Modelling Agent Communication in a First Order Logic

Paul Johannesson
Petia Wohed

Department of Computer and Systems Sciences
Stockholm University
Electrum 230
S -164 40 Kista
Sweden
email: petia,pajo@dsv.su.se
http://www.dsv.su.se/research/syslab/syslab.html

Abstract

In the design of information systems, the notion of agent has proven useful. When modelling communication among agents, deontic concepts, such as obligations, permissions, and prohibitions are essential. The dynamics of obligations, i.e., how obligations are created and destroyed, can effectively be described by means of notions from speech act theory. In this paper, we present a language that includes deontic and illocutionary constructs for the modelling of communication between agents. The language is a logic programming language, which gives it a simple semantics and makes it executable. A distinguishing feature of the language is that it is able to represent time explicitly, which is required to give an adequate semantics for deontic constructs.

1 Introduction

Decentralized and cooperating information systems have become increasingly important as a result of organizational demands and as a consequence of technical advances in computer networking. When specifying and designing these kinds of systems, the concept of intelligent and cooperating agents has proven useful. The term agent is commonly meant to be an entity, human or machine, that functions continuously and autonomously in an environment in which other processes take place and other agents exist. Agents are often taken to be ”high-level”, meaning that they can be described in mental terms such as beliefs, knowledge, capabilities, plans, goals, desires, intentions, obligations, commitments, etc. Ascribing such mental properties to human agents is certainly appropriate, but it can be questioned if it is appropriate to use this mental terminology also for mechanical agents. An answer to this question is that the issue is not if it is correct from a philosophical point of view to ascribe mental states to mechanical agents, but rather if doing so provides a coherent and convenient view of such agents. And it has turned out that in describing complex agents, mental terminology is most useful, see for example [Shoham93]. There exists a vast literature, ranging from philosophy and psychology to AI, on different mental aspects, in particular on belief and knowledge, [Russell95]. In recent years, there has been a growing interest in those aspects that have to do with obligations and commitments, aspects that are essential for cooperation and coordination among agents.

When coordinating their activities, agents create and fulfil obligations, that is the agents request each other to do things, they promise to do things, and they carry out what has been requested or promised. Thus, each agent holds an agenda describing what it is obliged to do in the future. This agenda is not static, but changes as the agent adds and removes obligations. The agent removes obligations by fulfilling them or by violating them, which in the latter case may result in some form of sanction. The agent adds new obligations to its agenda as a response to requests from other agents. This means that agents use speech acts (directives, commissives, and declaratives) to create and cancel their obligations. Speech acts, therefore, provide a natural basis for a structured description of the actions that take place in agent communication and interaction. In this paper, we introduce a logic based language for creating, specifying, and monitoring obligations between agents.

This language is intended to be used when analysing the social context in which non-human agents carry out their activities. In such situations the interaction between agents, human as well as non-human, needs to be analysed in a precise way. In contrast to human-human interaction where traditions and conventions guide the communication process, there cannot exist any informal principles for communication where one of the parties is a non-human agent. For example, if a manager tells an employee to order some item, the precise meaning of this utterance will be determined by contextual factors, such as the relationship between the manager and the employee, previous similar utterances, tone of voice, etc. These factors will determine whether the utterance should be interpreted as an order or not, whether there will be any sanctions if it is not obeyed, what these sanctions will be, etc. If, however, the same utterance is made by a machine, contextual factors become irrelevant and the precise meaning has to be given by a formal specification. Furthermore, the interaction process between several non-human agents is also important to analyse and establish. This kind of communication is often found in inter-organisational contexts, where it is based on some communication protocols such as EDI. A clarification of the communication processes should bring better understanding of these, and it could also increase the possibility for security supervision. We believe that a logic approach is a powerful tool when studying and specifying systems including non-human agents.

It is of course desirable that the interaction between a human and a non-human agent, or two non-human agents shall be similar to the interaction between two human agents whenever possible. With respect to this the examples in this paper are taken from human-human interaction. The paper is organised as follows. Section 2 gives a short overview of related research, primarily in deontic and illocutionary logic. Section 3 introduces the language, called First order Action Logic (FAL), and gives it syntax and semantics. Section 4 models a simple communication process in the language. Finally, Section 5 summarises the paper and suggests directions for further research.

2 Related Research

The social relations of obligations, prohibitions, and permissions have been studied in deontic logic. Surveys of several approaches to deontic logic can be found in [Gabbay84], [Meyer93]. One of the first axiomatic systems for deontic logic was the ”standard system” KD defined by von Wright in 1951 [Wright51], where the deontic operators O (for Obligation) and P (for Permission) were introduced together with a small number of axioms stating their properties and interrelationships. It soon turned out, however, that von Wright’s system entailed several nasty paradoxes, most notably the Chisholm paradox, [Chisholm63]. In attempts to avoid these paradoxes, several other deontic systems were developed during the following decades. In 1964, von Wright extended his original system to a system for dyadic logic [Wright64], where conditional obligations were introduced. In 1958, Anderson suggested in [Anderson58] a reduction of the system KD to a system in alethic modal logic. A novel feature of his system was the introduction of a special propositional atom V standing for violation of obligations. If V obtains in a state, it is regarded as undesirable and that there is need for some sanction or punishment. In spite of these and other efforts, several paradoxes remained in the new deontic systems.

In order to provide a sound foundation for deontic logic, Meyer suggested in 1988 to reduce it to a variant of dynamic logic, [Meyer88]. A consequence of using dynamic logic is a strict separation between actions and assertions, where an action may change the current situation whereas an assertion does not. In Meyer’s system, obligations pertain only to actions, not to assertions. This feature of the system renders some paradoxes inexpressible while others become improvable. The paradoxes that remain in the system seem to be relatively unproblematic, for a discussion see [Meyer93]. Meyer’s approach to deontic logic is, thus, an important step forward, but it has certain shortcomings. First, the fact that only actions, and not assertions, are allowed as the subject matter of obligations can in some situations be too restrictive. For example, an agent may have the obligation to ensure that the temperature in a particular vessel exceeds 25 degrees, and he may obtain this state of affairs in many different ways. Such an obligation would be inexpressible in Meyer’s system as only actions can be obliged. Another problem with Meyer’s approach is that its concept of obligation seems to be too strong. Intuitively, Meyer states that an action is obliged in a state if not performing it results in a violation. This forces a very strong interpretation of the obligation operator, in effect when an action is obliged the next action must be the obligatory one. This interpretation is too strong in many situations, since it is often required only that an obliged action should be carried out at some time in the future (or before some point in time), not that it should be carried out immediately. The cause of this problem is that Meyer’s approach does not provide an explicit representation of time.

