Pergamon

Information Systems Vol. 0, No. 0, pp. 1–22, 2001

????????/2001 $X.00 + 0.00

A FORMAL FRAMEWORK 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.
Email: manolis@ced.tuc.gr, URL: www.ced.tuc.gr/˜manolis,
[2]Dept. of Computer Science, University of Crete, 71305 Heraklion, Crete, Greece.
Email: dp@csd.uch.gr, URL: www.csd.uch.gr/˜dp

(Received .... ; in final revised form ....)

Abstract — We present a formal framework for enterprise and business process modeling. 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 constraints are maintained as a result of process execution.

Key words: enterprise modeling, business process modeling, situation calculus, ConGolog, methodology, formal verification

1. INTRODUCTION

The problem of *representing, analyzing* and *managing* knowledge about an enterprise 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 [16, 7, 48, 35, 17, 36, 15, 28, 19, 53, 22]. 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.

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.

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.

In this paper we advocate a formal approach to enterprise and business process modeling. 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

†AN EARLIER VERSION OF THIS PAPER APPEARED IN THE PROCEEDINGS OF CAISE*00, STOCKHOLM, SWEDEN, JUNE 5-9, 2000.

them. An advantage of formal models is that they can be verified mathematically, that it can be proved that they are self-consistent, and have or lack certain properties.

In summary, the original contributions of this paper are the following:

- We present a formal approach to enterprise and business process modeling. Our approach is built on *the situation calculus* (a knowledge representation formalism used in AI [33, 43]) and the established tradition of projects F^3 [2, 31] and EKD [20, 3]. An important feature of our model is how the use of situation calculus is combined with use of the concurrent logic programming language ConGolog [9]. ConGolog is used to capture operational knowledge about how actors behave when playing roles (e.g., when an actor in role "Postgraduate Secretary" is notified of an event of type "request for information", it performs action "reply to request".)
- We sketch the basic steps of a methodology which can be used by an enterprise for analysing and redesigning an existing business process, or developing a new one. The methodology starts with the objectives of the enterprise and produces a detailed formal specification of a business process which achieves these objectives. The formal specification is developed as a set of submodels that capture the business process from various viewpoints.

  Our methodology has a distinguishing powerful feature when compared with similar methodologies developed in other enterprise modeling projects e.g., EKD [3, 20]. In our framework a business analyst is able to *verify formally* that each role responsibility is fulfilled and each constraint is maintained as a result of process execution.

  We stress here that our verification techniques can be used as they are in similar methodologies such as EKD [3, 20] once formal models like ours are adopted. We view this as an important contribution of our work.

The rest of this paper is structured as follows. Section 2 introduces our approach to enterprise modeling and gives details of the formalism we will use. Sections 3, 4, 5 and 6 present the various parts of our enterprise model. Section 7 sketches the basic steps of our methodology while Section 8 concentrates on our process verification techniques. Finally, Section 9 discusses related work and Section 10 presents our conclusions.

2. FORMAL ENTERPRISE MODELING

Our approach to enterprise modeling follows the lead of F^3 [2, 31] and EKD [20, 3]. An *enterprise model* consists of five interconnected submodels that are used to describe *formally* different aspects of an organisation:

- *organisational submodel,* describing the actors in the enterprise, their roles, their responsibilities and their capabilities
- *objectives and goals submodel*, describing what the enterprise and its actors try to achieve
- *process submodel*, describing 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.

In the rest of this paper we will discuss how enterprise knowledge can be organised in terms of these submodels.

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 [33, 43]. 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 logical language $\mathcal{L}$ which is defined in the following way [11]. The symbols of $\mathcal{L}$ 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 $\mathcal{L}$ (constant, function and predicate symbols) will be defined step-by-step whenever a new modeling 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. We will also use the term first-order theory to refer to a set of sentences of $\mathcal{L}$ [11].

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.

One of the very appealing properties of 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 [2, 31, 20, 3] except by extending them so that they can become equivalent to first-order logic.

3. THE ORGANISATIONAL SUBMODEL

The main concepts of the organisational submodel are *actor* and *role*. An *actor* is an intentional entity, that is it has some idea of purpose that guides its actions. The concept of actor will be used to model people or groups of people or software/hardware systems in the context of the organisation we are modeling (e.g., an employee, a customer, a committee, an automated work-scheduling system 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 [37, 50, 10, 21]. Organisational roles can take many forms [37]: 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 or groups of actors.† Different actors can play different roles at different moments in 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 of the organisational submodel introduced above can be defined formally by introducing appropriate constructs of $\mathcal{L}$ and writing axioms that capture their semantics. We introduce unary predicates $Actor, HumanActor$, $AutomatedActor$ and $Role$, and binary predicate $PlaysRole$ with obvious meaning. The following axiom expresses the relation between actors, human actors and automated actors:

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

Example 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.

