Business Process Modelling and Design:

A Formal Model and Methodology[1]

Manolis Koubarakis
Dept. of Informatics
University of Athens
Panepistimioupolis, TYPA Buildings
157 81 Athens, Greece
manolis@di.uoa.gr
http://www.di.uoa.gr/ ~manolis

Dimitris Plexousakis[2]
Dept. of Computer Science
University of Crete
Heraklion, Crete
713 05 Greece
dp@csd.uch.gr
http://www.csee.usf.edu/ ~dimitris

Abstract

We present a formal framework for representing enterprise knowledge. The concepts of our framework (objectives and goals, roles and actors, actions and processes, responsibilities and constraints) allow business analysts to capture knowledge about an enterprise in a way that is both intuitive and mathematically formal. We also outline the basic steps of a methodology that allows business analysts to go from high-level enterprise objectives, to detailed and formal specifications of business processes that can be enacted to realise these objectives. The formal language used means that the specifications can be verified as having certain correctness properties: we can verify that responsibilities assigned to roles are fulfilled, and constraints are maintained as a result of process execution.

1 Introduction

The problem of representing, analysing and managing knowledge about an organisation and its processes has always been very important. Recently, management and computer science researchers have debated the use of information technology for tackling this complex problem

[1]The work of the first author was partially supported by BT Labs, Ipswich, U.K. through a Short Term Research Fellowship. Most of this work was carried out while the author was a lecturer in the Dept. of Computation, UMIST, U.K. The work of the second author was partially supported by the Univ. of South Florida through a Research and Creative Scholarship Grant.

[2] On leave from the Dept. of Computer Science and Engineering, University of South Florida, Tampa, FL.

[HC93, Dav93, WFM, NIS, IDE, Oul94, GHS95, LA94, JFJ+96, YML96, KO97]. Ultimately this research community is interested in improving the understanding of organisations and their processes, facilitating process design and analysis and supporting process management (i.e., process enactment, execution and monitoring). The topic is also of great practical importance to industry as an aid to designing organisational structures, processes and IT infrastructure that achieve business goals in an efficient and flexible way. A specific area of interest is in deriving, checking and improving business process definitions used as input to workflow systems.

An enterprise model is a description of the main constituents, purpose, processes, etc. of an organisation and how they relate to each other. It is essentially a representation (on paper or on a computer) of the organisation's knowledge about itself or what it would like to become. Here 'organisation' can mean anything from a large corporation or government department to a small team or a one-man company. Similarly, the level of detail represented in the model can vary depending on its purpose. In our model the various aspects of an enterprise are represented with distinct (though inter-related) submodels as follows:

- *organisational submodel:* this describes the 'actors' in the (extended) enterprise, their roles and relationships, responsibilities, capabilities, etc. An actor is an intentional entity, that is it has some idea of purpose that guides its actions. Actors will often model people, but could also represent groups of people (e.g. a department) or a software agent. Note that [Ken99] applies similar concepts to the software engineering of multi-agent systems'.
- *objectives and goals submodel*, describing what the enterprise and its actors try to achieve
- *process submodel*, desrcribing how it (intends to) to achieve them
- *concepts submodel,* describing non-intentional entities, and the
- *constraints submodel*, describing factors limiting what the enterprise and its components can do.

A similar approach has also been followed by F^3 [Bub94, LK95b] and EKD [KL98, BBS98] and our work has indeed been influenced by these projects.

Creating an enterprise model can be instructive in itself, revealing anomalies, inconsistencies, inefficiencies and opportunities for improvement. Once it exists (particularly in computerised form) it is a valuable means of sharing knowledge within the enterprise. It can also be used to formulate and evaluate changes, for example introducing a new product and associated business processes. The knowledge sharing role also extends to the enterprise's IT infrastructure. It is in principle possible, for example, to extract process definitions to be input to a workflow management system. Furthermore, it would be possible for business process support software to query the enterprise model, for example to find out who is fulfilling a give role in a given process.

Formal enterprise models, such as ours, are ones in which concepts are defined rigorously and precisely, so that mathematics can be used to analyse, extract knowledge from and reason about them. An advantage of formal models is that they can be verified mathematically, that is it can be proved that they are self-consistent, and have or lack certain properties. For example, using our approach, one can prove formally that responsibilities assigned to roles are fulfilled, and constraints are maintained as a result of process execution.

The majority of our model is built on a knowledge representation formalism used in AI, the *situation calculus* [MH69, Rei91]. This is unlike projects like F^3 [Bub94, LK95b] and EKD

[KL98, BBS98] where enterprise knowledge is organised using less formal representations (mainly entity-relationship models). An important additional feature of our model is how the use of situation calculus is combined (within the process submodel) with use of the concurrent logic programming language *ConGolog* [DGLL97]. ConGolog is used to represent operationally how actors behave when playing roles, e.g.. when an actor in role R is notified of an event of type E it performs action A then notifies its superior. This allows verification that a process designed to achieve some goal really will achieve that goal under a given set of circumstances. Verification of such properties is impossible under similar informal approaches such as F^3 [Bub94, LK95b] and EKD [KL98, BBS98]. In the conclusions of this paper (Section 8) we discuss some problems that might arise when a formal enterprise modelling effort is undertaken, and what can be done to overcome these problems.

The rest of this paper is structured as follows. Sections 2, 3, 4 and 5 present our enterprise model. We then present a methodology that enables business analysts to go from high-level enterprise objectives, to detailed and formal specifications of business processes for realising these objectives. The methodology can be used by an enterprise that wishes to develop a new business process, or alternatively model, document and analyse formally an existing process. The most distinguishing feature of our methodology (compared with similar approaches) is the utilisation of process verification techniques [Ple95]. Finally, Section 7 discusses related work and Section 8 presents our conclusions.

2 The Organisational Submodel

In this section, we initiate the presentation of the five submodels making up our enterprise modelling framework. The first submodel is the organisational submodel with main concepts *actor* and *role*. An *actor* is a person or a software/hardware system in the context of the organisation we are modelling (e.g., an employee, a customer, a printer etc.). Actors are distinguished as being either *human* or *automated* ones. Actors are capable of executing certain activities, but they might not be capable of executing others.

An *organisational role* involves a set of *responsibilities* and *actions* carried out by an actor or a group of actors within an organisation [Oul95, Yu94, DBCS94, KGR96]. Organisational roles can take many forms [Oul95]: a unique functional group (e.g., Systems Department), a unique functional position (e.g., Managing Director), a rank or job title (e.g., Lecturer Grade A), a replicated functional group (e.g., Department), a replicated functional position (e.g., Director), a class of persons (e.g., Customer) or an abstraction (e.g., Progress Chasing).

