$\mathcal{B}$-Tropos

Agent-oriented requirements engineering meets computational logic for declarative business process modeling and verification*

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

[1] DISI, University of Trento – Via Sommarive 14, 38100 Povo (TN), Italy
{bryl | zannone}@disi.unitn.it
[2] DEIS, University of Bologna – V.le Risorgimento 2, 40136 Bologna, Italy
{pmello | mmontali | ptorroni} @deis.unibo.it

Abstract. The work presented in this paper stands at the intersection of three diverse research areas: agent-oriented early requirements engineering, business process requirements elicitation and specification, and computational logic-based specification and verification. The analysis of business requirements and the specification of business processes are fundamental steps in the development of information systems. The first part of this paper presents $\mathcal{B}$-Tropos as a way to combine business goals and requirements with the business process model. $\mathcal{B}$-Tropos enhances a well-known agent-oriented early requirements engineering framework with declarative business process-oriented constructs, inspired by the DecSerFlow and ConDec languages. In the second part of the paper, we show a mapping of $\mathcal{B}$-Tropos onto $\mathcal{S}$CIFF, a computational logic-based framework for properties and conformance verification.

1 Introduction

This work proposes an integration of different techniques for information systems engineering, with the aim to reconcile requirements elicitation with declarative specification, prototyping, and analysis inside a single unified framework.

In tackling the requirements elicitation part, we take an agent-oriented perspective. Modeling and analyzing requirements of IT systems in terms of agents and their goals is an increasingly popular approach [20] which helps understanding the organizational setting where a system operates, as well as modeling the stakeholders' strategic interests, and finally documenting the rationale behind the design choices made. After system requirements elicitation is complete, one must define a corresponding business process. A very important issue that must be addressed at this stage is how to link the "strategic" business goals and

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

requirements with the business process model [23]. 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 and cooperation within economic activities.

For our purpose, we have chosen Tropos [8], an agent-oriented software engineering methodology which uses the concepts of agent and goal from the early phases of the system development. Tropos has a number of interesting features, such as its goal- and agent-orientation, intuitive and expressive modeling notation, etc., which have made it to become popular. However, a drawback of Tropos and a number of similar methodologies is that they do not clearly define how to move from a requirements model to a business process model. For example, Tropos does not allow the modeling of temporal and data constraints between tasks assigned to agents: this means that when developing a business process, the corresponding Tropos model does not have enough information to define a temporal ordering between activities. Likewise, start and completion times, triggering events, deadlines, and many other aspects not necessarily related to the temporal dimension are essential elements in the description of a business process model, but they are not represented in Tropos models.

How to enhance Tropos with information that can be automatically used in the generation of a business process model is one of the aspects we address in this work. In particular, we have extended Tropos with declarative business process-oriented constructs, inspired by two recent graphical languages: DecSerFlow [34] and ConDec [33]. We enhance the characteristic goal-oriented approach of Tropos agents by introducing 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 onto the $\mathcal{S}$CIFF language [4], which sits at the basis of a computational logic-based framework for the specification and verification of interaction protocols in open multi-agent systems. In the presentation of this work, we discuss the issue of time (ordering, deadlines, etc.) because it is an essential part of business process modeling, and because it is easy to explain by intuitive examples. However, $\mathcal{B}$-Tropos is not only a temporal extension of Tropos, but it covers also the treatment of conditions on process input/output data and other constraints.

The marriage of $\mathcal{B}$-Tropos with $\mathcal{S}$CIFF sets a link between specification, prototyping and analysis: in fact, $\mathcal{S}$CIFF specifications can be used to implement and animate logic-based agents [1], as well as to perform different verification tasks, such as properties verification [2] and conformance verification of a given execution trace [4]. Prototyping (animation) and analysis (properties and conformance verification) add value to $\mathcal{B}$-Tropos and can make it appealing to a large set of potential users. Early requirements engineers and process engineers will be able to test their models directly and get an immediate picture of the system being developed. Engineers testing the properties of the models will not have to resort to ad-hoc, error-prone translations of high-level models into the languages used to feed specifications into model checkers, since $\mathcal{B}$-Tropos can

directly generate $\mathcal{S}$CIFF programs. Managers who need to monitor the correct behavior of a running system will have a $\mathcal{S}$CIFF specification of the system generated out of a $\mathcal{B}$-Tropos model automatically, and based on this specification they will be able to automatically check the compliance of the system using the SOCS-SI runtime and offline checking facilities [3].

In this work, we focus on specific aspects of this global vision. We define $\mathcal{B}$-Tropos and the mapping of $\mathcal{B}$-Tropos constructs onto the $\mathcal{S}$CIFF framework. 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 organizational model is an excerpt of a large case study under consideration within the national FIRB TOCAI.IT project.[3]

The structure of the paper is as follows. Section 2 briefly presents the Tropos methodology. Section 3 describes $\mathcal{B}$-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 in Section 6 and conclusive remarks in Section 7.

2 The Tropos methodology

Tropos [8] 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 *what* or *how*.

The methodology is founded on models that use the concepts of actors (i.e., agent and roles), goals, tasks, resources, and social dependencies between two actors. 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 produce 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 in order to achieve some goal, execute some task, or deliver some resource. The former actor is called the *depender*, while the latter is called the *dependee*. The object around which the dependency centers, which can be a goal, a task, or a resource, is called the *dependum*. In the graphical representation, actors are represented as circles; goals, tasks 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 a system that is incrementally refined and extended. Specifically, *goal*

[3] The TOCAI.IT project (RBNE05BFRK, `http://www.dis.uniroma1.it/~tocai/`) is a three-year, 4.5 Ml euro project on "Knowledge-oriented technologies for enterprise aggregation in Internet." It involves a consortium of 11 Italian universities, the National Research Council, and three industrial partners in the ICT, engineering, and manufacturing sectors.

