Extending Agent-oriented Requirements with Declarative Business Processes: a Computational Logic-based Approach*

Volha Bryl[1], Marco Montali[2], Paola Mello[2], Paolo Torroni[2], and Nicola Zannone[1]

[1] DIT, University of Trento – Via Sommarive 14, 38050 Povo (TN), Italy
{bryl | zannone}@dit.unitn.it
[2] DEIS, University of Bologna – V.le Risorgimento 2, 40136 Bologna, Italy
{pmello | mmontali | ptorroni} @deis.unibo.it

Abstract. The analysis of business requirements and the specification of business processes are fundamental for the development of information system. The focus of this paper is on the combination of these two phases, that is, on linking the business goals and requirements to the business process model. To this end, we propose to extend the Tropos framework, which is used to model system and business requirements, with declarative business process-oriented constructs, inspired by DecSerFlow and ConDec languages. We also show how the proposed framework can be mapped into $\mathcal{S}$CIFF, a computational logic-based framework, for properties and conformance verification.

1 Introduction

Modeling and analyzing requirements of information systems in terms of agents and their goals has been a topic of a considerable interest during the last decades [1]. Tropos [2] is one of the existing approaches to agent-oriented software engineering, which emphasizes the concepts of agent and goal from the early phases of the system development. Agent-oriented requirements analysis helps to understand the organizational setting in which a system will operate, to model stakeholders' strategic interests and thus to represent the rationale beyond the introduction of the system and the design choices made.

The next step to be made after modeling and analyzing early system requirements is defining the corresponding business process. As it was pointed out in [3], linking the "strategic" business goals and requirements to the business process model is an utmost important issue. In this setting, many problems arise from organizational theory and strategic management perspectives due to limits on particular resources (e.g., cost, time, etc.). Business strategies have a fundamental impact on the structure of enterprises leading to efficiency in coordination

* This work has been partially funded by EU SENSORIA and SERENITY projects, by the MIUR-FIRB TOCAI project, by the MIUR PRIN 2005-011293 project, and by the PAT MOSTRO project.

and cooperation within economic activities. However, one of the drawbacks of Tropos, as well as many other agent-oriented modeling approaches, is in that the passage from a requirements model to a business process model is not clearly defined. For example, Tropos does not allow modeling temporal and data constraints between the tasks an agent is assigned to, which is essential when specifying the partial ordering between activities of a business process. Ability to represent start and completion times, triggering events, deadlines, etc. is strictly necessary when defining a business process model.

To support the development, optimization, and management of enterprise day-by-day activities, we propose a framework for facilitating the interaction between requirements analysis and the definition of business processes. In particular, we propose to extend Tropos with declarative business process-oriented constructs, inspired by two novel graphical languages, namely DecSerFlow [4] and ConDec [5]. In this way, the typical goal-oriented approach of Tropos agents is augmented with a high-level reactive, process-oriented dimension. We refer to the extended framework as to $\mathcal{B}$-Tropos. Furthermore, we show how both these complementary aspects could be mapped into a unique underlying formalism, called $\mathcal{S}$CIFF [6], a computational logic-based framework for the specification and verification of interaction protocols in an open multi-agent setting. Thanks to this mapping it is possible to exploit the possibility of directly using the $\mathcal{S}$CIFF specification to implement logic-based agents [7], as well as to perform different kinds of verification, such as properties verification [8] or conformance verification of a given execution trace w.r.t. the model it should follow [6]. To make the discussion more concrete, the proposed approach is applied to modeling and analyzing an intra-enterprise organizational model, focusing on the coordination of economic activities among different units of an enterprise collaborating to produce a specific product.

The structure of the paper is as follows. Section 2 briefly presents the Tropos methodology. Section 3 describes our process-oriented extensions of Tropos. The $\mathcal{S}$CIFF framework is presented in Section 4, whereas Section 5 defines the mapping of $\mathcal{B}$-Tropos concepts to $\mathcal{S}$CIFF specifications. The paper ends with the overview of related work and conclusive remarks in Sections 6 and 7, respectively.

2 The Tropos Methodology

Tropos [2] is an agent-oriented software engineering methodology tailored to describe and analyze socio-technical systems along the whole development process from requirements analysis up to implementation. One of its main advantages is the importance given to early requirements analysis. This allows one to capture *why* a piece of software is developed, behind the *what* or the *how*.

The methodology is founded on models that use the concepts of actor (i.e., agent and role), goal, task, resource, and social dependency. An *actor* is an active entity that has strategic goals and performs actions to achieve them. A *goal* represents a strategic interest of an actor. A *task* represents a particular course of actions that produces a desired effect. A *resource* represents a physical or an informational entity without intentionality. A *dependency* between two actors indicates that one actor depends on another to achieve some goal, execute some task, or deliver some resource. The former actor is called *depender*, while the latter is called *dependee*. The object around which the dependency centres is called *dependum*. In the graphical representation, actors are represented as circles; goals, plans and resources are respectively represented as ovals, hexagons and rectangles; and dependencies have the form $depender \rightarrow dependum \rightarrow dependee$.

From a methodological perspective, Tropos is based on the idea of building a model of the system that is incrementally refined and extended. Specifically, goal analysis consists of refining goals and eliciting new social relationships among actors. Goal analysis is conducted from the perspective of single actors using three reasoning techniques: means-end analysis, AND/OR decomposition, and contribution analysis. *Means-end analysis* aims at identifying tasks to be executed in order to achieve a goal. Means-end relations are graphically represented as arrows without any label on them. *AND/OR decomposition* combines AND and OR refinements of a root goal or a root task into subparts. In essence, AND-decomposition is used to define the process for achieving a goal or a task, whereas OR-decomposition defines alternatives for achieving a goal or executing a task. *Contribution analysis* identifies the impact of the achievement of goals and tasks over the achievement of other goals and tasks. This impact can be positive or negative and is graphically represented as edges labeled with "+" and "−", respectively.