Meyer’s approach only allows for the specification of obligations, it does not provide any specific constructs for the dynamic creation of obligations. In 1995, Dignum et.al. showed how such constructs can be provided through a combination of deontic logic and illocutionary logic, [Dignum95]. Illocutionary logic, [Searle85], is a formalisation of the theory of speech acts, [Searle79], which can be used for modelling communication structures. There is a clear distinction in [Dignum95] between speech acts and instrumental acts; speech acts are used for communication between agents, whereas instrumental acts are not. The main contribution of Dignum et al. is that they show how to formalise the dynamics of authorisations, i.e., how authorisations are created and deleted and how they influence obligations among agents. The theoretical foundation of the system of Dignum et al. is Meyer's system, which means that it has the same limitations as that system. The limitation from Meyer's system on time aspects has been addressed in a new "logic for action and norms" introduced by Dignum et al. in 1996 [Dignum96]. This logic, which is an extension of deontic dynamic logic facilitates reasoning about obligation and deadlines. A related approach to deontic reasoning is given in [Cohen90], which introduces a modal logic that combines operators for goals, beliefs, and intentions. The main focus of this paper is on the individual agent's intentionality and not on the communication among agents.

All the approaches discussed above are based on some sort of logic. However, there are also non-logic based languages that include deontic and illocutionary constructs. One of the most notable of these is Agent Oriented Programming, which was introduced by Shoham in 1993, [Shoham93]. The basic units in Agent Oriented Programming are agents with states consisting of beliefs, decisions, capabilities, and obligations. Shoham does not give a logical semantics to his language, instead he provides an operational semantics. The reason for this is that Shoham requires that his language be efficiently executable, and a logical semantics for the language would require a multi-modal, temporal theorem prover, which would become notoriously inefficient.

3 The Language FAL

In this section, we introduce a language for specifying, creating, and monitoring obligations, called First order Action Logic (FAL). Two basic requirements on this language are that it shall be executable and that it can be given a clear semantics. We have attempted to fulfil these requirements by basing the language on first order logic; this approach was inspired by the work by van Benthem et al. in [Benthem95], where first order logic is used to model dynamics. Furthermore, FAL is similar to the language proposed in [Lee88], which provides constructs for deontic modelling in a first order framework; important differences between this language and FAL is that FAL handles the explicit creation of obligations through speech acts and can handle states of affairs as the subject matter of obligations. FAL is also similar to the extended situation calculus as proposed in [Pinto93]. In fact, the syntax of FAL is restricted in such a way that it can be given a logic programming semantics. We claim that this approach provides several advantages:

- Simple semantics - The meanings of the language constructs are easy to understand as they are given using a first order semantics. This makes the semantics of FAL simpler to understand than the semantics of the approaches of the previous section, which make use of different forms of possible world semantics.
- Executable specifications - Specifications in the language are executable, since they can be interpreted as ordinary logic programs. This feature makes the specification language suitable for prototyping purposes.
- Explicit creation of obligations - The language makes it possible to explicitly create obligations by means of speech acts. In this respect, it is similar to the approach proposed by Dignum et al. in [Dignum95].
- States of affairs as the subject matter of obligations - The language makes it possible to have actions as well as states of affairs as the subject matter of obligations. This feature makes the language more expressive than some recent approaches to deontic logic, such as [Meyer88], which only allow actions.
- Representation of time - The language provides explicit representation of time, which makes it possible to express, for example, that a certain action should be carried out during a specified time interval or that a certain state of affairs should obtain at a specified point in time. We claim that an explicit representation of time is essential to obtain an adequate semantics for obligations.

An essential feature of the language proposed is that it is based on a temporal deductive approach to modelling, see for example [Olivé89]. This means that a theory (set of formulas) in the language does not describe a Universe of Discourse (UoD) only at a single point in time; instead, the theory describes the UoD for an extended period of time. It is assumed that a theory describing a UoD can be divided into two disjoint parts. The first part contains information about which actions that have occurred in the UoD. For instance:

- *30 of March Peter gets a quotation from Apple for a Macintosh LC475, valid until the 30 of May.*
- *5 of April Peter orders a Macintosh LC475 from Apple.*

This part corresponds to an information base and we will call it Extension module. The second part consists of rules that describe how information about the UoD can be derived based on knowledge about which actions that have taken place. For instance:

- *If a person gets a quotation from a company and thereafter makes an order according to the quotation, then the company is obliged to deliver.*
- *A person owns a computer at a certain point in time if he has bought it before that point in time and not sold it afterwards.*
- *If someone promises something then he/she is obliged to fulfil his/her promise.*

This set of rules can be divided into two disjoint modules. The first one, called the Domain module, contains domain dependent rules that model a particular domain; the first two rules in the preceding paragraph are examples of domain dependent rules for a domain about orders and purchases. The second module, called the General Intention module, contains domain independent rules describing how obligations, permissions, etc. can be derived from facts about speech acts that have been performed; the third rule in the preceding paragraph is an example of a rule in the General Intention module, which is intended to be valid for all domains. To summarise, a model of a UoD consists of three parts: domain independent rules in the General Intention module; domain dependent rules in the Domain module; and facts about actions that have occurred in the Extension module.

The temporal deductive approach requires that time is explicitly represented, which entails that each formula must include an explicit reference to a point in time telling when a certain fact is true or when an action has been performed. In this way, it becomes possible to maintain information about the UoD at each point in time of its entire life span. This also provides a framework in which it becomes simple to monitor obligations and in which it is possible to distinguish between different forms of obligations and authorisations.