Role *instances* are acted out by actors.[3] Different actors can play different roles at different moments of time (e.g., today the Managing Director can be John Smith, tomorrow it can be Tony Bates). Many instances of the same role can be active at any moment in time.

2.1 Formalism

[3] Role instances can also be acted out by *groups of actors* (e.g., a committee or a meeting). Currently our framework is rather poor with respect to role playing by groups of actors e.g., we have no explicit notion of joint goals, joint processes etc. [Tidhar93].

As we mentioned in the introduction, the distinguishing feature of our work is the emphasis on formality. Throughout the paper we will represent enterprise knowledge using the formalism of *situation calculus* [MH69, Rei91]. This formalism has been designed especially for knowledge representation and reasoning in dynamically evolving domains thus it is very appropriate for the business domain.

Technically, our basic tool will be a *first-order language* Λ which is defined in the following way. The symbols of Λ include parentheses, a countably infinite set of variables, the quantifiers (existential and universal), the equality symbol = and the standard sentential connectives (negation, conjunction, disjunction, implication and equivalence). The remaining machinery of Λ (constant, function and predicate symbols) will be defined step-by-step whenever a new modelling concept needs to be formalised. By convention, we will use strings of characters starting with a lower-case letter to denote variables; predicate and function symbols will start with an upper-case letter.

The language of first-order logic is particularly useful for representing enterprise knowledge. *Constant symbols* are used for denoting entities in an enterprise e.g., the engineer Mike Smith can be denoted by constant symbol *Mike*. *Predicate symbols* are used for denoting relations e.g., the relation 'works for' can be denoted by predicate *WorksFor*. Some relations are functional i.e., they relate any given object to at most one other object; these relations are denoted by *function symbols*. For example, the functional relation 'president of' can be denoted by function symbol *PresidentOf*. *Variables* and *quantifiers* enable us to make universally or existentially quantified statements e.g., '*all* employees should earn less than their managers' or 'the results of this project will be published *sometime* before the end of this month'. Finally, the *sentential connectives* are used to make complex sentences out of simpler ones. What is very appealing about first-order logic is its ability to represent *incomplete enterprise knowledge* e.g., 'the results of this project will be published *sometime* before the end of this month' or 'this job can be assigned to John *or* Mike'. Incomplete knowledge is impossible to represent in other enterprise knowledge frameworks (e.g., in the ones using entity-relationship models [Bub94, LK95b, KL98, BBS98]) except by extending them so that they can become equivalent to first-order logic.

2.2 Formalising the organisational submodel

The concepts of the organisational submodel introduced above can be defined formally by introducing appropriate constructs of Λ and writing axioms that capture their semantics. Thus, we introduce unary predicates *Actor, HumanActor, AutomatedActor* and *Role*, and binary predicate *PlaysRole*, so that *Actor(John)* means *'John is an actor'; PlaysRole(John, Manager)* means *'John plays the role of a manager'*, and so on. We adopt the convention of using symbols commencing with lower case characters to denote variables. The following axiom specifies the relation between actors, human actors and automated actors, i.e. that the set of all actors is the union of the sets of human and automated actors:

$$(\forall x)(Actor(x) \equiv HumanActor(x) \vee AutomatedActor(x))$$

Example 2.1 Throughout this paper we will demonstrate the features of our proposal by considering an imaginary Computer Science department DEPT as the organisation considered by our study. We assume that this department has so far no postgraduate program, and it is now considering the development of processes for the admission and education of postgraduate students.

Using the predicates introduced above, the following sentences of Λ can be introduced in the organisational submodel for DEPT:

HumanActor(John), HumanActor(Mary)
Role(Tutor), Role(Secretary)
PlaysRole(John,Tutor), PlaysRole(Mary,Secretary)

This completes our discussion of the organisational submodel. We now turn to the objectives and goals submodel.

3 The Objectives and Goals Submodel

An *enterprise goal* is a desired state of affairs [FN71, DvLF93, Yu94, LK95b, Lee94, Oul94, YML96, KL98]. Examples of enterprise goals are the following: "all customer enquiries are answered within one day", "profits are maximised" and so on.

In our framework goals are associated with the following components of other submodels:

- *Roles and actors* (organisational submodel). Goals are assigned to roles as a matter of policy by the organisation. Organisational goals become responsibilities of roles and the actors playing these roles.
- *Processes* (process submodel). The *purpose* of a process is the achievement of one or more goals. For example, the process of managing project X might have the purpose of achieving the goal "Y, the product of project X, is completed to specification by time T".
- *Entities* (concepts submodel). Every goal refers to certain enterprise entities. For example, the goal "two C++ programmers should be hired by the Systems Department" refers to entities "Systems Department" and "C++ programmer".

Explicit capturing of enterprise goals is important because it allows us to study organisations and their processes from an *intentional* point of view [YM94a, Yu94] . For example, this enables us to represent not only "what" information (e.g., what sub-processes form a process) as in standard process representations, but also "why" information (e.g., why a specific activity is done). When goals are combined with other intentional concepts like actors and roles, we are also enabled to represent "who" information (e.g., "who is responsible for bringing about a state of affairs").

3.1 Enterprise Goals

Enterprise goals can be *reduced* into alternative combinations of subgoals [DvLF93, Chu93, Yu94, Lee94, Bub94, LK95b, KL98, BBS98] by using *AND/OR goal graphs* originally introduced in the area of problem solving [FN71] . For example, the goal "our sales targets are achieved" can be AND-reduced to two goals "our sales targets for product A are achieved" and "our sales targets for product B are achieved".

We utilise the notion of goal reduction to define the concept of objective. An *enterprise objective* is a goal that is not derived from a higher-level goal through goal reduction. In other words, an

objective is a top-level goal; it is an *end* desired in itself, not a *means* serving some higher level end [LK95a].

Goals can *conflict* with each other [DvLF93, Chu93, YM94a, LK95b, vLDL98]. In our framework goals $G_1,\ldots,G_n$ conflict if they cannot be *satisfied* simultaneously given our knowledge about the enterprise [vLDL98]. Goals can also *influence* positively or negatively other goals [MCN92, Chu93, YM94a, Yu94, LK95b]. Such interactions between goals must be noted explicitly to facilitate goal-based reasoning (see Section 6).