Example 1. Figure 1 presents the requirements model of a product development process. In this scenario, different divisions of a company have to cooperate in order to produce a specific product. The Customer Care division is responsible for deploying products to customers, which refines it into subgoals manufacture product, for which it depends on the Manufacturing division, and present product, for which it depends on the Sales division. In turn, Manufacturing decomposes the appointed goal into subgoals define solution for product, for which it depends on the Research & Development (R&D) division, and make product that it achieves through task execute production line. To achieve goal define solution for product, R&D has to achieve goals provide solution, which it achieves through task design solution, evaluate solution, and deploy solution, which it achieves through task define production plan. The evaluation of the solution is performed in terms of costs and available resources. R&D executes task assess costs, which consists in calculating bill of quantities and evaluating bill of quantities, to achieve goal evaluate costs, and depends on the Warehouse for evaluate available resources. The Warehouse either can query the databases to find available resources or ask the Purchases division to buy resources from external Supplier. In the latter case, Purchases searches in company's databases for possible Suppliers and selects the one who provides the best offer.

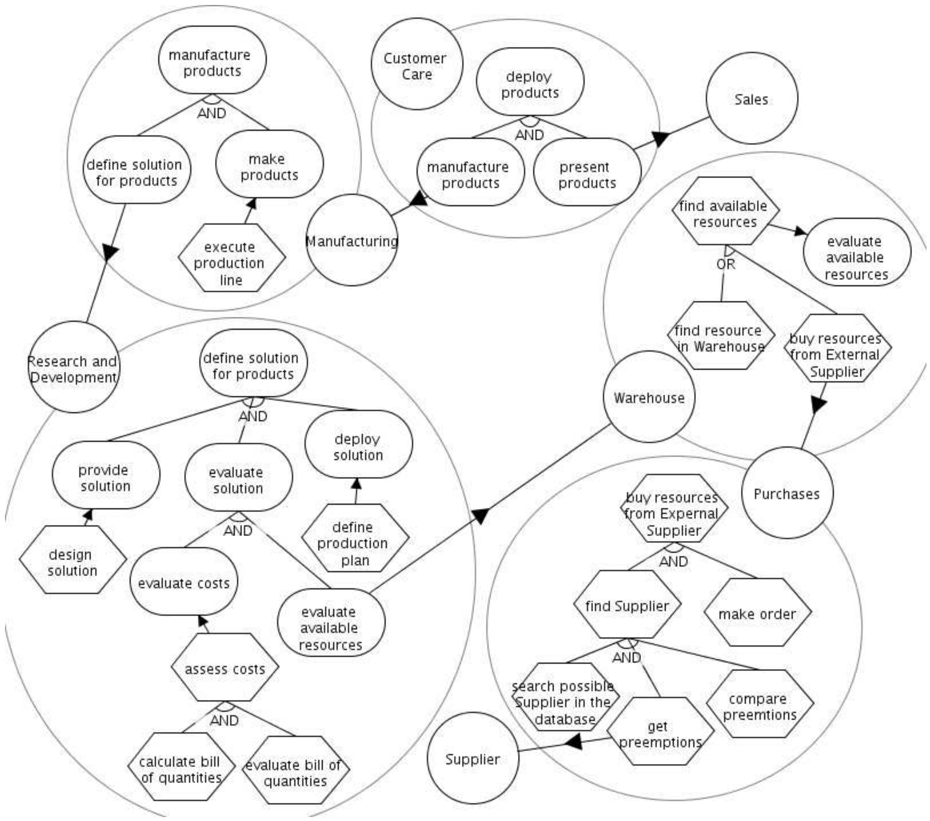


Fig. 1. Product Development Process in Tropos

3 Towards declarative process-oriented annotations

How business processes can be obtained from requirements analysis is an urgent issue for the development of a system. Unfortunately, Tropos is not able to cope with this issue mainly due to the lack of temporal constructs. In this section we discuss how Tropos can be extended in order to deal with high-level process-oriented aspects. The proposed extensions intend to support designers in defining durations, absolute time, and data-based decision constraints of goals and tasks as well as declaratively specifying relations between them. The latter extension is based on DecSerFlow [4] and ConDec [5], two novel graphical languages recently proposed by van der Aalst et al. to represent in a declarative and graphical way service flows and flexible business processes. We call Tropos extended with declarative business process-oriented constructs $\mathcal{B}$-Tropos.

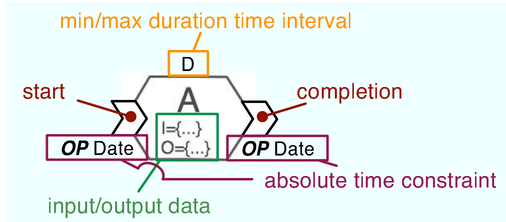


Fig. 2. Extended notation for tasks and goals

3.1 Some definitions

For the sake of clarity, we now give some informal definitions, which will be used to describe the Tropos extensions introduced in this section.

Definition 1 (Time interval). *A time interval is a definite length of time marked off by two (non negative) instants (T_{min} and T_{max}), which could be considered both in an exclusive or inclusive manner. As usually, we use parentheses ($(\ldots)$) to indicate exclusion and square brackets ($[\ldots]$) to indicate inclusion.*

Definition 2 (Relative time interval). *A time interval is relative if initial instant and final instant are defined in function of another instant. Given a time interval TI marked off by T_{min} and T_{max} and a time instant T, two relative time intervals could be defined w.r.t. T*

- *TI^{+T} to denote the time interval marked off by $T + T_{min}$ and $T + T_{max}$;*
- *TI^{-T} to denote the time interval marked off by $T - T_{max}$ and $T - T_{min}$.*

For example, $[10, 15)^{+T_1} \equiv [T_1 + 10, T_1 + 15)$ and $(0, 7]^{-T_2} \equiv [T_2 - 7, T_2)$.