3.1 Syntax

We now define the syntax of FAL. The alphabet, as usual for first order predicate logic, consists of a set C of constants, a set of variables, a set F of function symbols, a set P of predicate symbols, a set of connective symbols $\{\neg, \wedge, \vee, \leftarrow\}$, and a set of punctuation symbols. The UoD is considered to consist of different kinds of objects including agents, time points, actions, and states of affairs. To make it possible to distinguish between terms denoting these different kinds of objects, the terms are typed. We introduce the types Ag for agents, T for time points, A for actions, and SoA for states of affairs. We also introduce the type AS for the contents of speech acts, which may be actions, states of affairs, or combinations of these; A and SoA are subtypes of AS. Furthermore, we introduce the types SA for speech acts and IA for instrumental acts; these are subtypes of A. To make it possible to construct different actions and states of affairs, some special sets of function symbols are defined:

- a set $AC = \{\sim, \&, \cup\} \subset F$ of connectors;
- a set $IP = \{\mathtt{dir_c}, \mathtt{dir_a}, \mathtt{dir_p}, \mathtt{com}, \mathtt{cond\_com}, \mathtt{auth_1}, \mathtt{auth_m}, \mathtt{retract}\} \subset F$ of illocutionary points;
- a set $IAC \subset F$ of instrumental acts constructors;
- a set $SoAC \subset F$ of states of affairs constructors;
- a set $DO = \{\mathtt{O}, \mathtt{Pw}, \mathtt{A_1}, \mathtt{A_m}, \mathtt{P}, \mathtt{F}\} \subset SoAC \subset F$ of deontic operators;

These function symbols are typed as follows:

$\mathtt{dir_c}: Ag \times Ag \times AS \times T \to SA$	$\mathtt{O}: Ag \times AS \times T \times T \to SoA$
$\mathtt{dir_a}: Ag \times Ag \times AS \times T \to SA$	$\mathtt{Pw}: Ag \times Ag \times AS \times T \times T \to SoA$
$\mathtt{dir_p}: Ag \times Ag \times AS \times T \to SA$	$\mathtt{A_1}: Ag \times AS \times T \times T \to SoA$
$\mathtt{com}: Ag \times Ag \times AS \times T \to SA$	$\mathtt{A_m}: Ag \times AS \times T \times T \to SoA$
$\mathtt{cond\_com}: Ag \times Ag \times AS \times T \times AS \times T \to SA$	$\mathtt{P}: Ag \times AS \times T \times T \to SoA$
$\mathtt{auth_1}: Ag \times Ag \times AS \times T \to SA$	$\mathtt{F}: Ag \times AS \times T \times T \to SoA$
$\mathtt{auth_m}: Ag \times Ag \times AS \times T \to SA$	$\sim: AS \to AS$
$\mathtt{retract}: Ag \times Ag \times AS \to SA$	$\&: AS \times AS \to AS$
	$\cup: AS \times AS \to AS$

The predicate symbols in FAL are <, ≤, =, done and holds with arity two and fulfilled with arity three. The predicate symbols <, ≤, and = are used to give a partial order between the time points. The predicate symbols done, holds, and fulfilled are typed as follows:

$\mathtt{holds}: SoA \times T$
$\mathtt{done}: A \times T$

fulfilled: $AS \times T \times T$

A well formed formula in FAL is a clause [Lloyd87], i.e., an ordinary logic programming formula. We use Prolog notation and adopt the convention that constants are denoted with lower-case letters and variables with upper-case letters.

We now informally describe the meanings of the language constructs introduced above; a formal semantics is given in the next subsection.

- $\mathtt{dir}_c$(AgA, AgB, AS, T) - AgA asks AgB to fulfil AS latest at T;
- $\mathtt{dir}_a$(AgA, AgB, AS, T) - AgA requests that AgB shall fulfil AS latest at T;
- $\mathtt{dir}_p$(AgA, AgB, AS, T) - AgA orders AgB to fulfil AS latest at T;
- com(AgA, AgB, AS, T) - AgA commits himself to AgB to fulfil AS latest at T;
- cond_com(AgA, AgB, AS_1, T_1, AS_2, T_2) - AgA commits himself to AgB that AS_2 shall be fulfilled latest at T_2 if AS_1 is fulfilled at T_1;
- $\mathtt{auth}_1$(AgA, AgB, AS, T) - AgA authorizes AgB to request that AgA shall fulfil AS once before T;
- $\mathtt{auth}_m$(AgA, AgB, AS, T) - AgA authorizes AgB to request that AgA shall fulfil AS an indefinite number of times before T;
- retract(AgA, AgB, AS) - AgA withdraws the authorization from AgB to request that AgA shall fulfil AS;
- O(Ag, AS, T_1, T_2) - From T_1 it is obligatory for agent Ag to fulfil AS latest at T_2;
- Pw(AgA, AgB, AS, T_1, T_2) Between T_1 and T_2 AgA has the power to order AgB to fulfil AS;
- A_1(Ag, AS, T_1, T_2) - From T_1 until T_2 agent Ag is authorized to request once that AS shall be fulfilled;
- A_m(Ag, AS, T_1, T_2) - From T_1 until T_2 agent Ag is authorized to request an indefinite number of times that AS shall be fulfilled;
- P(Ag, AS, T_1, T_2) - From T_1 until T_2 agent Ag is permitted to fulfil AS;
- F(Ag, AS, T_1, T_2) - From T_1 until T_2 agent Ag is forbidden to fulfil AS;
- ~AS - The negation of AS;
- AS_1&AS_2 - The conjunction of AS_1 and AS_2;
- $AS_1 \cup AS_2$ - The disjunction of AS_1 and AS_2.

The predicates built by holds, done, and fulfilled shall be read as follows:

- holds(SoA, T) - The state of affairs SoA holds at T;
- done(A, T) - The action A has been performed at T;
- fulfilled(AS, T_1, T_2) - AS is fulfilled with respect to T_1 and T_2, i.e., AS is performed between T_1 and T_2 (if AS corresponds to an action), or AS holds at T_2 (if AS corresponds to a state of affairs).

