Auton Agent Multi-Agent Syst (2011) 23:193–223
DOI 10.1007/s10458-010-9135-4

Engineering and verifying agent-oriented requirements augmented by business constraints with $\mathcal{B}$-Tropos

Marco Montali · Paolo Torroni · Nicola Zannone · Paola Mello · Volha Bryl

Published online: 14 May 2010

Abstract We propose $\mathcal{B}$-Tropos as a modeling framework to support agent-oriented systems engineering, from high-level requirements elicitation down to execution-level tasks. In particular, we show how $\mathcal{B}$-Tropos extends the Tropos methodology by means of declarative business constraints, inspired by the ConDec graphical language. We demonstrate the functioning of $\mathcal{B}$-Tropos using a running example inspired by a real-world industrial scenario, and we describe how $\mathcal{B}$-Tropos models can be automatically formalized in computational logic, discussing formal properties of the resulting framework and its verification capabilities.

1 Introduction

Requirements engineering provided a number of approaches that address various aspects of information systems modeling. In particular, agent-oriented approaches support the understanding of the organizational setting in which a system will operate [9,13,24,52], and goal-oriented approaches support the modeling and analysis of stakeholders' strategic goals and thus they allow one to represent the rationale beyond the introduction of the system and the

M. Montali · P. Torroni (✉) · P. Mello
University of Bologna, Bologna, Italy
e-mail: paolo.torroni@unibo.it

M. Montali
e-mail: marco.montali@unibo.it

P. Mello
e-mail: paola.mello@unibo.it

N. Zannone
Eindhoven University of Technology, Eindhoven, The Netherlands
e-mail: n.zannone@tue.nl

V. Bryl
FBK-IRST, Povo, TN, Italy
e-mail: bryl@fbk.eu

design choices made [7,16,17]. Some requirements engineering methodologies have also been used to address the definition of business processes [12,27,35], but most of them result to be inadequate to represent all aspects of business processes, especially when spanning across different organizations (or different divisions of the same organization) [8,31].

Early requirements engineering techniques such as those we will refer to in this paper, are concerned with early stage requirements elicitation. At the far end of information systems design, we find implemented systems, and tools that support synthesis (system prototyping and simulation), run-time execution, monitoring and analysis. In between these two activities of information systems development, a number of other activities span from high-level architectural to low-level detailed design. Among them we find business process modeling and specification. Each one of these design activities typically relies on its own tools, languages and methodologies. The differences and variety in the approaches are justified by the different aims that each activity has. Early requirements focus on "why" questions, in order to support the definition of a model of an information system and on the corresponding goal model. Declarative business process modeling focuses on "what" questions, to specify in a declarative way the business process of an information system starting from a high-level model. The purpose of operational model development and tools is to throw bridges between "what" is declaratively specified and "how" this is operationally achieved, and to support execution-level tasks such as prototyping and system monitoring and analysis. Such heterogeneous aspects are typically not to be found in a single unified design framework.

Business process modeling is a fundamental part of the software development process [37]. However, understanding whether or not defined business processes provide support for the business strategies of an organization is still a challenge [8]. This issue has been partially addressed in both requirements engineering and business analysis research areas, but separately. The lack of interaction between these two areas caused that the duties of requirements engineers and business analysts, and their role in the software development process are not clear [43]. For instance, Haglind et al. addressed the issue of the integration of business analysis and requirements engineering by means of the impact that business analysis has on ensuring requirements engineering activities [23].

To allow for a systematic design of business processes, we have to understand the business goals and requirements of an organization, its structure and the dependencies among business partners, and then link business processes to business goals [31]. Many problems might also arise from organizational theory and strategic management perspectives due to limits on particular resources (e.g., cost, time, etc.). In this setting, if business processes are defined before identifying business goals and eliciting business requirements, the defined business processes may not meet the actual needs and capabilities of individual business partners participating in the process.

In our previous works [11], we have proposed $\mathcal{B}$-Tropos to facilitate the interaction between requirements engineers and business analysts in order to bridge the gap between requirements analysis and business process modeling. In particular, we proposed to model and analyze business goals and then define business processes upon goal models. To this end, we have extended Tropos [9], an agent-oriented, goal-oriented software engineering methodology, with declarative business process-oriented constructs inspired by DecSerFlow [46] and ConDec [45]. One of the main features of Tropos is the prominent role given to early requirements analysis that concerns the understanding of the domain by studying the organizational setting within which the system will operate. However, a drawback of Tropos, as well as of many other agent-oriented and goal-oriented approaches, is that it does not clearly define how to move from requirements models to business process models. 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 [29]. The integration with declarative business process languages provides Tropos with these capabilities. In [11] we have also shown how these complementary aspects (i.e., agent-, goal-, and process-oriented) can be formalized in the $\mathcal{S}$CIFFframework [6], a computational logic-based framework for the specification and verification of interaction protocols in an open multi-agent setting. In particular, we have demonstrated how the mapping of $\mathcal{B}$-Tropos into $\mathcal{S}$CIFF can be used to implement the skeleton of logic-based agents.

Since [11] was published, the application of the framework to a case study in collaboration with industrial partners has allowed us to evaluate its expressiveness and usability, and prove its applicability in industry. The attempt to capture and analyze the issues raised by the case study has pointed out a number of drawbacks in our initial proposal, which have demanded for a revision and an extension of $\mathcal{B}$-Tropos. In particular, bringing $\mathcal{B}$-Tropos up to capturing challenges of the case study has required to:

- revise the modeling constructs to increase the readability and manageability of the framework by industry partners;
- provide industry partners with analysis tools.

This paper presents a comprehensive and consolidated description of $\mathcal{B}$-Tropos that addresses the challenges raised from its application to the case study together with theoretical results. In particular, the paper presents:

- a consolidated version of the $\mathcal{B}$-Tropos modeling framework;
- a complete mapping of $\mathcal{B}$-Tropos constructs into $\mathcal{S}$CIFF specifications;
- a proof of the soundness, completeness and termination of the $\mathcal{S}$CIFF and g-$\mathcal{S}$CIFF proof procedures when reasoning on $\mathcal{B}$-Tropos models;
- a method for conformance and property verification.

The overall framework sets up a link between specification, simulation/execution and analysis. In particular, $\mathcal{B}$-Tropos allows requirements engineers to implement and execute logic-based agents [1], as well as to perform different verification tasks, such as verification of declarative business process specifications [40] and conformance verification of Web service choreographies [2], spanning from static verification of consistency and properties to run-time conformance verification, monitoring and a-posteriori analysis of the execution traces generated during the interaction. Simulation and analysis (property and conformance verification) constitute an important part of the system development process and the maintenance of deployed systems. Simulation and property verification allow requirements engineers and business analysts to test their models directly and get an immediate picture of the model being developed. Conformance verification allows system administrators to monitor the correct behavior of a running system using a $\mathcal{S}$CIFF specification of the system being automatically generated from the $\mathcal{B}$-Tropos model, and then, based on such specification, automatically check the compliance of the system using the SOCS-SI runtime and offline checking facilities [4]. The possibility to unify all these aspects of modelling, formal specifications, prototyping and verification in a single framework and with a single specification language is what makes $\mathcal{B}$-Tropos unique.

To make the discussion more concrete, the proposed approach is applied to an intra-enterprise organizational model, focusing on the coordination of economic activities among different units of an enterprise collaborating together in order to produce a specific product. This is an excerpt of a real case study analyzed in the context of the TOCAI project.[1]

[1] FIRB-TOCAIRBNE05BFRK—http://www.dis.uniroma1.it/tocai/.

This project aims at analyzing novel enterprise organizational models of integration, coordination, cooperation and interoperability, and their possible enhancement related to the integration of IT technologies in organizational processes and, in particular, in production processes.

The paper is organized as follows. Section 2 describes our process-oriented extensions of Tropos, and introduces an intra-enterprise organizational model used as a running example to explain the framework presented in this paper. Section 3 presents the $\mathcal{S}$CIFF framework and defines the mapping of $\mathcal{B}$-Tropos concepts to $\mathcal{S}$CIFF specifications, whereas Sect. 4 discusses formal properties of the mapping. Section 5 discusses how the proposed formal framework can be used to verify $\mathcal{B}$-Tropos models. The paper ends with an overview of related work and conclusive remarks in Sects. 6 and 7, respectively.

2 The $\mathcal{B}$-Tropos modeling framework

This section presents the agent-oriented and goal-oriented approach adopted by Tropos and discuss how such an approach can be augmented with a high-level reactive, process-oriented dimension. We call Tropos extended with declarative $\mathcal{B}$usiness process-oriented constructs $\mathcal{B}$-Tropos [11].

2.1 Tropos

Tropos [9] is an agent-oriented software engineering methodology developed to support the analysis of both the system-to-be and its organizational environment along the whole system development process. One of its main advantages is the importance given to early requirements analysis. This allows one to capture *why* system functionalities are required, 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* is an artifact that is consumed or produced by a task. A *dependency* between two actors indicates that one actor (the *depender*) depends on another actor (the *dependee*) for achieving some goal, executing some task, or furnishing some resource (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 are represented by edges marked by a solid triangle, connecting the *depender*'s *dependum* with the *dependee*.[2] Note that Tropos elements can appear in several instances inside the model. For example, if R is a resource of two actors, it will be represented by a rectangle inside each one of the circles symbolizing such actors. These two rectangles represent in fact a single output.

Requirements analysis in Tropos is conduced at two levels: strategic level and operational level. At the *strategic level*, the actors within the system are identified along with their goals and the inter-dependencies among them. Starting from this high-level view of a system, the analysis proceeds with an incremental refinement process. This process starts with a goal analysis where high-level goals are AND/OR refined into subgoals. In particular, *AND-decomposition* is used to define at high level the process for achieving a goal, whereas *OR-decomposition* defines the alternatives for achieving a goal.

[2] For ease of reading, we display actor names in bold face.

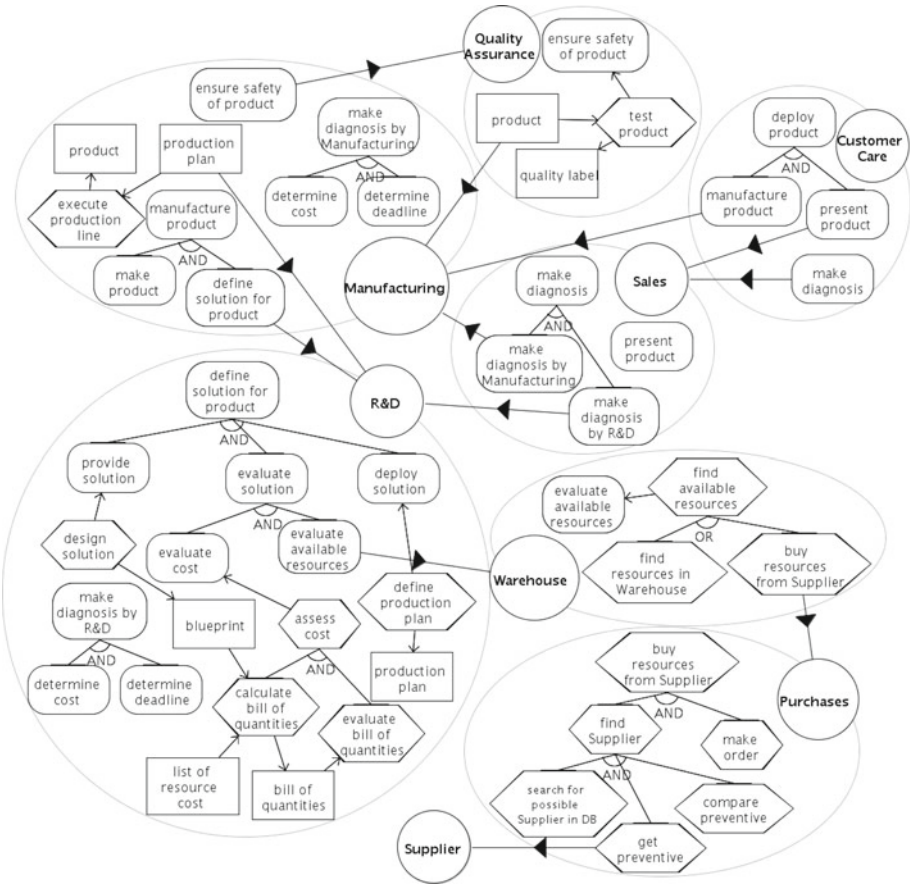


Fig. 1 Product development process in tropos

Requirements analysis at *operational level* focuses on what, where, and how system functionalities will be organized, integrated, and employed to achieve strategic goals. At this level requirements analysis establishes the tasks needed to accomplish the strategic objectives and the resources involved in their execution. In particular, *means-end analysis* is adopted to identify tasks used to achieve a goal and the resources consumed and produced by tasks. Together with means-end analysis, this phase uses AND/OR decomposition for refining requirements models, but here the focus of such analysis is on tasks. The requirements analysis process results in a graph, called *goal/task model*, whose nodes are goals and tasks, and whose edges are AND/OR decomposition and means-end relations linking tasks and goals together.