Definition 3 (Absolute time constraint). *An absolute time constraint is a unary constraint of the form T **OP** Date, where T is a time variable, Date is a date and **OP** $\in \{at, after, after\_or\_at, before, before\_or\_at\}$ (with their intuitive meaning).*

Definition 4 (Data-based decision). *A data-based decision formalizes a data-driven choice in terms of a CLP [12] constraint or Prolog predicate.*

Definition 5 (Condition). *A condition is a conjunction of data-based decisions and absolute time constraints.*

3.2 Tasks/Goals extension

In order to support the modeling and analysis of process-oriented aspects of systems, we have annotated goals and tasks with temporal information such as *start* and *completion* time (the notation is shown in Fig. 2). Each task/goal can also be described in terms of its allowed *duration* (D in Fig. 2). This allows one to constrain, for instance, the completion time to the start one: *completion time* $\in D^{+source\ time}$. Additionally, absolute temporal constraints can be used to define start and completion times of goals and tasks. Tasks can also be specified in terms of their *input* and *output*. Finally, goals and tasks can be annotated with a *fulfillment* condition, which defines when they are successfully executed.

	relation	weak relation	negation
responded presence	$\{c\}$ $\{r\}$	$\{c\}$ $\{r\}$	$\{c\}$ $\{r\}$
co-existence	$\{cr_1\}$ $\{cr_2\}$	$\{cr_1\}$ $\{cr_2\}$	$\{cr_1\}$ $\{cr_2\}$
response	Tb $\{c\}$ $\{r\}$	Tb $\{c\}$ $\{r\}$	Tb $\{c\}$ $\{r\}$
precedence	Tb $\{c\}$ $\{r\}$	Tb $\{c\}$ $\{r\}$	Tb $\{c\}$ $\{r\}$
succession	Tb $\{cr_1\}$ $\{cr_2\}$	Tb $\{cr_1\}$ $\{cr_2\}$	Tb $\{cr_1\}$ $\{cr_2\}$

Table 1. Tropos extensions to capture process-oriented constraints (grouped negation connections share the same intended meaning, as described in [4]).

3.3 Process-oriented constraints

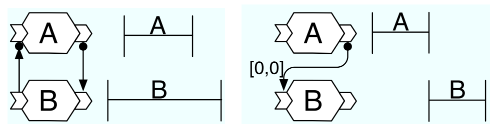
To refine a requirements model into a high-level and declarative process-oriented view, we have introduced different connections between goals and tasks, namely *relation*, *weak relation*, and *negation* (see Table 1). These connections allow designers to specify partial orderings between tasks under both temporal and data constraints. To make the framework more flexible, connections are not directly linked to tasks but to their start and completion time. A small circle is used to denote the connection source, which determines when the triggering condition is satisfied (co-existence and succession connections associate the circle to both end-points, since they are bi-directional).

Relation and negation connections are based on DecSerFlow [4] and ConDec [5] template formulas, extended with the possibility of bounding execution times (e.g., deadlines) and representing data-based and absolute time constraints. Conditions can be specified on both start and completion time and are delimited by curly braces (see $\{c\}$, $\{r\}$ and $\{cr_i\}$ in Table 1); the source condition is a triggering condition whereas the target condition represents a restriction on time and/or data.

The intended meaning of a responded presence relation is: if the source happens s.t. c is satisfied, then the target should happen and satisfy r. The co-existence relation applies the responded presence relation in both directions, by imposing that the two involved tasks, when satisfying cr_1 and cr_2, should co-exist (namely either none or both are executed). Other relation connections extend the responded presence relation by specifying a temporal ordering between source and target events; optionally, a relative time interval (denoted with T_b in Table 1) could be attached to these connections, bounding when the target is expected to happen w.r.t. the time at which the source happened.[3]

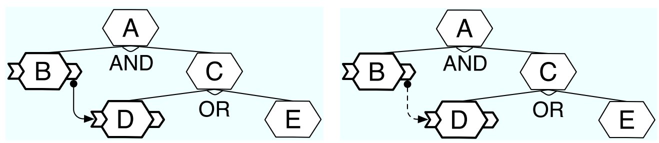
In particular, the response relation constrains the target to happen *after* the source. If T_b is specified, the minimum and maximum time are respectively

[3] If T_b is not specified, the default interval is $(0, \infty)$.



(a) A during B (b) A meets B

Fig. 3. Representation of two simple Allen's intervals in $\mathcal{B}$-Tropos



(a) Response relation (b) Weak response relation

Fig. 4. Integrating process-oriented and goal-directed dimensions in $\mathcal{B}$-Tropos

treated as a delay and a deadline, i.e. the target should occur between the minimum and the maximum time after the source ($target\ time \in T_b^{+source\ time}$). The precedence relation is opposite to response relation, in the sense that it constrains the target to happen *before* the source. A succession relation is used to mutually specify that two tasks are the response and precedence of each other. By mixing different relation connections, we can express complex temporal dependencies and orderings, such as Allen's intervals [9] (see Fig. 3). For example, Allen's *meets* relation is formalized by imposing that A's completion should be equal to B's start (see Fig. 3(b)).

As in DecSerFlow and ConDec, we assume an open approach. Therefore, we have to explicitly specify not only what is expected, but also what is forbidden. These "negative" dependencies are represented by negation connections, the counter-part of relation connections. For example, the negation co-existence between two task states that when one task is executed, the other task shall never be executed, either before or after the source.