In the following paragraphs, we explain in more detail the constructs above and give some examples. We follow Dignum et al. [Dignum95], in making a distinction between three kinds of directives: directives made by charity; directives made by authority; and directives made by power. For instance, the following is a charity directive, which in this case means that Apple is not obliged to comply to Peter's request:

Peter asks Apple to give him a quotation for a Macintosh LC475 latest at 200296

$\mathtt{dir}_c$(peter, apple, give_quotation(apple, peter, lc475, 1), 200296)

The next sentence is an authorisation directive, since Apple has authorised Peter to order by giving him a quotation, and Apple must therefore comply to Peter's order:

Peter requests that Apple shall deliver a Macintosh LC475, according to the quotation he received from Apple, i.e., Apple shall deliver an LC475 for 10' latest 300496.

$\mathtt{dir}_a$(peter, apple, deliver(apple, peter, lc475, 1, 10′), 300496)

Finally, an example of a power directive:

When Peter enters Sweden, the emigration control officer (eco) orders Peter to show his passport latest at the day he enters 120196.

$\mathtt{dir}_p$(eco, peter, show(peter, eco, peter′s_passport), 120196)

This is a power directive, since whether Peter authorises the emigration control officer to see his passport or not, the emigration control officer has the right to do that. These three kinds of directives are adopted in FAL, since they create different social relations and are thereby considered separately. They are represented by the indexes `c`, `a` and `p` for charity, authority and power, respectively and are used as subscripts for the symbol `dir`.

Authorisations are created and revoked by means of `auth` and `retract`, which are in fact different kinds of declaratives. We distinguish between two kinds of authorisations: authorisation to an agent to request once, that an action/a state of affairs, shall be fulfilled; and authorisation to an agent to request several times that an action/a state of affairs, shall be fulfilled. For instance, a quotation is an example for one time authorization. The term below represents that *Apple authorizes Peter to request once, to deliver an LC475 for price 10' latest at 300496. The authorization is valid until 300396*.

$$\mathtt{auth_1(apple, peter, deliver(apple, peter, lc475, 1, 10', 300496), 300396)}$$

An example for an indefinite number of authorizations is when *Peter receives a mail order catalogue from Elle valid for the winter 95-96, and he can order clothes several times during the winter 95-96*. This is represented by the term:

$$\mathtt{auth_m(elle, peter, deliver(elle, peter, clothes), 300396)}$$

The function symbol `retract` is used when an authorization is withdrawn. For instance the fact that *Apple withdraws the authorization from Peter to request a delivery of an LC475* is written:

$$\mathtt{retract(apple, peter, deliver(apple, peter, lc475, 1, 10', 300496))}$$

The connectors ~, & and ∪ are used to construct more complex contents of speech acts. The connector & is used for conjunction. For example, to express that *John orders Peter to buy an LC475 and to get information about the price for a PB165 until 300496*, we would write:

$$\mathtt{dir_p(john, peter, buy(peter, lc475)\&get\_information(price, PB165), 300496)}$$

The connector ∪ corresponds to disjunction, and ~ to negation. Note that it is possible to combine actions and states of affairs by these connectors, for example to express: *180296, John promises to Peter that the paper will be ready or John will call Peter latest at 200296*, we write

$$\mathtt{done(com(john, peter, is(paper, ready) \cup call(john, peter), 200296), 180296)}$$

The function symbols from the sets *IAC* and *SoAC* construct terms denoting instrumental acts and state of affairs respectively. Most of the elements of these sets are domain dependent and their specification is a part of a Domain module. Typical examples of instrumental acts in an order and purchase domain are `deliver` and `pay`, while typical elements of *SoAC* could be `owns` and `borrows`.

The terms described above can be divided into two categories: terms describing states of affairs and terms describing actions. It is convenient to talk about states of affairs holding and actions being performed. Therefore the predicate symbols `holds` and `done` are introduced. One of the arguments of both predicates is a time point, which shows when a state of affairs holds or when an action has been performed, respectively. For instance the fact that: *280296 Apple authorizes Peter to request once, to deliver an LC475 for price 10' latest at 300496. The authorization is valid until 300396.* shall be written:

$$\mathtt{done(auth_1(apple, peter, deliver(apple, peter, lc475, 1, 10', 300496), 300396), 280296)}$$

The fact that: *010396 Peter is authorized to request from Apple to deliver an LC475 for price 10' before 300496, if the request is made before 300396*, will be written:

$$\mathtt{holds(A_1(apple, peter, deliver(apple, peter, lc475, 1, 10', 300496), 300396), 010396)}$$

Finally, the predicate symbol `fulfilled` is introduced to handle terms like:

$$\mathtt{is(paper, ready) \cup call(john, peter)},$$

which can occur as the contents of speech acts. These terms can not be arguments of `done` and `holds`, since they are a combination of states of affairs and actions. Since the arguments of the connectors can be both states of affairs and actions the predicate `fulfilled` is used as a generalisation of the predicates `done` and `holds`. `fulfilled` is used to check whether obligations have been violated or not. For example to express that *the paper is ready at 200296 or John has called Peter between 100296 and 200296* we will have:

$\mathtt{fulfilled(is(paper, ready) \cup call(john, peter), 100296, 200296)}$

3.2 Basic Axioms

The well formed formulas of FAL are clauses, as stated in the previous section. This means that FAL inherits its semantics from logic programming. In order to give the meaning of constructs in the language, we will specify a number of axioms in the form of rules, and we first consider axioms for handling obligations. Roughly speaking, these express that someone is obliged to something if he/she has been ordered to do it or has promised to do it, and has not as yet fulfilled it. As we allow actions as well as states of affairs, and combinations of these as the contents of speech acts, we need language constructs to handle these in a uniform way. For this purpose the predicate `fulfilled` is introduced. The meaning of it is defined by the following rules:

GA1. $\mathtt{fulfilled(SoA, T_1, T_2)} \leftarrow$
$\quad\mathtt{holds(SoA, T_2)}$

GA2. $\mathtt{fulfilled(A, T_1, T_2)} \leftarrow$
$\quad\mathtt{done(A, T_3)},$
$\quad\mathtt{T_1 \leq T_3 \leq T_2}$

The general axiom GA1 states that if a state of affairs `SoA` holds at the time point $\mathtt{T_2}$ then `SoA` is fulfilled at $\mathtt{T_2}$. GA2 states that if an action `A` is performed at a time point $\mathtt{T_3}$, between the time points $\mathtt{T_1}$ and $\mathtt{T_2}$ then `A` is fulfilled between $\mathtt{T_1}$ and $\mathtt{T_2}$. Furthermore the axioms GA3 - GA9 are introduced to define the meaning of the connectors $\cup$, & and ~.