A sample Tropos model. Figure 1 shows a fragment of the Tropos diagram representing a product development process scenario, which has been modeled and analyzed in the course of the TOCAI project. In this scenario, the Customer Care and Manufacturing divisions have to cooperate in order to define the most suitable solution for a product and plan manufacturing activities. The Purchases and Sales divisions are involved in identifying different purchasing strategies and presenting the solution for a product, respectively. A key component of the enterprise is the Research & Development (R&D) unit. The R&D department deals with new technology development and existing products improvement. It is also in charge of

responding and anticipating customer needs and maintaining the competitive advantage. We refer to [32] for the entire model.

As shown in the model in Fig. 1, different divisions of a company have to cooperate in order to produce a specific product. The Customer Care division (top-right circle) is responsible for achieving goals make diagnosis and deploy product (represented by ovals inside the rationale of the actors) to customers. Customer Care depends on the Sales division for achieving make diagnosis (such a dependency is denoted by a connecting edge marked by a solid triangle). In turn, Sales appoints both R&D and Manufacturing divisions to achieve the assigned duty. These divisions decompose the goal into subgoals determine cost and determine deadline (see the AND-decomposition notation, marked by an "AND" label). Goal deploy product is refined by Customer Care 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 R&D, and make product, which it achieves through task execute production line (note the arrow from hexagon/task execute production line to oval/goal make product, denoting a means-end relation). 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). This task produces a production plan (note the means-end relation from task to resource), which is used by Manufacturing to execute production line. The evaluation of the solution is performed in terms of costs and available resources. To evaluate costs, R&D executes task assess costs, which consists in calculating bill of quantities and evaluating bill of quantities. The execution of calculating bill of quantities requires the blueprint produced by task design solution and the list of resources costs, and produces the bill of quantities that is analyzed by task evaluating bill of quantities. Moreover, R&D depends on the Warehouse for evaluate available resources. The Warehouse either queries the databases to find available resources or asks the Purchases division to buy resources from External Supplier. Purchases searches in the company's databases for possible Suppliers and selects the one who provides the best offer. Manufacturing is also responsible for achieving goal ensure safety of product, for which it depends on Quality Assurance. Essentially, before presenting the product to the customer, Quality Assurance achieves the appointed goal by means of task test product. Quality Assurance executes this task by taking the product produced by task execute production line and, in case of successful test results, produces a quality label. (Note that product, being a resource, is represented as a rectangle; it is present inside both Quality Assurance and Manufaturing, but it represents a single output).

Note that some parts of the diagram may be left unspecified. An extreme example is the external supplier, simply denoted by its name inside a circle. In general, Tropos diagrams are very high-level and considerably under-specified, to leave the focus on the global picture.

2.2 Constraint-based business process modeling

Tropos gives special attention to early system requirements, namely, to modeling and analyzing the organizational setting in which a system will operate, and thus, capturing the rationale behind the system functionalities. However, exactly because of Tropos's focus being far from operational modelling, moving from requirements to business process model in Tropos is problematic. Though in Fig. 1 we notice a correspondence between the tasks identified during the operational modeling phase and the activities characterizing business processes, Tropos lacks constructs suitable to interrelate them. In particular, a business process must

have clearly defined boundaries in terms of input and output and must consist of ordered activities [29]. Looking at Fig. 1 we can see that Tropos can only partially cope with the first issue and fails to accomplish the second one entirely.

A business process modeling approach, which shares the philosophy underlying Tropos, is the one of ConDec/DecSerFlow [45,46], two graphical languages proposed by van der Aalst and Pesic to represent service flows and flexible business processes in a declarative and graphical way. As with Tropos, ConDec and DecSerFlow aim at modeling the domain under study without over-specifying and over-constraining the model under development. To this end, instead of imposing a complete and rigid control-flow specification, they adopt a declarative style of modeling, helping the designer in the identification and modeling of the (minimal set of) business constraints that must be respected in order to correctly carry out the execution. Such an approach fits better than procedural ones with complex, unpredictable processes, such as those found in real-world scenarios, where a good balance between support and flexibility is of key importance [40,45]. Furthermore, since the approach guarantees what is called *flexibility by design* [44], it is able to support the modeler in the identification of early business requirements, abstracting away from implementation details.

Even if ConDec targets the business domain while DecSerFlow is focused on service interaction, the two languages follow the same principles and share the graphical notation. A ConDec/DecSerFlow model is composed by a set of activities, interconnected by business constraints. Unconstrained activities can be executed an arbitrary number of times, in an arbitrary order: the languages follow an *open* approach. Constraints can be imposed to regulate the execution of the involved activities; being the languages open, both positive and negative relationships can be imposed. A ConDec/DecSerFlow model *supports* all the execution traces which comply with all the constraints included in the model.

Positive relationships are used to state that a certain activity must be executed in a given situation, while negative relationships forbid the execution of an activity when a given situation holds. Different types of positive and negative constraints can be imposed; the difference between them relies in the degree of freedom they leave on the execution, for what regards both the temporal dimension and the repetition of the involved activities. For example, the *responded presence* constraint states that if the source activity is executed, then the target activity must be executed as well, either before or afterwards. The *response* constraint refines the *responded presence* one by imposing a temporal ordering between the two activities, i.e., by stating that the target activity must be executed after the source one.

The constraints semantics has been originally formalized in terms of Linear Temporal Logic (LTL), and then also by means of $\mathcal{S}$CIFF constraints, based on Abductive Logic Programming [38,39,45,46]. This unlocks the possibility of using a number of existing reasoning and verification techniques to verify, execute and monitor the developed models (see [19] for a survey on LTL verification tools and Sect. 3.1 for a presentation of $\mathcal{S}$CIFF-based verification methods).

While the LTL formalization can only be used to specify basic control-flow constraints, $\mathcal{S}$CIFF can also be used to specify new constraint features, such as data conditions and quantitative time constraints [39]. In this paper we will therefore rely on the $\mathcal{S}$CIFF formalization.

2.3 The $\mathcal{B}$-Tropos notation

Defining a business process and establishing its compliance with the business requirements that have motivated its definition is a challenging task. In this section we discuss how we extended Tropos to obtain a tool for building high-level, declarative business process models on top of requirements models. Typically, declarative business process specification

languages focus on activities and their relationships. The proposed extensions intend to support designers in defining durations, and absolute time constraints on goals and tasks as well as specifying temporal and data constraints among them.

The following definitions are needed to introduce the further description of $\mathcal{B}$-Tropos's extensions to Tropos.

Definition 2.1 (*Time interval*) A time interval over a numerical domain $\mathcal{D}$ is a defninite length of time marked off by two (non-negative) instants (T_{min} and $T_{max} \in \mathcal{D}$), 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:

- $(T_{min}, T_{max}) \equiv \{T \in \mathcal{D} \mid T > T_{min} \wedge T < T_{max}\}$
- $(T_{min}, T_{max}] \equiv \{T \in \mathcal{D} \mid T > T_{min} \wedge T \leq T_{max}\}$
- $[T_{min}, T_{max}) \equiv \{T \in \mathcal{D} \mid T \geq T_{min} \wedge T < T_{max}\}$
- $[T_{min}, T_{max}] \equiv \{T \in \mathcal{D} \mid T \geq T_{min} \wedge T \leq T_{max}\}$

Definition 2.2 (*Time interval shift*) Given a time interval $TI = [T_{min}, T_{max}]$ and a time variable T,

- TI^{+T} is the *positive shift* of TI by T, and it corresponds to the absolute time interval marked off by $T_{min} + T$ and $T_{max} + T$;
- TI^{-T} is the *negative shift* of TI by T, and it corresponds to the absolute time interval marked off by $T_{min} - T$ and $T_{max} - T$.

Being defnined in terms of another interval (TI in particular), TI^{-T} and TI^{+T} are said to be *relative* time intervals.

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

Definition 2.3 (*Absolute time constraint*) An absolute time constraint is a binary constraint of the form T **OP** i, where T is a time variable, i is a time instant and **OP** $\in \{at, after, after\_or\_at, before, before\_or\_at\}$ (with their intuitive meaning).

Instead of using a numerical value for i, we will often use a "date" notation, assuming that it is implicitly translated to its corresponding numerical representation (e.g., by applying a transformation to milliseconds).

Definition 2.4 (*Data-based decision constraint*) A data-based decision constraint is a boolean expression that formalizes a data-driven choice. It can be specified in terms of a CLP (Constraint Logic Programming) [28] constraint (e.g., $=$, $>$, $<$, etc.) or a Prolog predicate.

Definition 2.5 (*Condition*) A condition is a conjunction of data-based decision constraints and absolute time constraints.

For example, condition T *before_or_at* 11.06.2007 $\wedge$ $workingDay(T)$ states that T has November 6, 2007 as a deadline, and that it must be a working day.

Tasks/Goals extension. In order to support the modeling and analysis of process-oriented aspects of a system, we have annotated goals and tasks with temporal information such as

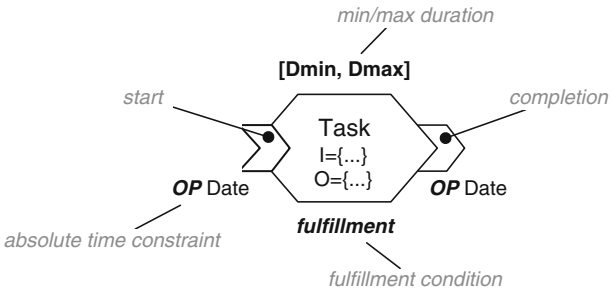


Fig. 2 Extended notation for tasks. Note that the absolute time constraints attached to the start/completion of the task lacks the left operand, which is implicitly bound to the time at which the task is started/completed

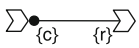
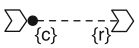
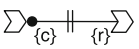
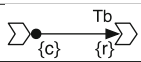
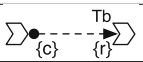
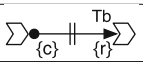
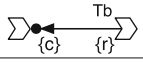
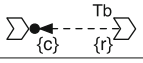
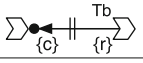
start and *completion* times. The notation for tasks is shown in Fig. 2 (the one for goals is identical except for the hexagon, which is replaced by an oval). Each task/goal can also be described in terms of its allowed *duration* ($[D_{min}, D_{max}]$ in Fig. 2). This allows one to constrain, for instance, the completion time to the start time, i.e., $completion\ time \in [D_{min}, D_{max}]^{+start\ time}$. Additionally, absolute temporal constraints can be used to defnine start and completion times of goals and tasks.

In the original Tropos proposal, resources needed to execute a task and produced by the execution of a task are linked to the task using means-end relations (see Sect. 2.1). Conversely, in $\mathcal{B}$-Tropos resources are treated as parameters that drive the execution of a business process. The advantage of this representation is that we can propagate resources bottom-up and uderstand which resourses are needed to achieve a goal and which is the outcome of the achievement of a goal. Along this direction, we represent resources as attributes of tasks and goals. In particular, each task/goal is associated with attribute I which specifies the list of resources consumed by the task, or needed to achieve the goal, and with attribute O, which specifies the list of resources produced as an outcome of task execution, or goal achievement. To indicate that the output of a task is taken in input by another task and to represent resource dependencies, we use process-oriented constraints as shown in the next section. Finally, tasks can be annotated with a *fulfillment* condition, which defnines when they are successfully executed.

Process-oriented constraints. To bring a high-level and declarative process-oriented view into a requirements model, we introduce different connections between goals and tasks, namely *relation*, *weak relation*, and *negation* connections (see Table 1). These connections allow requirements engineers and business analysts to specify partial ordering between goals/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 times. A small circle is used to denote the connection source, which determines when the triggering condition is satisfied.

Relation and negation connections are based on DecSerFlow [46] and ConDec [45] template formulas, extended with conditions (i.e., conjunctions of temporal and data constraints) [38]. Conditions can be specified on both start and completion time and are delimited by curly braces ($\{c\}$ and $\{r\}$ in Table 1). For instance, a condition may be $\{CBill = EBill\}$, indicating that a match between variables $CBill$ and $EBill$, such as unification or a form of equivalence, must be established. The source condition is a triggering condition, whereas the target condition is verified a posteriori.

Table 1 Tropos extensions to capture process-oriented constraints

	relation	weak relation	negation
responded presence			
response			
precedence			

A *responded presence* relation specifies that if the source happens such that c is satisfied, then the target has to happen and satisfy r. In LTL, by identifying with s the source activity and with t the target activity and by abstracting away from the conditions c and r, the constraint is formalized by the formula $\Diamond s \rightarrow \Diamond t$.[3] The other two relations extend the responded presence relation by specifying a temporal ordering between source and target events.