†This paper does not consider the concept of groups of actors in much depth e.g., we have no explicit notion of joint goals, joint processes etc. [45].

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 $\mathcal{L}$ can be introduced in the organisational submodel for DEPT:

$$HumanActor(John),\ HumanActor(Mary)$$

$$Role(Tutor),\ Role(Secretary)$$

$$PlaysRole(John, Tutor),\ PlaysRole(Mary, Secretary)$$

4. THE OBJECTIVES AND GOALS SUBMODEL

An *enterprise goal* is a desired state of affairs [12, 6, 50, 31, 24, 36, 53, 20]. 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 "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 [52, 50]. 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").

4.1. Organising Goals

Organisational goals can be *reduced* into alternative combinations of subgoals [6, 4, 50, 24, 2, 31, 20, 3] by using *AND/OR goal graphs* originally introduced in the area of problem solving [12]. 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 [30].

Goals can *conflict* with each other [6, 4, 51, 31, 46]. In our framework, goals conflict if they cannot be *satisfied* simultaneously given our knowledge about the enterprise [46]. Goals can also *influence* positively or negatively other goals. Such interactions between goals must be noted explicitly to facilitate goal-based reasoning (see Section 7) [34, 4, 51, 50, 31].

4.2. Defining Goals Formally

Organisational 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 [6].

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 2 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 $\mathcal{L}$:

$$(\forall a)(\forall e)(\forall x)(\forall s)(\forall s')$$

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

where predicates *Received* and *Answered* have 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. More details about the situation calculus and its machinery are given in Section 5. 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". In case goals are *temporal* (e.g., "enquiries are answered within a week") these can also be formalised in the situation calculus [18].

5. THE PROCESS SUBMODEL

A good 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" [5]. 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 [50] 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 [37]. The process submodel is also closely related with the objectives and goals submodel: processes are operationalisations of organisational goals [2, 1].

5.1. Primitive and Complex Actions

Our process submodel is built around the concepts of situation calculus [33, 43] and the concurrent logic programming language ConGolog [8]. 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 of the form $f(args)$ where f is a function symbol and $args$ is a list of first-order terms (e.g., $SendOfferLetter(a, app)$).

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 ϕ_1
  effect ϕ_2
endAction

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

Example 3 The following expression defines the action of forwarding an application *app* by actor $a1$ to actor $a2$:

> **action** $ForwardApp(a1, a2, app)$
> **precondition** $Has(a1, app)$
> **effect** $Has(a2, app) \wedge \neg Has(a1, app)$
> **endAction**

Our framework permits the recursive definition of *complex actions* (simply actions from now on) by adopting the exact syntax and semantics of ConGolog [8, 9]:

- *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 $\alpha_1 \mid \alpha_2$ 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 $\vec{x}$ is a list of variables, ϕ is a formula of $\mathcal{L}$ 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 $\beta(\vec{x})$.

An example of a complex action (procedure) is given in Figure 1. It is called *ProcessApp* (for "process application") and uses primitive action *ForwardApp* defined in Example 3. Other examples of complex actions are given in Section 7. The interested reader can also consult [8, 25] for more on ConGolog and its uses.

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

Fig. 1: A complex action in ConGolog

5.2. 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 such as $SendMsg$ in Figure 1) [44, 26]. It is known that knowledge-producing actions can be defined in the situation calculus formalism [44, 26].

Finally, actions can be *exogenous*. This concept corresponds to the notion of external events 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 [8].

5.3. 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 [5].

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 $resps$
  $PrimitiveActionDefs$
  $ProcedureDefs$
endRole

where id is a role identifier, $resps$ is a list of goals (separated by commas), $PrimitiveActionDefs$ is one or more primitive action specifications, and $ProcedureDefs$ is one more ConGolog procedures. The responsibility statement declares that role id is responsible for achieving the goals in list $resps$.

role $Tutor$
responsibility ...

action $ForwardApp(a_1, a_2, app)$
  precondition $Has(a_1, app)$
  effect $Has(a_2, app) \wedge \neg Has(a_1, app)$
endAction

action $SendMsg(sender, recipient, msg)$
  ...
endAction

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

endRole

Fig. 2: Role Tutor

As an example, consider the definition of role $Tutor$ given in Figure 2 (the responsibility specification has been omitted for brevity). The example uses the procedure $ProcessApp$ of Figure 1 that has now been renamed using the reserved keyword *main*. In the definition of each role, the business analyst has to specify a ConGolog procedure called *main*, which gives the detailed dynamics of the role. Of course, *main* can invoke other local procedures.