GA3. $\mathtt{fulfilled(AS_1 \cup AS_2, T_1, T_2)} \leftarrow$
$\quad\mathtt{fulfilled(AS_1, T_1, T_2)}$
$\mathtt{fulfilled(AS_1 \cup AS_2, T_1, T_2)} \leftarrow$
$\quad\mathtt{fulfilled(AS_2, T_1, T_2)}$

GA4. $\mathtt{fulfilled(AS_1 \& AS_2, T_1, T_2)} \leftarrow$
$\quad\mathtt{fulfilled(AS_1, T_1, T_2)},$
$\quad\mathtt{fulfilled(AS_2, T_1, T_2)}$

GA5. $\mathtt{fulfilled({\sim}AS, T_1, T_2)} \leftarrow$
$\quad\neg\mathtt{fulfilled(AS, T_1, T_2)}$

Axiom GA3 states that $\mathtt{AS_1 \cup AS_2}$ is fulfilled with respect to the time points $\mathtt{T_1}$ and $\mathtt{T_2}$, if either $\mathtt{AS_1}$ or $\mathtt{AS_2}$ is fulfilled with respect to $\mathtt{T_1}$ and $\mathtt{T_2}$. GA4 states that $\mathtt{AS_1 \& AS_2}$ is fulfilled with respect to $\mathtt{T_1}$ and $\mathtt{T_2}$, if both $\mathtt{AS_1}$ and $\mathtt{AS_2}$ are fulfilled with respect to $\mathtt{T_1}$ and $\mathtt{T_2}$. Furthermore GA5 states that ~`AS` is fulfilled with respect to $\mathtt{T_1}$ and $\mathtt{T_2}$, if `AS` is not fulfilled with respect to $\mathtt{T_1}$ and $\mathtt{T_2}$. A consequence of the logic programming semantics of the language is that the negation is interpreted as negation-by-failure and not as negation in classical first order logic. More precisely in this context GA5 states that ~`AS` is fulfilled with respect to $\mathtt{T_1}$ and $\mathtt{T_2}$ if there is no information to the contrary in the Extension module.

Summarising so far, the axioms GA3-GA5 correspond to disjunction introduction, conjunction introduction, and negation introduction, respectively. There is no need to introduce any correspondences to the elimination axioms from FOL as the terms of done and holds are simple in the sense that they are not built by elements of the set AC = {~,&,$\cup$}.

The predicate `fulfilled` makes it possible to concisely define some general axioms describing how obligations, authorisations, permissions and prohibitions are created. Axioms GA6-GA9 show different ways of creating obligations by speech acts.

GA6. $\mathtt{holds(O(AgA, AS, T_1, T_3), T_2)} \leftarrow$
$\quad\mathtt{done(dir_p(AgB, AgA, AS, T_3), T_1)},$
$\quad\mathtt{holds(Pw(AgB, AgA, AS, T_0, T_3), T_1)},$
$\quad\neg\mathtt{fulfilled(AS, T_1, T_2)},$
$\quad\mathtt{T_1 \leq T_2}\ ,\ \ \mathtt{T_0 \leq T_1 \leq T_3}$

GA7. $\mathtt{holds(O(AgA, AS, T_1, T_3), T_2)} \leftarrow$
$\quad\mathtt{done(dir_a(AgB, AgA, AS, T_3), T_1)},$
$\quad\mathtt{holds(A(AgA, AS, T_4, T_5), T_1)},$
$\quad\mathtt{T_1 \leq T_2 \leq T_3}\ ,$
$\quad\mathtt{T_4 \leq T_1 \leq T_5}$

GA8. $\mathtt{holds(O(Ag, AS, T_1, T_3), T_2)} \leftarrow$
$\quad\mathtt{done(com(Ag, \_, AS, T_3), T_1)},$
$\quad\neg\mathtt{fulfilled(AS, T_1, T_2)},$
$\quad\mathtt{T_1 \leq T_2}$

GA9. $\mathtt{holds(O(AgA, AS, T_1, T_4), T_2)} \leftarrow$
$\quad\mathtt{done(cond\_com(AgA, AgB, AS_1, T_3, AS_2, T_4), T_1)},$
$\quad\mathtt{fulfilled(AS_1, T_1, T_3)},$
$\quad\neg\mathtt{fulfilled(AS_2, T_3, T_4)},$
$\quad\mathtt{T_1 \leq T_2 \leq T_3 < T_4}$

Axiom GA6 states that if a power directive to an agent to perform an action is made, then the agent is obliged to perform the action. Axiom GA6 states as well that if a power directive to an agent that a state of affairs shall hold is made, then an obligation for the agent that the state of affairs shall hold exists. Now the purpose of the predicate `fulfilled` becomes clear, since with the use of it an axiom captures two different kinds of obligations: obligations for performing actions and obligations that a state of affairs should hold. These obligations are however similar in the way they are created. This observation is also valid for the rest of the axioms.

GA7 states that if an agent AgB has authority and requests that an agent AgA shall fulfil AS, then AgA is obliged to fulfil AS. The meaning of GA8 is that if an agent commits himself to something then he is obliged to fulfil his commitment. The obligation is valid until the commitment is fulfilled. An obligation is also created (GA9) when a conditional commissive exists and the condition, but not the commitment of this is fulfilled. The axioms GA6-GA9 show only the creation of obligations. Another important point is to reason about obligations. To make this possible the following axioms are introduced.