A *response* relation constrains the target to happen *after* the source. Therefore, it requires that if the source happens, the target has to happen after it: $\Box(s \rightarrow \Diamond t)$ A *precedence* relation constrains the source to happen *before* the target. In other words, it specifies that the target may not happen until the source happens: $\neg t \, \mathcal{U} s$. A relative time interval (denoted with T_b in Table 1) can also be attached to these relations. This time interval binds the expected time of the target to the time at which the source happened.[4] For instance, when T_b is specified in a response relation, the target should happen between the minimum and the maximum time after the source, i.e., $target\ time \in T_b^{+source\ time}$. In this way, the modeler can express different temporal constraints such as delays and deadlines, in a point algebra setting [48]. For example, attaching the time interval $[0, 7]$ to a response relation means that the target must occur after at most 7 time units after the source (deadline); conversely, the time interval $[8, \infty]$ can be applied to state that at least 8 time units must separate the execution of the source and the target (delay).

As in DecSerFlow and ConDec, we adopt an open approach, that is, we explicitly specify not only what is expected, but also what is forbidden. What is neither explicitly expected nor forbidden is implicitly allowed. This level of expressivity, not achieved by procedural business process notations, such as BPMN,[5] is one of the distinguishing features of declarative business process languages. These "negative" dependencies are represented by *negation connections* that are the counterpart of relation connections (last column of Table 1).

Summarizing, relation and negation connections allow system designers to add a horizontal declarative and high-level process-oriented dimension to the vertical goal-directed dimension. Note that, in the presence of OR decompositions, adding connections may affect the semantics of the requirements model. Figure 3 shows that 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

[3] We use literature notation for LTL operators. In particular, unary operators $\Diamond$ and $\Box$ respectively mean *at some future moment* and *at every future moment*, and binary operator $\mathcal{U}$ means *until*. Classical logic operators $\neg$ and $\rightarrow$ in LTL formulae respectively represent negation and implication. For a smooth introduction to temporal logic concepts and tools see [19].

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

[5] The Business Process Modeling Notation, see http://www.bpmn.org/.

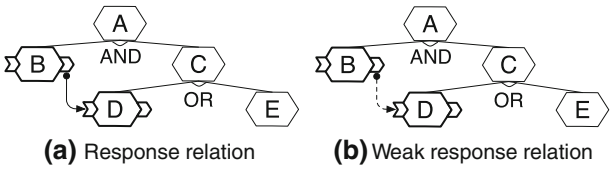


Fig. 3 Integrating process-oriented and goal-directed dimensions in $\mathcal{B}$-Tropos

mandatory (B has to be performed because of the AND-decomposition of A, hence D is also expected after B). This situation should be avoided. To this end, we have introduced *weak relation* connections that relax relation connections. Their intended meaning is: whenever both the source and the target happen, the target must satisfy the connection semantics and the corresponding restriction. The main difference between relations and weak relations is that weak relations are always considered a posteriori, that is, after both source and target have happened. Differently from Fig. 3, in Fig. 3 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 and goals, leading to the possibility of expressing some process-oriented patterns [47]. For instance, a relation connection whose source is the completion of a task, which is AND-decomposed, triggers when all subtasks have been executed. Therefore, the connection resembles the concept of synchronizing merge on leaf tasks.

To show how $\mathcal{B}$-Tropos models are annotated with process-oriented constraints, we have extended the requirements model presented in Fig. 1. An excerpt of the extended model is shown in Fig. 4. Some constraints concern the ordering of activities. For instance, R&D first provides a solution for the product, then evaluates such a solution, and finally deploys it. It is evident now that these activities should be executed sequentially. Other constraints aim at binding the input and output of different activities. For instance, bill of quantities produced by task calculate bill of quantities is passed in input to task evaluate bill of quantities. The ordering constraints can also be inter-actor ones. In particular, this happens when there is a resource dependency between two actors. This is the case, for instance, of Quality Assurance, which depends on Manufacturing for the product, and of Manufacturing, which depends on R&D for the production plan. Other extensions have the purpose of better detailing tasks. For instance, task buy resources from external supplier is associated with the maximum allowed duration, that is, the time by which the task must be completed.

We remark that building a business process model on top of the requirements model is a refinement process. Therefore, the designer may need to specify additional information in order to better characterize the tasks. For instance, to decide if task find resources in Warehouse has been successfully executed, the designer might introduce a datum (Found in Fig. 4), which describes whether or not resources have been found in the Warehouse within a certain time. Based on it, the designer can specify a condition stating when the task succeeds. In our example, task find resources in Warehouse is successfully executed only if Found evidence has been produced within a certain time.

3 $\mathcal{B}$-Tropos semantics in $\mathcal{S}$CIFF

In this section we present the $\mathcal{S}$CIFF framework and a mapping of $\mathcal{B}$-Tropos concepts into $\mathcal{S}$CIFF specifications.

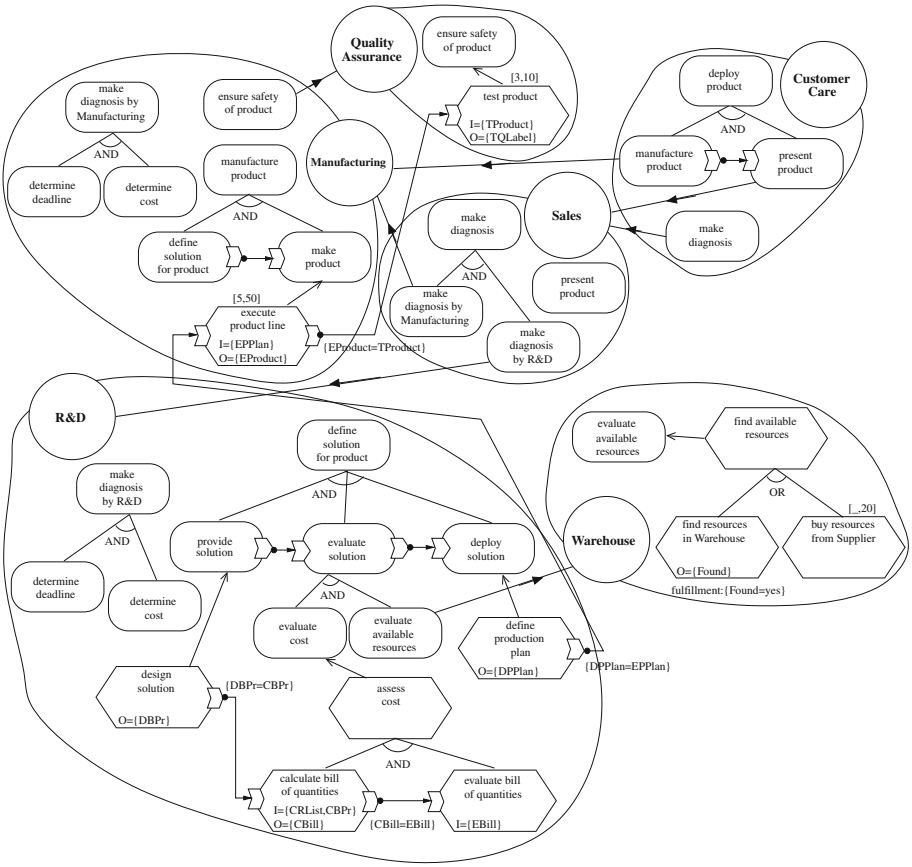


Fig. 4 Process-oriented extensions applied on a fragment of Fig. 1

3.1 The $\mathcal{S}$CIFF language and proof-procedures

$\mathcal{S}$CIFF [6] is a formal framework based on abductive logic programming [30], developed in the context of the SOCS project[6] 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 can decide what has to be considered as an event. For example, in an agent interaction setting, events may denote exchanged messages (e.g., $request(seller, buyer, give(10\$))$ or performed actions (e.g., $do(seller, buyer, give(10\$))$) [4]. This generality allows the designer 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. $\mathcal{B}$-Tropos events include atoms in the form $event(ID, X, Y, P)$ and $delegate(X, Y, A, T)$ (see below).

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 time instant representing the discrete time point at which the event happened. The time domain is

[6] SOcieties of heterogeneous ComputeeS, EU-IST-2001-32530—http://lia.deis.unibo.it/research/SOCS/.

not fixed, but depends on the chosen underlying CLP solver (see below). Currently, $\mathcal{S}$CIFF provides supports for integers and reals, to model both discrete and continuous time.

The set of all the events happened during a protocol execution constitutes its *log*, or *execution trace*. Moreover, the $\mathcal{S}$CIFF language supports the concept of *expectation* as first-class object, thus helping the user 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 are denoted by $\mathbf{E}(Ev, T)$ where Ev and T are atoms and may contain variables.

Social Integrity Constraints. The binding between happened events and expectations is given by means of *Social Integrity Constraints* (IC). These are forward rules, of the form $Body \rightarrow Head$. Constraint Logic Programming (CLP) constraints and Prolog predicates can be used to impose relations or restrictions on any of the variables (e.g., on time, by expressing orderings or deadlines). A sample IC is the following:

$$\begin{aligned}&\mathbf{H}(event(compl, r\&d, calc\_bill, [CBill]), T_f)\\ &\rightarrow \mathbf{E}(event(start, r\&d, eval\_bill, [EBill]), T_i)\\ &\quad \wedge T_f < T_i \wedge EBill = CBill.\end{aligned}$$

which intuitively means that if an event occurs which completes a calculate bill of quantities task, then an event is expected to occur, which starts an evaluate bill of quantities task, given some temporal (CLP) constraints, and such that the product ($CBill$) of the former is equal to the input of the latter ($EBill$). The whole set of IC is denoted by $\mathcal{IC}$.

In $\mathcal{S}$CIFF, the quantification of variables is left implicit. We recall here the essential quantification and scope rules. We forward the reader interested in their complete description to [6].

- Variables contained in an IC body (i.e., in happened events or Prolog predicates introduced in the body) are universally quantified with scope the entire rule. Indeed, the rule fires *each time* a set of concrete happened events occur s.t. the body of the rule evaluates to *true*.
- Variables contained in a positive expectation are existentially quantified with scope the IC head conjunct. Indeed, a positive expectation states that *at least one* corresponding matching event must occur during the execution.
- Variables contained (only) in a negative expectation are universally quantified with scope the IC head conjunct. Indeed, a negative expectation states that there must *not exist* a corresponding matching event during the execution.

In the example above, $CBill$ and T_f are universally quantified with scope the entire rule, while $EBill$ and T_i are existentially quantified with scope the head.

$\mathcal{IC}$ allows the designer to defnine how an interaction should evolve, given some previous situation encoded by a set of happened events. The static knowledge of the target domain is instead formalized inside the $\mathcal{S}$CIFF Knowledge Base (KB), which consists of clauses or backward rules. Here we find pieces of knowledge of the interaction model as well as the global organizational goal and/or objectives of single participants.

A sample rule is

$$achieve(r\&d, eval\_costs, T_i, T_f, I, O) \leftarrow execute(r\&d, assess\_costs, T_i, T_f, I, O).$$

which defnines a goal achievement in terms of a task execution. This knowledge is expressed in the form of clauses (i.e., a Prolog logic program). Here, quantification follows the standard Prolog conventions. As advocated in [21], this vision reconciles in a unique framework forward reactive reasoning with backward, goal-oriented deliberative reasoning.

$\mathcal{S}$CIFF semantics. In $\mathcal{S}$CIFF, an interaction model is interpreted in terms of an Abductive Logic Program (ALP) [30]. In general, an ALP is a triple $\langle P, A, \mathcal{IC}\rangle$, where P is a logic program, A is a set of predicates named *abducibles*, and $\mathcal{IC}$ is a set of ICs. In simple terms, the role of P is to defnine predicates, the role of A is to fill in the parts of P that are unknown, and the role of $\mathcal{IC}$ is to constrain the way elements of A are hypothesized, or "abduced". Reasoning in ALP is usually goal-directed, and it amounts to finding a set of abduced hypotheses Δ built from predicates in A such that $P \cup \Delta \models G$ (where G is a goal) and $P \cup \Delta \models \mathcal{IC}$.

$\mathcal{S}$CIFF-based verification methods. $\mathcal{S}$CIFF is both a formal specification language and a proof-procedure. The main usage of the latter is for run-time conformance verification. The idea is to adopt abduction to dynamically *generate* the expectations and to perform a *conformance checking* between expectations and happened events, to ensure that they are following the interaction model. The $\mathcal{S}$CIFF proof-procedure makes hypotheses about how participants should behave, thus expectations are defnined as abducibles. Conformance is verified by trying to confirm the expectations. A concrete running interaction is evaluated as conformant if it *fulfills* the specification. The $\mathcal{S}$CIFF proof-procedure is defnined as a transition system, and it is sound and complete with respect to its declarative semantics [6]. It is implemented and publicly available.[7] It is embedded inside SOCS-SI [4], a JAVA-based tool capable of parsing different event sources (or execution traces) and verifying, at run-time or in batch mode, if such events conform with a given $\mathcal{S}$CIFF specification.