Notice symbol $self$ which is a pseudo-variable that enables us to attach responsibilities to roles in a structured way. Whenever $self$ appears in a formula of $\mathcal{L}$, it is assumed to range over all actors playing the role inside which the variable appears.

Each actor playing the role $Tutor$ can perform the causal action $ForwardApp$ (defined in Example 3) 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 [27]. Role $Tutor$ also needs to watch for condition $Has(self, app)$ where $self$ denotes the actor playing the role $Tutor$ and *app* is an application. More examples of role definitions are given in Section 7.

6. THE CONCEPTS AND CONSTRAINTS SUBMODELS

The *concepts* submodel contains information about non-intentional aspects of 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 enterprise a predicate $Has(a, app)$ might be used to denote that actor a 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) [40]. An example of a static constraint is the following sentence of $\mathcal{L}$

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

which intends to enforce the policy that no applicant can be both accepted and rejected. This constraint is regarded as a static constraint since it refers to any single situation in which an applicant is either accepted or rejected. Dynamic constraints on the other hand refer to multiple states and can also be expressed in the situation calculus. Sometimes dynamic constraints are expressed in some variant of temporal logic [40]. An example of such a constraint is presented in the Appendix, where an example of process specification and verification for a simple elevator controller is given.

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

7. A GOAL-ORIENTED METHODOLOGY FOR BUSINESS PROCESS DESIGN

This section and the next one outline the basic steps of a methodology which is based on the enterprise modeling framework developed in the previous sections. The methodology is intended to guide an enterprise wishing 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 should be implemented to achieve 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.

Although this paper only concentrates on using the proposed methodology for the specification of a new business process, the methodology can also be used (with some changes) for modeling, documenting formally and redesigning an existing process.

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.

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.

In this paper we do not intend to present the methodology and its application in detail (for this the interested reader should go to [23] where more details and a substantial example are given). We will only discuss some of the issues involved in Steps 1, 2 and 3, and then concentrate our attention to Steps 4 and 5, where our approach significantly improves on related methodologies e.g., EKD [3, 20] or GEM [42]. This paper will also avoid to cover certain other issues related to applying this methodology to an industrial application (i.e., who must participate in the modeling team, what techniques can be used for eliciting the required knowledge etc.). For these problems one can utilise the guidelines provided by similar frameworks and tested in industrial projects (e.g., EKD [3, 20]). We stress here that all of our ideas can be used as they are in similar methodologies such as EKD once formal models such as the ones we discussed in the previous section are adopted. We view this as an important contribution of our work.

7.1. 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 modeled (and possibly other enterprises in the same industry sector), and so on [3]. 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 [6]. 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 [51, 34, 6, 47] and related goal-oriented enterprise modeling frameworks [31, 20, 3].

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 [47].

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 formalized as has already been shown in Section 4.2.

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 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 [36].

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 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 handling promptly 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 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 formalized as follows:

$$(\forall a)(\forall e)(\forall x)(\forall s)(\forall s')$$

$$(Actor(a) \wedge Enquiry(e) \wedge Action(x) \wedge Situation(s) \wedge Situation(s') \wedge$$

$$PlaysRole(a, Secretary) \wedge Received(e, a, s) \wedge s' = Do(x, s) \supset Answered(e, a, s'))$$

role *Secretary*

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

action $AnswerEnquiry(a, e)$
...
endAction

action $ForwardApp(a_1, a_2, app)$
...
endAction

action $SendOfferLetter(a, app)$
...
endAction

action $SendRejectionLetter(a, app)$
...
endAction

action $SenseMsg$
...
endAction

proc *main*
$\langle e : \ Enquiry(e) \wedge Received(e, self) \rightarrow AnswerEnquiry(self, e)\rangle$
$\gg$
$\langle app : \ Application(app) \wedge Has(self, app) \rightarrow$
for $a : Actor(a) \wedge PlaysRole(a, Tutor)$ **do**
$Forward(self, a, app)$
endFor$\rangle$
$\gg$
while True do
$SenseMsg$;
if $\neg Empty(MsgQ(self))$ **then**
if $First(MsgQ(self)) = (lect, \ulcorner INFORM(WantsToSupervise(lect, app))\urcorner)$
then $SendOfferLetter(self, app)$
else if $First(MsgQ(self)) = (tut, \ulcorner INFORM(Unacceptable(app))\urcorner)$ **then**
$SendRejectionLetter(self, app)$
endIf
endIf
endWhile
endProc

endRole

Fig. 3: Role Secretary

role $Faculty$
responsibility ...