3.2 Defining Goals Formally

Enterprise goals can be described formally or informally. Organisational objectives and other high-level goals are usually difficult to formalise. These goals should initially be described informally, and reduced step by step to more concrete and formal goals. Appropriate formal concepts and tools for assisting goal reduction (in the context of requirements modelling) are discussed in [DvLF93] .

Because a goal is a desired state of affairs many concrete and formal goals can be formalised as *sentences* of L as demonstrated by the following example.

Example 3.1 The operational goal "enquiries are answered by a member of staff as soon as they are received" can be formalised by the following sentence of Λ:

$$(\forall a)(\forall e)(\forall x)(\forall s)(\forall s')(Staff(a) \wedge Enquiry(e) \wedge Action(x) \wedge Situation(s) \wedge Situation(s') \wedge Received(e,a,s) \wedge (s' = Do(x,s)) \supset Answered(a,e,s'))$$

where predicates have obvious meaning and $s' = Do(x,s)$ means that s' is the situation resulting from the execution of action x in situation s (the concept of situation is equivalent to the concept of state and will be discussed in detail in Section 4.1). The sentence can be read as 'any situation in which a member of staff receives an enquiry gives rise to an action that causes the enquiry to be answered by that member of staff'. More details about the situation calculus and its machinery are given in Section 4. Note also that the use of a formal language forces one to be very precise and dispense with informal concepts such as "as soon as".

4 The Process Submodel

A complete process model should allow representation of "*what* is going to be done*, who* is going to do it, *when* and *where* it will be done, *how* and *why* it will be done, and *who is dependent* on its being done" [CKO92]. The process model presented in this section allows one to answer five of these seven questions. We do not include a spatial attribute for processes and we do not consider dependencies [Yu94] explicitly.

The main concepts of the process submodel are: *action, process*, *role, actor* and *goal*. The process submodel is connected to the organisational submodel through the concepts of *actor* and *role*. All actions carried out as part of a process are executed in the context of an organisational role by an actor playing that role. In this respect we have been inspired by the Role-Activity diagrams of

[Oul95]. The process submodel is also closely related to the objectives and goals submodel: processes are operationalisations of organisational goals [Bub94, AMP94] .

4.1 Primitive and Complex Actions

Our process submodel is built around the concepts of situation calculus [MH69, Rei91] and the concurrent logic programming language ConGolog [DGLL97] . The situation calculus is a first-order language for representing dynamically evolving domains. A *situation* is a state of affairs in the world we are modelling. Changes are brought about in situations as results of actions performed by actors. Actions are distinguished into *primitive* and *complex* . Usually an action is considered to be primitive if no decomposition will reveal any further information of interest. To deal with these new concepts, we enrich our language with two unary predicates: *Action* for actions and *Situation* for situations (these predicates have already been used in Example 3.1).

Actions are denoted by first-order terms e.g., $SendOfferLetter(act, app)$. For an action α and a situation s, the term $Do(\alpha, s)$ denotes the situation that results from the execution of action α in situation s. Relations whose truth values may differ from one situation to another are called *fluents*. They are denoted by predicate symbols having a situation term as their last argument. Similarly, the term *functional fluent* is used to denote functions whose denotation varies from one situation to another. Primitive actions are introduced formally by expressions of the following form:

action α
  precondition $\phi 1$
  effect $\phi 2$
endAction

where α is an action, and $\phi 1, \phi 2$ are formulas of Λ. To enhance readability of our specifications, our syntax for actions avoids the detailed syntactic conventions used traditionally in situation calculus [Rei91].

Example 4.1 The following expression defines the action of forwarding an application *app* by actor $act1$ to actor $act2$:

action $ForwardApp(act1, act2, app)$
  precondition $Actor(act1) \wedge Application(app) \wedge Has(act1, app)$
  effect $Has(act2, app) \wedge \neg Has(act1, app)$
endAction

Our framework permits the recursive definition of *complex actions* (simply actions from now on) by adopting the exact syntax and semantics of ConGolog [DGLL97]. ConGolog is a concurrent programming language based on logic developed by the Cognitive Robotics group of the University of Toronto (see www.cs.toronto.edu/~cogrobo for more details). ConGolog was originally developed as a high-level language for programming robots and software agents.

Recently, it has also been used for business process modelling and analysis [Ple95, Ple96, YML96, LKMY99].

In ConGolog actions are defined recursively as follows:

- *Primitive actions* are actions.
- The special action of *doing nothing* is an action and is denoted by **noOp** .
- *Sequencing*. If $\alpha 1, \alpha 2$ are actions, then $\alpha 1; \alpha 2$ is the action that consists of $\alpha 1$ followed by $\alpha 2$.
- *Waiting for a condition*. If ϕ is a formula of Λ then $\phi?$ is the action of waiting until condition ϕ becomes true.
- *Non-deterministic choice of actions*. If $\alpha 1, \alpha 2$ are actions, then $\alpha 1 \mid \alpha 2$ is the action consisting of non-deterministically choosing between $\alpha 1$ and $\alpha 2$.
- *Non-deterministic choice of action parameters.* If α is an action, then $\prod_x(\alpha)$ denotes the non-deterministic choice of parameter x for α .
- *Non-deterministic iteration*. If α is an action, then $\alpha *$ denotes performing α sequentially zero or more times.
- *Conditionals and iteration*. If $\alpha 1, \alpha 2$ are actions, then **if** ϕ **then** $\alpha 1$ **else** $\alpha 2$ defines a conditional and **while** ϕ **do** $\alpha 1$ defines iteration.
- *Concurrency.* If $\alpha 1, \alpha 2$ are actions, then $\alpha 1 \| \alpha 2$ is the action of executing $\alpha 1$ and $\alpha 2$ concurrently.
- *Concurrency with different priorities*. If $\alpha 1, \alpha 2$ are actions, then $\alpha 1 >> \alpha 2$ denotes that $\alpha 1$ has higher priority than $\alpha 2$, and may only execute when $\alpha 1$ is done or blocked.
- *Non-deterministic concurrent iteration*. If α is an action, then $\alpha^{\|}$ denotes performing α concurrently zero or more times.
- *Interrupts*. If $\vec{x}$ is a list of variables, ϕ is a formula of Λ and α is an action then $\langle \vec{x} : \phi \rightarrow \alpha \rangle$ is an interrupt. If the control arrives at an interrupt and the condition ϕ is true for some binding of the variables then the interrupt triggers and α is executed for this binding of the variables. Interrupts are very useful for writing *reactive* processes.
- *Procedures*. Procedures are introduced with the construct **proc** $\beta(\vec{x})$ **endProc**. A *call* to this procedure is denoted by $\beta(\vec{x})$.