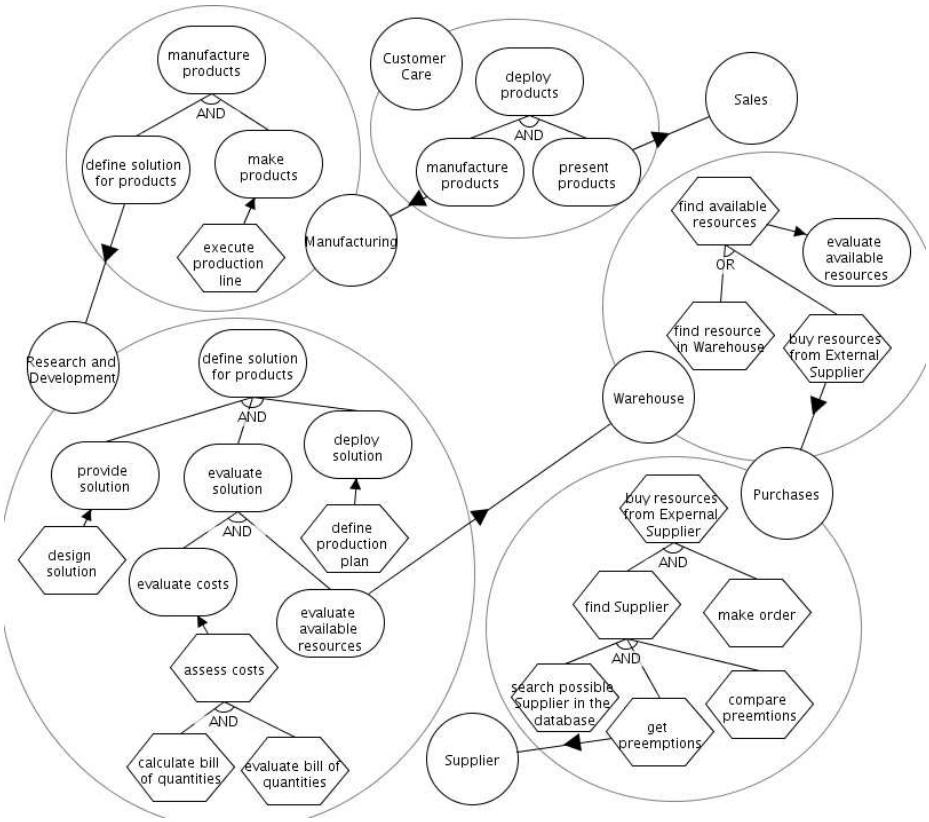


Fig. 1. Product development process

analysis consists of refining goals and eliciting new social relationships among actors. Goal analysis is conducted from the perspective of single actors using means-end analysis and AND/OR decomposition. *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 high-level process for achieving a goal or a task, whereas OR-decomposition defines alternatives for achieving a goal or executing a task. Fig. 1 presents the Tropos diagram representing an excerpt of the product development process studied in the course of the TOCAI project.

Example 1. 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 it refines into subgoals manufacture product, for which Customer Care depends on the Manufacturing division, and present product, for which it depends on the Sales division. In turn, Manufacturing de-

composes 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 by executing 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. In order to evaluate costs, R&D executes task assess costs, which consists of calculate bill of quantities and evaluate bill of quantities. Moreover, this division depends on the Warehouse for the goal evaluate available resources. The Warehouse either queries the databases to find available resources or asks the Purchases division to buy resources from external Supplier. The Purchases division searches in company's databases for possible Suppliers and selects the one who provides the best offer.

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 domain-based constraints for goals and tasks, as well as declaratively specifying relations between them. These extensions are based on the DecSerFlow [34] and ConDec [33] graphical languages for the declarative representation of service flows and flexible business processes. The enhanced Tropos is called $\mathcal{B}$-Tropos.

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 can 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 $(0, 7]^{-T_2} \equiv (T_2 - 7, T_2]$.

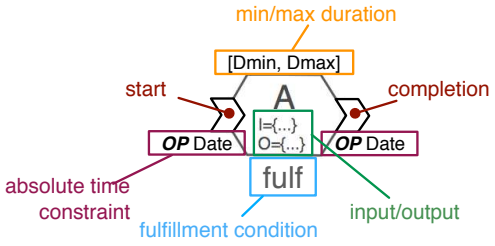


Fig. 2. Extended notation for tasks and goals

Definition 3 (Absolute time constraint). *An absolute time constraint is a 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 (Domain-based constraint). *A domain-based constraint formalizes specific application domain requirements and is specified using CLP constraints [21] (e.g.,* $>$, $<$, $=$, *etc.) or Prolog predicates.*

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

For example, condition T $before\_or\_at$ 11.26.2007 $\wedge$ $workingDay(T)$ states that T has 26 November 2007 as deadline and that it must be a working day.

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* times (the notation is shown in Fig. 2). Each task/goal can also be described in terms of its allowed *duration* ($[Dmin, Dmax]$ in Fig. 2). This allows one to constrain, for instance, the completion time to the start time, i.e., *completion time* $\in [Dmin, Dmax]^{+start\ time}$. Additionally, absolute temporal constraints can be used to define start and completion times of goals and tasks.