action $SendMsg(sender, recipient, msg)$
...
endAction

proc $Eval(self, app)$
if $GoodUniversity(Univ(app)) \wedge AvgMark(app) > MinMark(self) \wedge NoOfStudents(self) < MaxNoOfStudents(self)$ **then**
for $a : Actor(a) \wedge PlaysRole(a, Secretary)$ **do**
$SendMsg(self, a, \ulcorner INFORM(WantsToSupervise(self, app))\urcorner)$
endFor
endIf
endProc

proc $main$
$\langle app : Application(app) \wedge Has(self, app) \rightarrow Eval(self, app)\rangle$
endProc

endRole

Fig. 4: Role Faculty

7.2. Defining Roles Using ConGolog

The next step in our methodology is to specify the details of each role by identifying the *primitive actions* that are available to it, the *conditions* to be monitored and the *interactions* with other roles. In many cases this will be done in parallel with the last stages of goal reduction and responsibility assignment. For example, given the responsibilities assigned to role $Secretary$, each actor playing this role needs to be capable of carrying out primitive actions $AnswerEnquiry, ForwardApp, SendOfferLetter$, $SendRejectionLetter$ as well as various communicative actions. Similar considerations for roles $Tutor$ and $Faculty$ lead us to define these roles as in Figures 2, 3 and 4.

The ConGolog code for these roles should be easy to understand but the following comments are in order. First, notice that in the interest of brevity we have omitted specifying explicitly the responsibilities assigned to each role. Only role *Secretary* has an explicitly specified responsibility which, according to the semantics of pseudovariable $self$, is equivalent to the formalisation of goal G_1 given in Section 7.1.

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 than the others because a message queue (in the spirit of [27]) 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 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 5.

At the end of this step, we have a complete formal specification of our enterprise and its processes. Before we go on to the next step, let us comment on the suitability of ConGolog as a modeling language. In the processes we have modeled 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.

8. FORMAL VERIFICATION

The final step in our methodology is to *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 [39, 40], which are based on a systematic solution to the frame and ramification problems [43]. 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.

8.1. Constraint Verification

We now present the details of our method for constraint verification. The most important concept in our analysis of constraint verification and responsibility fulfillment is the concept of *ramification of a constraint or a goal* [13].

Definition 1 Let ϕ be a sentence expressed in our first-order language $\mathcal{L}$. Also, let Th be a set of sentences in $\mathcal{L}$ formalising a world state and G a goal. The sentence ϕ will be called a *ramification* of goal G and theory Th iff $Th \cup \{ G \} \models \phi$.

Example 4 Consider our imaginary Computer Science department and the constraint that expresses that no applicant can be both accepted and rejected. This constraint needs to be maintained at all times and is expressed by the formula

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

as we saw in Section 6.

If sentence $Accepted(p)$ holds, as will be the case with the execution of an acceptance action, then the sentence $\neg Rejected(p)$ is a ramification of the above constraint and any consistent first-order theory including the sentence $Accepted(p)$. Similarly, a rejection action would have $\neg Accepted(p)$ as a ramification. We will return to this example in the sequel.

It is easy to verify the following properties of ramifications [13, 40]:

- If a ramification of a constraint/goal is known to be unsatisfiable, then the constraint/goal is unsatisfiable.
- Constraint/goal ramifications can reduce the search space for constraint/goal satisfaction.
- Transformations may be applicable to constraint/goal ramifications but not to the constraint/goals themselves.

Hence, if there exists a systematic way to generate ramifications from a set of constraints/goals, then the derived ramifications can be used for making the task of meeting the constraints/goals simpler. Our method exploits this observation: it derives ramifications of constraints/goals and action preconditions and effects, and uses them to strengthen the action specifications [39, 40]. For the case of constraints our method will be explained below; the case of goals is covered in Section 8.2.

Note that generating ramifications may require an arbitrary amount of inferencing. Thus from the semi-decidability of first-order entailment, it follows that the problem of finding ramifications is, in its generality, intractable. Tractability can be achieved by restricting the class of constraints/goals for which ramifications are sought. For instance, the task is tractable for the case of conjunctions of atomic formulas [13]. Our previous work [39, 40] depicts the applicability of the generation of ramifications to an extension of the class of binary constraints in prenex normal form. Specifically, constraints are of the form:

$$(\forall x_1,\ldots,x_k),(\forall t_1,t_2)\ \phi(x_1,\ldots,x_k,t_1,t_2)\ \vee\ (\neg)p_1(x_1,\ldots,x_k,t_1)\ \vee\ (\neg)p_2(x_1,\ldots,x_k,t_2)$$

where, p_1, p_2 are $(k+1)$-ary predicates, intensional or extensional, t_1, t_2 are temporal variables and $\phi(x_1,\ldots,x_k,t_1,t_2)$ is a formula in which variables $x_1,\ldots,x_k,t_1,t_2$ occur free, if at all, and does not mention any predicate other than evaluable predicates. This class of constraints is an extension of the class of *binary constraints* of [38], since it allows the temporal variables to occur in evaluable predicates. It includes static constraints and transition constraints, i.e., constraints referring to two consecutive states of a knowledge base, but not general dynamic constraints.