Examples of complex actions are given in Figures 1, 2 and 3 (see Section 6). Other examples of ConGolog programs can be found in [DGLL97, LKMY99].

4.2 Categories of Actions

We distinguish actions into *causal* and *knowledge-producing.* Causal actions change the state of affairs in the enterprise we are modelling (e.g., the action of forwarding an application form). Knowledge-producing actions do not change the state of the enterprise but rather the mental state of the enterprise actors (e.g., a perceptual or a communicative action) [SL93, LLR99]. It is known that knowledge-producing actions can be defined in the situation calculus formalism [SL93, LLR99].

Finally, actions can be *exogenous*. This concept corresponds to the notion of external event in other process frameworks. Exogenous actions are necessary in an enterprise modelling framework since they allow us to "scope" our modelling and consider certain parts of the enterprise (or its environment) as being outside of the area we are modelling. Exogenous actions can also be handled by the situation calculus formalism [DGLL97] .

4.3 Business Processes

A *business process* can now be informally defined as a network of actions performed in the context of one or more organisational roles in pursuit of some goal. Formally, a business process is defined by an expression of the following form:

```
process     id
  purpose   goals
    RoleDefs
endProcess
```

where *id* is a process identifier, *goals* is a list of goals (separated by commas) and *RoleDefs* is a sequence of statements defining roles and their local ConGolog procedures. The purpose statement in a process definition describes the organisational goals *achieved* by the process. The concept of purpose captures why a process is done [CKO92]. Explicating the purpose of a process is very useful; it forces business analysts to think carefully, at 'the intentional level', about the processes they are designing. Later on, we will present a method for verifying formally that a process specification achieves its purpose. This allows business analysts to deal with incomplete or incorrect process specifications.

The actions that make up the processes are distributed among organisational roles, and ConGolog procedures are used to capture their details. Roles and their procedures are defined by expressions of the following form:

```
role  id
  responsibility resps
    ProcedureDefs
endRole
```

where *id* is a role identifier, *resps* is a list of goals (separated by commas) and *ProcedureDefs* is a set of ConGolog procedures. The responsibility statement declares that role *id* is responsible for achieving the goals in list *resps*. The procedures associated with a role define the behaviour of an actor playing that role in pursuit of the goals. Examples of role definitions are given in Figures 1, 2 and 3 (see Section 6).

Our formal framework permits the detection of conflicts that may arise due to the presence of multiple roles or the association of multiple procedures with a single role. This and other cases of incomplete or incorrect process specifications can be detected using the machinery presented in Section 6.5.

5 The Concepts and Constraints Submodels

The *concepts* submodel contains information about enterprise entities, their relationships and attributes. Information in this submodel is formally expressed by sentences of Λ using appropriate predicate and function symbols (e.g., for our DEPT enterprise a predicate $Has(act, app)$ might be used to denote that actor act has application app). Enterprise data are part of this submodel.

The *constraints* submodel is used to encode restrictions imposed on the enterprise. Constraints can be formally expressed by sentences of Λ using the machinery of the situation calculus and the symbols defined in the other submodels. Constraints can be static (i.e., referring to a single situation) or dynamic (i.e., referring to more than one situation) [Ple96]. An example of a static constraint is given in Example 6.1 (see Section 6.5).

This section ends the presentation of our enterprise model. Let us now turn to our methodology.

6 A goal-oriented methodology for business process design

The objective of this section is to outline a methodology which can be used by an enterprise that wishes to develop a *new* business process. The methodology starts with the objectives of the enterprise concerning this new development and produces a detailed formal specification of a business process that achieves these objectives. The formal specification is developed as a set of submodels (based on the concepts discussed in previous sections) that capture the new process from various viewpoints.

The steps of the proposed methodology are the following:

1. Identify the organisational objectives and goals. Initiate goal reduction.
2. Identify roles and their responsibilities. Match goals with role responsibilities.
3. For each role specify its primitive actions, the conditions to be noticed and its interaction with other roles
4. Develop ConGolog procedures local to each role for discharging each role's responsibilities.
5. Verify formally that the ConGolog procedures local to each role are sufficient for discharging its responsibilities, and that constraints are maintained as a result of process execution.

The steps of the methodology are presented above as if strictly ordered, but some of them will in practice need to run concurrently. Also, backtracking to a previous step will often be useful in practice. The final product of an application of the methodology is a complete enterprise model that can be used to *study* and *analyse* the proposed business process. The process specification can also serve as a guide for the development of an information system that implements the process.

The above methodology is presented in more detail in [KP99]. In the rest of this section we only discuss some of the issues involved in Steps 1 and 2, and then concentrate our attention to Steps 4 and 5 where our approach significantly improves on other related methodologies (e.g., EKD [KL98,BBS98] or GEM [Rao96]).

6.1 Goal Reduction and Responsibility Assignment

The first step of the proposed methodology is the elicitation of an *initial statement* of the enterprise objectives and goals concerning the new process. This will involve brainstorming sessions with the enterprise stakeholders, studying documents (e.g., mission statement) outlining the strategy of the enterprise to be modelled and so on. During this activity the analyst using our methodology must try to uncover not only prescriptive goals, but also descriptive ones [AMP94].

After we have a preliminary statement in natural language of the enterprise objectives and goals, then the process of constructing a corresponding AND/OR goal graph by asking "why" and "how" questions can begin [DvLF93]. This process involves reducing goals, identifying conflicts and detecting positive and negative interactions between goals. The process of goal reduction will lead to a better understanding of the organisational goals, and very often to a reformulation of their informal definition. This step of our methodology is similar to goal reduction steps in goal-oriented requirements modelling frameworks [YM94a, MCN92, DvLF93, vLDM95] and related goal-oriented enterprise modelling frameworks [Bub94, LK95b, KL98, BBS98].

An important issue that needs to be addressed at this stage is the distinction between *achievable* and *unachievable* (or *ideal*) goals. An ideal goal may be something that one strives to achieve, knowing that it is impossible to do so, e.g. achieving total customer satisfaction. Alternatively, it may not be possible to associate the goal with a measurable or observable state of affairs. Ideal goals need to be considered, but in the process of AND/OR-reduction they need to be substituted by weaker goals that are actually achievable [vLDM95].

After the AND/OR graph corresponding to informal goals is sufficiently developed and stable, the process of *goal formalisation* can start. For example, one of the goals in our postgraduate program example can be the following goal G_1 : "enquiries are answered by a member of staff as soon as they are received". This goal can be formalised as has already been shown in Example 3.1.