The same $\mathcal{S}$CIFF language can also be used as a basis for static verification. In particular, domain-dependent properties can be verified in contexts such as agent interaction protocols [5] and business process specifications [40], starting from $\mathcal{S}$CIFF specifications and using g-$\mathcal{S}$CIFF, i.e., the generative extension of the $\mathcal{S}$CIFF proof-procedure. In a nutshell, g-$\mathcal{S}$CIFF considers the desired property as the initial goal, and checks, for each expectations, if it already has a matching happened event. If that is not the case, it "hypothesizes" such an event. Hence, g-$\mathcal{S}$CIFF operates by simulating a sequence of intensional happened events which fulfill the expectations. It does so via a rule $\mathbf{E}(X, T) \rightarrow \mathbf{H}(X, T)$, which incurs the automatic generation of a corresponding happened event for each positive expectation. Simulated events are intensional in the sense that they are partially specified (i.e., they could contain variables). Thus the output of g-$\mathcal{S}$CIFF is an execution schema. If the given property can be actually satisfied, g-$\mathcal{S}$CIFF also returns as a proof a partially specified execution trace capable of fulfilling both $\mathcal{IC}$ and the property. As well as $\mathcal{S}$CIFF, also g-$\mathcal{S}$CIFF is implemented and publicly distributed via the $\mathcal{S}$CIFF Web site. Empirical evidence shows that in the domain addressed by this work g-$\mathcal{S}$CIFF greatly outperforms state-of-the-art verification tools [40].

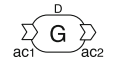
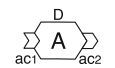
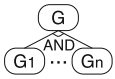
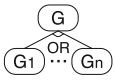
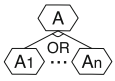
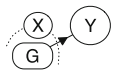
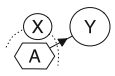
3.2 Mapping $\mathcal{B}$-Tropos concepts into the $\mathcal{S}$CIFF framework

The goal-oriented part of a $\mathcal{B}$-Tropos model represents the static knowledge of the application domain. As such, it is modeled in the $\mathcal{S}$CIFF KB by Prolog clauses. Table 2 summarizes the formalization of this part.

Goal achievement and task execution. The achievement of a goal and the execution of a task 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 with input I inside the time interval $[T_i, T_f]$, producing the output O. $execute(X, A, T_i, T_f, I, O)$ holds if actor X starts

[7] See the $\mathcal{S}$CIFF Web site: http://lia.deis.unibo.it/research/sciff/.

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

Unspecified leaf goal		$achieve(X,G,T_i,T_f,I,O) \leftarrow$ **achieved**(X,G,T_i,T_f,I,O), $T_f \in [D_{min},D_{max}]^{+T_i}, ac_1, ac_2.$
Leaf task		$execute(X,A,T_i,T_f,I,O) \leftarrow$ $\mathbf{E}(event(start,X,A,I),T_i)$, $\mathbf{E}(event(compl,X,A,O),T_f)$, $T_f \in [D_{min},D_{max}]^{+T_i}, ac_1, ac_2, fulfillment\_condition.$
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		$achieve(X,G,T_i,T_f,I,O) \leftarrow \mathbf{E}(delegate(X,Y,G,T_f),T_d),$ $T_d \geq T_i, T_d \leq T_f,$ $achieve(Y,G,T_d,T_f,I,O).$
Task Dependency		$execute(X,A,T_i,T_f,I,O) \leftarrow \mathbf{E}(delegate(X,Y,A,T_f),T_d),$ $T_d \geq T_i, T_d \leq T_f,$ $execute(Y,A,T_d,T_f,I,O).$

to execute task A with input I at time T_i (start time), and completes the execution of A at time T_f (completion time), producing the output O.

Therefore, parameters I and O represent the resource, respectively, needed and produced by a task execution or goal achievement. Start and completion times should satisfy both duration and absolute time constraints possibly associated to the goal/task.[8]

Execution times and I/O resources depend on how the goal/task is specified in the $\mathcal{B}$-Tropos model. When the task is a leaf task with associated duration D, pre-condition $ac1$ and post-condition $ac2$ (second row of Table 2), then it represents a concrete course of actions that must be executed by its responsible agent X. The concrete execution is in turn modeled in

[8] Note that the satisfaction of the duration constraint implicitly imposes an ordering between the start and the completion of the task

terms of the execution of the tasks's start and completion atomic events. These two events are represented using literals of the form $event(Ev, X, A, R)$ where $Ev \in \{start, compl\}$, 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. The fact that actor X executes the leaf task is then modeled as a pair of expectations concerning these two atomic events, bounding the input/output resource and the start/completion time of the task to the resource and expected time of the start/completion event.

Reasoning from partial specifications. The case of a leaf goal has a different meaning. It reflects the case in which the model must be left partially unspecified, either because the designer may prefer to keep the model at an abstract level, or because she has incomplete knowledge about the domain. In this situation, goals can be neither refined nor associated to tasks nor delegated. Abduction enables us to handle lack of information by reasoning on goal achievement in a hypothetical way. To this end, we introduced a new abducible, **achieved**, to indicate our hypothesis that the actor has reached the goal inside the expected time interval and the expected I/O resources. An example of this is Table 2's first row.

AND/OR decomposition. Composite tasks and goals are instead formalized by (recursively) linking a super-goal/task with its underlying components, according to the semantics of the employed decomposition (AND vs OR). "Linking" means that the start and completion times as well as the resources of the composite task are defnined in terms of the execution times and resources of the components.

In particular, the formalization of the $\mathcal{B}$-Tropos AND-decomposition relation by $\mathcal{S}$CIFF rules is done according to the following schema:

- an AND-decomposed goal/task is achieved/executed if all its underlying components are achieved/ executed;
- the resources involved in an AND-decomposed goal/task are determined by considering all the resources consumed and produced by the sub-goals/tasks;
- the starting time of an AND-decomposed goal/task is determined by the time at which the first sub-goal/task is achieved/starts to be executed;
- the completion time of an AND-decomposed goal/task is determined by the time at which the last sub-goal/task is completed.

Notice that AND-decompositions do not impose any constraint on the order of the achievement of subgoals (execution of subtasks). Essentially, by defnault AND-subgoals (AND-subtasks) are executed in parallel. Temporal constraints on the achievement of AND-subgoals (execution of AND-subtasks) are expressed using process-oriented constraints (see below).

The formalization of the $\mathcal{B}$-Tropos OR-decomposition relation by $\mathcal{S}$CIFF rules instead states that an OR-decomposed goal/task is achieved/executed if one of its underlying components is achieved/executed; start/completion times and resources of the goal/task are bound to the times and resources of such a sub-goal/task.

Means-end relation. The means-end relation sets the link between the concept of goal achievement and task execution: the goal is achieved during a certain interval and with certain involved resources if its underlying task is executed within that interval and by exploiting those resources.

Goal-task dependency. By goal and task dependencies (last two rows of Table 2), we introduce an expectation: that the depender will communicate to the dependee that the goal must be achieved before a certain deadline. To this end, we use a special event, $delegate(X, Y, G, T)$, to indicate the delegation of the achievement of goal G from actor X to actor Y. Moreover,

the formalization considers the latest time T when the goal should be achieved. A delegation is observable and so it has a corresponding event in the execution trace.

Composite events. The reactive part of $\mathcal{B}$-Tropos completes the formalization by encompassing process-oriented constraints as it illustrated by Table 3. As we mentioned above, the framework also permits to constrain non-leaf tasks and goals. However, only start and completion events of leaf tasks are considered to be observable events. To address this issue, we have introduced the intensional predicate **hap** to represent the happening of (possibly) composite events.

More specifically, a leaf task starts/completes only if there is evidence of it, i.e., a corresponding happened event is observable in the system. This is formalized by Table 3's first row: for leaf tasks, the **hap** predicate is defnined directly in terms of **H**. All other rows, which do not refer to leaf tasks, denote happening of composite event, recursively referring to the happening of the underlying events. In particular, the representation of composite events through AND/OR decompositions strictly resembles the one of the *achieve*/*execute* predicates used for representing goal achievement and task execution:

- for leaf goals which have a means-end task, their start/completion happens when such a task is started/completed (first row);
- the start/completion of an AND-decomposed task happens when its first/last (sub)task is started/ completed (second and fourth row);
- the start/completion of an OR-decomposed task happens when one of its (sub)tasks start/completion happens (third and fifth row);
- the start/completion of a delegated goal/task happen when the delegation is performed (sixth row).

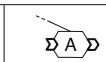
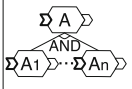
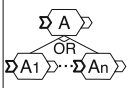
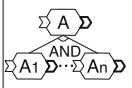
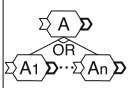
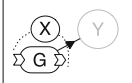
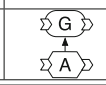
The defninition of happened events through the **hap** predicate accommodates $\mathcal{S}$CIFF rules in which composite events are treated as if they were simple events (i.e., a composite event too can "happen"). $\mathcal{S}$CIFF will employ the specific defnition of the **hap** predicates contained in the rule to *unfold* it, finally obtaining a set of rules which employ only **H** predicates related to leaf tasks (see the example below).

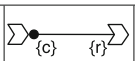
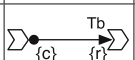
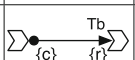
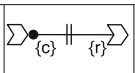
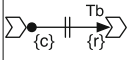
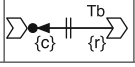
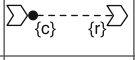
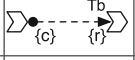
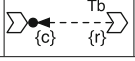
Note that ordering between the start and completion of composite task is guaranteed by the ordering imposed over the atomic events of start and completion of their underlying leaf task. The ordering on leaf task events is explicitly imposed by the corresponding $\mathcal{S}$CIFF formalization (see Table 2).

Process-oriented constraints. Process-oriented constraints are inspired by DecSerFlow/ConDec template formulas, for which a complete mapping to $\mathcal{S}$CIFF has been defnined in [39]. They are reported here in the bottom half of Table 3. Connections belonging to the same family (i.e., relations, weak relations, and negations) are translated to very similar sets of ICs. They mainly differ in the way they constrain time. Take for example the first block of three rows, representing the relation connections. They all relate the (possibly composite) happening of the source with the execution/achievement of the target task/goal. However, while *responded presence* does not state any temporal constraint on the involved execution times, *response*/*precedence* impose that the target execution time should be greater/lower than the source one.

Let us briefly review the formalization of reactive relationships in more detail. A **relation connection** states that when the source event *occurs* s.t. the attached condition is satisfied, then the target task is *expected to occur*, satisfying the corresponding condition. In the context of $\mathcal{B}$-Tropos the occurring of a (possibly composite) event is represented by its happening (denoted by the **hap** predicate applied on the event), while the expectation of its occurrence is modeled by way of task execution (denoted by the *execute* predicate). Time constraints

Table 3 Mapping of the reactive process-oriented part of $\mathcal{B}$-Tropos in $\mathcal{S}$CIFF

Leaf happening		$\mathbf{hap}(event(Ev, X, A, R), T) \leftarrow \mathbf{H}(event(Ev, X, A, R), T).$
Start events decomposition		$\mathbf{hap}(event(start, X, A, I), T) \leftarrow$ $\mathbf{hap}(event(start, X, A_1, I_1), T_1), \ldots, \mathbf{hap}(event(start, X, A_n, I_n), T_n),$ $T = \min\{T_1, \ldots, T_n\}, I = I_1 \cup \ldots \cup I_n.$
		$\mathbf{hap}(event(start, X, A, I), T) \leftarrow \mathbf{hap}(event(start, X, A_1, I), T).$ $\ldots$ $\mathbf{hap}(event(start, X, A, I), T) \leftarrow \mathbf{hap}(event(start, X, A_n, I), T).$
Completion events decomposition		$\mathbf{hap}(event(compl, X, A, O), T) \leftarrow$ $\mathbf{hap}(event(compl, X, A_1, O_1), T_1), \ldots, \mathbf{hap}(event(compl, X, A_n, O_n), T_n),$ $T = \max\{T_1, \ldots, T_n\}, O = O_1 \cup \ldots \cup O_n.$
		$\mathbf{hap}(event(compl, X, A, O), T) \leftarrow \mathbf{hap}(event(compl, X, A_1, O), T).$ $\ldots$ $\mathbf{hap}(event(compl, X, A, O), T) \leftarrow \mathbf{hap}(event(compl, X, A_n, O), T).$
Events of a delegated goal/task		$\mathbf{hap}(event(Ev, X, A, R), T) \leftarrow \mathbf{hap}(event(Ev, Y, A, R), T).$
Means-end events		$\mathbf{hap}(event(Ev, X, G, R), T) \leftarrow \mathbf{hap}(event(Ev, X, A, R), T)$

