Applying Tropos to Socio-Technical System Design and Runtime Configuration

Fabiano Dalpiaz*, Raian Ali*, Yudistira Asnar*, Volha Bryl* and Paolo Giorgini*
*Dipartimento di Ingegneria e Scienza dell'Informazione
Università degli Studi di Trento
Email: {dalpiaz,ali,yudis.asnar,bryl,pgiorgio}@disi.unitn.it

***Abstract*—Recent trends in Software Engineering have introduced the importance of reconsidering the traditional idea of software design as a socio-tecnical problem, where human agents are integral part of the system along with hardware and software components. Design and runtime support for Socio-Technical Systems (STSs) requires appropriate modeling techniques and non-traditional infrastructures. Agent-oriented software methodologies are natural solutions to the development of STSs, both humans and technical components are conceptualized and analyzed as part of the same system. In this paper, we illustrate a number of Tropos features that we believe fundamental to support the development and runtime reconfiguration of STSs. Particularly, we focus on two critical design issues: risk analysis and location variability. We show how they are integrated and used into a planning-based approach to support the designer in evaluating and choosing the best design alternative. Finally, we present a generic framework to develop self-reconfigurable STSs.**

I. Introduction

Socio-technical systems, introduced by Emery and Trist [1], [2], identify a particular class of systems characterized by an interplay between their social and technical components; in other words, a socio-technical system is composed not only of hardware and software, but also of human agents. STSs present specific properties, among which [3]:

- emergent properties arising from the system as a whole, rather than from the individual components;
- non-determinism, since humans do not always react in the same way;
- dynamic organizational objectives, because objectives can have different (subjective) interpretations and may vary over time.

A particularly relevant and promising application area for socio-technical systems is Ambient Intelligence (AmI). Figure 1 presents an AmI scenario concerning *crisis management*, which we will use further in the paper as a motivating case study. A camera detects a possible fire in a building and, in order to avoid false alerts, it asks another camera for confirmation (1). A sound alert cannot be activated, because the (in-place) alarm ring is out of order (2). The system should therefore *self-reconfigure*: the alternative is to call the firemen (3), check the current traffic status to support the rescue teams (4), and alert the fire warden (6). The socio-technical nature of this scenario becomes clear during this reconfiguration step: calling firemen and alerting the fire warden involves human activities, whose outcome is unpredictable, whereas checking the traffic status involves humans only in a further step, when traffic should be re-routed (5). Even in such a simplified scenario, the interplay between humans and technical subsystems is manifest, and shows the complex nature of designing and supporting STS at runtime.

Fig. 1. An Ambient Intelligence crisis management scenario for Socio-Technical Systems.

The use of *agents*, with sociality, autonomy, and proactivity as key characteristics, can be beneficial for socio-technical systems (and, consequently, AmI systems) design and runtime support. Indeed, in an STS the social interaction among the computational units is essential in pursuing the system goals. System components (both technical and social) need autonomy to take decisions locally, to choose when and how they need to achieve their objectives.

Tropos [4] is an agent-oriented software engineering methodology, which bases on the Belief-Desire-Intention (BDI) paradigm [5], [6]. Tropos models the system as a set of interacting agents[1]. Each agent has a set of *goals* to fulfill, and a number of *tasks* that describe how to achieve goals. An agent can provide or require *resources* to execute tasks. Soft-goals represent those goals, such as software qualities, for which fulfillment there is no clear-cut criteria. Goal-to-goal connections can be set through (a) *and-decomposition* to split a goal into a number of concurrent sub-goals; (b) *or-decomposition*

[1]An agent can be a human or an artificial agent.

to represent a number of alternative sub-goals. Goals are connected to tasks through *means-end* decomposition: the task is a means to achieve an end (the goal). *Contribution* relations link goals, tasks, and resources to soft-goals; a contribution from an element x to a soft-goal s represents how well x contributes to the satisfaction of s. Finally, agents depend on each other – via *dependencies* – for goal achievement, task execution, and resource provision.