In parallel with the process of goal reduction, the business analyst should engage in the identification of roles and their responsibilities (Step 2 of the methodology). Role identification is achieved by interacting with the enterprise stakeholders and by considering goals at the lowest level of the developed goal hierarchy. Given one of these goals and the roles currently existing in the organisation, the analyst should then decide whether one of these roles (or a new one) can be designated as responsible for achieving the goal. If this is possible then the goal becomes a role responsibility, otherwise it needs to be refined further. This might sound simple, but role identification and responsibility assignment is a rather difficult task and business analysts could benefit from the provision of guidelines for dealing with it. Such guidelines are discussed in [Oul94].

In our example we assume that the following roles are introduced: Postgraduate Tutor (notation: *Tutor*), Postgraduate Secretary (notation: *Secretary*) and Member of Academic Staff (notation: *Faculty*). For the purposes of our discussion it is not necessary to consider a role for students enquiring about or applying to the postgraduate program. Students will be considered to be outside of the process and interaction with them could be captured through the concept of exogenous actions.

Let us also assume that the following responsibility assignments are made. The Postgraduate Secretary will be responsible for handling all correspondence with applicants but also for forwarding applications to the Postgraduate Tutor. The Postgraduate Tutor will be responsible for doing an initial evaluation of applications and forwarding applications to appropriate members of academic staff. Finally, members of academic staff will be responsible for evaluating promptly all applications they receive.

Once roles have been identified and responsibilities assigned, the goal hierarchy should be revisited. Now goal statements can be made more precise by taking into account the introduced roles, and formal definitions of goals can be rewritten. For example, goal G_1 can be rephrased as "enquiries are answered by the Postgraduate Secretary as soon as they are received". This is formalised as follows:

$$(\forall a)(\forall e)(\forall x)(\forall s)(\forall s')(Actor(a) \wedge Enquiry(e) \wedge Action(x) \wedge Situation(s) \wedge Situation(s') \wedge$$
$$PlaysRole(a, Secretary) \wedge Received(e, a, s) \wedge (s' = Do(x, s)) \supset Answered(a, e, s'))$$

6.3 Defining Roles using ConGolog

The first step in specifying a role is to identify the *primitive actions* that are available to it, the *conditions* to be monitored and the interactions with other roles. Then the detailed specification of the dynamics of each roles is given using the syntax of Section 4. For each role, the business analyst has to specify a ConGolog procedure called *main* which gives the details of the behaviour of the role. Of course, *main* can invoke other local procedures.

In the processes we have modelled so far, we have found ConGolog very natural and easy to use. In most cases it was straightforward to write a piece of ConGolog code for each responsibility of a role, and then combine those pieces to form a complete specification of the dynamics of the role. We expect to come up with more precise guidelines for using the language as our experience with it increases.

For our example let us first consider the role *Tutor*. This role can perform the causal action *ForwardApp* (defined in Example 4.1) and the knowledge producing action $SendMsg(sender, recipient, msg)$ which means that actor *sender* sends message *msg* to actor *recipient*. A precise specification of *SendMsg* and other useful communicative actions in situation calculus can be found in [LLL+95].

Role *Tutor* also needs to watch for condition $Has(actor, app)$ where *act* is the actor playing the role *Tutor* and *app* is an application. The complete specification of role *Tutor* is shown in Figure 1. Figures 2 and 3 show the specifications of roles *Secretary* and *Faculty.*

```
role   Tutor
  responsibility  ...
  proc   main
      ⟨  app : Has(self , app) →
        if  AvgMark(app) < 70  then
            for  act : PlaysRole(act, Secretary)  do
               SendMsg(self , act, 'INFORM (Unacceptable(app))')
            endFor
        else
            for  act : PlaysRole(act, Lecturer)   do
               ForwardApp(self , act, app)
            endFor
        endIf    ⟩
   endProc
 endRole
```

Figure 1: Role Postgraduate Tutor

```
 role   Secretary
   responsibility G1 , ...
   proc  main
     ⟨  infoReq:  Received(self, infoReq )  →  ReplyTo(self, infoReq)   ⟩
     >>
     ⟨  app : Has(self , app) →
       for  act : PlaysRole(act, Tutor)  do
           Forward (self , act, app)
       endfor   ⟩
     >>
     while True do
       SenseMsg(self);
       if  ¬Empty(MsgQ(self ))  then
         if  First(MsgQ(self )) = (lect, 'INFORM (WantsToSupervise(lect, app))')  then
            SendOfferLetter(self , app)
         else if  First(MsgQ(self )) = (tut, 'INFORM (Unacceptable(app))')  then
            SendRejectionLetter(self, app)
         endIf
       endIf
     endWhile
   endProc
 endRole
```

Figure 2: Role Postgraduate Secretary

```
role Faculty
  responsibility ...

  proc  Eval(self, app)
    if  GoodUniv(Univ(app)) ∧ AvgMark(app) > MinMark(self) ∧
        NoOfStud(self) < MaxNoOfStud(self) then
        for act : PlaysRole(act, Secretary) do
            SendMsg(self, act, 'INFORM(WantsToSupervise(self, app))')
        endFor
    endIf
  endProc

  proc  main
    ⟨ app : Has(self, app) → Eval(self, app) ⟩
  endProc
endRole
```

Figure 3: Role Academic Member of Staff (Faculty)

The ConGolog code should be easy to understand but the following comments are in order. First, notice that in the interest of brevity we have omitted unary predicates like *Actor, Application* etc. that are used to type variables. We have also omitted specifying explicitly the responsibilities assigned to each role; only G_1 is specified as a responsibility for role *Tutor*. Symbol *self* is a pseudo-variable denoting the actor playing the role inside which *self* appears. The reader should notice how natural it is to specify in ConGolog reactive processes using interrupts and concurrency.

The specification of role *Secretary* is perhaps more involved because a message queue is used in the spirit of [LLL+95]. The specification for secretary does not handle the case where more than one member of academic staff wants to supervise the same applicant (this is not a problem for ConGolog; we simply omit this case). We also omit the specification of exogenous actions that capture the interaction between the role *Secretary* and the applicants (that are part of the outside environment).

Given the above specifications for roles *Secretary*, *Tutor* and *Faculty*, the specification of the complete business process is straightforward using the syntax of Section 4.

6.5 Formal Verification

Let us now discuss the final step of our methodology where we *verify formally* that each role responsibility is fulfilled and each constraint is maintained by the ConGolog procedures defined locally for each role.