Relation connections (when connected to the start of a task)		
Responded Presence		$\mathbf{hap}(event(Ev, X, A_1, R_1), T) \wedge c$ $\rightarrow execute(Y, A_2, T_i, T_f, I, O) \wedge r.$
Response		$\mathbf{hap}(event(Ev, X, A_1, R_1), T) \wedge c$ $\rightarrow execute(Y, A_2, T_i, T_f, I, O) \wedge r \wedge T_i \in T_b^{+T}.$
Precedence		$\mathbf{hap}(event(Ev, X, A_1, R_1), T) \wedge c$ $\rightarrow execute(Y, A_2, T_i, T_f, I, O) \wedge r \wedge T_i \in T_b^{-T}.$
Negation connections		
Responded Absence		$\mathbf{hap}(event(Ev_1, X, A_1, R_1), T_1) \wedge c$ $\wedge \mathbf{hap}(event(Ev_2, Y, A_2, R_2), T_2) \wedge r \rightarrow \bot .$
Negation Response		$\mathbf{hap}(event(Ev_1, X, A_1, R_1), T_1) \wedge c$ $\wedge \mathbf{hap}(event(Ev_2, Y, A_2, R_2), T_2) \wedge r \wedge T_2 \in T_b^{+T_1} \rightarrow \bot .$
Negation Precedence		$\mathbf{hap}(event(E_1v, X, A_1, R_1), T_1) \wedge c$ $\wedge \mathbf{hap}(event(Ev_2, Y, A_2, R_2), T_2) \wedge r \wedge T_2 \in T_b^{-T_1} \rightarrow \bot .$
Weak relation connections		
Responded Presence		$\mathbf{hap}(event(Ev_1, X, A_1, R_1), T_1) \wedge c$ $\wedge \mathbf{hap}(event(Ev_2, Y, A_2, R_2), T_2) \rightarrow r.$
Response		$\mathbf{hap}(event(Ev_1, X, A_1, R_1), T_1) \wedge c$ $\wedge \mathbf{hap}(event(Ev_2, Y, A_2, R_2), T_2) \rightarrow r \wedge T_2 \in T_b^{+T_1}.$
Precedence		$\mathbf{hap}(event(Ev_1, X, A_1, R_1), T_1) \wedge c$ $\wedge \mathbf{hap}(event(Ev_2, Y, A_2, R_2), T_2) \rightarrow r \wedge T_2 \in T_b^{-T_1}.$

can be expressed by CLP constraints over the involved times, i.e., the execution time related to the source event and the start or completion time related to the target task (depending on the relation).

Differently from relation connections, **weak relations** do not impose any expectation about the execution of a certain task, but they impose instead some data-related and/or temporal conditions *only when* the involved events have already occurred. Therefore, they are represented by a rule containing the occurrence of both the source and the target events in the body (i.e., the rule fires only when the two events happen), and containing the involved data-related and/or temporal conditions in the head (i.e., if the rule fires then such conditions must be satisfied).

Finally, **negation connections** have the form $body \rightarrow \bot$, meaning that the execution is considered incorrect if *body* becomes *true*. $Body$ is used to express the situation that must be avoided. For example, the *negation response* constraint shown in Table 3 states that when the source event happens s.t. condition c holds, then the target event making r true is forbidden afterwards. Such a constraint is reformulated in $\mathcal{S}$CIFF by stating that an execution in which the two events occur in "ascending" order, making conditions c and r true, must be avoided.

The Product Development Process $\mathcal{B}$-Tropos model in $\mathcal{S}$CIFF. Table 4 shows the $\mathcal{S}$CIFF formalization of the bottom-left fragment of the $\mathcal{B}$-Tropos diagram presented in Fig. 4.

In particular, $KB_{r\&d}$ maps R&D's evaluate solution goal, KB_{wh} maps the entire Warehouse's content, plus a delegation from Warehouse to Purchase shown in Fig. 1, and finally $ICs_{r\&d}$ maps the responded presence relation between R&D's calculate bill of quantities and evaluate bill of quantities tasks.

To clarify on the example how the $\mathcal{S}$CIFF formalization of reactive constraints works, let us now briefly describe how the reactive rule contained in Table 4 is implicitly rewritten (technically, *unfolded* [6]) by the $\mathcal{S}$CIFF or g-$\mathcal{S}$CIFF proof procedure during the verification. As we have previously pointed out, variables contained in the body of a rule are universally quantified with scope the entire rule, and the rule will trigger in any possible situation making the rule true. The practical impact of this intuitive notion is that the body of the rule in Table 4 is unfolded by the proof procedure considering all the possible defninitions of the **hap** predicate contained in the body.[9]

In the example, the *calc_bill* completion is an event attached to a leaf task, and it is therefore directly represented by means of a corresponding $\mathcal{S}$CIFF happened event (**H**). The unfolding of the reactive rule contained in Table 4 would therefore lead to obtain the following rule:

$$\begin{aligned}&\mathbf{H}(event(compl, r\&d, calc\_bill, [CBill]), T)\\ &\rightarrow execute(r\&d, eval\_bill, T_i, T_f, [EBill], [])\\ &\quad \wedge T_i > T \wedge EBill = CBill.\end{aligned}$$

Let us now suppose to modify the $\mathcal{B}$-Tropos model under study as follows: the calculate bill of quantities is now defnined by means of an OR-decomposition, which reduces it either to the manual calculation or automatic calculation leaf tasks, associated to the same I/O resources. In that case, the **hap**$(event(compl, r\&d, calc\_bill, [CBill]), T)$ would be defnined in terms of the completion of the manual calculation task *or* the completion of the automatic calculation task:

[9] In particular, the unfolding step will lead to obtain a set of replicated rule, each one considering a possible defninition of the predicate.

Table 4 $\mathcal{S}$CIFF specification of a fragment of the $\mathcal{B}$-Tropos model shown in Fig. 4; for the sake of simplicity, the defnination of the **hap** predicates is not presented

$KB_{r\&d}$:

$$achieve(r\&d, eval\_solution, T_i, T_f, I, O) \leftarrow achieve(r\&d, eval\_costs, T_{i1}, T_{f1}, I_1, O_1), achieve(r\&d, eval\_resources, T_{i2}, T_{f2}, I_2, O_2), \min(T_i, [T_{i1}, T_{i2}]), \max(T_f, [T_{f1}, T_{f2}]), I = I_1 \cup I_2, O = O_1 \cup O_2.$$

$$achieve(r\&d, eval\_costs, T_i, T_f, I, O) \leftarrow execute(r\&d, assess\_costs, T_i, T_f, I, O).$$

$$execute(r\&d, assess\_costs, T_i, T_f, I, O) \leftarrow execute(r\&d, calc\_bill, T_{i1}, T_{f1}, I_1, O_1), execute(r\&d, eval\_bill, T_{i2}, T_{f2}, I_2, O_2), \min(T_i, [T_{i1}, T_{i2}]), \max(T_f, [T_{f1}, T_{f2}]), I = I_1 \cup I_2, O = O_1 \cup O_2.$$

$$execute(r\&d, calc\_bill, T_i, T_f, [CRList, CBPr], [CBill]) \leftarrow \mathbf{E}(event(start, r\&d, calc\_bill, [CRList, CBPr]), T_i), \mathbf{E}(event(compl, r\&d, calc\_bill, [CBill]), T_f), T_f > T_i.$$

$$execute(r\&d, eval\_bill, T_i, T_f, [EBill], []) \leftarrow \mathbf{E}(event(start, r\&d, eval\_bill, [EBill]), T_i), \mathbf{E}(event(compl, r\&d, eval\_bill, []), T_f), T_f > T_i.$$

$$achieve(r\&d, eval\_resources, T_i, T_f, I, O) \leftarrow \mathbf{E}(delegate(r\&d, wh, eval\_resources, T_f), T_d), achieve(wh, eval\_resources, T_d, T_f, I, O), T_d > T_i, T_i < T_f.$$

KB_{wh} :

$$achieve(wh, eval\_resources, T_i, T_f, I, O) \leftarrow execute(wh, find\_resources, T_i, T_f, I, O).$$

$$execute(wh, find\_resources, T_i, T_f, I, O) \leftarrow execute(wh, find\_in\_wh, T_i, T_f, I, O).$$

$$execute(wh, find\_resources, T_i, T_f, I, O) \leftarrow execute(wh, buy, T_i, T_f, I, O).$$

$$execute(wh, find\_in\_wh, T_i, T_f, [], [Found]) \leftarrow \mathbf{E}(event(start, wh, find\_in\_wh, []), T_i), \mathbf{E}(event(compl, wh, find\_in\_wh, [Found]), T_f), T_f \geq T_i, Found = yes.$$

$$execute(wh, buy, T_i, T_f, [], []) \leftarrow \mathbf{E}(event(start, wh, buy, []), T_i), \mathbf{E}(event(compl, wh, buy, []), T_f), T_f > T_i, T_f \leq T_i + 20.$$

$$execute(wh, buy, T_i, T_f) \leftarrow \mathbf{E}(delegate(wh, purchase, buy, T_f), T_d), T_d > T_i.$$

$ICs_{r\&d}$:

$$\mathbf{hap}(event(compl, r\&d, calc\_bill, [CBill]), T) \rightarrow execute(r\&d, eval\_bill, T_i, T_f, [EBill], []) \wedge T_i > T \wedge EBill = CBill.$$

$$\mathbf{hap}(event(compl, r\&d, calc\_bill, [CBill]), T) \leftarrow \mathbf{hap}(event(compl, r\&d, manual\_calc, [CBill]), T).$$

$$\mathbf{hap}(event(compl, r\&d, calc\_bill, [CBill]), T) \leftarrow \mathbf{hap}(event(compl, r\&d, auto\_calc, [CBill]), T).$$

The reactive rule of Table 4 would then be unfolded in two different ways, leading to the following rules:

$$\mathbf{H}(event(compl, r\&d, manual\_calc, [CBill]), T) \rightarrow execute(r\&d, eval\_bill, T_i, T_f, [EBill], []) \wedge T_i > T \wedge EBill = CBill.$$

$$\mathbf{H}(event(compl, r\&d, auto\_calc, [CBill]), T) \rightarrow execute(r\&d, eval\_bill, T_i, T_f, [EBill], []) \wedge T_i > T \wedge EBill = CBill.$$

Similarly to what has been presented for the unfolding of the body, the *execute* predicate will be unfolded when the body of the rule evaluates to $true$. The unfolding of predicates in the head is however different: the presence of multiple defninitions of a predicate (modeling design alternatives, such as OR-decompositions) leads to introducing a disjunction of heads (each one considering a possible defninition of the predicate) in the same rule. This is to express that there exist different ways to make the "original" head *true*.

4 Formal properties

Tables 2 and 3 defnine the semantics of $\mathcal{B}$-Tropos constructs. Those tables represent a complete mapping, in the sense that no additional specifications are needed, other than those obtained automatically by following the table to fully specify a $\mathcal{B}$-Tropos model. Besides, they show that $\mathcal{B}$-Tropos is a conservative extension of Tropos. In fact, all constraints would be trivially verified if the $\mathcal{B}$-Tropos model was projected into its corresponding Tropos model.

The translation of $\mathcal{B}$-Tropos to a computational logic-based framework, equipped with a clear declarative semantics, allows us to identify and address important formal properties, namely soundness, completeness and termination.

We briefly recall the main theoretical results investigated in the general case, and then show how they apply in the specific context of $\mathcal{B}$-Tropos.

The properties of soundness, completeness and termination of the $\mathcal{S}$CIFF proof procedure were studied in [6], where they were proven for *acyclic* knowledge bases and *bounded* goals and implications. In [3,38], these properties have been extended to g-$\mathcal{S}$CIFF by imposing the acyclicity conditions on the KB and the $\mathcal{IC}$, by taking into account the rule $\mathbf{E}(X, T) \rightarrow \mathbf{H}(X, T)$ as well.

The notion of acyclicity of an abductive logic program is an extension of the corresponding notion given for SLD resolution. Intuitively, for SLD resolution a level mapping must be defnined, such that the head of each clause has a higher level than the body (in this case, the logic program under analysis is said to be acyclic). For the sake of clarity, we report here the defnition of level mapping given in [50]:

Definition 4.1 (*Level Mapping*) Given a logic program P, a level mapping $||$ is a function that maps all ground atoms in the Herbrand base of P (B_P) to $\mathbb{N}^+$ (the set of positive natural numbers), and $false$ to 0. Also, $||$ is extended to map a ground negative literal $\neg A$ to $|A|$ where $A \in B_P$.