Summarizing, through relation and negation connections designers can add a horizontal declarative and high level process-oriented dimension to the vertical goal-directed decomposition of goals and tasks. It is worth noting that, in presence of OR decompositions, adding connections may affect the semantics of the requirements model. The decomposition of task A in Fig. 4(a) shows that A can be satisfied by satisfying D or E. On the contrary, the response relation between B's completion and D's start makes D mandatory (B has to be performed because of the AND-decomposition, hence D is expected to be performed after B). This kind of interaction is not always desirable. Therefore, we have introduced *weak relation* connections that relax relation connections. Their intended meaning is: whenever both the source and the target happen, then the target must satisfy the connection semantics and the corresponding restriction. The main difference between relations and weak relations is that in weak relations the execution is constrained a posteriori, after both source and target have happened.

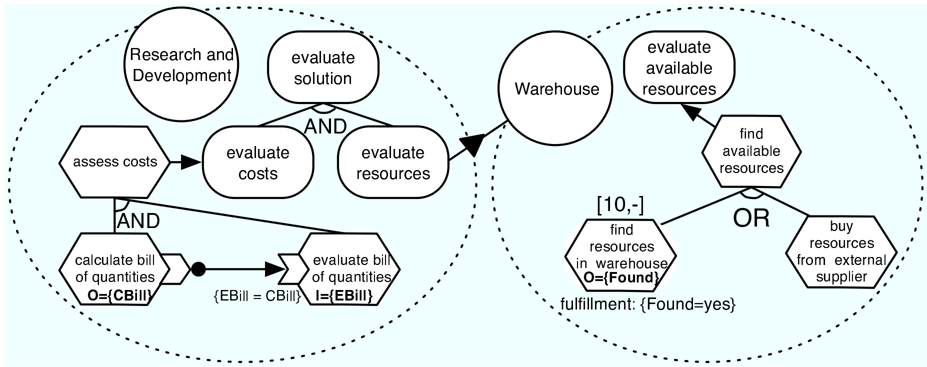


Fig. 5. Process-oriented extensions applied on a fragment of Fig. 1

Differently from Fig. 4(a), in Fig. 4(b) the response constraint between B and D should be satisfied only if D is executed.

Finally, $\mathcal{B}$-Tropos permits to constrain non-leaf tasks, leading to the possibility of expressing some process-oriented patterns [10]. For instance, a relation connection whose source is the completion of a task, which is AND-decomposed into two subtasks, triggers when both subtasks have been executed. Therefore, the connection resembles the concept of a synchronizing merge on the leaf tasks.

To show how process-oriented constraints could be added to a Tropos model, we extend a fragment of the diagram represented in Fig. 1; the result is shown in Fig. 5. The first extension concerns the decomposition of task assess costs: the bill of quantities can be evaluated only after having been calculated. Such a constraint could be modeled in $\mathcal{B}$-Tropos by (1) indicating that the calculation produces a bill of quantities, whereas the evaluation takes a bill as an input, and (2) attaching a response relation connection between the completion of task calculate bill of quantities and the start of task evaluate bill of quantities. The second extension has the purpose of better detailing task find resources in Warehouse, namely representing that (1) task duration is at least of 10 time units, (2) the task produces as an output a datum (called *Found*), which describes whether or not resources have been found in the Warehouse, and (3) the task is considered fulfilled only if resources have been actually found, i.e., *Found* is equal to *yes*.

4 $\mathcal{S}$CIFF

$\mathcal{S}$CIFF [6] is a formal framework based on abductive logic programming [11], developed in the context of the SOCS project[4] for specifying and verifying interaction protocols in an open multi-agent setting. $\mathcal{S}$CIFF introduces the concept of event as an atomic observable and relevant occurrence triggered at execution time. The designer has the possibility to decide what has to be considered as an

[4] SOcieties of heterogeneous ComputeeS, EU-IST-2001-32530 (home page http://lia.deis.unibo.it/research/SOCS/).

event; this generality allows him to decide how to model the target domain at the desired abstraction level, and to exploit $\mathcal{S}$CIFF for representing any evolving process where activities are performed and information is exchanged.

We distinguish between the description of an *event*, and the fact that an event has happened. Happened events are represented as atoms $\mathbf{H}(Ev, T)$, where Ev is a *term* and T is an integer, representing the discrete time point at which the event happened. The set of all the events happened during a protocol execution constitutes its log (or execution trace). Furthermore, the $\mathcal{S}$CIFF language supports the concept of *expectation* as first-class object, pushing the user to think of an evolving process in terms of reactive rules of the form *"if A happened, then B is expected to happen"*. Expectations about events come with form $\mathbf{E}(Ev, T)$ where Ev and T are variables, eventually grounded to a particular term/value.

The binding between happened events and expectations is given by means of *Social Integrity Constraints* ($\mathcal{IC}$s). They are forward rules, of the form $Body \rightarrow Head$, where $Body$ can contain literals and (conjunctions of happened and expected) events and $Head$ can contain (disjunctions of) conjunctions of expectations. CLP constraints and Prolog predicates can be used to impose relations or restrictions on any of the variables, for instance, on time (e.g., by expressing orderings or deadlines). Intuitively, $\mathcal{IC}$ allows the designer to define how an interaction should evolve, given some previous situation represented in terms of happened events; the static knowledge of the target domain is instead formalized inside the $\mathcal{S}$CIFF Knowledge Base. Here we find pieces of knowledge of the interaction model as well as the global society goal and/or objectives of single participants. Indeed, $\mathcal{S}$CIFF considers interaction as goal-directed, i.e., envisages environments in which each actor, as well as the overall society, could have some objective only achievable through interaction; by adopting such a vision, the same interaction protocol could be seamlessly exploited for achieving different strategic goals. This knowledge is expressed in the form of clauses (i.e., a logic program); a clause's body may contain expectations about the behavior of participants, defined literals, and constraints, while their heads are atoms. As advocated in [13], this vision reconciles in a unique framework forward reactive reasoning with backward, goal-oriented deliberative reasoning.