To perform verification we utilise the techniques reported in [Ple95, Ple96], which are based on a systematic solution to the frame and ramification problems [Rei91]. Specifically, we are interested

in determining whether: (i) responsibilities of roles can be fulfilled, and (ii) constraints defined in the constraints submodel are preserved or violated as a result of process execution. These questions are answered by reasoning with the process specification resulting from the previous step of the methodology. In the case where such a proof or disproof is not possible at process specification time, strengthenings to the specifications of actions that are relevant to the responsibilities/constraints are proposed, so that any process implementation meeting the strengthened specifications provably guarantees that the responsibilities/constraints will be satisfied in the state resulting from action execution. The method proceeds by deriving ramifications of constraints and action preconditions and effects, and by using these ramifications to strengthen the action specifications [Ple95,Ple96] .

Example 6.1 Let us consider the specification of the action *SendOfferLetter* shown below (this is a simplified version of the action used in role *Secretary*). The predicate $Accepted(App)$ denotes that application *app* has been accepted by DEPT. Similarly, $WantsToSupervise(lect, app)$ means that academic *lect* would like to supervise the student of application *app* .

action $SendOfferLetter(app)$
  precondition
    $Application(App) \wedge (\exists lect)(PlaysRole(lect, Faculty) \wedge WantsToSupervise(lect, app))$
  effect $Accepted(app)$
endAction

Assume that we wish to enforce the policy that no applicant can be both accepted and rejected. This constraint may be expressed by the following sentence of Λ (and belongs to the constraints submodel):[4]

$$(\forall p)(Application(p) \wedge Accepted(p) \supset \neg Rejected(p))$$

It is evident that the action specification given above does not exclude a situation in which both $Accepted(app)$ and $Rejected(app)$ are satisfied. We can easily see that if the constraint is to be preserved in the situation resulting from performing action *SendOfferLetter* , then $\neg Rejected(p)$ is a logical implication of the constraint, i.e., a ramification of the constraint and the action specification. Our ramification generator proposes that the term $\neg Rejected(p)$ be used to strengthen the action specification (by conjoining the term with the action precondition or effect). The strengthened specification is now guaranteed not to violate the constraint in any possible execution of the action *SendOfferLetter* .

Albeit short[5] and simple, the above example conveys the idea behind the derivation of ramifications for strengthening action specifications. More complex examples and details of the generation process can be found in [Ple95, Ple96]. The same ideas can be used to verify formally that roles fulfil the responsibilities assigned to them.

The aforementioned work provides results for verifying properties of primitive actions and of processes including sequencing of actions, when the constraints refer to at most two distinct states.

[4] The predicate *Rejected* denotes that some applicant has been rejected by DEPT.
[5] We have intentionally omitted presenting all the steps in the generation process due to lack of space.

The derivation of similar results for processes synthesised using any of the remaining ConGolog constructs - including concurrency and non-determinism - and for general dynamic constraints is a topic of current research. Our previous work can also accommodate knowledge-producing actions in a single-agent environment. The theoretical basis of ConGolog has been extended to include exogenous and knowledge-producing actions in a multi-agent environment [LLR99]. The adaptation of these ideas in our analysis and verification techniques is an ongoing effort.

We argue that the ability to verify properties of processes is essential for business process design and re-engineering. The process specifier realises the implications of actions as far as goal achievement is concerned and the implementor is saved the burden of having to find ways to meet postconditions and maintain invariants. Furthermore, optimised forms of conditions to be verified can be incorporated into process specifications and consistency is guaranteed by the soundness of the verification process [Ple96].

7 Discussion

The first paper to propose situation calculus and ConGolog (more precisely its earlier version Golog) for business process modelling was [Ple95]. Since then similar ideas have appeared in [Ple96, YML96, LKMY99]. But so far, ConGolog has not been used in conjunction with a more general framework like ours that offers intentional concepts like actors, roles and goals. This is an important contribution of the current paper.

Situation calculus is also the formalism of choice for the TOVE enterprise modelling project [FG98]. However TOVE concentrates mostly on enterprise ontologies and uses situation calculus for their formalisation. To the best of our knowledge TOVE does not concentrate on process modelling and has not proposed anything corresponding to our methodology.

The concepts of goals, actors and roles also appear prominently in the i^* framework [Yu94] where the need for *intentional concepts* in enterprise modelling (and requirements modelling) is emphasised. i^* also supports the concept of dependency between actors, something which is not offered by our framework (and would be a nice addition to it). Our methodology could possibly benefit by incorporating some features of the i^* framework (e.g., dependencies, strategic rationale etc.).

There is a clear connection of our work to goal-oriented methodologies for requirements engineering especially KAOS [DvLF93]. This connection has been explained in detail in previous sections of this paper so we will not elaborate on it here.

Our work is also related to the enterprise modelling frameworks of F^3 [Bub94, LK95b] and its successor EKD [KL98, BBS98]. In EKD knowledge about an organisation is partitioned into the following submodels: the goals submodel (corresponds to our objectives and goals submodel), the actors and resources submodel (roughly corresponds to our organisational submodels), the business processes submodel (corresponds to our process submodel), the concepts submodel (corresponds exactly to concepts submodel), the business rules submodel (it is more involved than our constraints submodel), and the technical components and requirements submodel (no corresponding concept in our work). In terms of formalisms, EKD uses entity-relationship models

to represent structural information and Role-Activity Diagrams [Oul94] to represent roles and their activities. Our proposal, compared with EKD, offers more expressive languages (situation calculus and ConGolog) and is therefore more amenable to formal reasoning. On the other hand, we have not attempted to be as comprehensive as EKD in our coverage of issues related to enterprise modelling, and, so far, we have not tried our models and methodology in significant industrial applications.

Lee's Goal-based Process Analysis (GPA) is also related to our research [Lee94]. GPA is a goal-oriented method and can be used to analyse existing processes in order to identify missing goals, ensure implementation of all goals, identify non-functional parts of a process, and explore alternatives to a given process.

Finally, our work has many common ideas with the GEM models and methodology [Rao96]. According to GEM business processes are collections of suitably ordered activities, enacted by individual persons, depending on their role within an organisation. Every process has a purpose which is to achieve a goal or react to an event. GEM offers a number of models that can be used for specifying processes: the role interaction model, the purpose model, the procedure model, the internal data model and the corporate data model. The GEM methodology consists of three steps: defining the scope of the business process, doing process analysis and doing system design. The process analysis step consists of the following stages: goal hierarchy analysis, basic procedure analysis, detailed procedure analysis, input/output data analysis and performance metrics specification. We have been unable to make a more detailed comparison of our work with GEM because the only related document publicly available [Rao96] gives only a short informal description of the models and methodology. A methodology for developing multi-agent systems based on concepts similar to the ones in GEM appears in [KGR96] .