For $\mathcal{S}$CIFF and g-$\mathcal{S}$CIFF, which contain forward rules (integrity constraints), the level mapping should also map atoms in the body of an IC to higher levels than the atoms in the head [50]. In our specific case, we impose some reasonable restrictions on the structure of a $\mathcal{B}$-Tropos model by defnining the class of $\mathcal{B}$-*acyclic* models. Then, we prove that such restrictions are sufficient to guarantee termination of both $\mathcal{S}$CIFF and g-$\mathcal{S}$CIFF proof-procedures when they are used to execute verification tasks on $\mathcal{S}$CIFF specifications derived from $\mathcal{B}$-Tropos models.

Definition 4.2 ($\mathcal{B}$-*acyclic* $\mathcal{B}$-Tropos model) A $\mathcal{B}$-Tropos model is $\mathcal{B}$-*acyclic* iff it satisfies the following conditions:

1. the goal/task models (see Sect. 2.1) are acyclic graphs;
2. there does not exist a cyclic chain of dependencies for the same goal/task;
3. there does not exist a cyclic chain of reactive constraints;

4. fulfillment conditions and data constraints do not contain any goal achievement, task execution, or **H/E** atoms.

These conditions do not significantly limit the expressiveness of the language. The first condition, which also hold in the original Tropos, states that it is not possible to reduce a sub-goal (sub-task) in terms of one of its super-goals (super-tasks); the presence of this kind of loop would lead to an infinite decomposition. The violation of the second corresponds to an error in the model, because it is not acceptable that an agent, even indirectly, receives a delegation from itself: the delegation process would be indefninitely re-iterated and the delegated goal/task would never be achieved/executed. A similar motivation lies behind the necessity of avoiding cycles through reactive connections; the presence of a cycle would imply that the task execution will never reach an end, which contrasts with the intuitive fact that agents must be able to achieve their goals in finite time, i.e., by executing a finite process. Finally, the last condition ensures that data constraints effectively work on data, and do not involve the execution of further events. Notice that "cycles" including different types of relations are allowed in the framework. For instance, a model in which an actor, A, depends on another actor, B, for the achievement of a goal G, and B depends on A for the achievement of a subgoal of G, is *$\mathcal{B}$-acyclic*.

Theorem 4.3 (*$\mathcal{B}$-acyclic*ity of $\mathcal{B}$-Tropos implies acyclicity of the underlying $\mathcal{S}$CIFF specification) *The $\mathcal{S}$CIFF formalization of a* $\mathcal{B}$-acyclic *$\mathcal{B}$-Tropos model is acyclic (in spite of the additional "generative" rule* $\mathbf{E}(X, T) \rightarrow \mathbf{H}(X, T)$*).*

Proof (Sketch) To guarantee termination of a $\mathcal{S}$CIFF program corresponding to a $\mathcal{B}$-Tropos model, we have to find a level mapping for the specification, which satisfies all acyclicity conditions for abductive logic programs. To this aim, we first map the atoms that express fulfillment conditions and data constraints as well as the $achieved$ abducible and the happening of a delegation to the level 0. The fourth hypotheses of *$\mathcal{B}$-acyclic* models ensures that fulfillment conditions and data constraints have no relationships with other parts of the program, and therefore data constraints respect acyclicity. Moreover, $achieved$ and $\mathbf{H}(delegate(\ldots), \_)$ do not occur in the body of any IC, and therefore they could be directly associated to specific values in the level mapping.

For other (ground) atoms, namely achievement/execution/composite happenings as well as expectations and happenings of start/completion events, we build a "level mapping graph" whose nodes are associated to such atoms and whose edges express the acyclicity conditions associated to each of the rules shown in Tables 2 and 3. The intended meaning of an edge is that the level mapping of the (ground atom attached to the) source should be greater than the level mapping of the (one of the) target.

An intuitive description of how the edges are inferred from the rules in Tables 2 and 3 is as follows:

- if a goal/task is a child of another goal/task w.r.t. AND/OR decomposition, add an edge from the node which maps the achievement/execution (composite start/completion) of the latter to the one which maps the achievement/execution (composite start/completion, respectively) of the former;[10]
- if a task is means-end of a goal, add an edge from the node which maps the achievement (composite start/completion) of the latter to the one which maps the execution (composite start/completion, respectively) of the former;

[10] If we consider the example illustrated in Fig. 5, this step would amount to adding four edges: two from t_1 start to t_2 and t_3 start, two from t_1 completion to t_2 and t_3 completion. Such edges are visible in the central area of the figure. Other steps below are also illustrated in Fig. 5 but we omit their description in the text for the sake of brevity.

- for each node mapping the execution of a leaf task, add an outgoing edge to connect it with the expectations about its start/completion (see the *leaf task* rule);
- for each node mapping the composite happening of a leaf task, add an outgoing edge to connect it with the happenings of its start/completion (see the *leaf (observable) events* rules, Table 2);
- for each node mapping the execution/achievement of a delegated task/goal, add an outgoing edge to connect it with the expectation about its delegation and another outgoing edge to connect it with the node which maps the execution/achievement of the corresponding delegatee's task/goal;
- for each node mapping the composite start/completion of a delegated task/goal, add an outgoing edge to connect it with the node which maps start/completion happening of the corresponding delegatee's task/goal;
- for each node mapping the happening of a delegation, add an outgoing edge to connect it with the corresponding achievement (see the *reaction to delegation* rule);
- for each reactive connection, add an edge from the set of nodes representing the composite start/ completion of the leaf tasks which descends from the connection source (following the AND/OR decomposition) to the node which maps the execution of the connection target. Note that this rule suffices to realize the acyclicity conditions in the case of ICs [50], due to the already existing edges derived from the AND/OR decomposition of both connection source and target.

Such rules are sufficient to express all the acyclicity conditions requested by $\mathcal{S}$CIFF to ensure termination. To guarantee the termination of g-$\mathcal{S}$CIFF, we should also add the following rule:

- add an edge from the node which maps an expectation to the node which maps the corresponding happened event.

The constructed graph resembles the structure of a $\mathcal{B}$-Tropos model. This is not surprising, since the $\mathcal{S}$CIFF mapping expresses in a natural way the corresponding model. By exploiting this similarity, it is easy to show that the $\mathcal{B}$-*acyclicity* conditions imposed on the $\mathcal{B}$-Tropos model guarantee that the obtained graph is acyclic:

- Conditions 1 and 2 of Definition 4.2 avoids loops between nodes containing execute/achieve atoms;
- Condition 3 of Definition 4.2 avoids loops between nodes containing **H/E** atoms;
- Condition 4 of Definition 4.2 avoids loops between execute/achieve and **H/E** atoms.

Note, that to guarantee the termination of $\mathcal{S}$CIFF, only conditions 1, 2 and 4 of Definition 4.2 are needed, whereas the termination of g-$\mathcal{S}$CIFF also requires condition 3, because relating event expectations with event occurrences leads to relating, for each element, its incoming reactive connections with the outgoing ones.

Since the obtained graph is acyclic, a partial ordering among nodes can be found so that a suitable level mapping can be defnined. More specifically, let us consider the following node numbering function:

$$num(v) = \begin{cases} 1 & \text{if } v \text{ is a sink node} \\ 1 + \max\{num(v') \text{ s.t. } vv' \text{ is an edge}\} & \text{otherwise} \end{cases}$$

By defnining the level mapping of each atom contained in the level mapping graph to the value of the *num* function applied on the corresponding node, all the acyclicity conditions represented in the graph are satisfied by construction. Being 1 the minimum number assigned

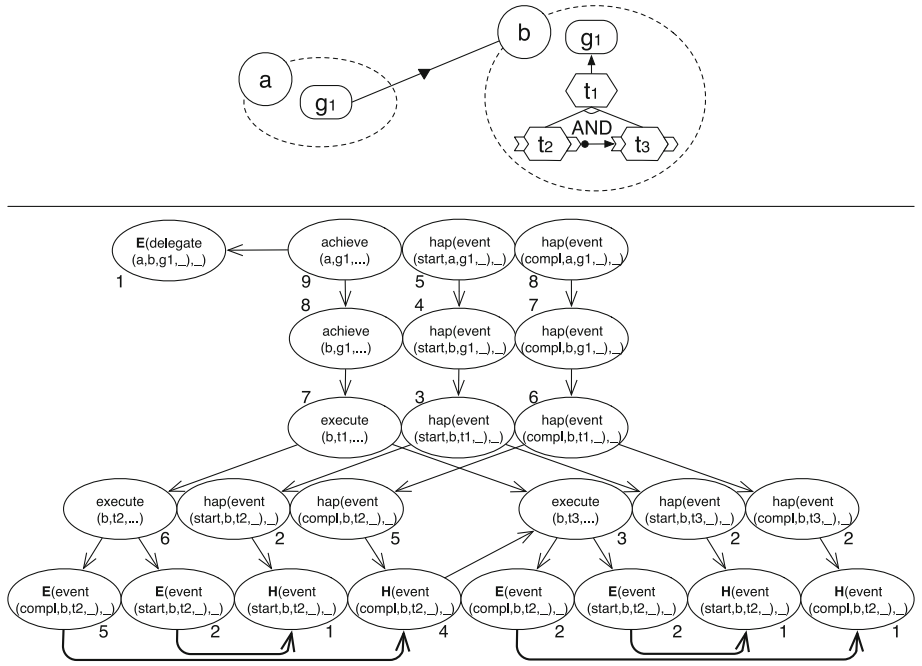


Fig. 5 A simple $\mathcal{B}$-Tropos model together with its corresponding level mapping graph (annotated with a suitable level mapping). Fat edges denote the extended conditions for g-$\mathcal{S}$CIFF

by the *num* function, all the acyclicity conditions between atoms which are represented in the graph and atoms mapped to the level 0 are preserved. Figure 5 shows an example of $\mathcal{B}$-Tropos model together with its corresponding level mapping graph. Its bubbles contain elements of the $\mathcal{S}$CIFF formalization given in Table 5. □

Theorem 4.4 (Soundness, completeness and termination) *The $\mathcal{S}$CIFF and g-$\mathcal{S}$CIFF proof procedures are sound and complete and always terminate their computations when reasoning on specifications obtained by mapping* $\mathcal{B}$-acyclic $\mathcal{B}$*-Tropos models.*

Proof (Sketch) Soundness and completeness results are inherited from the soundness and completeness of $\mathcal{S}$CIFF and g-$\mathcal{S}$CIFF (see [3,6,38]). Note that it is possible to inherit these results thanks to the declarative nature of the $\mathcal{S}$CIFF framework. Termination for $\mathcal{S}$CIFF is guaranteed if specifications are acyclic. Termination for g-$\mathcal{S}$CIFF is guaranteed if specifications, augmented with the generative rule (see Sect. 3.1), are acyclic. Thanks to Theorem 4.3, both conditions are fulfilled by $\mathcal{B}$*-acyclic* $\mathcal{B}$-Tropos models. □

All these formal properties are guaranteed for $\mathcal{B}$*-acyclic* $\mathcal{B}$-Tropos models. Therefore, to ensure that an arbitrary $\mathcal{B}$-Tropos model is correctly handled by the proof procedures, a pre-processing step is needed so as to identify whether the model is effectively $\mathcal{B}$*-acyclic* or not; in the positive case, the model can be translated to $\mathcal{S}$CIFF and verified by exploiting the proof procedures, otherwise the presence of an error can be directly reported to the user.

The characterization of such a pre-processing step is out of the scope of the paper. However, one possibility would be to translate the $\mathcal{B}$-Tropos model to a graph, thus reducing a $\mathcal{B}$*-acyclicity* identification to a loop detection problem.

Table 5 $\mathcal{S}$CIFF formalization of the Tropos model shown in Fig. 5

$achieve(b, g_1, T_i, T_f, I, O) \leftarrow execute(b, t_1, T_i, T_f, I, O).$

$achieve(a, g_1, T_i, T_f, I, O) \leftarrow \mathbf{E}(delegate(a, b, g_1, T_f), T_d), T_d > T_i, T_d < T_f, achieve(b, g_1, T_d, T_f, I, O).$

$execute(b, t_2, T_i, T_f, I, O) \leftarrow \mathbf{E}(event(start, b, t_2, I), T_i), \mathbf{E}(event(compl, b, t_2, O), T_f).$

$execute(b, t_3, T_i, T_f, I, O) \leftarrow \mathbf{E}(event(start, b, t_3, I), T_i), \mathbf{E}(event(compl, b, t_3, O), T_f).$

$execute(b, t_1, T_i, T_f, I, O) \leftarrow execute(b, t_2, T_{i2}, T_{f2}, I_2, O_2), execute(b, t_3, T_{i3}, T_{f3}, I_3, O_3), T_i = \min\{T_{i2}, T_{i3}\}, T_f = \max\{T_{f2}, T_{f3}\}, I = I_1 \cup I_2, O = O_1 \cup O_3.$

$\mathbf{hap}(event(Ev, b, t_2, R), T) \leftarrow \mathbf{H}(event(Ev, b, t_2, R), T).$

$\mathbf{hap}(event(Ev, b, t_3, R), T) \leftarrow \mathbf{H}(event(Ev, b, t_3, R), T).$

$\mathbf{hap}(event(start, b, t_1, I), T) \leftarrow \mathbf{hap}(event(start, b, t_2, I_2), T_2), \mathbf{hap}(event(start, b, t_3, I_3), T_3), T = \min\{T_2, T_3\}, I = I_2 \cup I_3.$

$\mathbf{hap}(event(compl, b, t_1, O), T) \leftarrow \mathbf{hap}(event(compl, b, t_2, O_2), T_2), \mathbf{hap}(event(compl, b, t_3, O_3), T_3), T = \max\{T_2, T_3\}, O = O_2 \cup O_3.$

$\mathbf{hap}(event(Ev, b, t_2, R_1), T) \rightarrow execute(b, t_3, T_i, T_f, I, O) \wedge T_i > T.$

5 Conformance and property verification

By Theorem 4.3, a query consisting of the conjunction of subsets of top-level goals of a $\mathcal{B}$-*acyclic* $\mathcal{B}$-Tropos model always results in a terminating $\mathcal{S}$CIFF and g-$\mathcal{S}$CIFF derivation. This result ensures that property and conformance verification can be performed in a finite number of steps.