In $\mathcal{S}$CIFF an interaction model is interpreted in terms of an Abductive Logic Program (ALP)[11]. In general, an ALP is a triple $\langle P, A, IC\rangle$, where P is a logic program, A is a set of predicates named *abducibles*, and IC is a set of Integrity Constraints. Roughly speaking, the role of P is to define predicates, the role of A is to fill-in the parts of P that are unknown, and the role of IC is to control the way elements of A are hypothesized, or "abduced". Reasoning in abductive logic programming is usually goal-directed (being G a goal), and it accounts to finding a set of abduced hypotheses Δ built from predicates in A such that $P \cup \Delta \models G$ and $P \cup \Delta \models IC$. The idea underlying $\mathcal{S}$CIFF is to adopt abduction to dynamically *generate* the expectations and to perform the *conformance checking* between expectations and happened events (to ensure that they are following the interaction model). Expectations are defined as abducibles: the framework makes hypotheses about how participants should behave. Conformance is verified by trying to confirm the hypothesized expectations: a concrete running interaction is evaluated as conformant if it *fulfills* the specification. Operationally, expectations are generated and verified by the $\mathcal{S}$CIFF proof procedure,[5] a transition system which has been proved sound and complete w.r.t. the declarative semantics [6]. The proof procedure is embedded within SOCS-SI,[6] a JAVA-based tool capable to accept different event-sources (or previously collected execution traces) and to check if the actual behavior is conformant w.r.t. a given $\mathcal{S}$CIFF specification.

5 Mapping $\mathcal{B}$-Tropos concepts to the $\mathcal{S}$CIFF framework

In this section we present the mapping of $\mathcal{B}$-Tropos concepts into $\mathcal{S}$CIFF specifications, briefly describing how the obtained formalization is used to implement the skeleton of logic-based agents.

Table 2 summarizes the formalization of the goal-oriented part of $\mathcal{B}$-Tropos in $\mathcal{S}$CIFF. Tasks and goals refer to a whatsoever actor X. Being goal-oriented, all the concepts of such a part are modeled inside the $\mathcal{S}$CIFF knowledge base. When formalizing this part, two fundamental concepts emerge: the achievement of a goal and the execution of a task. Both concepts are modeled in $\mathcal{S}$CIFF by considering the actor who is trying to achieve the goal or executing the task and the involved start and completion times; such times should satisfy both duration and absolute time constraints eventually associated to the goal/task. Furthermore, by taking into account a specific goal, different (possibly overlapping) cases may arise:

- AND/OR-decompositions and means-end relations can be trivially translated to Prolog.
- Positive contributions are implemented with a clause specifying that the target is achieved if the contribution's source is achieved.
- Negative contributions are implemented as denials, by imposing that achieving both the involved goals leads to inconsistency.
- In goal and task dependencies, it is expected that the depender will communicate to the dependee that he/she requires the goal to be achieved inside a certain time interval. The communication of this kind of delegation is explicit (i.e. observable), so it can be directly mapped to a $\mathcal{S}$CIFF expectation about depender's behavior.
- In some cases the designer may prefer to keep the model at an abstract level, so goals can be neither refined nor associated to tasks. Abduction allows us to face such a lack of information by reasoning on goal's achievement in a hypothetical way; in particular, we introduce a new abducible called **achieved** to hypothesize that the actor has actually reached the goal.

Task execution mainly differs from goal achievement in that task start and completion events are verified by a fulfillment condition. As for dependency

[5] Available at http://lia.deis.unibo.it/research/sciff/.
[6] Available at http://www.lia.deis.unibo.it/research/socs_si/socs_si.shtml.

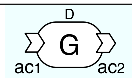
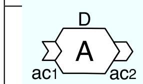
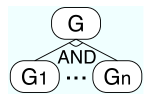
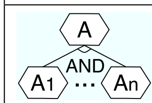
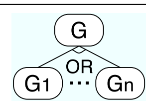
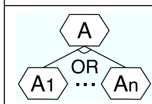
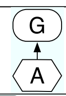
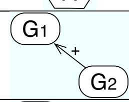
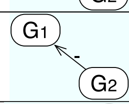
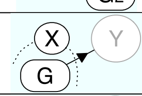
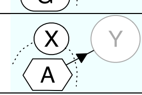
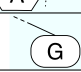
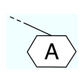
$\mathcal{B}$-Tropos Goal/ Task		$achieve(X, G, T_i, T_f) \leftarrow T_f \in D^{+T_i}, ac_1, ac_2, \ldots.$
		$execute(X, A, T_i, T_f) \leftarrow T_f \in D^{+T_i}, ac_1, ac_2, \ldots.$
AND decomposition		$achieve(X, G, T_i, T_f) \leftarrow$ $achieve(X, G_1, T_{i1}, T_{f1}), \ldots, achieve(X, G_n, T_{in}, T_{fn}),$ $T_i = \min\{T_{i1}, \ldots, T_{in}\}, T_f = \max\{T_{f1}, \ldots, T_{fn}\}.$
		$execute(X, A, T_i, T_f) \leftarrow$ $execute(X, A_1, T_{i1}, T_{f1}), \ldots, execute(X, A_n, T_{in}, T_{fn}),$ $T_i = \min\{T_{i1}, \ldots, T_{in}\}, T_f = \max\{T_{f1}, \ldots, T_{fn}\}.$
OR decomposition		$achieve(X, G, T_i, T_f) \leftarrow achieve(X, G_1, T_i, T_f).$ $\ldots$ $achieve(X, G, T_i, T_f) \leftarrow achieve(X, G_n, T_i, T_f).$
		$execute(X, A, T_i, T_f) \leftarrow execute(X, A_1, T_i, T_f).$ $\ldots$ $execute(X, A, T_i, T_f) \leftarrow execute(X, A_n, T_i, T_f).$
Means-end		$achieve(X, G, T_i, T_f) \leftarrow execute(X, A, T_i, T_f).$
Positive contribution		$achieve(X, G_1, T_i, T_f) \leftarrow achieve(X, G_2, T_i, T_f).$
Negative contribution		$achieve(X, G_1, T_i, T_f), achieve(X, G_2, T_i, T_f) \rightarrow \bot$
Goal Dependency		$achieve(X, G, T_i, T_f) \leftarrow \mathbf{E}(delegate(X, Y, G, T_f), T_i).$
Task Dependency		$execute(X, A, T_i, T_f) \leftarrow \mathbf{E}(delegate(X, Y, A, T_f), T_i).$
Leaf goal		$achieve(X, G, T_i, T_f) \leftarrow \mathbf{achieved}(X, A, T_i, T_f).$
Leaf task		$execute(X, A, T_i, T_f) \leftarrow \mathbf{E}(event(start, X, A), Ti),$ $\mathbf{E}(event(compl, X, A), Tf),$ $fulfillment\_condition, Tf > Ti.$