A goal/task can also be described in terms of the resources needed and produced by the goal/task itself. We represent the resources needed by a goal/task through attribute *input* and the resources produced by a goal/task through attribute *output*. Finally, tasks can be annotated with a *fulfillment condition*, which defines when they are successfully executed.

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

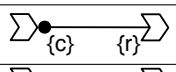
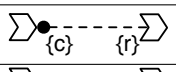
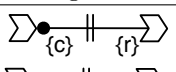
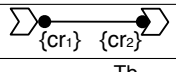
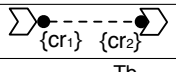
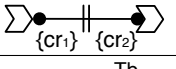
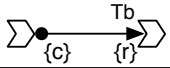
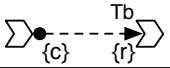
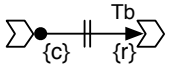
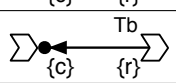
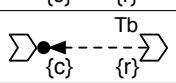
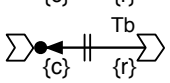
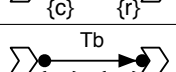
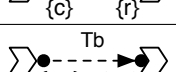
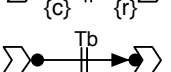
	relation	weak relation	negation
responded presence			
co-existence			
response			
precedence			
succession			

Table 1. Tropos extensions to capture process-oriented constraints (grouped negation connections share the same intended meaning, as described in [34])

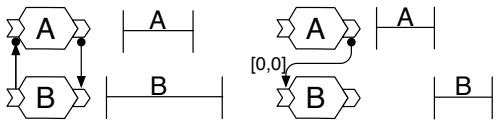
designers to specify partial orderings between tasks under both temporal and domain-based constraints. To make the framework more flexible, connections are not directly linked to tasks but to their start and completion times. This solution, for instance, enables the representation of interleaving concurrency. 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 [34] and ConDec [33] template formulas, extended with constraints on execution times (e.g., deadlines) as well as domain-based and absolute time constraints. Conditions can be specified on both start and completion times and are delimited by curly braces ($\{c\}$, $\{r\}$ and $\{cr_i\}$ in Table 1); the source condition is a triggering condition, whereas the target imposes restrictions on time and/or data.

The intended meaning of a *responded presence* relation is: if the source happens such that 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).

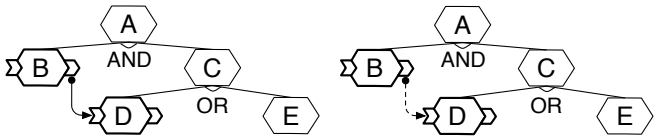
Other relation connections extend the responded presence relation by specifying a temporal ordering between source and target events; optionally, a relative time interval (denoted with T_b in Table 1) could be attached to these connections, bounding when the target is expected to happen with respect to the time at which the source happened.[4] In particular, the *response* relation constrains the target to happen *after* the source. If T_b is specified, the minimum and maximum times are respectively treated as a delay and a deadline, that is, 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.

[4] 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

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 [5] (Fig. 3). For example, Allen's *duration* relation is formalized by imposing that A's start should happen after B's start and A's completion should happen before B's completion (Fig. 3(a)), whereas *meets* relation is formalized by imposing that A's completion should be equal to B's start (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, negation co-existence between two tasks states that when one task is executed, the other task shall never be executed, neither before nor 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 its subtask C 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 with the intent of relaxing relation connections. Their intended meaning is: whenever both the source and the target happen and the trigger condition is satisfied, the target must satisfy the restriction condition. The main difference between relations and weak relations is that in weak relations the execution is constrained a posteriori, after both

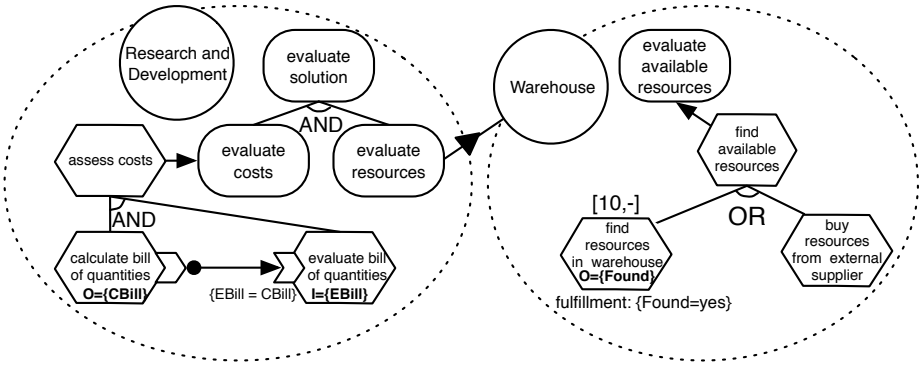


Fig. 5. Process-oriented extensions applied on a fragment of Fig. 1

source and target have happened. 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 [35]. 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 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, that is, $Found$ is equal to yes. Finally, one can notice the absence of constraints between goals evaluate costs and evaluate resources. Such an absence enables the two sets of activities aimed at achieving those goals to be executed concurrently.

