A Formal Model for Business Process Modeling and Design

Manolis Koubarakis[1] and Dimitris Plexousakis[2]

[1] Dept. of Electronic and Computer Engineering
Technical University of Crete
73100 Chania, Crete, Greece
manolis@ced.tuc.gr
[2] Dept. of Computer Science
University of Crete
71305 Heraklion, Crete, Greece
dp@csd.uch.gr

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 enterprise knowledge in a way that is both intuitive and mathematically formal. We also outline the basic steps of a methodology that allows business analysts to produce detailed, formal specifications of business processes from high-level enterprise objectives. The use of a formal language permits us to verify that the specifications possess certain correctness properties, namely that the responsibilities assigned to roles are fulfilled and that the constraints are maintained.

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 [10,22,9,11,33]. Ultimately this community is interested in improving the understanding of organisations and their processes, facilitating process design and analysis and supporting process management. 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 wrokflow systems.

In this paper we present a formalism that can be used to represent knowledge about organisations and their business processes. Motivated by F^3 [19] and EKD [12,2] we develop an *enterprise model* which consists of five interconnected submodels (organisational submodel, objectives and goals submodel, process submodel, concepts submodel and constraints submodel) that can be used to describe *formally* different aspects of an organisation. However, unlike

B. Wangler, L. Bergman (Eds.): CAiSE 2000, LNCS 1789, pp. 142–156, 2000.

these projects, our framework emphasises formality and advocates the use of situation calculus [20,27] and the concurrent logic programming language ConGolog [5] for representing knowledge about organisations and their processes. In this respect, we continue the work of one of us [24,25] (but also [33,15]) who suggested to use these formal tools to model organisations. Early steps towards the development of our model were presented in [13].

Creating an enterprise model can be instructive in itself, revealing anomalies, inconsistencies, inefficiencies and opportunities for improvement. Once it exists it is a valuable means for sharing knowledge within the enterprise. It can also be used to formulate and evaluate changes. 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 to find out who is fulfilling a given 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 analyze extract knowledge from and reason about them. An advantage of formal models is that they are self-consistent and have certain properties. For instance, one can prove formally that responsibilities assigned to roles are fulfilled, and constraints are maintained as a result of process execution. A few words about our representational framework are in order here. We will represent enterprise knowledge using an extension of the formalism of *situation calculus* [20,27]. This formalism has been designed especially for knowledge representation and reasoning in dynamically evolving domains. Technically, our basic tool will be a *many-sorted first-order language* $\mathcal{L}$ which is defined in the following way. The logical symbols of $\mathcal{L}$ include parentheses, a countably infinite set of variables, the equality symbol $=$ and the standard sentential connectives. The remaining machinery of $\mathcal{L}$ (sort, predicate and function symbols) will be defined in Sections 2, 3 and 4 where intuitive modeling concepts will need to be formalised.

The rest of this paper is structured as follows. Sections 2, 3 and 4 present our enterprise model. We then sketch a methodology that enables business analysts to go from high-level enterprise-objectives, to detailed formal specifications of business processes for realizing these objectives. Finally, section 6 discusses related work and presents our conclusions.

2 Organisational and Goal Modeling

In this section we initiate the presentation of the five submodels making up our enterprise modeling framework. Throughout the paper we will demonstrate the features of our proposal by considering an imaginary Computer Science department DEPT as our enterprise. 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.

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 modeling (e.g., an employee, a customer, a printer etc.).

Actors are distinguished into *human* and *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 [23,6]Organisational roles can take many forms [23]: 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. 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.

The concepts introduced above can be defined formally by introducing appropriate constructs of $\mathcal{L}$ (e.g., unary predicates $Actor$, $HumanActor$, $Automated\text{-}Actor$ and $Role$, and binary predicate $PlaysRole$) and writing axioms that capture their semantics.

The second component of our enterprise model is the objectives and goals submodel. The central concept in this submodel is an enterprise goal. An *enterprise goal* is a desired state of affairs [19,14,22,33,12]. Examples of enterprise goals are the following: "all customers 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 "project X is completed successfully".
- *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 [32]. 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").

2.1 Enterprise Goals

Organisational goals can be *reduced* into alternative combinations of subgoals [4,31,14,19,12,2] by using *AND/OR goal graphs* originally introduced in the area

of problem solving [7]. 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 *organisational objective* is a goal that does not present itself 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 [18].

Goals can *conflict* with each other [4,31,19,29]. In our framework goals $G_1, \ldots, G_n$ conflict if they cannot be *satisfied* simultaneously given our knowledge about the enterprise [29]. Goals can also *influence* positively or negatively other goals [21,31,19]. Such interactions between goals must be noted explicitly to facilitate goal-based reasoning (see Section 5).

