked out in this paper (but see [DiWe94]), our deontic language also allow the specification of deviating behaviour (failure handling, sanctions etc). In the other approaches, authorizations are not taken into account, and the communication is modelled as a mechanical process. Being based on speech act theory, our communication language also contains higher-level primitives than other approaches based on the concept of communication as data flow, or communication as synchronization.

In the Appendix, an EDI example is presented that illustrates the specification of objects, actions, messages and agents. The specification language has not been fixed yet. This is one topic for future research. Another research project we have planned concerns the implementation of the model in an "agent-oriented DBMS" to be built as an extra layer around a traditional DBMS. A design methodology that is in accordance with the framework presented in this paper, is the subject of a forthcoming Ph.D. thesis.

References

[Aus62] J.L. Austin *How to do things with words.* Oxford: Clarendon Press, 1962.

[BalB62] T. Balmer, W. Brennenstuhl *Speech Act Classification: A Study in the Lexical Analysis of English Speech Activity Verbs.* Springer-Verlag, Berlin, 1981.

[ByS92] R.A. de By, H.J. Steenhagen Interfacing Heterogeneous Systems through Functionally specified Transactions (extended abstract). In: *Proc. IFIP DS-5 Semantics of Interoperable Database Systems (D. Hsiao et al, eds)* Lorne, Australia, 1982.

[ChaW94] M.K. Chang, C.C. Woo A Speech-Act Based Negotiation Protocol: Design, Implementation, and Test Use *ACM Trans. on Inf. Systems*, Vol.12, No4, 1994, pp360-382.

[Date90] C. Date *An Introduction to Database Systems.* Volume 1, 5th Edition, Addison-Wesley, 1990.

[DieW92] J.L.G. Dietz and G.A.M. Wiederhoven A comparison of the linguistic theories of Searle and Habermas as a basis for communication supporting systems. In: *Linguistic Instruments in Knowledge Engineering (Riet, R.P van de, and R.A. Meersman, eds)* North-Holland, 1992.

[DiWe94] F. Dignum, H. Weigand Communication and Deontic Logic. In: *Proc. ISCORE '94 Workshop (R. Wieringa, R. Feenstra, eds)*, Vrije Universiteit, Amsterdam, 1994, pp.401-415.

[Gua92] N. Guarino Concepts, Attributes, and Arbitrary Relations: Some Linguistic and Ontological Criteria for Structuring Knowledge Bases. In: *Linguistic Instruments in Knowledge Engineering (Riet, R.P van de, and R.A. Meersman, eds)* North-Holland, 1992, pp.195-211.

[ISO84] Concepts and Terminology for the Conceptual Schema and the Information Base. (J.J.van Griethuysen, ed.) ISO/TC97/SC21 N197, 1984.

[Lee88] R. Lee Bureaucracies as Deontic Systems. *Trans. on Office Information Systems* Vol.6, No2, 1988, pp87-108.

[LehL86] E. Lehtinen, K. Lyytinen Action Based Model of Information Systems. *Information Systems*, Vol.12, No3, 1986, pp299-317.

[Mey88] J.-J.Ch. Meyer A different approach to deontic logic: deontic logic viewed as a variant of dynamic logic. *Notre Dame Journal of Formal Logic* 29(1), 1988, pp.109-136.

[NguM94] A. Ngu, R.A. Meersman, H. Weigand Specification and verification of communication for interoperable transactions. *Int. Journal of Intelligent and Cooperative Information Systems* 3(1), 1994, pp.47-65.

[Sear69] J.R. Searle *Speech Acts.* Cambridge University Press. 1969.

[SeaV85] J.R. Searle and D. Vanderveken *Foundations of illocutionary logic.* Cambridge University Press. 1985.

[Shoh93] Y. Shoham Agent-oriented programming. *Artificial Intelligence 60* 1993, pp. 51-92.

[Sowa84] J.F. Sowa *Conceptual Structures: Information Processing in Mind. and Machine* Addison-Wesley, Reading, Mas. 1984.

[VerW94] E. Verharen, H. Weigand, and O. De Troyer Agent-oriented information system design. In: *Proc. ISCORE '94 Workshop (R. Wieringa, R. Feenstra, eds)*, Vrije Universiteit, Amsterdam, 1994, pp.378-392.

[Wei90] H. Weigand *Linguistically Motivated Principles of Knowledge Base Systems.* Foris, Dordrecht, 1990.

[Wei91] H. Weigand An object-oriented approach in a multimedia database project. In: *DS-4 Object-Oriented Databases (R. Meersman, W. Kent, S. Khosla, eds)* North-Holland, 1991, pp.393-413.

[Wei93] H. Weigand Deontic aspects of communication. In: *Deontic Logic in Computer Science (J.-J.Ch. Meyer,R. Wieringa eds)* Wiley Professional Computing, 1993.

[WMW89] R.J. Wieringa, J.-J.Ch. Meyer and H. Weigand Specifying dynamic and deontic integrity constraint. *Data & Knowledge Engineering 4*, 1989, pp.157-189.

[WWMD91] R.J. Wieringa, H. Weigand, J.-J.Ch. Meyer and F. Dignum The Inheritance of Dynamic and Deontic Integrity Constraints. *Annals of Mathematics and Artificial Intelligence*, nr. 3, pp.393-428.

A EDI Example

In this section we give a short example about ordering products.