These aspects of Tropos appear useful for the development of socio-technical systems, where hardware/software agents coexist with human actors. However, the original [4] Tropos modeling framework alone is not sufficient to capture all aspects of STSs, and in this paper, we illustrate a collection of Tropos extensions that can be used to better address both the development and the support of STSs at runtime. The first design-time extension concerns the modeling of location variability; the location where an agent is situated can requires specific strategies to be used at runtime. The second extensions we introduce is a framework to handle uncertainty in the development of STSs: the Goal-Risk (GR) framework is a modeling technique, accompanied by analysis tools, whose objective is the minimization of risk. These extensions to the original Tropos framework can be integrated in a planning-based framework that can support a designer in exploring the design-time space of alternatives. The framework allows the designer to look at all possible designs and on the base of a number of criteria decide on such alternatives. Finally, we introduce two approaches to support self-reconfigurable STSs. Essentially, we propose two approaches to use the Tropos goal diagrams to monitor the execution of a system. The former adopts a centralized reconfiguration engine, while the latter a decentralized reconfiguration enacted by each agent.

The paper is organized as follows: Section II discusses two concerns which are relevant in the design of STSs in an AmI setting: location variability and risk; Section III shows how AI planning techniques can be used to automate design-time analysis; Section IV analyzes autonomic STSs and the property of runtime self-reconfiguration. Finally, Section V presents final remarks and future work.

II. Modeling STS in AmI settings

We present two different techniques to support the design of socio-technical systems. The rationale behind these proposals comes from two important aspects in STSs, and whose importance gets even wider when considering Ambient Intelligence settings; these concerns are location-based variability and risk.

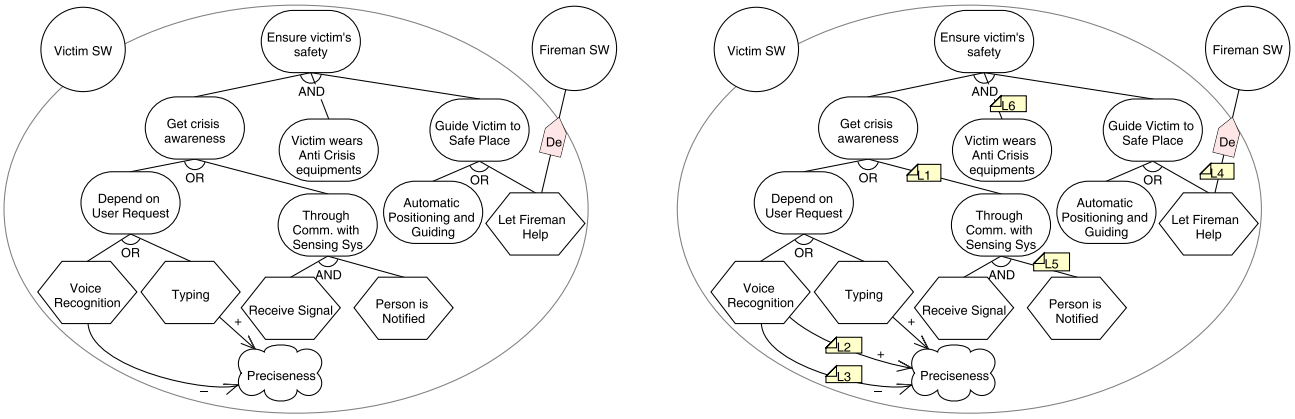
A. Modeling Location-based Variability

The rationale of an agent often contains behavioral variability, where the agent can choose among alternative strategies (*tasks*) to fulfill the same objective (*goal*). A proper variability modeling should include means to represent how the selection between these alternatives is performed, specifying *when* and *where* a certain alternative is applicable. STSs are characterized by a dynamic location, in which both technical and social components vary over time. We claim that location is an important criteria that can constrain and guide the selection of the most suitable alternative.