Furthermore, not all derivable ramifications may be useful for simplifying the task of verifying a constraint/goal. For that, the generator may be guided to derive only "useful" ramifications if provided with appropriate input formulas. In fact, the solution to the ramification problem that we sketch here generates exactly those ramifications that are needed for proving that the specifications will not violate any constraints/goals. The systematic solution presented here was initially proposed in [29] and [38], and was extended in [39, 40, 41].

Example 5 Consider the specification of the action $SendOfferLetter$ shown below:

action $SendOfferLetter(a, app)$
  precondition $(\exists\ lect)\ WantsToSupervise(lect, app)$
  effect $Accepted(app)$
endAction

This action is available to role $Secretary$ (see Figure 3). 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*.

Assume that we wish to enforce the policy that no applicant can be both accepted and rejected (see Example 4). It is evident that the action specification given above does not exclude a situation in which both $Accepted(app)$ and $Rejected(app)$ are satisfied (where *app* is the second parameter of action $SendOfferLetter$). 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 $SendOfferLetter$.

To make this example more concrete, we elaborate on the idea of the generation of ramifications in more detail. The process of strengthening postconditions seeks for conditions that, if conjoined with the given process postcondition, would guarantee the preservation of the constraint in the state resulting from the execution of the process. To derive this condition we reason as follows:

Consider the negation of the constraint instantiated in the state resulting from the action execution and parameterized with the action arguments. Let I' denote this sentence of $\mathcal{L}$. Then

$$I' \equiv Accepted(app)\ \supset\ \neg Rejected(app)$$

We proceed by negating this sentence and conjoining it with the process postcondition (effect), resulting in the sentence

$$Accepted(app)\ \wedge\ Rejected(app)\ \wedge Accepted(app)$$

which after applying the absorption rule for common terms is equivalent to

$$Accepted(app)\ \wedge\ Rejected(app).$$

This sentence can be further simplified by substituting the term $Accepted(app)$ with the Boolean constant $True$, since the effect of the process execution is to make the fluent *Accepted* true. Hence, the resulting sentence is $Rejected(app)$.

This sentence is a logical consequence, i.e., a ramification, of the negated constraint and the process postcondition. Consequently, its negation is a condition sufficient for the satisfaction of the constraint, provided that the postcondition is met. Hence, the required strengthening of the initial postcondition is expressed by the sentence $\neg Rejected(app)$.

As a result, the action specification should be strengthened as follows:

action $SendOfferLetter(a, app)$
  precondition $(\exists\ lect)\ WantsToSupervise(lect, app)$
  effect $Accepted(app)\ \wedge \neg Rejected(app)$
endAction

Albeit short and simple, the above example conveys the idea behind the derivation of ramifications for strengthening action specifications. A more detailed example is given in the Appendix. The reader can also find more complex examples and other details in [39, 40]. Apart from simple actions, our verification ideas also apply to processes composed of action sequencing. We are currently investigating the applicability of verification to processes composed of the remaining complex actions of ConGolog.

Before we turn to our next topic, a number of remarks are in order. First, the process of strengthening process specifications that was exhibited by the above example, relies on the assumption that no inconsistency is present when an action or process execution takes place. This permits the verification of processes at process specification time. Second, ramifications can be generated systematically from the specification of processes as shown in [39, 40]. Third, the results presented therein show that the process is applicable to the cases where multiple constraints need to be preserved by a process and the case where multiple processes need to observe the same constraints. Fourth, strengthenings are applied to the postcondition rather than the precondition of processes. One could argue, that the same strengthenings could be applied to the process precondition so as not to allow the execution of processes that do not guarantee the preservation of constraints. This is true for the case of static constraints, but does not apply to the case of dynamic constraints. In the latter case, there may very well be cases in which, the required strengthening is a temporal sentence which refers to future states. Thus, it would not be meaningful to conjoin with the process precondition a sentence that refers to a not yet achieved stated. Last, but not least, specification strengthening has been shown to be sound [40].