2.2 Defining Goals Formally

Organizational goals can be described formally or informally. Organisational objectives and other high-level goals are usually difficult to formalise. These goals should be described only 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 modeling) are discussed in [4].

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

Example 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 $\mathcal{L}$:

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

In the above sentence predicates have the obvious meaning and $s' = Do(x, s)$ means that s' is the situation (i.e., state) resulting from the execution of action x in situation s. 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". 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".

3 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" [3]. Our process model allows one to answer five of these seven questions. We do not include a spatial attribute for processes and we do not consider dependencies explicitly [32].

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 [23]. The process submodel is also closely related with the objectives and goals submodel: processes are operationalisations of organisational goals [1].

3.1 Primitive and Complex Actions

Our process submodel is built around the concepts of situation calculus [20,27] and the concurrent logic programming language ConGolog [5]. 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 modeling. Changes are brought to being 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 which is of interest. To deal with these new concepts, we enrich our language $\mathcal{L}$ with a sort *Action* for actions and a sort *Situation* for situations. 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. Primitive actions are introduced formally by expressions of the following form:

action α
  precondition ϕ_1
  effect ϕ_2
endAction

where α is an action, and ϕ_1, ϕ_2 are formulas of $\mathcal{L}$.

Example 2. The following expression defines the action of forwarding an application *app* by actor *act*1 to actor *act*2:

action $ForwardApp(act1, act2, app)$
  precondition $Has(act1, app)$
  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 syntax and semantics of ConGolog [5]:

- *Primitive actions* are actions.
- The special action of *doing nothing* is an action and is denoted by **noOp**.
- *Sequencing.* If α_1, α_2 are actions, then $\alpha_1; \alpha_2$ is the action that consists of α_1 followed by α_2.
- *Waiting for a condition.* If ϕ is a formula of $\mathcal{L}$ then $\phi?$ is the action of waiting until condition ϕ becomes true.

- *Non-deterministic choice of actions.* If α_1, α_2 are actions, then $\alpha_1|\alpha_2$ is the action consisting of non-deterministically choosing between α_1 and α_2.
- *Non-deterministic choice of action parameters.* If α_1, α_2 are actions, then $\Pi_x(\alpha_1)$ denotes the non-deterministic choice of parameter x for α_1.
- *Non-deterministic iteration.* If α is an action, then α^* denotes performing α sequentially zero or more times.
- *Conditionals and iteration.* If α_1, α_2 are actions, then **if** ϕ **then** α_1 **else** α_2 defines a conditional and **while** ϕ **do** α_1 defines iteration.
- *Concurrency.* If α_1, α_2 are actions, then $\alpha_1 \parallel \alpha_2$ is the action of executing α_1 and α_2 concurrently.
- *Concurrency with different priorities.* If α_1, α_2 are actions, then $\alpha_1 \gg \alpha_2$ denotes that α_1 has higher priority than α_2, and α_2 may only execute when α_1 is done or blocked.
- *Non-deterministic concurrent iteration.* If α is an action, then $\alpha^{\parallel}$ denotes performing α concurrently zero or more times.
- *Interrupts.* If $\overline{x}$ is a list of variables, ϕ is a formula of $\mathcal{L}$ and α is an action then $\langle \overline{x} : \phi \rightarrow \alpha \rangle$ 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** $\beta(\overline{x})$ **endProc**. A *call* to this procedure is denoted by $\beta(\overline{x})$.

Examples of complex actions are given in Figures 1 and 2 (see Section 5).

3.2 Categories of Actions

We distinguish actions into *causal* and *knowledge-producing*. Causal actions change the state of affairs in the enterprise we are modeling (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) [28,16]. It is known that knowledge-producing actions can be defined in the situation calculus formalism [28,16].

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 modeling framework since they allow us to "scope" our modeling and consider certain parts of the enterprise (or its environment) as being outside of the area we are modeling. Exogenous actions can also be handled by the situation calculus formalism [5].

3.3 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 introduces the purpose of a process i.e., the organisational goals *achieved* by the process. The concept of purpose captures *why* a process is done [3].

Processes are distributed among organisational roles and ConGolog procedures are used to capture the details of a process. 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*. Examples of role definitions are given in Figures 1 and 2 (see Section 5).

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 5.3.

4 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 $\mathcal{L}$ 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 $\mathcal{L}$ using the machinery of the situation calculus and the symbols defined in the rest of the submodels. Constraints can be static (i.e., referring to a single situation) or dynamic (i.e., referring to more than one situation) [25]. An example of a static constraint is given in Example 3 (Section 5.3).