Figure 2a shows a partial goal model for a software agent working on behalf of a victim of a crisis (e.g., a fire). To ensure victim safety, the software has to be aware of the crisis, which can be done through an explicit request from the victim by a voice or typed command or through the continuous automated analysis of the signal that comes from some sensing system. To ensure safety, the victim might need to wear special dress (e.g. anti-fire coat). Then, the victim has to be guided to a safe place through an automatic tracing and directing, or through the help of a fireman.

The original Tropos goal model supports modeling alternatives for satisfying goals, but lacks tools for specifying the locations where specific alternatives are applicable. (see the model in Figure 2a). In our previous work [7], [8], [9], we extended Tropos goal model to represent the relation between goal satisfaction alternatives and location. Our modeling enables an agent to answer several important questions such as: what are the possible, impossible, or recommended alternatives to satisfy a goal in a specified location. In our approach, (i) a *location property* is a boolean predicate evaluated against the current location; (ii) a *location-based variation point* is an element in the goal model to which a location property (Li on Figure 2b) can be associated. We defined five location-based variation points:

1) Location-based Or-decomposition: Or-decomposition is the basic variability construct; the choice of a specific Or-alternative might be based on a location properties that inhibits, allows, or recommends some alternatives. E.g. having crisis awareness through communicating with a sensing system requires both that a sensing system exists and the user's PDA has the ability to connect to it ($L1$).
2) Location-based contribution to softgoals: the value of the contributions to the softgoals can vary from one location to another. We need to specify the relation between the location and the value of the contribution. E.g. receiving the user request through voice recognition contributes positively to the softgoal *"Preciseness"* when the level of noise is low and the system is trained enough for recognizing that user voice ($L2$), while it contributes negatively in the opposite case ($L3$).
3) Location-based dependency: in some locations, an actor might be unable to satisfy a goal using its own alternatives. In such case, the actor might delegate this goal to another actor that is able to satisfy it. E.g. guiding the person in crisis by a fireman is an alternative that needs a free and skilled fireman that is close to and can reach that person ($L4$).
4) Location-based goal / task activation: an actor, and depending on the location settings, might find necessary or possible triggering (or stopping) the desire of satisfying a goal / the execution of a task. E.g. notifying a person about crisis has to be triggered when the analysis of the signal that comes from the sensing system addresses



(a) Modeling with Tropos

(b) Modeling with location-based Tropos

Fig. 2. Partial goal model for the crisis management scenario.

some potential danger ($L5$).

5) Location-based And-decomposition: a sub-goal might (or might not) be needed in a certain location, that is some sub-goals are not always mandatory to fulfil the top-level goal in And-decomposition. E.g. the need of a person in crisis to wear special equipments depends on the category of the crises, and the skills the person in danger has ($L6$).

The analysis of the location properties will lead to the definition of the location model that can be modeled using class diagram as we did in [8]. In [9], we described a process to derive a location model – describing the location in terms of its entities and the links between them – from the location properties in a goal model. The location model can be instantiated to represent a certain location and enable automated reasoning. The evaluation of the location properties will enable an agent to derive the possible alternatives for satisfying its goals. The proposed extension of Tropos goals modeling constructs are colored in gray in the metamodel of Figure 3.

By formalizing the goal model, the location model, and the location preperties, it becomes possible performing several kinds of analysis. We outline now three types of automated analysis:

1) Location-based goal satisfiability (LGS): it verifies whether a goal is achievable by choosing a certain alternative in a specific location.
2) Location properties satisfiability (LPS): this analysis checks if the current location structure is compliant with a set of goals. This techniques can be used to identify what is missing in a particular location where some top-level goals have been identified as unsatisfiable by LGS.
3) Preferences analysis (PA): this type of analysis requires the specification of preferences over alternatives. Preferences can be modeled using soft-goals as in [10]. We need this analysis in two cases:
   a) when there are several alternatives to satisfy a goal: the selection will be based on the contributions to preferred soft-goals.
   b) when there is no applicable alternative: in this case, LPS might provide several proposals about the needed location modifications.

   The adopted modifications are those leading to better satisfaction of the preferences expressed over soft-goals.