GA10. $\mathrm{holds}(O(AgA, A\&B, T_1, T_3), T_2) \leftarrow$
$\quad \mathrm{holds}(O(AgA, A, T_1, T_3), T_2),$
$\quad \mathrm{holds}(O(AgA, B, T_1, T_3), T_2),$

GA11. $\mathrm{holds}(O(AgA, A, T_1, T_3), T_2) \leftarrow$
$\quad \mathrm{holds}(O(AgA, A\&B, T_1, T_3), T_2),$
$\quad \neg \mathrm{fulfilled}(A, T_1, T_2)$
$\mathrm{holds}(O(AgA, b, T_1, T_3), T_2) \leftarrow$
$\quad \mathrm{holds}(O(AgA, A\&B, T_1, T_3), T_2),$
$\quad \neg \mathrm{fulfilled}(B, T_1, T_2)$

GA12. $\mathrm{holds}(O(AgA, A \cup B, T_1, T_3), T_2) \leftarrow$
$\quad \mathrm{holds}(O(AgA, A, T_1, T_3), T_2)$
$\mathrm{holds}(O(AgA, A \cup B, T_1, T_3), T_2) \leftarrow$
$\quad \mathrm{holds}(O(AgA, B, T_1, T_3), T_2)$

GA13. $\mathrm{holds}(O(AgA, C, T_1, T_3), T_2) \leftarrow$
$\quad \mathrm{holds}(O(AgA, A \cup B, T_1, T_3), T_2),$
$\quad \mathrm{holds}(O(AgA, \sim A \cup C, T_1, T_3), T_2),$
$\quad \mathrm{holds}(O(AgA, \sim B \cup C, T_1, T_3), T_2)$

GA14. $\mathrm{holds}(O(AgA, B, T_1, T_3), T_2) \leftarrow$
$\quad \mathrm{holds}(O(AgA, A \cup B, T_1, T_3), T_2),$
$\quad \mathrm{holds}(O(AgA, \sim A, T_1, T_3), T_2),$
$\quad \neg \mathrm{fulfilled}(B, T_1, T_2)$

According to GA10 an agent is obliged to fulfil both A and B between T_1 and T_3 if the agent is obliged to fulfil the action or state of affair A between T_1 and T_3 and if he also is obliged to fulfil B between T_1 and T_3. Furthermore by GA11 an agent is obliged to fulfil an action or state of affair A if he is obliged to fulfil both A and some other action or state of affair B, and he has not yet fulfilled A. Note that GA10 and GA11 are nothing more than conjunction introduction and conjunction elimination, respectively. In the same way GA12-GA14 represent disjunction introduction, disjunction elimination, and resolution, by numerical order. However, there is no rule corresponding to negation introduction. Handling negation as negation-by-failure would be inappropriate in this context - if a person is not obliged to an action, this does not entail that he is obliged to the negation of the action. The absence of a rule for negation introduction means that we obtain only a restricted reasoning capability. This may be sufficient for applications where the focus is on specifying, creating, and monitoring obligations, but more advanced reasoning would be needed in many applications. Extending the reasoning capabilities of FAL is an important topic for further work.

We now define with GA15 and GA16 when an authorisation holds.

GA15. $\mathrm{holds}(A_m(Ag, AS, T_1, T_4), T_3) \leftarrow$
$\quad \mathrm{done}(\mathrm{auth}_m(\_, Ag, AS, T_4), T_1),$
$\quad \neg \mathrm{done}(\mathrm{retract}(\_, Ag, AS, T_4), T_2),$
$\quad T_1 \leq T_2 \leq T_3 \leq T_4$

GA16. $\mathrm{holds}(A_1(Ag, AS, T_1, T_4), T_3) \leftarrow$
$\quad \mathrm{done}(\mathrm{auth}_1(\_, Ag, AS, T_4), T_1),$
$\quad \neg \mathrm{done}(\mathrm{retract}(\_, Ag, AS, T_4), T_2),$
$\quad \neg \mathrm{fulfilled}(AS, T_1, T_2),$
$\quad T_1 \leq T_2 \leq T_3 \leq T_4$

So, a many time authorisation is valid after it has been given until it expires, unless it has been withdrawn (GA15). A one time authorization is valid until it has been performed, if it has not been withdrawn before (GA16).

4 An Example

In this section, we give an example illustrating how a Domain module and an Extension module can be constructed in FAL. The example is taken from [Dignum95] and models the communication process starting with a customer requesting a quotation from a company. The communication process is illustrated in the following figure.

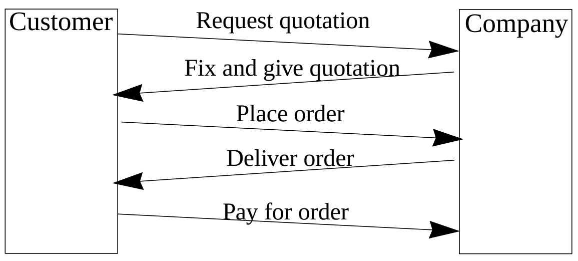

Figure1: Ordering process

After a request for a quotation, a company may fix and give a quotation to the customer. A quotation is a kind of a contract and is binding. This means that if the customer makes an order according to the quotation, the company has to fulfil it by delivering the requested items. After delivery, the customer has to pay the price quoted for the article. We will now construct the Domain module and in the second part of this section give some examples of Extension modules.

The first thing to do when constructing the Domain module for a domain is to analyse the actions which may occur in the domain and define corresponding function symbols. From the figure above, it becomes clear that the actions in this domain are request quotation, fix quotation, give quotation, order, deliver and pay. Some of them like request quotation, fix quotation, give quotation and order correspond to speech acts and some of them like delivery and pay are instrumental acts. The function symbols are thereby
{request_quotation, give_quotation, fix_quotation, order} $\subset SA$ and {deliver, pay} $\subset IA$.
For instance, the action that *170196 Peter requests a quotation for an LC475 before 170296* will in FAL be written as:

```
done(request_quotation(peter, apple, lc475, 170296), 170196)
```

Secondly, speech acts have to be classified in the categories after their illocutionary points. Instrumental acts have, if possible, to be associated to speech acts. For instance, the action that Peter requests a quotation from Apple at 170196 is considered to be a charity directive from Peter to Apple that Apple shall give a quotation to Peter. This is written:

$\texttt{done}(\texttt{dir}_c(\texttt{peter, apple, give\_quotation(apple, peter, lc475), 170296), 170196}) \leftarrow$
$\quad \texttt{done(request\_quotation(peter, apple, lc475, 170296), 170196)}$

We are now generalising our example by replacing the constants like `peter` and `apple` by variables like `AgA` and `AgB`. So, the action that a customer requests a quotation from a company is in effect a charity directive from the customer to the company that the company shall give him a quotation. This is formalised as domain axiom DA1.