5 A Goal-Oriented Methodology for Business Process Design

This section outlines 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 which 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:

- Identify the organisational objectives and goals. Initiate goal reduction.
- Identify roles and their responsibilities. Match goals with role responsibilities.
- For each role specify its primitive actions, the conditions to be noticed and its interaction with other roles.
- Develop ConGolog procedures local to each role for discharging each role's responsibilities.
- Verify formally that the ConGolog procedures local to each role are sufficient for discharging its responsibilities.

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 specification can also serve as a guide for the development of an information system implementing the process.

This section does not intend to present the methodology and its application in detail (for this the interested reader should go to [13]. We will only discuss some of the issues involved in Steps 1 and 2, and then concetrate our attention to Steps 4 and 5, where our approach significantly improves on related methodologies (e.g., EKD [2,12] or GEM [26]).

5.1 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 possibly other enterprises in the same industry sector), and so on [2]. During this activity the analyst using our methodology must try to uncover not only prescriptive goals, but also descriptive ones [1].

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 [4]. 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 identical with goal reduction steps in goal-oriented requirements modeling frameworks [31,21,4,30] and related goal-oriented enterprise modeling frameworks [19,12,2].

An important issue that needs to be addressed at this stage is the distinction between *achievable* and *unachievable* (or *ideal*) goals. 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 [30].

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 follwoing goal G_1: "enquiries are answered by a member of staff as soon as they are received". This goal can be formalized as has already been shown in example 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 the roles currently existing in the organization, 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 [22].

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: *Staff*). For the purposes of our discussion it is not necessary to consider a role for students enquiring about or applying to the prostgraduate program. Students are considered to be outside of the process and interaction with them is 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 hadling all correspondence with applicants but also for forwarding applications to the Postgraduate Tutor, who will be responsible for doing an initial evaluation of applications and forwarding applications to appropriate members of academic staff. The latter will be responible for evaluating promptly all applications they receive. Once roles have been identified and responsibilites 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 formalized as follows:

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

5.2 Defining Roles Using ConGolog

The first step in specifying a role is to identify the *primitive actions* that are available to each role, the *conditions* to be monitored and the *interactions* with other roles. Then the detailed specification of the dynamics of each role is given

using the syntax of Section 3. 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 process 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 2) 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 [17]. Role $Tutor$ also needs to watch for condition $Has(actor, app)$ where *actor* is the actor playing the role $Tutor$ and *app* is an application. The complete specification of roles *Tutor*, *Secretary* and *Faculty* are shown in Figures 1 and 2 respectively.

The ConGolog code should be easy to understand but the follwoing 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 responibilities assigned to each role; only G_1 is specified as a responibility got role *Tutor*. Symbol $self$ is a pseudo-variable denoting the actor playing the role inside which the variable appears. The reader should notice how natural it is to specify in ConGolog reactive processes using interrupts and concurrency. The specification of the role *Secretary* is perhaps more involved because a message queue (in the spirit of [17]) is used. The case where more than one members of academic staff want to supervise the same applicant is omitted. We also omit the specification of exogenous actions that capture the interaction between the role *Secretary* and the applicants (that

role $Tutor$
responsibility G_{132}

proc *main*
$\langle app : Has(self, app) \rightarrow$
  if $AvgMark(app) < 70$ **then**
    for $act : PlaysRole(act, Secretary)$ **do**
      $SendMsg(self, act, \ulcorner INFORM(Unacceptable(app)) \urcorner)$
    endFor
  else for $act : PlaysRole(act, Lecturer)$ **do**
      $ForwardApp(self, act, app)$
    endFor
  endIf $\rangle$
endProc
endRole

Fig. 1. Role Postgraduate Tutor

```
role Secretary
responsibility G_12, G_131, G_134

proc main
⟨infoReq : Received(self, infoReq) → ReplyTo(infoReq)⟩
≫
⟨app : Has(self, app) →
   for act : PlaysRole(act, Tutor) do
      Forward(self, act, app)
   endFor⟩ ≫
while True do
   SenseMsg;
   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

role Faculty
responsibility G_133

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
```

Fig. 2. Roles Post Graduate Secretary and Faculty

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 3.

5.3 Formal Verification

In this step we *verify formally* that each role responsibility is fulfilled and each constraint is maintained by the ConGolog procedures defined for each role. To perform verification we utilize the techniques reported in [24,25], which are based on a systematic solution to the frame and ramification problems [27]. Specifically, we are interested in determining whether: (i) responsibilities of roles can be fulfilled, and (ii) constraints are preserved or violated as a result of process execution. In 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 derives ramifications of constraints and action preconditions and effects, and uses them to strengthen the action specifications [24,25].

Example 3. Consider the specification of the action $SendOfferLetter$ shown below. 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 $(\exists\ lect)\ WantsToSupervise(lect, app)$
  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 $\mathcal{L}$ (and belongs to the constraints submodel):

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

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[1] and simple, the above example conveys the idea behind the derivation of ramifications for strengthening action specifications. More complex examples and details can be found in [24,25]. The same ideas can be used to verify formally that roles fulfill their assigned responsibilities.