Table 2. Mapping of the goal-oriented proactive part of $\mathcal{B}$-Tropos in $\mathcal{S}$CIFF.

relations, these events are mapped to expectations and should appear in the execution trace.

The reactive part of $\mathcal{B}$-Tropos encompasses both the reaction to a request for achieving a goal and process-oriented constraints. As already pointed out, process-oriented constraints are inspired by DecSerFlow/ConDec template formulas, for which a preliminary mapping to $\mathcal{S}$CIFF has been already established

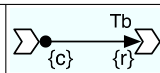
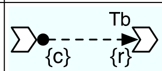
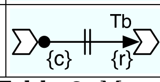
Response		$\mathbf{hap}(event(Ev, A, X), T_1) \wedge c$ $\rightarrow \mathbf{exp}(event(Ev, A, X), T_2) \wedge r \wedge T_2 \in T_b^{+T_1}$.
Weak Response		$\mathbf{hap}(event(Ev, A, X), T_1) \wedge c$ $\wedge \mathbf{hap}(event(Ev, A, X), T_2) \rightarrow r \wedge T_2 \in T_b^{+T_1}$.
Negation Response		$\mathbf{hap}(event(Ev, A, X), T_1) \wedge c$ $\wedge \mathbf{hap}(event(Ev, A, X), T_2) \wedge r \wedge T_2 \in T_b^{+T_1} \rightarrow \bot$.

Table 3. Mapping of $\mathcal{B}$-Tropos response connections in $\mathcal{S}$CIFF.

[14]. Connections belonging to the same family (i.e. relations, weak relations and negations) are translated to very similar $\mathcal{IC}$s: the only main difference is the way in which the involved times are constrained, to reflect the connection semantics. An example is given in Table 3, where response connections have been formalized; they specify in a straightforward way the informal description given in Section 3.

Predicates **hap** and **exp** respectively represent the happening and the expectation of a complex or simple event (remember indeed that also non-leaf tasks could be constrained). Since the start and completion of leaf tasks are considered as observable events, then for a leaf-task A ($Ev \in \{start, completion\}$):

$$\mathbf{hap}(event(Ev, A, X), T) \leftarrow \mathbf{H}(event(Ev, A, X), T).$$
$$\mathbf{exp}(event(Ev, A, X), T) \leftarrow \mathbf{E}(event(Ev, A, X), T).$$

Complex events recursively follows the AND/OR decomposition philosophy:

- the start/completion of an OR-decomposed task happen (resp. is expected to happen) when one of its (sub)tasks start/completion happens (resp. is expected to happen);
- the start of an AND-decomposed task happens (resp. is expected to happen) when its first (sub)task is started (resp. expected to started);
- the completion of an AND-decomposed task happens (resp. is expected to happen) when its last (sub)task is completed (expected to be completed).

To model the reaction to a request for achieving a goal G, we simply assume that when a dependee Y receives from depender X a request for achieving goal G, then Y should react by assuming the commitment of actually achieving G:[7]

$$\mathbf{H}(delegate(X, Y, G, T_f), T_d) \rightarrow achieve(Y, G, T_i, T_f) \wedge T_i > T_d.$$

Table 5.1 shows the $\mathcal{S}$CIFF formalization corresponding to the $\mathcal{B}$-Tropos diagram of Figure 5. Here Research & Development and Warehouse are respectively represented as $r\&d$ and wh, and the equality symbol = is used to denote unification.

The provided formalization could be used to directly implement the skeleton of logic-based agents, as for example the ones described in [7]. Such agents follow

[7] Anyway, more complex dependency protocols should be seamlessly modeled in $\mathcal{S}$CIFF.

Table 5.1 Formalization of the $\mathcal{B}$-Tropos model fragment shown in Figure 5