DA1. $\texttt{done}(\texttt{dir}_c(\texttt{AgA, AgB, give\_quotation(AgB, AgA, A)}, \texttt{T}_2), \texttt{T}_1) \leftarrow$
$\quad \texttt{done(request\_quotation(AgA, AgB, A}, \texttt{T}_2), \texttt{T}_1)$

To be able to give a quotation for a customer the company has to prepare it. As soon as a quotation has been fixed we consider it as given to the customer.

DA2. $\texttt{done(give\_quotation(AgB, AgA, A), T)} \leftarrow$
$\quad \texttt{done(fix\_quotation(AgB, AgA, A, Am, P}, \texttt{T}_1, \texttt{T}_v), \texttt{T})$

The fix_quotation action is also an authorisation, where a company authorises a customer for a delivery, i.e., if the customer requests a delivery then the company is obliged to comply. This is expressed in the following axiom:

DA3. $\texttt{done}(\texttt{auth}_1(\texttt{AgB, AgA, deliver(AgB, AgA, A, Am, P}, \texttt{T}_1), \texttt{T}_v), \texttt{T}) \leftarrow$
$\quad \texttt{done(fix\_quotation(AgB, AgA, A, Am, P}, \texttt{T}_1, \texttt{T}_v), \texttt{T})$

The action that the customer gives an order to the company after receiving the quotation is an authority directive. The order has, of course, to be made in the time valid for the quotation.

DA4. $\texttt{done}(\texttt{dir}_a(\texttt{AgA, AgB, deliver(AgB, AgA, A, Am, P}, \texttt{T}_1), \texttt{T}_1), \texttt{T}) \leftarrow$
$\quad \texttt{done(order(AgA, AgB, A, Am, P}, \texttt{T}_1), \texttt{T})$

Giving an order is in fact not only a directive, it is also a conditional commissive stating that if the company delivers in time the customer will pay. This example illustrates that a single action may correspond to several speech acts, cf. also the notion of speech moves in [Johannesson95].

DA5. done(cond_com(AgA, AgB, deliver(AgB, AgA, A, Am, P, T_1), T_1, pay(AgA, AgB, A, Am, P), T_1),T) $\leftarrow$
    done(order(AgA, AgB, A, Am, P, T_1), T),

The Domain module is thereby consisting of DA1-DA5. The axioms can, without any transformations, be implemented as rules in Prolog. Note that the Domain module is very concise and easy to understand; it simply specifies which speech acts each action corresponds to.

Finally in this section we will give an example of an Extension module. The example is given in both FAL and natural language. Furthermore we give some examples of how by using the Extension, Domain and General modules more information about the universe of discourse at a particular time point can be derived.

E1. done(request_quotation(peter, apple, lc475, 170296), 170196)
170196 Peter requests, latest at 170296, a quotation from Apple for an LC475.

E2. done(fix_quotation(apple, peter, lc475, 1, 10′, 300396, 200396), 200196)
200196 Apple gives quotation to Peter that a LC475 for 10' can be delivered until 300396. The quotation is valid until 200396.

From E2 and DA3 we can derive that:

D1. done(auth$_1$(apple, peter, deliver(apple, peter, lc475, 1, 10′, 300396), 200396), 200196)
200196 Apple authorizes Peter, latest at 200396 to request them to deliver latest at 300396 an LC475 for 10'.

Now from D1 and GA11 we can derive that:

D2. holds(A$_1$(peter, deliver(apple, peter, lc475, 1, 10′, 300396), 200196, 200396), 210196)
210196 it holds that from 200196 until 200396 Peter is authorized to request once, that Apple shall deliver an LC475 for 10', latest at 300396.

We add the following predicate to the Extension module.

E3. done(order(peter, apple, lc475, 1, 10′, 300396), 200296)
200296 Peter places an order to Apple for an LC475 for 10'.

Then from E3, DA4 follows:

D3. done(dir$_a$(peter, apple, deliver(apple, peter, lc475, 1, 10′, 300396), 300396), 200296)
200296 Peter places an order to Apple for an LC475 for 10'.

D3, D2 and GA8 gives now that

D4. holds(O(apple, deliver(apple, peter, lc475, 1, 10′, 300396), 200297, 300396), 210296)
At 210296 Apple is obligated to deliver an LC475 for 10' to Peter latest between 200296 and 300396.

An Extension module consists, as shown in the example, of predicates like E1-E3. These predicates show the actions that have been performed and when they have been performed. From the predicates in the Extension module and the axioms, it can be derived which states of affairs hold at a certain time point, like D2 and D4.

5 Conclusions

In this paper, we have introduced a language for modelling communication and obligations between agents. The language does not only allow for the specification of obligations as the work provided by Meyer in [Meyer93], but it also supports explicit and dynamic creation of obligations by agents through illocutionary constructs, such as directives, commissives, and authorisations. This work is characterized by focusing on the communication process between agents, and not on the individuals' intentionality as the work presented in [Dignum96]. We have argued that an adequate semantics for obligations requires an explicit representation of time, and we have shown how such a representation can be provided within a first order framework. The use of first order logic makes the paradoxes from some modal logic approaches, as discussed in section two, inexpressible in FAL.