4 $\mathcal{S}$CIFF

$\mathcal{S}$CIFF [4] is a formal framework based on abductive logic programming [22], developed in the context of the SOCS project[5] for specifying and verifying inter-

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

action 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 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 in 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). Such constraints 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 on the interaction model as well as the global organizational goal and/or objectives of single participants. Indeed, $\mathcal{S}$CIFF considers interaction as goal-directed, i.e., it envisages environments in which each actor as well as the overall organization 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 body may contain expectations about the behavior of participants, defined literals, and constraints, while their heads are atoms. As advocated in [17], 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) [22]. 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 "abducted". Reasoning in abductive logic programming is usually goal-directed, and accounts for finding a set of abducted hypotheses Δ built from predicates in A such that $P \cup \Delta \models G$ (being G a goal) 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,[6] a transition system which has been proved sound and complete with respect to the declarative semantics [4]. The proof procedure is embedded within SOCS-SI [3], a JAVA-based tool capable of accepting different event sources (or previously collected execution traces) and checking if the actual behavior is conformant with respect to 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. The idea behind the mapping is to define a formal statement in $\mathcal{S}$CIFF for each $\mathcal{B}$-Tropos graphical element. This allows for the automatic generation of $\mathcal{S}$CIFF specifications from $\mathcal{B}$-Tropos models.

Table 2 summarizes the formalization of the goal-oriented part of $\mathcal{B}$-Tropos in $\mathcal{S}$CIFF. This part represents the static knowledge of the application domain, so it is modeled inside the $\mathcal{S}$CIFF knowledge base. Two fundamental concepts are *goal achievement* and *task execution*. These concepts are modeled in $\mathcal{S}$CIFF using the 6-ary predicates *achieve* and *execute*. Intuitively, $achieve(x, g, t_i, t_f, i, o)$ is true if actor x achieves goal g where t_i is the start time and t_f is the completion time. $execute(x, a, t_i, t_f, i, o)$ holds if actor x executes task a where t_i and t_f are start and completion time, respectively. Parameters i and o represent the resources respectively needed and produced by the execution of the task or achievement of a goal. Start and completion times should satisfy both duration and absolute time constraints (ac in Table 2) eventually associated to a goal/task.

The execution of tasks is also determined by the satisfaction of fulfillment conditions and the generation of task start and completion events. These events are represented using literals of the form $event(ev, x, a, r)$ where $ev \in \{start, end\}$, a is the task that has generated the event, x is the actor who has executed the task, and r is a list of resources. In particular, resources associated with start events represent the input of the task, whereas resources associated with completion events refer to the output.

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 achievement in a hypothetical way. In particular, we have introduced a new abducible called **achieved** to hypothesize that the actor has actually reached the goal.

Tropos relations are then formalized in $\mathcal{S}$CIFF as rules on the basis of the following concepts:

[6] Available at `http://lia.deis.unibo.it/research/sciff/`.

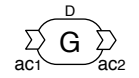
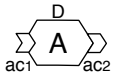
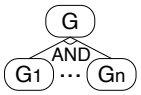
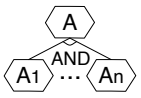
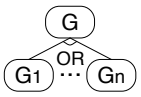
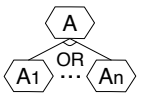
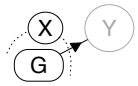
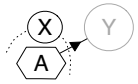
Leaf goal		$achieve(X, G, T_i, T_f, I, O) \leftarrow$ $\mathbf{achieved}(X, G, T_i, T_f, I, O)$, $T_f \in [D_{min}, D_{max}]^{+T_i}, ac_1, ac_2.$
Leaf task		$execute(X, A, T_i, T_f, I, O) \leftarrow$ $\mathbf{E}(event(start, X, A, I), T_i)$, $\mathbf{E}(event(end, X, A, O), T_f)$, $T_f \in [D_{min}, D_{max}]^{+T_i}, ac_1, ac_2,$ $fulfillment\_condition.$
AND decomposition		$achieve(X, G, T_i, T_f, I, O) \leftarrow$ $achieve(X, G_1, T_{i1}, T_{f1}, I_1, O_1), \ldots,$ $achieve(X, G_n, T_{in}, T_{fn}, I_n, O_n),$ $T_i = \min\{T_{i1}, \ldots, T_{in}\}, T_f = \max\{T_{f1}, \ldots, T_{fn}\},$ $I = I_1 \cup \ldots \cup I_n, O = O_1 \cup \ldots \cup O_n.$
		$execute(X, A, T_i, T_f, I, O) \leftarrow$ $execute(X, A_1, T_{i1}, T_{f1}, I_1, O_1), \ldots,$ $execute(X, A_n, T_{in}, T_{fn}, I_n, O_n),$ $T_i = \min\{T_{i1}, \ldots, T_{in}\}, T_f = \max\{T_{f1}, \ldots, T_{fn}\},$ $I = I_1 \cup \ldots \cup I_n, O = O_1 \cup \ldots \cup O_n.$
OR decomposition		$achieve(X, G, T_i, T_f, I, O) \leftarrow achieve(X, G_1, T_i, T_f, I, O).$ $\ldots$ $achieve(X, G, T_i, T_f, I, O) \leftarrow achieve(X, G_n, T_i, T_f, I, O).$
		$execute(X, A, T_i, T_f, I, O) \leftarrow execute(X, A_1, T_i, T_f, I, O).$ $\ldots$ $execute(X, A, T_i, T_f, I, O) \leftarrow execute(X, A_n, T_i, T_f, I, O).$
Means-end		$achieve(X, G, T_i, T_f, I, O) \leftarrow execute(X, A, T_i, T_f, I, O).$
Goal dependency		$achieve(X, G, T_i, T_f, I, O) \leftarrow$ $\mathbf{E}(delegate(X, Y, G, T_f), T_d)$, $T_d > T_i, T_d < T_f.$
Task dependency		$execute(X, A, T_i, T_f, I, O) \leftarrow$ $\mathbf{E}(delegate(X, Y, A, T_f), T_d)$, $T_d > T_i, T_d < T_f.$