8.2. Responsibility Verification

Let us now turn to the problem of verifying the fulfillment of responsibilities assigned to roles. Consider, for instance, the goal assigned as responsibility to the role *Secretary* (see Figure 3). This goal states the requirement that enquiries are answered by the Postgraduate Secretary as soon as they are received and is formalized equivalently as follows:[†]

$$(\forall a)(\forall e)(\forall x)(\forall s)(\forall s')$$

$$(Actor(a) \wedge Enquiry(e) \wedge Action(x) \wedge Situation(s) \wedge Situation(s') \wedge$$

$$PlaysRole(a, Secretary) \wedge Received(e, a, s) \wedge s' = Do(x, s) \supset Answered(e, a, s'))$$

The task of verifying that responsibilities are fulfilled amounts to checking whether any action specification may result in a state that will prohibit the goal from being satisfied. Assume now that the action $AnswerEnquiry$ is available to $Secretary$ and is specified as follows:

action $AnswerEnquiry(a, e)$
  precondition $Actor(a) \wedge PlaysRole(a, Secretary) \wedge$
    $Enquiry(e) \wedge Received(e, a)$
  effect $Answered(e, a)$
endAction

Intuitively, as given, the action specification is sufficient for guaranteeing that the assigned responsibility will be fulfilled. However, one needs to have a systematic way of discovering that the specification is sufficient in this respect. The verification of the sufficiency of the specification proceeds as follows:

[†]Pseudovariable $self$ has been substituted according to its semantics.

1. The responsibility statement is instantiated with respect to the particular actor that is responsible for carrying out the action in any state where the action's precondition is satisfied. The result of this instantiation is a simplified statement of the responsibility, in which variables corresponding to action parameters have been substituted for the parameter values, and fluents that are true by virtue of the precondition being satisfied have been substituted by the Boolean constant $True$. For instance, the fluent $PlaysRole(a, Secretary)$ is substituted by $True$ for all actions that are enacted by an actor playing the role of $Secretary$, including the one in our example. Furthermore, the statement is instantiated to the state in which the precondition is evaluated. The result is the statement:

$$(\forall s')(Situation(s') \wedge s' = Do(AnswerEnquiry(a,e), s) \supset Answered(e,a,s'))$$

2. To evaluate the effect of the action's specification in the fulfillment of the responsibility in the resulting state, the responsibility statement is subsequently instantiated in the resulting state, negated and conjoined to the action effect. Note that the quantified situation variable can now be eliminated. Thus we have:

$$\neg(True \supset Answered(e,a)) \wedge Answered(e,a)$$

3. The result is a ramification of the action specification and the responsibility statement whose negation is the condition that needs to be conjoined with the action effect in order to guarantee that the responsibility will be fulfilled. In this particular case, it is easy to see that the result of step 2 is the Boolean constant $False$. Hence, its negation ($True$) must be conjoined to the effect of the action.

This example shows that the derivation of certain ramifications can not only designate the appropriate strengthenings of action specifications but also reveal that specifications are sufficient as given. In particular, it has been shown in [39, 40] that, if the Boolean constant $False$ is derived as a ramification of a goal and an action specification, then the goal is maintained in the state resulting after the action's execution provided that its effect is met.

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. Hence, tha class of constraints for which the verification method can be applied encompasses static and transition constraints. 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 [26]. The adaptation of these ideas in our analysis and verification techniques is an ongoing effort.

Let us close this section by emphasising 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 [40].

9. DISCUSSION

The first paper to propose situation calculus and ConGolog (more precisely its earlier version Golog) for business process modeling was [39]. Since then similar ideas have appeared in [40, 53, 25]. 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 [14]. However, TOVE concentrates mostly on enterprise ontologies and uses situation calculus for their formalization. To the best of our knowledge, TOVE does not concentrate on process modeling and has not proposed anything corresponding to our modeling, design and verification methods.

The concepts of goals, actors and roles also appear prominently in the i^* framework [50] where the need for *intentional concepts* in enterprise modeling (and requirements modeling) is emphasized. 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 can undoubtedly benefit by incorporating some features of the i^* framework (dependencies, strategic rationale etc.).

There is a strong connection of our work to goal-oriented methodologies for requirements engineering especially KAOS [6, 47]. The common elements of our research and KAOS is the emphasis on formality, and the adoption of intentional concepts such as goal and actor (agent in KAOS). But our work also differs from KAOS by concentrating on business process modeling (thus the new concepts of role and process) and using situation calculus and ConGolog instead of first-order logic with temporal operators.

As we have already said, our work follows the lead of the enterprise modeling frameworks of F^3 [2, 31] and its successor EKD [3]. 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 [36] 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 modeling, 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 [24]. 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 [42]. 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 [42] gives only a short informal description of the models and methodology. Methodologies for developing multi-agent systems based on concepts similar to the ones in GEM appear in [21, 49]. It is an open question whether our work could be used as a basis for the design of better methodologies for requirements modeling, analysis and design of multi-agent systems ([21, 49] concentrate only on analysis and design).

The vast majority of business process modeling 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 [25]. Strengthening of specifications using inference rules has also been proposed in [6] 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 [47] 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 [25]. The tool also includes a module for progressing an initial situation, allowing it to simulate the execution of long-running processes.

10. 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 modeling 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 modeling domain by projects such as KAOS [6]. In this paper we have demonstrated that formal methods can be valuable in the domain of business modeling 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 8 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 modeling:

1. 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.