$$
\begin{aligned}
KB_{r\&d}: achieve(r\&d, eval\_solution, T_i, T_f) &\leftarrow achieve(r\&d, eval\_costs, T_{i1}, T_{f1}),\\
&\quad achieve(r\&d, eval\_resources, T_{i2}, T_{f2}),\\
&\quad \min(T_i, [T_{i1}, T_{i2}]), \max(T_f, [T_{f1}, T_{f2}]).\\
achieve(r\&d, eval\_costs, T_i, T_f) &\leftarrow execute(r\&d, assess\_costs, T_i, T_f).\\
execute(r\&d, assess\_costs, T_i, T_f) &\leftarrow execute(r\&d, calc\_bill, T_{i1}, T_{f1}),\\
&\quad execute(r\&d, eval\_bill, T_{i2}, T_{f2}),\\
&\quad \min(T_i, [T_{i1}, T_{i2}]), \max(T_f, [T_{f1}, T_{f2}]).\\
execute(r\&d, calc\_bill, T_i, T_f) &\leftarrow \mathbf{E}(event(start, r\&d, calc\_bill), T_i),\\
&\quad \mathbf{E}(event(compl, r\&d, calc\_bill, [CBill]), T_f), T_f > T_i.\\
execute(r\&d, eval\_bill, T_i, T_f) &\leftarrow \mathbf{E}(event(start, r\&d, eval\_bill, [EBill]), T_i),\\
&\quad \mathbf{E}(event(compl, r\&d, eval\_bill), T_f), T_f > T_i.\\
achieve(r\&d, eval\_resources, T_i, T_f) &\leftarrow \mathbf{E}(delegate(r\&d, wh, eval\_resources, T_f), T_i).\\
KB_{wh}: achieve(wh, eval\_resources, T_i, T_f) &\leftarrow execute(wh, find\_resources, T_i, T_f).\\
execute(wh, find\_resources, T_i, T_f) &\leftarrow execute(wh, find\_in\_wh, T_i, T_f).\\
execute(wh, find\_resources, T_i, T_f) &\leftarrow execute(wh, buy, T_i, T_f).\\
execute(wh, find\_resources, T_i, T_f) &\leftarrow \mathbf{E}(event(start, wh, find\_in\_wh), T_i),\\
&\quad \mathbf{E}(event(compl, wh, find\_in\_wh, Found), T_f),\\
&\quad T_f \geq T_i + 10, Found = yes.\\
execute(wh, buy, T_i, T_f) &\leftarrow \mathbf{E}(event(start, wh, buy), T_i),\\
&\quad \mathbf{E}(event(compl, wh, buy), T_f), T_f > T_i.
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{IC}s_{r\&d}: \mathbf{H}(event(compl, r\&d, calc\_bill, [CBill]), T_1) &\rightarrow \mathbf{E}(event(start, r\&d, eval\_bill, [EBill]), T_2)\\
&\quad \wedge T_2 > T_1, EBill = CBill.\\
\mathcal{IC}s_{wh}: \mathbf{H}(delegate(r\&d, wh, eval\_resources, T_f), T_i) &\rightarrow achieve(wh, eval\_resources, T_i, T_f).
\end{aligned}
$$

the Kowalsky-Sadri cycle for intelligent agents, by realizing the *think* phase with the $\mathcal{S}$CIFF proof-procedure and the *observe* and *act* phases in JADE. The proof-procedure embedded into $\mathcal{S}$CIFF-agents is equipped with the possibility to transform expectations about the agent itself into happened events, and with a selection rule for choosing a behavior when more different choices are available.

In particular, each actor represented in a $\mathcal{B}$-Tropos model could be mapped into a $\mathcal{S}$CIFF-agent whose deliberative pro-active part (formalized in the agent's knowledge base) is driven by the goal/task decomposition of its root goal, and whose reactive behavior (formalized as a set of $\mathcal{IC}$s) is determined by the delegation mechanism and the process-oriented constraints. The agent that wants to achieve the global goal (such as Customer Care in Figure 1) starts by decomposing it, whereas other actors wait until an incoming request from a depender is observed; in this case, the delegation reactive rule of the agent is triggered, and the agent tries to achieve its root goal. The root goal is decomposed until finally one or more expectations are generated. Such expectations could be either requests or start/completions of tasks, and thus are transformed to happened events, i.e. actions performed by the agent.

Table 5.1 shows how the formalized $\mathcal{S}$CIFF specification is assigned to the two agents under study, i.e., the Warehouse and R&D unit. To have an intuition about how the two agents act and interact, let us consider the case in which the R&D unit should achieve its top goal (because it has received the corresponding delegation from the Manufacturing division). The unit will decompose the goal obtaining, at last, the following set of expectations about itself:[8]

$$\begin{aligned}
&\mathbf{E}(event(start, r\&d, calc_bill), T_{scb}), \ldots,\\
&\mathbf{E}(event(compl, r\&d, calc_bill, [Bill]), T_{ccb}), T_{ccb} > T_{scb},\\
&\mathbf{E}(event(start, r\&d, eval_bill, [Bill]), T_{seb}), T_{seb} > T_{ccb},\\
&\mathbf{E}(event(compl, r\&d, eval_bill), T_{ceb}), T_{ceb} > T_{seb},\\
&\mathbf{E}(del(r\&d, wh, eval_resources, T_{cer}), T_{ser}).
\end{aligned}$$

This set of expectations could be read as an execution plan, consisting of two concurrent parts: (1) a sequence about start/completion of leaf tasks, ordered by the response relation which constrains the bill calculation and evaluation; (2) the delegation of resources evaluation, which should be communicated to the Warehouse. In particular, when the expectation about the delegation is transformed to a happened event by the R&D agent, the Warehouse agent is committed to achieve the delegated goal inside the time interval (T_{ser}, T_{cer}).

Besides the implementation of logic-based agents, $\mathcal{S}$CIFF can also be used to perform different kinds of verification, namely performance verification and conformance verification. Performance verification is devoted to prove that stakeholders can achieve their strategic goals in a given time. Such a verification can also be used to evaluate different design alternatives in terms of system performances. Conformance verification [6] is related to the auditing measures that can be adopted for monitoring the activities performed by actors within the system. The idea underlying conformance verification is to analyze system logs and compare them with the design of the system. This allows system administrators to understand whether or not stakeholders have achieved their goals and, if it is not the case, predict future actions. For the lack of space, we do not discuss here the details of these kinds of verification.

6 Related Works

Several formal frameworks have been developed to support the Tropos methodology. For instance, Giorgini et al. [15] proposed a formal framework based on logic programming for the analysis of security and privacy requirements. However, the framework does not take into account temporal aspects of the system. In [16] a planning approach has been proposed to analyze and evaluate design alternatives. Though this framework explores the space of alternatives and determines a (sub-)optimal plan, that is, a sequence of actions, to achieve the goals of stakeholders, it does not permit to define temporal constraints among

[8] By imposing, through a special integrity constraint, that two different expectations about the same event should be fulfilled by one happened event.

tasks. Fuxman et al. [17] proposed Formal Tropos that extends Tropos with annotations that characterize the temporal evolution of the system, describing for instance how the network of relationships evolves over time. Formal Tropos provides a temporal logic-based specification language for representing Tropos concepts together with temporal constructs, which are verified using a model-checking technique such as the one implemented in NuSMV. This framework has been used to verify the consistency of the requirements model [17] as well as business processes against business requirements and strategic goal model [3]. However, Formal Tropos does not support conformance verification.