As we mentioned above, the $\mathcal{S}$CIFF proof-procedure can be used to address conformance verification [6]. This analysis is twofold. On one side, it can be used to verify that the actual system implementation is compliant with the requirements model. On the other side, it is related to the auditing measures adopted by an information system for monitoring the activities performed by actors within the system. Both kinds of analysis share the same idea, that is, to analyze events produced by the system, which can either be generated by an automated reasoning tool or a simulator starting from the specifications, or become available at runtime or via system logs, and compare them with the design of the system. In the first case, we verify if the system logs can be generated from the requirements model. In second case, the objective is to verify if stakeholders have achieved their goals without violating process-oriented constraints and to determine (by means of expectations about tasks) the next possible actions. $\mathcal{S}$CIFF realizes such analysis by checking whether the system execution trace actually matches the expectations generated by the proof-procedure without violating the ICs. Expectations are generated starting from the given (conjunction of) goals by unfolding the corresponding *achieve* predicates.

Unexpected situations, i.e., run-time behaviors which deviate from the model, lead to violate a certain generated expectation, either because a positive expectation does not have a corresponding happened event, or because an event forbidden by a negative expectation occurs. These violations are detected by $\mathcal{S}$CIFF as soon as they happen. Note that neither $\mathcal{B}$-Tropos nor the underlying $\mathcal{S}$CIFF formalization contain primitives and mechanisms to handle such unexpected situations. However, exception handling and compensation mechanisms can be realized in $\mathcal{S}$CIFF by exploiting the (reactive) Event Calculus axiomatization discussed in [14,15].

The g-$\mathcal{S}$CIFF proof-procedure can be exploited to check whether a given property holds in the model by generating, in case of a positive answer, a simulated intensional execution

compliant with the model which satisfies the property. Differently from the $\mathcal{S}$CIFF proof-procedure, g-$\mathcal{S}$CIFF addresses static verification, to ensure that the model under study is consistent and that it enjoys certain properties a-priori. Let us spend a few words to discuss how g-$\mathcal{S}$CIFF relates with other approaches developed by researchers in the formal methods community. In [40,41] g-$\mathcal{S}$CIFF has been compared with other verification techniques, namely explicit, symbolic and SAT-based model checkers, showing that it scales better as the number of constraints grows. As pointed out in [40,41], this behaviour can be explained by the well known "state-explosion" problem experienced by model checking techniques, which is especially true if we consider declarative specifications such as the ones of ConDec, in which the system is not represented as a Kripke structure, but it is itself specified as a set of declarative logical formulae.

Static verification helps designers to evaluate different design alternatives in terms of system performances, resources needed to achieve a goal, etc. A detailed discussion about the properties that can be expressed in g-$\mathcal{S}$CIFF can be found in [38,41]. Properties can express many different requirements, such as for example an external regulation or a desirable situation. In the case of an external regulation, the intended meaning is that the property must be respected in any possible execution of the system; conversely, the reachability of a certain situation (which is expressed, in $\mathcal{B}$-Tropos, as a conjunction of goal achievements) holds if there exists at least a possible execution which leads to that situation.

In the general case, properties quantify over execution traces in two different ways: existential and universal. A property is existentially entailed by the model under study if at least one execution trace compliant with the model satisfies the property as well; a property is instead universally entailed if all the execution traces compliant with the model also entail the property.

g-$\mathcal{S}$CIFF provides support for verifying properties which quantify over execution traces in an existential way: based on a query, it tries to generate an (intensional) execution trace which is compliant with the model under study and, at the same time, lead to the achievement of the goals in the query, if any. Such an execution trace represents a proof attesting that the property can be satisfied. Universal properties, i.e., properties that must be satisfied by any possible execution of the modeled system, can be negated and reduced to existential properties.[11] If g-$\mathcal{S}$CIFF founds an execution trace entailing the negated property, then such an execution trace can be interpreted as a counter-example which disproves the entailment.

For example, a "never claim", that is, a property stating that a certain situation will never hold in any execution of the system, is expressed in the opposite way, by searching for an execution in which such a situation holds; if g-$\mathcal{S}$CIFF finds an execution trace which entails such a negated property, then the trace can be considered as a counter-example attesting that the never claim is not satisfied.

In the context of $\mathcal{B}$-Tropos, if the property is given in terms of the root goals of the whole $\mathcal{B}$-Tropos model, then g-$\mathcal{S}$CIFF verifies if there exists at least one execution trace that satisfies the entire requirements model, together with process-oriented constraints. For example, by considering the $\mathcal{B}$-Tropos model in Fig. 4, a general and simple query is

$$\leftarrow \; achieve(man, ensure\_safety\_of\_products, \_, \_, \_, \_),$$
$$achieve(cust\_care, deploy\_product, \_, \_, \_, \_).$$

It is possible to verify more sophisticated properties by constraining data and time variables in the query. For instance, one can verify if a stakeholder achieves its goals within a

[11] In the general case, the reduction must be manually executed by the modeler. However, the reduction can often be automatized for fragments of the language [38].

Table 6 Execution trace generated by $\mathcal{S}$CIFF

$\min(0, [T_{scb}, T_{ser}]), T_f < 50, \max(T_f, [T_{ceb}, T_{cer}]),$
$\mathbf{H}(event(start, r\&d, calc\_bill, [Blueprint, Res\_cost]), T_{scb}),$
$\mathbf{H}(event(compl, r\&d, calc\_bill, [Bill]), T_{ccb}), T_{ccb} > T_{scb}, T_{ccb} < T_{seb},$
$\mathbf{H}(event(start, r\&d, eval\_bill, [Bill]), T_{seb}),$
$\mathbf{H}(event(compl, r\&d, eval\_bill, []), T_{ceb}), T_{ceb} > T_{seb},$
$\mathbf{H}(delegate(r\&d, wh, eval\_resources, T_{cer}), T_{ser}),$
$\mathbf{H}(event(start, wh, find\_resources, []), T_{sfr}), T_{sfr} > T_{ser},$
$\mathbf{H}(event(compl, wh, find\_resources, [yes]), T_{cer}), T_{cer} > T_{sfr}.$

certain maximum execution time. To concretely show the verification of this property, we have analyzed the fragment of the $\mathcal{B}$-Tropos model formalized in Table 4. Here, the problem is to ensure that R&D can evaluate the solution for the product in at most 50 time units (in fact, even less). Such a property is formalized as

$$\leftarrow achieve(r\&d, evaluate\_solution, 0, T_f), T_f > 0, T_f < 50.$$

Table 6 shows the intensional execution trace generated by g-$\mathcal{S}$CIFF, which proves formally that R&D can indeed achieve its goal within 50 time units. If we assume discrete, integer time, a possible labeling of time variables is the following: $T_{scb} = T_{ser} = 0$, $T_{ccb} = T_{sfr} = 1$, $T_{seb} = T_{cer} = 0$, $T_{ceb} = T_f = 3$. Indeed, if we further specify the model by associating minimum duration constraints with leaf tasks, the result may change. If we continue the analysis by asking for another solution, g-$\mathcal{S}$CIFF generates a second execution trace where the upper-level task find available resources is performed by executing task buy resources from an external supplier instead of task find resources in warehouse.

Other interesting properties could exploit the negative expectations of the $\mathcal{S}$CIFF language. For example, we could augment the property discussed so far by adding an **EN** conjunct stating that a certain task cannot be executed. The fact that g-$\mathcal{S}$CIFF finds an execution trace attests that the top-level goals of the agents can be achieved avoiding the execution of such a task.

6 Related work

There exist many agent-oriented software engineering methodologies in the market. Their comparison is out of the scope of this paper. The interested reader can refer to one of the many surveys (e.g., [25,49]). In practice, the choice of one methodology over another depends on the domain specifics, on the focus of design and engineering tasks, and not least on the experience and background of the designer. The Tropos methodology is particularly well-suited for early requirements engineering. Augmenting Tropos with an operational framework and with tools for handling process-level concepts is useful to get a better idea of a possible future system at the early engineering phases, which is a missing element of other related proposals. The translation of $\mathcal{B}$-Tropos models to $\mathcal{S}$CIFF specifications follows a fully-automated, push-button procedure, which implements the mapping shown in Sect. 3. This makes $\mathcal{B}$-Tropos and its verification features accessible to the non-IT-savvy, because most of the engineering work is essentially done using a graphical tool.

Requirements engineering for business processes is emerging in the last years, spurred by the realization that the business process obtained from business requirements shall be consistent with the requirements and the goals it aims to achieve [31]. Bleistein et al. [8] proposed a requirements engineering framework based on problem frames [26], goal modeling [34], and role activity diagrams [42] to incorporate a business strategy dimension in requirements analysis. De la Vara Gonzáles et al. [18] proposed an approach to align business processes and system requirements. Here, business strategies are used to drive both organizational design choices and the design of the IT infrastructure that supports them. Business processes are modeled using BPMN on the basis of the business infrastructure, and then used to derive and analyze business goals, which are in turn used to elicit system requirements. Loucopoulos [35] defnined the S^3 framework, a requirements engineering modeling framework, in which business processes are modeled along three orthogonal dimensions, namely strategy-oriented process modeling, which addresses the "why", service-oriented process modeling, which addresses the "what", and support-oriented process modeling, which addresses the "how". However, the S^3 framework does not provide formal analysis support in order to verify the consistency of the different business process representations. Yi and Johannesson [51] propose a formal approach for representing and reasoning with business goals. In their approach goals are specified in terms of business rules that defnine what the system should do and how in different business situations. These rules are specified in a first order many sorted temporal logic. Logic is very good to provide a precise model of the system behavior, but very bad at communicating such a model [33] to requirements engineers and stakeholders that may not have a background in formal methods.

The problem of filling the gap between the defninition of business processes and requirements engineering is not new also in the Tropos community. Kazhamiakin et al. [31] used Tropos to capture the strategic goals of an enterprise. The business process is then defnined in BPEL4WS on the basis of Tropos concepts extended with formal annotation expressed in Formal Tropos [22]. Formal Tropos extends Tropos with temporal logic-based annotations that characterize the evolution of a system, describing, for instance, how the network of relationships evolves over time. The consistency of the requirements model, as well as business processes, against business requirements and a strategic goal model is verified using a model-checking technique such as the one implemented in NuSMV. Differently from our approach, Formal Tropos does not permit to explicitly defnine temporal and data constraints between tasks. On the contrary, it defnines the partial ordering between tasks through message exchange using logic formulas. This makes Formal Tropos difficult to be used by non experts. Frankova et al. [20] proposed a framework for deriving the skeleton of secure business processes from early requirements analysis. This framework uses Secure BPEL, a specification language for secure business process based on BPEL, and provides a mapping between SI* [36], a requirements modeling framework that extends Tropos with concepts specific to security, and Secure BPEL. However, it does not offer any facility for formal analysis. In [10] a planning approach has been proposed to explore the space of alternatives and determine a (sub-)optimal plan (a sequence of actions) to achieve the goals of stakeholders. This framework mainly focuses on the analysis and evaluation of design alternatives, rather than the defninition of business process. As a consequence, the determined plan contains actions, such as goal decomposition, that are not proper to describe business processes.

The use of computational logic for the flexible specification and rigorous verification of agent interaction is adopted by many proposals.

A critical review of other works related to $\mathcal{S}$CIFF is given in [6]. For the purposes of $\mathcal{B}$-Tropos, some essential features of $\mathcal{S}$CIFF which are not to be found in other frameworks are: its ability to accommodate partially-specified models by way of abduction, and its explicit

representation of time, which enables specification and reasoning upon expressive temporal constraints, such as deadlines, not to mention its efficient implementation.