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

- AND/OR-decompositions and means-end are trivially translated to $\mathcal{S}$CIFF.
- In goal (task) dependencies, it is expected that the depender appoints the dependee to achieve a goal (execute a task) before a certain time instant. To this end, we have introduced event $delegate(x, y, g, t)$ to indicate that actor x delegates the achievement of goal g to actor y and y have to achieve g by

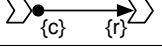
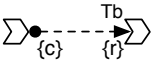
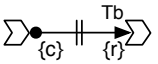
Response		$\mathbf{hap}(event(Ev, X_1, A_1, R_1), T_1) \wedge c$ $\rightarrow \mathbf{exp}(event(Ev, X_2, A_2, R_2), T_2) \wedge r \wedge T_2 \in T_b^{+T_1}$.
Weak Response		$\mathbf{hap}(event(Ev, X_1, A_1, R_1), T_1) \wedge c$ $\wedge \mathbf{hap}(event(Ev, X_2, A_2, R_2), T_2) \rightarrow r \wedge T_2 \in T_b^{+T_1}$.
Negation Response		$\mathbf{hap}(event(Ev, X_1, A_1, R_1), T_1) \wedge c$ $\wedge \mathbf{hap}(event(Ev, X_2, A_2, R_2), T_2) \wedge r \wedge T_2 \in T_b^{+T_1} \rightarrow \bot$.

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

time t. A delegation is observable and so it is kept trace of in the execution trace.

The reactive part of $\mathcal{B}$-Tropos encompasses both the reaction to a dependency 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 [11]. Connections are translated using $\mathcal{IC}$s. For the sake of space, we refer to [4] for a detailed description on how $\mathcal{S}$CIFF handles constraints. Here we present some examples of how process-oriented constraints are formalized (Table 3). Such formulas specify the informal description given in Section 3. Response connection constraint states that if the source, $event(Ev, X_1, A_1, R_1)$, happens and the trigger condition, c, is satisfied, then the target, $event(Ev, X_2, A_2, R_2)$, is expected to happen and the restriction condition imposed on the target, r, must be satisfied. In addition, the target is expected to occur within $T_b^{+T_1}$. Weak response constraints are verified a posteriori. In particular, when the connected events happen and the triggering condition is satisfied, the restriction imposed by the target must be satisfied. Similarly to response connections, the constraint is verified if the target event occurs within $T_b^{+T_1}$. Negative response constraints spot an inconsistency when the connected events happen and all conditions are satisfied.

We remark that the framework allows one to constrain non-leaf tasks and goals, but only start and completion events of leaf tasks are considered as observable events. To address this issue, we have introduced intensional predicates **hap** and **exp** to represent the happening and expectation of (possibly) composite events. For instance, a leaf task starts (or is completed) only if there is evidence for it (i.e., the corresponding event happened). Accordingly, for a leaf-task A:

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

Composite events recursively follow the goal analysis approach:

- 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 starts (resp. expected to start);

– 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 dependency, we assume that when a dependee Y receives from a depender X a request for achieving a goal G, Y reacts by undertaking the commitment of achieving G:[7]

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

The provided formalization can be used to directly implement the skeleton of logic-based agents, as for example the ones described in [1]. Such agents follow 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 in $\mathcal{S}$CIFF-agents is equipped with the possibility to transform expectations about the agent into happened events, and with a selection rule for choosing a behavior when several choices are available. In particular, each actor represented in a $\mathcal{B}$-Tropos model can 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 (e.g., Customer Care in Fig. 1) starts by decomposing it, whereas other agents wait until an incoming request is observed. In this case, the dependency reactive rule of the agent is triggered, and the agent attempts to achieve the assigned goal. This goal may be either decomposed or delegated to other agents until expectations proving its achievement are generated. Such expectations thus are transformed to happened events, that is, actions performed by the agent.

Figure 6 presents the $\mathcal{S}$CIFF formalization corresponding to the $\mathcal{B}$-Tropos diagram of Fig. 5. Here Research & Development and Warehouse are respectively represented as $r\&d$ and wh, and symbol = is used to denote unification. In that figure one can see how the formalized $\mathcal{S}$CIFF specification is assigned to the Warehouse and R&D units. To have an intuition about how the two agents act and interact, let us consider the case in which the R&D unit intends to achieve the goal assigned by the Manufacturing division. The unit decomposes goal evaluate solution in its subparts until a set of expectations, which lead to the achievement of the goal, is determined. Below we list a possible set of expectations:

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

[7] For the sake of brevity we do not present here the reaction rule for task dependency that has the same intuition as the one for goal dependency.