2. The use of complex mathematical notation may put off the average manager, business analyst or user.

3. Special skills are required (in our case, familiarity with situation calculus and ConGolog).

The first criticism is not really a criticism of "formal" enterprise modeling but rather of any kind of enterprise modeling. There is a price to pay for undertaking an enterprise modeling 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 [36] 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 modeling techniques and methodology. In parallel we would like to apply our techniques to the modeling 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 [39, 40] 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 work of the first author was partially supported by BT Labs, Ipswich, U.K. through a Short Term Research Fellowship. 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. Paul Kearney in particular provided many comments on previous versions of this paper; these comments have all found their way into this paper.

The work of the second author was partially supported by the Univ. of South Florida through a Research and Creative Scholarship Grant.

We would also like to 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

[1] A.I. Anton, M.W. McCracken, and C. Potts. Goal decomposition and scenario analysis in business process reengineering. In *Proceedings of CAISE'94*, pp. 94–104 (1994).

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

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

[4] L. Chung. *Represening and Using Non-Functional Requirements: A Process-Oriented Approach.* PhD thesis, Dept. of Computer Science, University of Toronto (1993).

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

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

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

[8] 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*, pp. 1221–1226 (1997).

[9] G. De Giacomo, Y. Lesperance, and H.J. Levesque. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, **121**(1-2) (2000).

[10] 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*, pp. 87–106. Academic Press (1994).

[11] H.B. Enderton. *A Mathematical Introduction to Logic.* Academic Press (1972).

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

[13] J. Finger. Exploiting Constraints in Design Synthesis. Technical Report STAN-CS-88-1204, Stanford University (1988).

[14] M.S. Fox and M. Gruninger. Enterprise Modelling. *The AI Magazine*, pp. 109–121 (1998).

[15] 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).

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

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

[18] Pinto J. and R. Reiter. Adding a Time Line to the Situation Calculus. In *The Second Symposium on Logical Formalizations of Commonsense Reasoning*, pp. 172–177 (1993).

[19] 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).

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

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

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

[23] 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).

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

[25] Y. Lesperance, T.G. Kelley, J. Mylopoulos, and E. Yu. Modeling dynamic domains with congolog. In *Proceedings of CAISE'99*, pp. 365–380 (1999).

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

[27] 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).

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

[29] F. Lin and R. Reiter. State Constraints Revisited. In *Proceedings of the Symposium on Logical Formalizations of Commonsense Reasoning*, pp. 114–121 (1992).

[30] P. Loucopoulos and V. Karakostas. *System Requirements Engineering.* McGraw Hill (1995).

[31] 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).

[32] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems.* Springer Verlag, Vol. 1: Specification (1991).

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

[34] 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).

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

[36] 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).

[37] M. A. Ould. *Business Processes: Modeling and Analysis for Re-engineering and Improvement.* Wiley (1995).

[38] J. Pinto. *Temporal Reasoning in the Situation Calculus.* PhD thesis, Department of Computer Science, University of Toronto (1994).

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

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

[41] D. Plexousakis and J. Mylopoulos. Accommodating Integrity Constraints During Database Design. In *Proceedings of the International Conference on Extending Database Technology*, pp. 497–513, Avignon, France (1996).

[42] A. Rao. Modeling the service assurance process for Optus using GEM. Technical Report Technical Note 69, Australian Artificial Intelligence Institute (1996).

[43] 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*, pp. 359–380. Academic Press (1991).

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

[45] G. Tidhar. Team-oriented programming: Social structures. Technical Report Technical Note 47, Australian Artificial Intelligence Institute (1993).

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

[47] 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).

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

[49] M. Wooldridge, N.R. Jennings, and D. Kinny. The Gaia Methodology for Agent-Oriented Analysis and Design. *Journal of Autonomous Agents and Multi-Agent Systems*, **3**(3):285–312 (2000).

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

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

[52] 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*, pp. 234–243, Hawaii (1994).

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

A. APPENDIX: A COMPLEX EXAMPLE OF CONSTRAINT VERIFICATION

In this section, we would like to give a more complex example of constraint verification to complement the examples given in Section 8. We depict the application of specification strengthening to a simple example of an elevator controller borrowed from [6]. Constraints here are expressed in a variant of first-order temporal logic [32]. The example at the same time aims at showing the applicability of the proposed analysis techniques to the case where specifications are given in a temporal logic. Hence, we will focus on postcondition strengthening rather than giving a fairly complete specification of the domain in question.