7 Conclusions

In this paper, we have presented an extension of Tropos agent-oriented software engineering methodology with declarative process-oriented constructs, defnined a complete mapping into the $\mathcal{S}$CIFF language and discussed its formal properties. Our work goes in the direction of narrowing the gap between the defninition of business processes and the agent-oriented analysis of organizational and business requirements. Most importantly, it puts a powerful tool in the hands of requirements engineers who want to follow an intuitive and flexible agent-oriented approach, and who are not necessarily familiar with logic formalisms, but wish nevertheless to benefit from the rigour of state-of-the-art formal verification methods. To the best of our knowledge, this is the first work that encompasses all these elements. We provide specification and verification tasks based on a single formal language, $\mathcal{S}$CIFF, and on a graphical interface to the user, the $\mathcal{B}$-Tropos notation.

This work is still in progress to better support requirements engineers in the modeling and analysis of business requirements. A limitation of the proposed framework concerns scalability issues. Large $\mathcal{B}$-Tropos models can be complicated and hard to understand. To address this issue, we are developing a CASE tool able to handle model complexity. The tool will also provide a front-end with $\mathcal{S}$CIFF for requirements analysis. Ideally, graphical models are automatically translated into $\mathcal{S}$CIFF specifications that are verified using the $\mathcal{S}$CIFF engine. The results of the reasoning tool are interpreted by the CASE tool that graphically shows possible property violations. Future research will also focus on the generation of executable business process specifications (such as BPEL and BPMN) from $\mathcal{B}$-Tropos models.

Acknowledgements 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.

References

1. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., & Mello, P. (2006). A verifiable logic-based agent architecture. In *Proceedings of the 16th international syposium on methodologies for intelligent systems*. Lecture Notes in Computer Science (Vol. 4203, pp.188–197). Springer.
2. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Montali, M., et al. (2006). Computational logic for run-time verification of Web services choreographies: Exploiting the SOCS-SI tool. In: M. Bravetti, M. Núñez, & G. Zavattaro, (Eds.), *Proceedings of the 3rd international workshop on Web services and formal methods*, Lecture Notes in Computer Science (Vol. 4184, pp. 58–72). Springer.
3. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., & Torroni, P. (2005). On the automatic verification of interaction protocols using g-SCIFF. Technical Report DEIS-LIA-04-004, University of Bologna.
4. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., & Torroni, P. (2006). Compliance verification of agent interaction: A logic-based software tool. *Applied Artificial Intelligence, 20*(2–4), 133–157.
5. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., & Torroni, P. (2006). Security protocols verification in Abductive Logic Programming: A case study. In: O. Dikenelli, M.-P. Gleizes, & A. Ricci (Eds.), *Proceedings of the 6th international workshop on engineering societies in the agents world*, Lecture Notes in Artificial Intelligence (Vol. 3963, pp. 106–124). Springer.
6. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., & Torroni, P. (2008). Verifiable agent interaction in abductive logic programming: The SCIFF framework. *ACM Transactions on Computational Logic (TOCL), 9*(4), 29.

7. Antón, A. I., & Potts, C. (1998) The use of goals to surface requirements for evolving systems. In *Proceedings of 20th the international conference on software engineering* (pp 157–166). IEEE Press.
8. Bleistein, J., Cox, K., Verner, J., & Phalp, T. (2005). Requirements engineering for e-business advantage. *Requirements Engineering, 11*(1), 4–16.
9. Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., & Perini, A. (2004). TROPOS: An agent-oriented software development methodology. *Journal of Autonomous Agents and Multi-Agent Systems, 8*(3), 203–236.
10. Bryl, V., Massacci, F., Mylopoulos, J., & Zannone, N. (2006). Designing security requirements models through planning. In *Proceedings of the 18th international conference on advanced information systems engineering*, Lecture Notes in Computer Science (Vol. 4001, pp. 33–47). Springer.
11. Bryl, V., Mello, P., Montali, M., Torroni, P., & Zannone, N. (2008) $\mathcal{B}$-Tropos: Agent-oriented requirements engineering meets computational logic for declarative business process modeling and verification. In: F. Sadri & K. Satoh (Eds.), *Proceedings of the 8th international workshop on computational logic in multi-agent systems*, Lecture Notes in Artificial Intelligence (Vol. 5056, pp. 157–176). Springer.
12. Buhr, R. J. A. (1998). Use case maps as architectural entities for complex systems. *TSE, 24*(12), 1131–1155.
13. Caire, G., Coulier, W., Garijo, F., Gomez-Sanz, J., Pavon, J., & Kearney, P., et al. (2004). The message methodology. *Methodologies and Software Engineering for Agent Systems, 11*, 177–194.
14. Chesani, F., Mello, P., Montali, M., & Torroni, P. (2009). Commitment tracking via the reactive event calculus. In: C. Boutilier (Ed.), *IJCAI 2009, Proceedings of the 21st International joint conference on artificial intelligence*, Pasadena, CA, USA, July 11–17, 2009, pp. 91–96.
15. Chesani, F., Mello, P., Montali, M., & Torroni, P. (2009). Verification of choreographies during execution using the reactive event calculus. In: R. Bruni & K. Wolf (Eds.), *Web services and formal methods, 5th international workshop, WS-FM 2008*, Milan, Italy, September 4–5, 2008, Revised Selected Papers, Lecture Notes in Computer Science (Vol. 5387, pp 55–72). Springer.
16. Chung, L. K., Nixon, B. A., Yu, E., & Mylopoulos, J. (2000). *Non-functional requirements in software engineering*. Kluwer Publishing.
17. Dardenne, A., van Lamsweerde, A., & Fickas, S. (1993). Goal-directed requirements acquisition. *Science of Computer Programming, 20*, 3–50.
18. De la Vara González, J. L., & Díaz, J. S. (2007) Business process-driven requirements engineering: A goal-based approach. In *Proceedings of the 8th workshop on business process modeling, development, and support*.
19. Fisher, M. (2005). Implementing temporal logics: Tools for execution and proof (tutorial paper). In: F. Toni & P. Torroni (Eds.), *CLIMA VI*, Lecture Notes in Computer Science (Vol. 3900, pp. 129–142). Springer.
20. Frankova, G., Massacci, F., & Seguran, M. (2007). From early requirements analysis towards secure workflows. In *Proceedings of the joint iTrust and PST conferences on privacy, trust management and security*.
21. Fung, T. H., & Kowalski, R. A. (1997). The IFF proof procedure for abductive logic programming. *Journal of Logic Programming, 33*(2), 151–165.
22. Fuxman, A., Liu, L., Mylopoulos, J., Pistore, M., Roveri, M., & Traverso, P. (2004). Specifying and analyzing early requirements in tropos. *Requirements Engineering, 9*(2), 132–150.
23. Haglind, M., Johansson, L., & Rantzer, M. (1998) Experiences integrating requirements engineering and business analysis an empirical study of operations & management system procurement projects. In *Proceedings of the 3rd international conference on requirements engineering* (pages 108–117). IEEE Computer Society.
24. Henderson-Sellers, B., & Giorgini, P. (Eds.). (2005). *Agent-oriented methodologies*. Idea Group Publishing
25. Iglesias, C. A., Garijo, M., & Centeno-González, J. (1999). A Survey of agent-oriented methodologies. In *Proceedings of the 5th international workshop on intelligent agents V, agent theories, architectures, and languages* (pp. 317–330). London, UK. Springer.
26. Jackson, M. (2001). *Problem frames: Analysing and structuring software development problems*. Addison Wesley.
27. Jacobson, I., Ericsson, M., & Jacobson, A. (1994) *The object advantage: Business process reengineering with object technology.* ACM Press/Addison-Wesley Publishing Co.
28. Jaffar, J., & Maher, M. (1994). Constraint logic programming: A survey. *Journal of Logic Programming, 19*(20), 503–582.
29. Johansson, H. J., McHugh, P., Pendlebury, A. J., & Wheeler, W. A. (1993). *Business process reengineering—Breakpoint strategies for market dominance*. Wiley.

30. Kakas, A. C., Kowalski, R. A., & Toni, F. (1993). Abductive logic programming. *Journal of Logic and Computation, 2*(6), 719–770.
31. Kazhamiakin, R., Pistore, M., & Roveri, M. (2004). A framework for integrating business processes and business requirements. In *Proceedings of the 8th international enterprise distributed object computing conference* (pp. 9–20). IEEE Press.
32. Kiyavitskaya, N., Moretti, R., Sebastianis, M., & Zannone, N. (2007). Project Report on the Initial Analysis of (Early) Requirements of Domain 1. TOCAI Deliverable D2.1, University of Rome "La Sapienza,". Online resource, http://www.dis.uniroma1.it/~tocai/
33. Lamport, L. (1994). How to write a long formula. *Formal Aspects of Computing, 6*(5), 580–584.
34. Liu, L., & Yu, E. (2001). From requirements to architectural design: Using goals and scenarios. In *Proceedings of the 1st international workshop on software requirements and architectures.*
35. Loucopoulos, P. (2003). The S3 (Strategy-Service-Support) framework for business process modelling. In *Proceedings of the workshop on requirements engineering for business process support*, CEUR Workshop Proceedings (Vol. 75). CEUR-WS.org
36. Massacci, F., Mylopoulos, J., & Zannone, N. (2007). An ontology for secure socio-technical systems. In *Handbook of ontologies for business interaction*. The IDEAGroup,
37. Mayr, H. C., Kop, C., & Esberger, D. (2007). Business process modeling and requirements modeling. In *Proceedings of the 1st international conference on the digital society* (p. 8). IEEE Computer Society.
38. Montali, M. (2009). *Specification and verification of declarative open interaction models: A Logic-based framework*. PhD thesis, University of Bologna.
39. Montali, M., Pesic, M., van der Aalst, W. M. P., Chesani, F., Mello, P., & Storari S. (2009). Declarative specification and verification of service choreographies. *ACM Transactions on the Web*, (accepted with minor revision.)
40. Montali, M., Torroni, P., Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., et al. (2008). Verification from declarative specifications using logic programming. In: M. G. de la Banda & E. Pontelli (Eds.), *Proceedings of the 24th international conference on logic programming*, Lecture Notes in Computer Science (Vol. 5366, pp. 440–454). Springer.
41. Montali, M., Torroni, P., Alberti, M., Chesani, F., Lamma, E., & Mello, P. (2010). Abductive logic programming as an effective technology for the static verification of declarative business processes. *Special Issue of Journal of Algorithms in Cognition, Informatics and Logic* (expected).
42. Ould, M. (1995). *Business process: Modelling and analysis for re-engineering and improving*. Wiley.
43. Rubens, J. (2007). Business analysis and requirements engineering: the same, only different?. *Requirements Engineering, 12*(2), 121–123.
44. Schonenberg, H., Mans, R., Russell, N., Mulyar, N., & van der Aalst, W. M. (2008). Towards a taxonomy of process flexibility. In: Z. Bellahsene, C. Woo, E. Hunt, X. Franch, & R. Coletta (Eds.), *Proceedings of the forum at the CAiSE'08 conference. CEUR workshop proceedings* (Vol. 344, pp. 81–84).
45. van der Aalst, W. M. P., & Pesic, M. (2006) A declarative approach for flexible business processes management. In *Proceedings of the 4th international conference on business process management*, Lecture Notes in Computer Science (Vol. 4103, pp. 169–180). Springer.
46. van der Aalst, W. M. P., & Pesic, M. (2006) DecSerFlow: Towards a truly declarative service flow language. In *Proceedings of the 3rd international workshop on web services and formal models*, Lecture Notes in Computer Science (Vol. 4184).
47. van der Aalst, W. M. P., ter Hofstede, A. H. M., Kiepuszewski, B., & Barros, A. P. (2003). Workflow patterns. *Distributed and Parallel Databases, 14*(1), 5–51.
48. Vilain, M., Kautz, H., & van Beek, P. (1990). Constraint propagation algorithms for temporal reasoning: A revised report. In: *Readings in qualitative reasoning about physical systems* (pp. 373–381). San Francisco, CA: Morgan Kaufmann Publishers Inc.
49. Wood, M. F., & DeLoach, S. A. (2001). An overview of the multiagent systems engineering methodology. In *Proceedings of first international workshop on agent-oriented software engineering* (pp. 207–221). Springer-Verlag New York, Inc.
50. Xanthakos, I. (2003). *Semantic integration of information by abduction*. PhD thesis, Imperial College London. Available online, http://www.doc.ic.ac.uk/~ix98/PhD.zip
51. Yi, C.-H., & Johannesson, P. (1999). Beyond goal representation: Checking goal-satisfaction by temporal reasoning with business processes. In *Proceedings of CAiSE'99* LNCS (Vol. 1626, pp.462–466). Springer.
52. Zambonelli, F., Jennings, N. R., & Wooldridge, M. (2003). Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering and Methodology (TOSEM), 12*(3), 317–370.