$$
\begin{aligned}
KB_{r\&d} : achieve(r\&d, eval\_solution, T_i, T_f, I, O) \leftarrow & achieve(r\&d, eval\_costs, T_{i1}, T_{f1}, I_1, O_1),\\
& achieve(r\&d, eval\_resources, T_{i2}, T_{f2}, I_2, O_2),\\
& \min(T_i, [T_{i1}, T_{i2}]), \max(T_f, [T_{f1}, T_{f2}]),\\
& I = I_1 \cup I_2, O = O_1 \cup O_2.\\
achieve(r\&d, eval\_costs, T_i, T_f, I, O) \leftarrow & execute(r\&d, assess\_costs, T_i, T_f, I, O).\\
execute(r\&d, assess\_costs, T_i, T_f, I, O) \leftarrow & execute(r\&d, calc\_bill, T_{i1}, T_{f1}, I_1, O_1),\\
& execute(r\&d, eval\_bill, T_{i2}, T_{f2}, I_2, O_2),\\
& \min(T_i, [T_{i1}, T_{i2}]), \max(T_f, [T_{f1}, T_{f2}]),\\
& I = I_1 \cup I_2, O = O_1 \cup O_2.\\
execute(r\&d, calc\_bill, T_i, T_f, [], [CBill]) \leftarrow & \mathbf{E}(event(start, r\&d, calc\_bill, []), T_i),\\
& \mathbf{E}(event(end, r\&d, calc\_bill, [CBill]), T_f), T_f > T_i.\\
execute(r\&d, eval\_bill, T_i, T_f, [EBill], []) \leftarrow & \mathbf{E}(event(start, r\&d, eval\_bill, [EBill]), T_i),\\
& \mathbf{E}(event(end, r\&d, eval\_bill, []), T_f), T_f > T_i.\\
achieve(r\&d, eval\_resources, T_i, T_f, I, O) \leftarrow & \mathbf{E}(delegate(r\&d, wh, eval\_resources, T_f), T_d),\\
& achieve(wh, eval\_resources, T_d, T_f, I, O),\\
& T_d > T_i, T_d < T_f.\\
KB_{wh} : achieve(wh, eval\_resources, T_i, T_f, I, O) \leftarrow & execute(wh, find\_resources, T_i, T_f, I, O).\\
execute(wh, find\_resources, T_i, T_f, I, O) \leftarrow & execute(wh, find\_in\_wh, T_i, T_f, I, O).\\
execute(wh, find\_resources, T_i, T_f, I, O) \leftarrow & execute(wh, buy, T_i, T_f, I, O).\\
execute(wh, find\_resources, T_i, T_f, [], [Found]) \leftarrow & \mathbf{E}(event(start, wh, find\_in\_wh, []), T_i),\\
& \mathbf{E}(event(end, wh, find\_in\_wh, [Found]), T_f),\\
& T_f \geq T_i + 10, Found = yes.\\
execute(wh, buy, T_i, T_f, [], []) \leftarrow & \mathbf{E}(event(start, wh, buy, []), T_i),\\
& \mathbf{E}(event(end, wh, buy, []), T_f), T_f > T_i.
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{ICS}_{r\&d} : \mathbf{hap}(event(end, r\&d, calc\_bill, [CBill]), T_1) \rightarrow & \mathbf{exp}(event(start, r\&d, eval\_bill, [EBill]), T_2)\\
& \wedge T_2 > T_1 \wedge EBill = CBill.\\
\mathcal{ICS}_{wh} : \mathbf{H}(delegate(r\&d, wh, eval\_resources, T_f), T_d) \rightarrow & achieve(wh, eval\_resources, T_i, T_f, I, O)\\
& \wedge T_i > T_d.
\end{aligned}
$$

Fig. 6. Formalization of the $\mathcal{B}$-Tropos model fragment shown in Fig. 5

This set of expectations can be read as an execution plan, consisting of two concurrent parts: (1) a sequence of events related to 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}). It is worth noting that the framework can identify inconsistencies in temporal and/or data requirements specification by means of unsatisfiable constraints. This is, for instance, the case in which the R&D unit requires an evaluation of the availability of resources, e.g., in 5 time units, whereas the Warehouse needs at least 10 time units to verify the presence of resources. In these situations, the

designer needs either to relax constrains (e.g., extending the time) or to adopt new solutions for increasing the performance of the system (e.g., providing the Warehouse with a more efficient search application).

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 aims at proving that stakeholders can achieve their strategic goals in a given time. Such verification can also be used to evaluate different design alternatives in terms of system performance. For example, one could ask $\mathcal{S}$CIFF to verify whether an execution exists such that the top goal of one of the stakeholders is achieved within a given deadline. $\mathcal{S}$CIFF will then try to generate such an execution, by means of an intensional (i.e., partially specified) execution trace; generally speaking, this is achieved by transforming expectations into happened events. Conformance verification [4] 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, to verify whether the actual behavior of a system effectively complies with model expectations. This allows system administrators to understand whether or not stakeholders have achieved their goals and, if it is not the case, to predict future actions.

6 Related Work

While the literature on single aspects of the framework is huge (many references can be found to the papers describing Tropos, $\mathcal{S}$CIFF, and DecSerFlow/CondDec), not much work has been done at the intersection of the corresponding domains. Several formal frameworks have been developed to support the Tropos methodology. For instance, Giorgini et al. [19] proposed a formal framework based on logic programming for the analysis of security requirements. However, the framework does not take into account temporal aspects of the system. In [9] 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 is limited in defining temporal constraints among tasks. Fuxman et al. [18] 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 requirements models [18] as well as business processes against business requirements and strategic goal model [23]. However, Formal Tropos does not support abduction, and thus, it is not able to generate expectations and perform conformance checking between expectations and happened events. Finally, we mention the work by Cares et al [10], who proposed to implement

software agents in Prolog starting from Tropos models. In particular, they proposed to specify the programming activation time through four implementation attributes, namely at begin, at end, at call, and always. The difference with our proposal lies in the generation of implementation besides the employed temporal constructs. Actually, they do not provide an encoding of Tropos models into Prolog so that the implementation is manual.