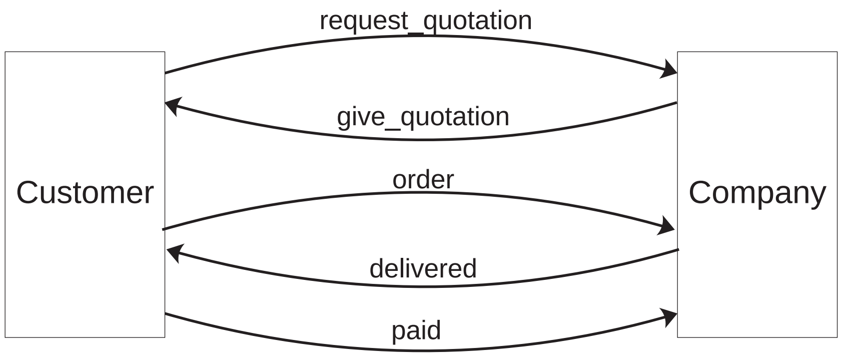

Figure 6: ordering procedure

In business communication, we usually distinguish three phases of communication. The first phase is the negotiation about the terms of a contract. In this phase authorizations can be established on the basis of which some actions can be performed in the following phases. In the above figure this corresponds to the requesting and sending of a quotation (also called the pre-order process in business communication). Here, sending a quotation implies authorizing the customer to order some products on some specified conditions. The second phase concerns the ordering communication itself. In the above example this is included in the order which implies an acceptance of the quotation. The last phase is the fulfilment of the contract. In the example this corresponds to the delivery and the payment sequence.

We can now model all messages, action, objects and agents from our ordering example.

```
agent Customer                    agent Company
is-a Person                       attributes
attributes                                   name
           name                              address
           address                           customers: SET_OF Customer
           account                actions
actions                                      give_quotation
           request_quotation                 deliver
           order                             remind_payment
           pay
```

Figure 7: the agents

The messages supported by agents are part of their specification. We now provide the message definitions one by one.

After a request for a quotation (i.e. a directive) the company is obliged to give the quotation or send a refusal. Although not modelled here, a difference can be made between directions based on authorization or on charity. In other words, is the customer authorized to request quotes or not. In Fig. 8, it is assumed that no authorization exists.

If a company gives a quotation for a certain price (p) the client is authorized to order the product (o) for that price.

The obligation for the company to deliver a product arises either directly from an authorized order message (after a quotation has been made) or from an order transaction (an order without previous quotation, followed by a commitment of the company). The differ-

message	request_quotation
directive	
	DIR(give_quotation(m,o,p,c))(c,m)
parameters	
	c: Customer
	o: Order
	p: price
	m: Company
description	
	the Customer requests a quotation
preconditions	
	-
postconditions	
	-

Figure 8: request quotation

message	give_quotation
authorization	
	AUT(order(c,o,p,m))(m,c)
parameters	
	c: Customer
	o: Order
	p: price
	m: Company
description	
	the Company gives a quotation for a certain price to the Customer
preconditions	
	p NOT NULL
postconditions	
	-

Figure 9: give quotation

message	order
directive	
	DIR(deliver(m,o,p,c))(c,m)
parameters	
	c: Customer
	o: Order
	p: price
	m: Company
description	
	the Customer places an order for the given price
preconditions	
	AUT(order(c,o,p,m))(m,c)
postconditions	
	OBL(deliver(m,o,p,c))(m,c)

Figure 10: order product for given price

ence between having a quotation and not having one is not that the customer would not be able to order without it, but having a quotation, he can order for a specific price. Fig. 10 models the first case.

Another possibility is that customers are authorized to order products anyway (by commercial law, let's say), although not necessarily for a specific price. This situation could be modeled by a variant of fig. 10 in which the price p is NULL in the precondition.

```
message         delivered
assertive
                ASS(delivered(m,o,p,c))(m,c)
parameters
                c: Customer
                o: Order
                p: price
                m: Company
description
                the Company informs the Customer
                of the delivery of the Order
preconditions
                OBL(deliver(m,o,p,c))(m,c)
postconditions
                OBL(pay(m,p,c))(c,m)
                NOT OBL(deliver(m,o,p,c))(m,c)
```

Figure 11: delivery

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. The delivery message is to be distinguished from the material action of delivery itself. It is aimed at consensus between the agents that the material action has taken place. After that, the obligation to deliver has been fulfilled.

After delivery of the order, the customer is obliged to pay for it. We assume here that an invoice is not necessary. The only thing left is that the customer informs the company that the order has been paid. The transaction is closed when the company accepts this statement. When the customer makes use of a bank transfer, this message is typically mediated by the bank who informs the company as soon as the money has been transferred.

(If we had included an invoice message, we would have to change the delivery message. The delivery would not create an obligation to pay then, but an authorization for the company to request payment).

```
message         paid
assertive
                ASS(paid(m,o,p,c))(c,m)
parameters
                c: Customer
                o: Order
                p: price
                m: Company
description
                the Customer informs the Company
                that the Order has been paid
preconditions
                -
postconditions
                NOT OBL(pay(m,p,c))(c,m)
```

Figure 12: payment of the order

As the delivery of the product, the payment itself is a (material) action. The delivery and payment action are modelled in fig. 13 and 14.

```
action deliver(m,o,p,c)
parameters
            m: Company
            o: Order
            p: price
            c: Customer
description
            the Company delivers the Order for
            the given price
preconditions
            in_stock(o.part.product)
postconditions
```

Figure 13: deliver action

```
action pay(m,p,c)
parameters
            m: Company
            p: price
            c: Customer
description
            the Customer pays the price of the Order
preconditions
            account(c)=x
postconditions
            account(c)=x+p
```

Figure 14: payment action

Finally, the object type specifications. The Order has already been given in Fig.3 above. The object Order Part is defined in fig. 15.

```
type Order_Part
attributes
            product: Product
            quantity: num[4]
methods
            create
            destroy
```

Figure 15: order part