Several open issues are left for further research. One major limitation of our approach is the way violations of obligations are handled, where a violation is signalled solely by a certain fact being true. In addition to this, it would be desirable that a violation gave rise also to compensating actions. Such compensating actions for an obligation should be possible to specify at the same time as the obligation is created through a directive or a commissive. Including mechanisms for compensating violations of obligations would make it possible to specify and monitor more complex contracts between agents. Another limitation of our approach is that it does not support revoking obligations; it therefore needs to be extended with language constructs that enable agents to revoke commissives and directives that they have performed. Revoking an obligation is similar to violating it and can therefore require compensating actions in certain cases. Consequently, it must be possible to specify compensating actions not only for violations of obligations, but also for revoking of obligations. Still another issue, which remains to be worked out, concerns the establishment of power and authorisation relationships between agents. Moreover the general axioms for permissions and prohibitions have to be defined. Different kinds of permissions have to be analysed like single permissions and multiple permissions. For instance a single entrance visa and a multiple entrance visa are both permissions, but they are still very different kinds of permissions. Furthermore, the language FAL is not appropriate for constructing specifications by a user, as the syntax is not very user-friendly. To overcome this problem, a graphical interface of an environment for the language will be constructed. This interface will include two different modes - one for the definition of domain modules based on forms, and one for the graphical presentation of the agenda and history of an agent's obligations.

We envisage that an important application of our approach will be domain abstractions for the communicative aspects of information systems. A *domain abstraction* is a small abstract schema that describes the generic features of a type of situation that may occur in many different contexts. The instantiations of a domain abstraction can, therefore, be parts of the schemas of information systems of different types. An example of a domain abstraction is a provider transferring an acquisition to a recipient, and an instantiation of this could be a library lending a book to a borrower. In the area of software engineering, in particular software reuse, domain abstractions have been utilized in several forms, e.g. clichés [Reubenstein91], reusable patterns [Biggerstaff87], generalised application frames [Constantopoulos92], and analysis patterns [Fowler97]. We believe that an important class of domain abstractions is the one consisting of schemas for communicative structures, and that a language with deontic and illocutionary constructs is adequate for describing such domain abstractions. Our hypothesis is that there is some basic communicative structure, cf. for example Flores' circle, [Flores88], which can be elaborated in various ways to more complex structures, thereby giving rise to a set of interrelated domain abstractions.

Acknowledgements

Thanks to the anonymous reviewers, Prof. Jan Dietz, and Prof. Kalle Lyytinen for their valuable comments on earlier versions of this paper.

References

[Anderson58] A. R. Anderson, "A Reduction of Deontic Logic to Alethic Modal Logic", *Mind*, vol. 67, pp. 100-103, 1958.

[Benthem95] J. v. Benthem and J. Bergstra, "Logic of Transition Systems", *Journal of Logic, Language and Information*, vol. 3, pp. 247-283, 1995.

[Biggerstaff87] T. Biggerstaff and C. Richter, "Reusability Framework, Assessment, and Directions", *IEEE Software*, March, 1987.

[Cohen90] P.R. Cohen and H. Levesque, "Intention is Choice with Commitment", *Artificial Intelligence*, vol.42, pp. 213-261, 1990.

[Constantopoulos92]P. Constantopoulos, M. Jarke, J. Mylopoulos and Y. Vassiliou, "Software Information Base: A Server for Reuse", ICS-Forth, Greece, 1992.

[Chisholm63] R. M. Chisholm, "Contrary-to-duty Imperatives and Deontic Logic", *Analysis*, vol. 24, pp. 33-36, 1963.

[Dignum95] F. Dignum and H. Weigand, "Modelling Communication between Cooperative Systems", in *CAiSE*, 1995.

[Dignum96] F. Dignum, H. Weigand, E Verharen, “Meeting the Deadline: On the Formal Specification of Temporal Deontic Constraints”, in *Fondations of Intelligent Systems, LNAI 1079*, pp. 243-252, Springer-Verlag, Berlin, 1996.

[Flores88] F. Flores, M. Graves, B. Hartfield and T. Winograd, “Computer Systems and the Design of Organizational Interaction”, *ACM TOIS*, vol. 6, no. 2, pp. 153-172, 1988.

[Fowler97] M. Fowler, *Analysis Patterns - Reusable Object Models*, Addison-Wesley, 1997.

[Gabbay84] *Handbook of Philosophical Logic*, Reidel, Dordrecht, 1984.

[Johannesson95] P. Johannesson, “Representation and Communication - A Speech Act Based Approach to Information Systems Design”, *Information Systems*, pp. 291 - 303, 1995.

[Lee88] R. M. Lee, “Bureaucracies as Deontic Systems”, *ACM Transactions on Office Information Systems*, vol. 6, no. 2, pp. 87-108, 1988.

[Lloyd87] J. Lloyd, *Foundations of Logic Programming*, Springer Verlag, 1987.

[Meyer88] J.-J. C. Meyer, “A Different Approach to Deontic Logic: Deontic Logic Viewed as a Variant of Dynamic Logic”, *Notre Dame J. of Formal Logic*, vol. 29, no. 1, pp. 109-136, 1988.

[Meyer93] *Deontic Logic in Computer Science*, Wiley Professional Computing, 1993.

[Olivé89] A. Olivé, “On the Design and Implementation of Information Systems from Deductive Conceptual Models”, in *VLDB89*, Amsterdam, 1989.

[Pinto93] Pinto and Reiter, “Temporal Reasoning in Logic Programming: A Case for the Situation Calculus”, in *Tenth International Conference on Logic Progamming*, Budapest, 1993.

[Reubenstein91] H. Reubenstein and R. Waters, “The Requirements Apprentice: Automated Assistance for Requirements Acquistion”, *IEEE Transactions on Software Engineering*, vol. March, pp. 226-240, 1991.

[Russell95] S. Russell and R. Norvig, *Artificial Intelligence A Modern Approach*, Prentice Hall, 1995.

[Searle79] J. R. Searle, “A Taxonomy of Illocutionary Acts”, in *Expression and Meaning: Studies in the Theory of Speech Acts*, pp. 1-29, Cambridge University Press, Binghamton, New York, 1979.

[Searle85] Searle and Vanderveken, *Foundations of Illocutionary Logic*, Cambridge University Press, 1985.

[Shoham93] Y. Shoham, “Agent-Oriented Programming”, *Artificial Intelligence*, vol. 60, pp. 51-92, 1993.

[Wright51] G. H. v. Wright, “Deontic Logic”, *Mind*, vol. 60, pp. 1-15, 1951.

[Wright64] C. H. v. Wright, A New System of Deontic Logic, *Danish Yearbook of Philosophy 1*, 1964.