The last years have seen the need for bridging the gap between requirements engineering and business process design by providing support for developing business processes on top of requirements models and verifying whether a business process actually meets its business goals. For instance, Lapochnian et al. [25] proposed a systematic requirements-driven approach for business process design and configuration management, which adopts goal models to capture alternative process configurations. Differently from our work, they do not consider the relationships between agents so that framework is inadequate to describe business processes spanning across multi-agent systems. Frankova et al. [16] have used the SI* modeling language [27], an extension of Tropos addressing security and privacy issues, as a basis for the definition of Secure BPEL, a specification language that extends WS-BPEL [6] for modeling secure business processes. The objective of this framework is to assist business process analysts in deriving the skeleton of secure business processes from early requirements analysis. Finally, López et al. [26] presented a reasoning method for verifying the consistency between SI* models and BPMN specifications [7]. In particular, the authors have investigated the connection between business processes and requirements models, introducing the notion of goal equivalence based on trace semantics.

Several works also attempt to define a formal semantics underlying graphical business process models and to design agent systems. In the business process domain, Wong et al. [37] provided a formal semantics for a subset of BPMN in terms of the process algebra CSP [30], whereas Dijkman et al. [15] used Petri Nets [28]. Their objective is to formally analyze and compare business process models. We differ from these proposals since the objective of our work is to provide a requirements-driven framework for business process and agent system design. The use of computational logic for the flexible specification and rigorous verification of agent interaction is adopted by many proposals. While other works (e.g., [36]) use temporal logic to model the temporal dimension of interaction, $\mathcal{S}$CIFF exploits a constraint solver and adopts an explicit representation of time.

Event Calculus [24] was introduced by Kowalsky and Sergot as a logic programming formalism for representing events and their effects. This formalism explicitly reasons upon properties (fluents) holding during time intervals. Differently from Event Calculus, our framework treats time like other variables, in association with domains, which makes it possible to express constraints (e.g., deadlines) and to exploit an underlying constraint solver. Among the works based on Event Calculus, we cite the work by Shanahan [32], who proposed the abductive event calculus that includes the concept of expectation, and the work by Cicekli et al. [12], who formalized workflows using Event Calculus. In Shanahan's work events and expectations are of the same nature and both are

abduced, while our expectations should match the actual events. This is due to the different underlying assumptions and, consequently, the different focus: while we assume that the history is known, Shanahan proposes to abduce events. Similarly to [15, 37], Cicekli et al. focus on the execution of business processes, whereas the reconciliation between a business process and the business goals that have motivated the process definition are completely ignored.

Finally we mention that a mapping of DecSerFlow into Linear Temporal Logic (LTL) [29] has been proposed in [34]. It can be used to verify or enforce conformance of service flows and also to directly enact their execution. The advantages of using SCIFF instead of LTL is that SCIFF can handle time and data in an explicit and quantitative way, exploiting CLP to define temporal and data-related constraints.

7 Conclusions

In this work we have proposed to integrate a number of techniques for information systems engineering, with the aim to reconcile requirements elicitation with specification, prototyping and analysis, inside a single unified framework. We have presented $\mathcal{B}$-Tropos, an extension of Tropos with declarative process-oriented constraints, and its mapping into the $\mathcal{S}$CIFF language. We have mainly focused on the modeling and mapping of aspects related to declarative business processes using connections inspired by DecSerFlow and ConDec languages. Augmenting a Tropos model with such constraints has the effect that both the proactive and the reactive, process-oriented agent behavior could be captured within the same diagram.

The mapping of $\mathcal{B}$-Tropos onto $\mathcal{S}$CIFF makes it possible to directly implement logic-based agents starting from the enhanced Tropos 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 is a first step towards the integration of a business process in the requirements model. We are currently running experiments on prototyping as well as on property and conformance verification. Some results are presented in [13], where $\mathcal{B}$-Tropos models are also used to generate possible executions traces, and to animate agents in the context of the CLIMA Contest Food Collection problem [14], in line with the aforementioned work by Cares and colleagues [10]. We are also investigating in depth the formal properties of our proposed mapping, and are trying to understand how to better exploit the underlying $\mathcal{S}$CIFF constraint solver by introducing more complex scheduling and resource constraints so as to capture more detailed business requirements and agent interactions. As a future activity, we plan to investigate the generation of executable business process specifications (such as WS-BPEL) from $\mathcal{B}$-Tropos models. Another direction under investigation concerns business process compliance. In particular, we are interested in the problem of the interplay between business and control objectives during business process design [31]. Finally, we

intend to conduct empirical studies on large scale, industrial size case studies for a practical evaluation of the framework.