Let us assume that the constraint expressing the requirement that doors be closed while the elevator is moving ensures the abstract goal of safe transportation in the elevator system. The constraint is formally expressed as the following many-sorted temporal formula, where the operators $\bigcirc$ and $\odot$ mean "in then next state" and "in the previous state" respectively:

$$\begin{aligned} I \;\equiv\; & \forall l/Lift\ d/Door\ f, f'/Floor\ \neg PartOf(d,l)\ \Rightarrow \\ & (LiftAt(l,f)\ \wedge\ \bigcirc LiftAt(l,f')\ \wedge\ (f \neq f')\ \Rightarrow \\ & ((d.state = \text{"}closed\text{"})\ \wedge\ \bigcirc(d.state = \text{"}closed\text{"}))) \end{aligned}$$

For simplicity, let us assume that the only actions relevant to the constraints are the actions $GotoFloor$ and $OpenDoors$, whose specification is given in Figure 5. We need to add the axiom

$$\forall d/Door\ (d.state = \text{"}open\text{"})\ \Rightarrow\ \neg(d.state = \text{"}closed\text{"})$$

in order to be able to reason about the state of elevator doors.

We first consider the action $GotoFloor$ for the derivation of ramifications. The first step is to instantiate the constraint I in the state after the action's execution. This amounts to parameterizing the universally quantified variables corresponding to the parameters of the action and to replacing every occurrence of $\bigcirc P$ by P and every occurrence of P with $\odot P$ for every predicate P. Negating the instantiated constraint and conjoining it with the action's postcondition yields

$$\begin{aligned} \neg I'\ \wedge\ Post\ \equiv\ & \odot PartOf(d,l)\ \wedge\ \odot LiftAt(l,f)\ \wedge\ \quad LiftAt(l,f')\ \wedge\ (f \neq f')\ \wedge \\ & (\neg\odot(d.state = \text{"}closed\text{"})\ \vee\ \quad \neg(d.state = \text{"}closed\text{"})) \end{aligned}$$

By exploiting the facts that the predicates occurring positively in the action's postcondition must evaluate to $True$ in the state after the action's execution and that the precondition must have been satisfied, $\neg I'\ \wedge\ Post$ implies the formula

$$\odot PartOf(d,l)\ \wedge\ (\neg\odot(d.state = \text{"}closed\text{"})\ \vee\ \neg(d.state = \text{"}closed\text{"}))$$

We now exploit the assumption that the constraint was known to be satisfied in the state prior to the action's execution. This means that either the antecedent $\odot PartOf(d,l)$ of I is $False$ or, both the antecedent of I and its consequent are $True$. In the former case we derive $False$ as a ramification and we can conclude that the constraint remain unaffected by the transaction. In the latter case, $\odot PartOf(d,l)$ can be substituted by $True$ and the derived formula is $\neg R\ \equiv\ \neg\odot(d.state = \text{"}closed\text{"})\ \vee\ \neg(d.state = \text{"}closed\text{"}))$, which yield the ramification $R \equiv\ \odot(d.state = \text{"}closed\text{"})\ \wedge\ (d.state = \text{"}closed\text{"})$. Intuitively, this means that both in the state before and in the state after the action execution, the state of the elevator door must be "closed".

Similarly, we can derive ramifications with respect to the action $OpenDoors$. In this case, the ramification derived is expressed by the formula

$$R\ \equiv\ \exists f, f'/Floor\ (\odot LiftAt(l,f)\ \wedge\ LiftAt(l,f'))\ \Rightarrow\ \neg(f \neq f')$$

which means that both before and after the action execution the elevator must be at the same floor. It is easy to verify that if we strengthen the action specifications by conjoining their postconditions with the derived ramifications, then the constraint is maintained by any implementation of the actions that meet the new specifications. Figure 6 shows the strengthened specification of the actions $GotoFloor$ and $OpenDoors$.

Action *GotoFloor* (l, f, f')
Precondition: $LiftAt(l, f)$
Postcondition: $LiftAt(l, f') \ \wedge \ (f \neq f')$
end

Action *OpenDoors(l,d)*
Precondition: $PartOf(d, l) \ \wedge \ (d.state = "closed")$
Postcondition: $(d.state = "open")$
end

Fig. 5: Specification of an Elevator Controller

Action *GotoFloor* (l, f, f')
Precondition: $LiftAt(l, f) \ \wedge \ (d.state = "closed")$
Postcondition:
$LiftAt(l, f') \ \wedge \ (f \neq f') \ \wedge (d.state = "closed")$
end

Action *OpenDoors(l,d)*
Precondition: $PartOf(d, l) \ \wedge \ (d.state = "closed")$
Postcondition:
$PartOf(d,l) \ \wedge \ (d.state = "open") \ \wedge$
$\exists f, f'/Floor \ (\odot LiftAt(l, f) \ \wedge \ LiftAt(l, f')) \ \Rightarrow$
$\neg(f \neq f')$
end

Fig. 6: Strengthened Specification of an Elevator Controller