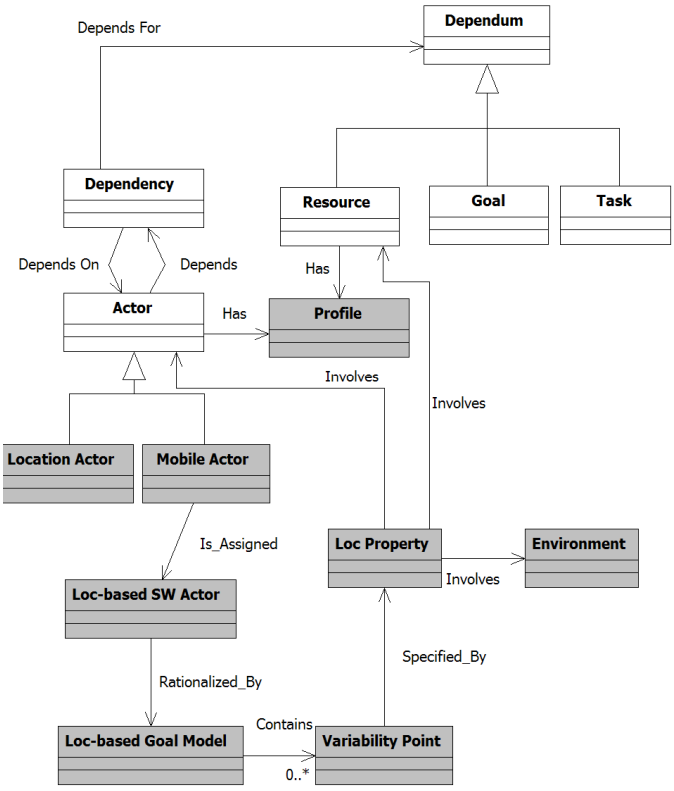


Fig. 3. Metamodel showing the extension of Tropos with location.

B. Modeling Uncertainty through Risk Analysis

STSs are exposed to a wide range of uncertainty during their development, runtime, and maintenance. Some uncertainties can result into system failures, and can even put human lives

in danger. There is no such systems that are free of failure: if something *can go wrong* then it *will go wrong* [11]. Therefore, designers should consider uncertainties that could lead to failures that harm the system and treat them.

In [12], we proposed the Goal-Risk (GR) modeling framework, that extends Tropos by providing modeling constructs to represent uncertain events that may affect the organization negatively (called risks) or positively (called opportunities). Indeed, it is hardly possible to nullify the risks that threaten a system since the system can fail as well in case of normal operations (e.g., operator errors, bad maintenance) or malicious intentions (e.g., attacks, frauds).