The use of computational logic for the flexible specification and rigorous verification of agent interaction is adopted by many proposals. While other works [18] use temporal logics to model the temporal dimension of interaction, $\mathcal{S}$CIFF exploits a constraint solver and adopts an explicit representation of time, making it possible to specify and reason upon expressive temporal constraints (such as deadlines). With [19, 20], $\mathcal{S}$CIFF shares the vision of a multi-agent system as an open society of heterogeneous and autonomous interacting entities. While in [20] Event Calculus is applied to commitment-based protocol specification, $\mathcal{S}$CIFF semantics is given in terms of an Abductive Logic Program. Many abductive proof-procedures exist in literature (e.g., [13, 21]), but none of them deals with hypotheses confirmation used in $\mathcal{S}$CIFF to verify if the abduced expectations have indeed a corresponding happened event.

7 Conclusions

In this work we have proposed $\mathcal{B}$-Tropos, an extension of Tropos with declarative process-oriented constraints, to the aim of making the first step towards the definition of a business process from an early requirements model. More specifically, we have introduced the possibility to mutually constrain task/goal execution times, by using connections inspired by DecSerFlow and ConDec languages. Augmenting a Tropos model with such constraints has the effect that both the proactive agents behavior and the reactive, process-oriented one could be captured within the same diagram.

We have also shown how both goal-oriented and process-oriented dimensions of $\mathcal{B}$-Tropos can be mapped into the $\mathcal{S}$CIFF framework. Such a mapping makes it possible to directly implement logic-based agents following what is prescribed by the model, as well as to perform different kinds of verification, namely to check if the model satisfies a given property and to monitor if the execution trace of a real system is actually compliant with the model.

The work presented here it is a first step towards the integration of a business process in the requirements model. The next step will be the generation of executable business process specifications (such as BPEL) from $\mathcal{B}$-Tropos models. Moreover, we intend to better exploit the underlying $\mathcal{S}$CIFF constraint solver by introducing more complex scheduling and resource constraints in order to capture more details of business requirements and agent interactions.

References

1. Henderson-Sellers, B., Giorgini, P., eds.: Agent-Oriented Methodologies. Idea Group Publishing (2005)
2. Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., Perini, A.: TROPOS: An Agent-Oriented Software Development Methodology. JAAMAS **8**(3) (2004) 203–236
3. Kazhamiakin, R., Pistore, M., Roveri, M.: A Framework for Integrating Business Processes and Business Requirements. In: EDOC'04, IEEE Press (2004) 9–20
4. van der Aalst, W.M.P., Pesic, M.: DecSerFlow: Towards a truly declarative service flow language. In: WS-FM'06. Volume of LNCS 4184., Springer (2006)
5. van der Aalst, W.M.P., Pesic, M.: A declarative approach for flexible business processes management. In: BPM'06. LNCS 4103, Springer (2006) 169–180
6. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Verifiable agent interaction in abductive logic programming: the SCIFF framework. TOCL (2007) Accepted for publication.
7. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P.: A verifiable logic-based agent architecture. In: ISMIS'06. LNCS 4203, Springer (2006) 188–197
8. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Security Protocols Verification in Abductive Logic Programming: A Case Study. In: ESAW'05. LNCS 3963, Springer (2005) 106–124
9. Allen, J., Ferguson, G.: Actions and events in interval temporal logic. Journal of Logic and Computation **4**(5) (1995)
10. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distributed and Parallel Databases **14**(1) (2003) 5–51
11. Kakas, A.C., Kowalski, R.A., Toni, F.: Abductive Logic Programming. Journal of Logic and Computation **2**(6) (1993) 719–770
12. Jaffar, J., Maher, M.: Constraint logic programming: a survey. Journal of Logic Programming **19-20** (1994) 503–582
13. Fung, T.H., Kowalski, R.A.: The IFF proof procedure for abductive logic programming. Journal of Logic Programming **33**(2) (1997) 151–165
14. Chesani, F., Mello, P., Montali, M., Storari, S.: Towards a decserflow declarative semantics based on computational logic. Technical Report DEIS-LIA-07-002, DEIS, Bologna, Italy (2007)
15. Giorgini, P., Massacci, F., Zannone, N.: Security and Trust Requirements Engineering. In: FOSAD 2004/20005. LNCS 3655, Springer (2005) 237–272
16. Bryl, V., Massacci, F., Mylopoulos, J., Zannone, N.: Designing Security Requirements Models through Planning. In: CAiSE'06. LNCS 4001, Springer (2006) 33–47
17. Fuxman, A., Liu, L., Mylopoulos, J., Pistore, M., Roveri, M., Traverso, P.: Specifying and Analyzing Early Requirements in Tropos. REJ **9**(2) (2004) 132–150
18. Venkatraman, M., Singh, M.P.: Verifying compliance with commitment protocols. JAAMAS **2**(3) (1999) 217–236
19. Artikis, A., Pitt, J., Sergot, M.J.: Animated specifications of computational societies. In: AAMAS'02. (2002) 1053–1061
20. Yolum, P., Singh, M.P.: Flexible protocol specification and execution: applying event calculus planning using commitments. In: AAMAS'02, ACM (2002) 527–534
21. Kakas, A., Mancarella, P.: On the relation between truth maintenance and abduction. In: PRICAI'90. (1990) 438–443