References

1. M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, and P. Mello. A Verifiable Logic-Based Agent Architecture. In *Proc. of ISMIS'06*, LNCS 4203, pages 188–197. Springer, 2006.
2. M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Security Protocols Verification in Abductive Logic Programming: A Case Study. In *Proc. of ESAW'05*, LNCS 3963, pages 106–124. Springer, 2005.
3. M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Compliance verification of agent interaction: a logic-based software tool. *Applied Artificial Intelligence*, 20(2-4):133–157, 2006.
4. M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Verifiable Agent Interaction in Abductive Logic Programming: the SCIFF framework. *TOCL*, 2007. Accepted for publication.
5. J. Allen and G. Ferguson. Actions and events in interval temporal logic. *Journal of Logic and Computation*, 4(5), 1995.
6. Web Services Business Process Execution Language (WS-BPEL) Version 2.0. OASIS Standard, April 11, 2007, Organization for the Advancement of Structured Information Standards, 2007.
7. Business Process Modeling Notation (BPMN) Specification. OMG Final Adopted Specification, February 6, 2006, Object Management Group, 2006.
8. P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. TROPOS: An Agent-Oriented Software Development Methodology. *JAAMAS*, 8(3):203–236, 2004.
9. V. Bryl, F. Massacci, J. Mylopoulos, and N. Zannone. Designing Security Requirements Models through Planning. In *Proc. of CAiSE'06*, LNCS 4001, pages 33–47. Springer, 2006.
10. C. Cares, X. Franch, and E. Mayol. Extending Tropos for a Prolog Implementation: A Case Study Using the Food Collecting Agent Problem. In *Proc. of CLIMA VI*, LNCS 3900, pages 396–405. Springer, 2006.
11. F. Chesani, P. Mello, M. Montali, and S. Storari. Towards a DecSerFlow declarative semantics based on computational logic. Technical Report DEIS-LIA-07-001, University of Bologna, Italy, Jan. 2007. LIA Series no. 79.
12. N. K. Cicekli and Y. Yildirim. Formalizing Workflows Using the Event Calculus. In *Proc. of DEXA'00*, pages 222–231. Springer, 2000.
13. D. Corapi. Traduzione di un linguaggio per l'ingegneria dei requisiti orientato agli agenti in logica computazionale. Technical Report DEIS-LIA-07-007, University of Bologna, Italy, Oct. 2007. Master's Thesis. LIA Series no. 85. In Italian.
14. M. Dastani, J. Dix, and P. Novák. The First Contest on Multi-agent Systems Based on Computational Logic. In *Proc. of CLIMA VI*, LNCS 3900, pages 373–384. Springer, 2005.
15. R. Dijkman, M. Dumas, and C. Ouyang. Formal Semantics and Automated Analysis of BPMN Process Models. Preprint 7115, Queensland University of Technology, 2007.
16. G. Frankova, F. Massacci, and M. Seguran. From Early Requirements Analysis towards Secure Workflows. In *Proc. of IFIPTM'07*,

2007. The full version appears as Technical Report, DIT-07-036, at http://eprints.biblio.unitn.it/archive/00001220/.
17. T. H. Fung and R. A. Kowalski. The IFF proof procedure for abductive logic programming. *Journal of Logic Programming*, 33(2):151–165, 1997.
18. A. Fuxman, L. Liu, J. Mylopoulos, M. Pistore, M. Roveri, and P. Traverso. Specifying and Analyzing Early Requirements in Tropos. *REJ*, 9(2):132–150, 2004.
19. P. Giorgini, F. Massacci, and N. Zannone. Security and Trust Requirements Engineering. In *FOSAD 2004/20005*, LNCS 3655, pages 237–272. Springer, 2005.
20. B. Henderson-Sellers and P. Giorgini, editors. *Agent-Oriented Methodologies*. Idea Group Publishing, 2005.
21. J. Jaffar and M. Maher. Constraint logic programming: a survey. *Journal of Logic Programming*, 19-20:503–582, 1994.
22. A. C. Kakas, R. A. Kowalski, and F. Toni. Abductive Logic Programming. *Journal of Logic and Computation*, 2(6):719–770, 1993.
23. R. Kazhamiakin, M. Pistore, and M. Roveri. A Framework for Integrating Business Processes and Business Requirements. In *EDOC'04*, pages 9–20. IEEE Press, 2004.
24. R. Kowalski and M. Sergot. A logic-based calculus of events. *New Gen. Comput.*, 4(1):67–95, 1986.
25. A. Lapouchnian, Y. Yu, and J. Mylopoulos. Requirements-Driven Design and Configuration Management of Business Processes. In *Proc. of BPM'07*, LNCS 4714, pages 246–261. Springer, 2007.
26. H. A. López, F. Massacci, and N. Zannone. Goal-Equivalent Secure Business Process Re-engineering. In *Proc. of SeMSoC'07*, 2007.
27. F. Massacci, J. Mylopoulos, and N. Zannone. An Ontology for Secure Socio-Technical Systems. In *Handbook of Ontologies for Business Interaction*. The IDEA Group, 2007.
28. J. L. Peterson. Petri Nets. *ACM Comput. Surv.*, 9(3):223–252, 1977.
29. A. Pnueli. The Temporal Semantics of Concurrent Programs. In *Proc. of the International Symposium on Semantics of Concurrent Computation*, pages 1–20. Springer, 1979.
30. A. W. Roscoe, C. A. R. Hoare, and R. Bird. *The Theory and Practice of Concurrency*. Prentice Hall PTR, 1997.
31. S. W. Sadiq, G. Governatori, and K. Namiri. Modeling control objectives for business process compliance. In *Proc. of BPM'07*, LNCS 4714, pages 149–164. Springer, 2007.
32. M. Shanahan. Reinventing shakey. In *Logic-based artificial intelligence*, pages 233–253. Kluwer Academic Publishers, 2000.
33. W. M. P. van der Aalst and M. Pesic. A declarative approach for flexible business processes management. In *Proc. of BPM'06*, LNCS 4103, pages 169–180. Springer, 2006.
34. W. M. P. van der Aalst and M. Pesic. DecSerFlow: Towards a truly declarative service flow language. In *WS-FM'06*, LNCS 4184. Springer, 2006.
35. W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.
36. M. Venkatraman and M. P. Singh. Verifying Compliance with Commitment Protocols. *JAAMAS*, 2(3):217–236, 1999.
37. P. Wong and J. Gibbons. A Process Semantics for BPMN, 2007. Preprint, Oxford. University Computing Laboratory. Available at `http://web.comlab.ox.ac.uk/`.