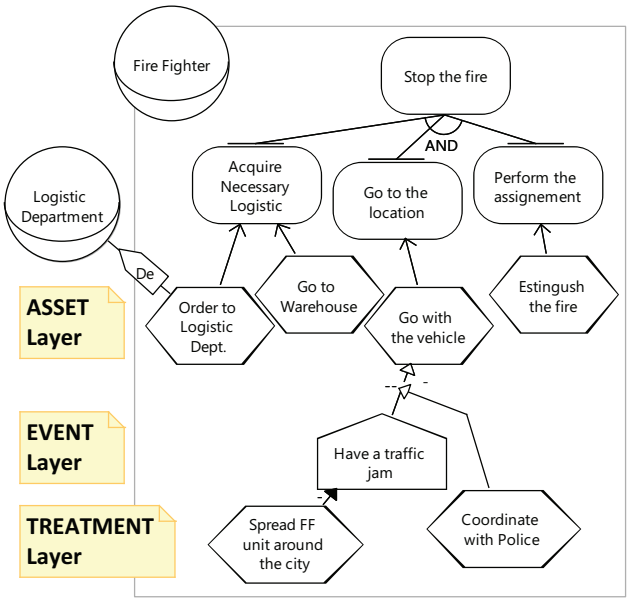


Fig. 4. The Goal-Risk Framework

Conceptually, a Goal-Risk (GR) model (see Figure 4) is composed by three-layers: *asset* to capture the goals of the stakeholders (e.g., firefighter intends to stop a fire), and tasks and resources required to achieve the goals (e.g., "Go to warehouse to obtain the logistic"), *event* (pentagons) to model uncertainty events (e.g., risks, opportunities) that affect the asset layer (e.g., having a traffic jam), and *treatment*, depicted as tasks, to capture additional measures that are required to treat the risks (e.g., spread firefighter units around the city). This framework is equipped with two basic reasoning mechanisms to help designers in making decisions. First, *forward reasoning* aims at calculating the risk level of an organization for a given setting (e.g., value of goals, adopted treatments) and inputs (e.g., likelihood-severity of event). Second, *backward reasoning* aims at eliciting the possible solutions (e.g., strategy to achieve the goals and necessary treatments to mitigate the risks) for a given set of constraints (e.g., tolerable risk level).

A socio-technical system is composed by several agents, each having its own goals, tasks, resources, and, moreover, each exposed to different risks. An agent often cannot fulfill all its goals, and needs to depend on others to satisfy some subgoals, execute some tasks, or provide resources. Thus, a system can be viewed as a network of agent dependencies. In secure and dependable systems, this phenomena emerges as one of the critical points because a vulnerable agent can put the entire system at risk.

In [13], we extend the GR framework to the case of a multi-agents setting and illustrate how risks are propagated from an agent across the organization. In certain cases, agents should depend on other agents that they do not trust due to some reasons (e.g., there are not other choices, the regulation orders to do it). In such a setting, the agents often perceives a higher level of risk as if they depend on the ones they trust. Essentially, we can infer how much risks that an agent perceives is based on its trust relations [14] and evaluate whether adopted treatments are perceived to be effective by agents in mitigating the risks.

Finally, considering risks is critical to ensure the socio-technical systems being dependable and operating securely. Risk analysis is a continuous process, and therefore risks must be monitored during runtime and be reviewed regularly as long as the system is still in use.

III. Automating the Design: a Planning-Based Approach

The planning-based extension of Tropos [15] has been proposed to support a designer in exploring the space of alternative designs of a socio-technical system. Indeed, the fulfillment of each of the system goals is related to a number of choices of how the goal is decomposed and which are the actors the goal (or its subgoals) are delegated to. The idea behind planning-based framework is that the task of constructing a requirements model for a socio-technical system, i.e. a network of delegations among actors for goals, can be framed as a *planning problem* where selecting a suitable social structure corresponds to selecting a plan that satisfies the stakeholders goals.

This work adopts AI (Artificial Intelligence) planning [16] techniques to the domain of requirements engineering. AI planning is about automatically determining a course of actions (i.e., a plan) needed to achieve a certain goal where an action is a transition rule from one state of the world to another. To define a planning problem, one should specify (i) the initial state of the world, (ii) the desired state of the world, and (iii) the actions. In our planning-based framework, *goal decomposition*, *delegation* and *fulfillment* are seen as actions that the designer ascribes to the actors of the system-to-be and of its organizational environment. We use PDDL (Planning Domain Definition Language) 2.2 [17] to formally specify the initial organizational setting and actions of the domain. An off-the-shelf planning tool, LPG-td [18], is adopted for the implementation of the planning domain.

In [15] we have presented the basic set of first-order predicates used to formalize the organizational setting in terms

of actors and goals, their properties (e.g. actor capabilities), and social dependencies among actors.

The flexibility of the PDDL specification language makes it possible to accommodate various criteria into the planning problem definition. In the following, we list the extensions of our framework related to three different aspect of socio-technical system development and deployment.

An application of our planning-based framework to the domain of **secure system design** presented in [19] supports trust and permission concepts of Secure Tropos [20]. The planning domain is defined so that it guarantees that the resulting socio-technical model satisfied the trust and permission related constraints imposed on it (e.g., no goal is delegated along an untrusted link). In the crisis management scenario, presented in Section I, the examples of such security constraints can be related to the permission to activate the alarm, which only a limited set of actors possess, or to trust relations between the firemen and the fire warden of the building.