The aforementioned work provides results for verifying properties of primitive actions and of processes including sequencing of actions, when the constraints

[1] We have intentionally omitted presenting all the steps in the generation process due to lack of space.

refer to at most two distinct states. The derivation of similar results for processes synthesized 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 [16]. 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 realizes 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, optimized forms of conditions to be verified can be incorporated into process specifications and consistency is guaranteed by the soundness of the verification process [25].

6 Discussion

The first paper to propose situation calculus and ConGolog (more precisely its earlier version Golog) for business process modeling was [24]. Since then similar ideas have appeared in [25,33,15]. 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. Situation calculus is also the formalism of choice for the TOVE enterprise modeling project [8]. However, TOVE concentrates mostly on enterprise ontologies rather than process design and verification.

The concepts of goals, actors and roles also appear prominently in the i^* framework [32,33] where the need for *intentional concepts* in enterprise modeling is emphasized. There is also clear connection of our work to goal-oriented methodologies for requirements engineering especially KAOS [4]. 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 (and has been inspired by) the enterprise modeling frameworks of F^3 [19] and its successor EKD [2]. Lee's Goal-based Process Analysis (GPA) is also related to our research [14]. 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 [26]. Detailed comparisons of our work with these related efforts appears in the extended version of this paper (available from the authors).

The vast majority of business process modeling efforts lack formal methods for verifying properties of processes. Exceptions to this rule is the efforts in [15] where the use of ConGolog is advocated as well. We share similar long term research goals with these researchers, and would like to demonstrate that formal languages and methods can offer significant advantages in the design and analysis of business processes.

References

1. 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.
2. J. Bubenko, D. Brash, and J. Stirna. EKD user guide, 1998. Available from ftp://ftp.dsv.su.se/users/js/ekd_user_guide.pdf.
3. B. Curtis, M. Kellner, and J. Over. Process Modelling. *Communications of ACM*, 35(9):75–90, 1992.
4. A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-Directed Requirements Acquisition. *Science of Computer Programming*, 20:3–50, 1993.
5. 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.
6. J.E. Dobson, A.J.C. Blyth, J. Chudge, and R. Strens. 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.
7. R. Fikes and N. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.
8. M.S. Fox and M. Gruninger. Enterprise Modelling. *The AI Magazine*, pages 109–121, Fall 1998.
9. 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.
10. M. Hammer and J. Champy. *Reengineering the Corporation: A Manifesto for Business Revolution.* Harper Collins, 1993.
11. 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.
12. V. Kavakli and P. Loucopoulos. Goal-Driven Business Process Analysis - Application in Electricity Deregulation. In *Proceedings of CAISE'98*, 1998.
13. M. Koubarakis and D. Plexousakis. Business Process Modeling and Design: AI Models and Methodology. In *Proceedings of IJCAI-99 Workshop on Intelligent Workflow and Process Management: the New Frontier for AI in Business*, 1999.
14. J. Lee. Goal-Based Process Analysis: A Method for Systematic Process Redesign. In *Proceedings of the Conference on Organizational Computing Systems (COOCS'94)*, 1994.
15. Y. Lesperance, T.G. Kelley, J. Mylopoulos, and E. Yu. Modeling dynamic domains with congolog. In *Proceedings of CAISE'99*, 1999.
16. 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/.
17. 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.
18. P. Loucopoulos and V. Karakostas. *System Requirements Engineering.* McGraw Hill, 1995.

19. 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.
20. 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.
21. J. Mylopoulos, L. Chung, and Nixon B.A. Representing and Using Non-Functional Requirements: A Process-Oriented Approach. *IEEE Transactions on Software Engineering*, 18(6):483–497, 1992.
22. 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.
23. M. A. Ould. *Business Processes: Modeling and Analysis for Re-engineering and Improvement*. Wiley, 1995.
24. 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.
25. D. Plexousakis. *On the efficient maintenance of temporal integrity in knowledge bases*. PhD thesis, Dept. of Computer Science, University of Toronto, 1996.
26. A. Rao. Modeling the service assurance process for Optus using GEM. Technical Report Technical Note 69, Australian Artificial Intelligence Institute, 1996.
27. 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.
28. R. Scherl and H. Levesque. The frame problem and knowledge producing actions. In *Proceedings of AAAI-93*, 1993.
29. 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.
30. A. van Lamsweerde, R. Darimont, and Massonet P. Goal-Directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learned. In *Proceedings of RE'95*, 1995.
31. 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.
32. 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.
33. E. Yu, J. Mylopoulos, and Y. Lesperance. AI Models for Business Process Reengineering. *IEEE Expert*, 11(4):16–23, 1996.