The vast majority of business process modelling efforts lack formal methods for verifying properties of processes. A user-assisted verification tool handling arbitrary ConGolog theories is currently under development as reported in [LKMY99]. Strengthening of specifications using inference rules has also been proposed in [DvLF93] in the context of the KAOS project. Verification of process properties however is not treated systematically.

Orthogonally to verification, validation tools may be employed for testing whether processes execute as expected in various conditions. In [vLDM95] the use of operational scenarios is proposed for discovering overlooked aspects of the specified model, such as, e.g., missing goals. A simulation tool based on logic programming has been developed for validating ConGolog processes [LKMY99]. The tool also includes a module for progressing an initial situation, allowing it to simulate the execution of long-running processes.

8 Conclusions

We presented a formalism that can be used to represent knowledge about organisations and their business processes. We also discussed a methodology that enables business analysts to go from high-level enterprise objectives, to detailed and formal specifications of business processes for realising these objectives. The methodology can be used by an enterprise that wishes to develop a new business process, or alternatively model, document and analyse formally an existing process.

The main contribution of our work is the use of formal languages from Artificial Intelligence (situation calculus and ConGolog) for business process modelling and analysis. We strongly believe that the use of formal methods such as the ones discussed in this paper can be of significant benefit to business analysts. In the past formal methods have been shown to offer significant advantages in the requirements modelling domain by projects such as KAOS [DvLF93]. In this paper we have demonstrated that formal methods can be valuable in the domain of business modelling and analysis as well. The main advantage of formal methods (compared with more informal approaches such as EKD) is that they can be used by sophisticated business analysts to capture business knowledge in an *intuitive* and *unambiguous* way. They can also be used to analyse processes in a formal way (e.g., see Section 6.5 on process verification); this would have been impossible if the business analyst used an informal approach.

Several possible criticisms can be voiced against the use of formal methods in enterprise modelling:

- It is a lot of work to create a formal enterprise model initially. Additionally, it is hard to maintain it to retain consistency with the actual enterprise.
- The use of complex mathematical notation may put off the average manager, business analyst or user.
- Special skills are required (in our case, familiarity with situation calculus and ConGolog).

The first criticism is not really a criticism of 'formal' enterprise modelling but rather of 'any kind' of enterprise modelling. There is a price to pay for undertaking an enterprise modelling effort but we would argue that the long-term benefits will outweigh the investment in resources.

The second and third criticism are valid. Formal tools such as the ones proposed in this paper are somewhat complex, and business analysts may not bother to become familiar with them, opting for more informal methods (in our case, they could use EKD). This can be a problem with formal methods but only if the people advocating them are not careful. The solution lies in developing supporting tools -that offer the possibility of working with *formal* and *informal* versions of the same concept (e.g., allow the possibility to describe a business procedure using a role-activity diagram [Oul94] but also using our ConGolog-based notation). In this way any business analyst will find the supporting tools attractive and easy to use, while more sophisticated analysts will be able to resort to the formal machinery whenever they feel that they will gain advantage from doing so. As time goes by, even less formally-inclined business analysts might also be tempted to invoke the formal functionalities.

Our future work will concentrate on demonstrating that the proposed formal methods are useful in practice. In the spirit of the previous discussion, we would like to develop a set of user-friendly supporting tools for our enterprise modelling techniques and methodology. In parallel we would like to apply our techniques to the modelling of large processes so that we can evaluate our methodology and quantify any benefits over other approaches.

We would also like to extend the techniques of [Ple95, Ple96] to accommodate all features of ConGolog. Finally, we are also interested in extending our methodology to deal with the problem of *business change* and investigate what formal techniques and reasoning can be beneficial in this case.

Acknowledgements

The first author would like to thank Paul Kearney, Paul O'Brien, Mark Weigand and Nader Azarmi for support and encouragement during his collaboration with BT Labs.

We also acknowledge the kindness of Vagelio Kavakli who was always there with comments, pointers and encouragement that helped us understand the features of EKD. We would also like to thank Liz Kendall for providing us with a pointer to the GEM methodology.

References

[AMP94] A.I. Anton, M.W. McCracken, and C. Potts. *Goal decomposition and scenario analysis in business process reengineering.* In Proceedings of CAISE'94 , pages 94-104, 1994.

[BBS98] J. Bubenko, D. Brash, and J. Stirna. *EKD user guide*, 1998. Available from ftp://ftp.dsv.su.se/users/js/ekd_user_guide.pdf.

[Bub94] J. Bubenko. *Enterprise Modelling.* Ingenierie des Systems d' Information, 6(2), 1994.

[Chu93] L. Chung. *Representing and Using Non-Functional Requirements: A Process-Oriented Approach.* Ph.D. thesis, Dept. of Computer Science, University of Toronto, 1993.

[CKO92] B. Curtis, M. Kellner, and J. Over. *Process Modelling.* Communications of ACM 35(9):75-90, 1992.

[Dav93] P.T. Davenport. *Process Innovation: Re-Engineering Work Through Information Technology.* Harvard Business School Press, 1993.

[DBCS94] J.E. Dobson, A.J.C. Blyth, J. Chudge, and R. Sterns. *The ORDIT Approach to organisational requirements.* In M. Jirotka and J. Goguen, editors, Requirements Engineering: Social and Technical Issues, pages 87-106. Academic Press, 1994.

[DGLL97] G. De Giacomo, Y. Lesperance, and H. Levesque. *Reasoning About Concurrent Execution, Prioritised Interrupts and Exogenous Actions in the Situation Calculus.* In Proceedings of IJCAI'97, pages 1221-1226, August 1997.

[DvLF93] A. Dardenne, A. van Lamsweerde, and S. Fickas. *Goal-Directed Requirements Acquisition.* Science of Computer Programming, 20:3-50, 1993.

[FG98] M.S. Fox and M. Gruninger. *Enterprise Modelling.* The AI Magazine, pages 109-121, Fall 1998.

[FN71] R. Fikes and N. Nilsson. *STRIPS: A new approach to the application of theorem proving to problem solving.* Artificial Intelligence, 2:189-208, 1971.

[GHS95] D. Georgakopoulos, M. Hornick, and A. Sheth. *An Overview of Worklfow Management: From Process Modelling to Workflow Automation Infrastructure.* Distributed and Parallel Databases, 3:119-153, 1995.

[HC93] M Hammer and J Champy. *Reengineering the Corporation: A Manifesto for Business Revolution*. Harper Collins, 1993.

[IDE] http://www.idef.com/

[JFJ + 96] N.R. Jennings, P.Faratin, M.J. Johnson, P. O'Brien, and M.E. Wiegand. *Using Intelligent Agents to Manage Business Processes.* In Proceedings of the First International Conference on The Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM96), 1996.

[Ken99] L. Kendall. *Role Models: Patterns of Agent System Analysis and Design.* BTTJ Vol. 17 No. 4, October 1999.

[KGR96] D. Kinny, M. Georgeff, and A. Rao. *A methodology and modelling technique for systems of BDI agents.* In Proceedings of MAAMAW-96, 1996.

[KL98] V. Kavakli and P. Loucopoulos. *Goal-Driven Business Process Analysis - Application in Electricity Deregulation.* In Proceedings of CAISE'98, 1998.

[KP99] M. Koubarakis and D. Plexousakis. *Business Process Modeling and Design: AI Models and Methodology.* Proceedings of the IJCAI-99 Workshop on Intelligent Workflow and Process Management: The New Frontier for AI in Business, Stockholm, Sweden, August 1-2, 1999.

[KO97] S. Kirn and G. O'Hare. *Cooperative Knowledge Processing: The Key Technology for Intelligent Organisations.* Springer, 1997.

[LA94] F. Leymann and W. Altenhuber. *Managing Business Processes as an Information Resource IBM Systems Journal,* 33(2):326-348, 1994.

[Lee90] J. Lee. *SIBYL: A Qualitative Decision Management System.* In P.H. Winston and S.A Shellard, editors, Artificial Intelligence at MIT: Expanding Frontiers, Vol. 1, pages 105-133. MIT Press, 1990.

[Lee94] J. Lee. *Goal-Based Process Analysis: A Method for Systematic Process Redesign.* In Proceedings of the Conference on Organisational Computing Systems (COOCS'94) , 1994.

[LK95a] P. Loucopoulos and V. Karakostas. *System Requirements Engineering.* McGraw Hill, 1995.

[LK95b] P. Loucopoulos and V. Kavakli. *Enterprise Modelling and the Teleological Approach to Requirements Engineering.* International Journal of Intelligent and Cooperative Information Systems, 4(1):45-79, 1995.

[LKMY99] Y. Lesperance, T.G. Kelley, J. Mylopoulos, and E. Yu. *Modelling dynamic domains with Congolog.* In Proceedings of CAISE'99, 1999.

[LLL + 95] Y. Lesperance, H.J. Levesque, F. Lin, D. Marcu, R. Reiter, and R.B. Scherl. *Foundations of a Logical Approach to Agent Programming* . In M. Wooldridge, J.P. Muller, and M. Tambe, editors, Intelligent Agents Volume II - Proceedings of ATAL-95, Lecture Notes in Artificial Intelligence. Springer Verlag, 1995.

[LLR99] Y. Lesperance, H. Levesque, and R. Reiter. *A situation calculus approach to modeling and programming agents, 1999.* Available from http://www.cs.toronto.edu/~ cogrobo/.

[MCB92] J. Mylopoulos, L. Chung, and B. Nixon. *Representing and Using Non-Functional Requirements: A Process-Oriented Approach.* IEEE Transactions on Software Engineering 18(6):483-497, 1992.

[MH69] John McCarthy and Patrick J. Hayes. *Some Philosophical Problems From the Standpoint of Artificial Intelligence.* In B. Meltzer and D. Mitchie, editors, Machine Intelligence, pages 463-502. Edinburg University Press, 1969.

[NIS] http://www.mel.nist.gov/psl/.

[Oul94] M. Ould. *Modelling Business Processes for Understanding, Improvement and Enactment.* Tutorial Notes, 13th International Conference on the Entity Relationship Approach (ER' 94), Manchester, U.K., 1994.

[Oul95] M. Ould. *Business Processes: Modeling and Analysis for Re-engineering and Improvement.* Wiley, 1995.

[Ple95] D. Plexousakis. *Simulation and Analysis of Business Processes Using GOLOG.* In Proceedings of the Conference on Organizational Computing Systems (COOCS'95), pages 311-323, 1995.

[Ple96] D. Plexousakis. *On the efficient maintenance of temporal integrity in knowledge bases.* Ph.D. thesis, Dept. of Computer Science, University of Toronto, 1996.

[Rao96] A. Rao. *Modelling the service assurance process for Optus using GEM.* Technical Note 69, Australian Artificial Intelligence Institute, 1996.

[Rei91] R. Reiter. *The Frame Problem in the Situation Calculus: A Simple Solution (Sometimes) and a Completeness Result for Goal Regression.* In Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy, pages 359-380. Academic Press, 1991.

[SL93] R. Scherl and H. Levesque. *The frame problem and knowledge producing actions.* In Proceedings of AAAI-93, 1993.

[Tid93] G. Tidhar. *Team-oriented programming: Social structures.* Technical Note 47, Australian Artificial Intelligence Institute, 1993.

[vLDL98] A. van Lamsweerde, R. Darimont, and E. Letier. *Managing Conflicts in Goal-Driven Requirements Engineering.* IEEE Transactions on Software Engineering, November 1998. Special Issue on Managing Inconsistency in Software Development.

[vLDP95] A. van Lamsweerde, R. Darimont, and P. Massonet. *Goal-Directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learned.* In Proceedings of RE'95, 1995.

[WFM] http://www.wfmc.org/.

[YM94a] E. Yu and J. Mylopoulos. *Understanding "Why" in Software Process Modelling.* In Proceedings of the 16th International Conference on Software Engineering, pages 135-147, Sorrento, Italy, 1994.

[YM94b] E. Yu and J. Mylopoulos. *Using Goals, Rules and Methods to Support Reasoning in Business Process Reengineering.* In Proceedings of the 27th Annual Hawaii International Conference on Systems Sciences , pages 234-243, Hawaii, 1994.

[YML96] E. Yu, J. Mylopoulos, and Y. Lesperance. *AI Models for Business Process Reengineering.* IEEE Expert, 11(4):16--23, 1996.

[Yu94] E. Yu. *Modelling Strategic Relationships For Process Reengineering.* PhD thesis, Dept. of Computer Science, University of Toronto, 1994.