Yet another extension of our framework [21] uses risk-based evaluation metrics for selecting a suitable design alternative, and aims at **agent-based safety critical** applications. In this work, the risk-based criteria (e.g. related to the criticality of a goal satisfaction or minimum acceptable level of trust between agents) and the respective framework discussed in Section II-B of the present paper, are incorporated into the planning-based procedure which supports a socio-system design (as well as a system redesign at runtime). This work aims at proposing a design that maintain the risk level within the acceptable limits. In the crisis management scenario, the examples of risk constraints are the ones related to the way to alert the workers about the danger (the most reliable one should be chosen among the available alternatives), or to the level of trust between the firemen and the fire warden of the building and, accordingly, the goals that can be delegated between these actors.

Location can be used as a metrics for evaluating alternatives, as well. As we showed in Section II-A, location properties associated to variation points can be used to (a) limit the range of alternatives an agent can choose among; (b) express location-dependent contribution to soft-goals. In such a way, the planning-based approach we suggest here can be customized to discard unavailable options and to exploit soft-goal satisfaction for ranking available alternatives.

IV. Autonomic Socio-Technical Systems: Runtime Self-Reconfiguration

The previous sections focused on various facets of the design of STSs, proposing both modeling languages and analysis/reasoning techniques. The design of an STS is a fundamental activity, which helps preventing the development of a system that violates its requirements (both functional and non-functional) at runtime.

Nevertheless, design-time support is not sufficient to provide a comprehensive support for socio-technical systems. Runtime violation of requirements [22] is recognized as an open problem and has been explored since several years. Feather et al. [23] propose an approach to reconcile requirements with runtime behavior, where both the design- and run-time phases are covered: (a) anticipate as many violations as possible at specification time, and (b) detect and resolve the remaining violations at runtime. Though Feather's work is not targeted for socio-technical systems, it points out problems and proposes solutions which apply also in the context of STSs.

There is hence a clear need for a runtime framework which complements the design-time techniques we have presented. An STS exhibits particular properties that set specific requirements for the execution infrastructure:

- humans play an active role and should interact with the technical sub-systems at runtime;
- the location (both the physical and the social aspects of location) is in continuous evolution;
- the system should self-reconfigure adapting to the changing environment where it operates;
- failures should be compensated and an alternative plan should replace the failed plan.

Multi-agent system infrastructures represent a good candidate to support socio-technical systems at runtime. In particular, those based on the BDI paradigm are particularly suitable, since Tropos is founded on BDI. However, STSs exhibit some features which are not considered in the classical BDI paradigm, such as the interplay between software and human agents. Therefore, existing infrastructures need customization to result an effective solution for STSs.

The approach we have taken in our research is to link Tropos to BDI-based software architectures. This solution enables us to combine different state-of-the-art techniques extending the capabilities of BDI. Agents represent the core concept at runtime; each agent has a goal-based specification, executes plans to achieve its active goals, and depends on other agent for plan execution, goal achievement, and resource provision. Two different but complementary approaches can drive the self-reconfiguration process, with distinct properties and application scenarios:

- centralized self-configuration: some types of STS, such as a scientific institution, can work properly only if a centralized knowledge of the various agents is available, and self-reconfiguration is therefore controlled centrally. We explored this first type of self-reconfiguration in [24].
- decentralized self-configuration: this approach presents self-configuration from the local perspective of an individual agent. Each agent commits to achieve its own goals at best, without having a complete knowledge of the STS. This solution cannot achieve the same level of optimality a centralized approach guarantees, but is the only available solution whenever the internals of the agents cannot be disclosed to a centralized supervisor. An example of this situation is two software systems that interact but belong to different companies: the interfaces are available, but the companies will not disclose the internal reasoning. We proposed an initial approach supporting this vision in [25].

In [24] we have presented an approach to dynamic reconfiguration of a socio-technical system structure in response to internal or external changes. The paper suggests a **centralized reconfiguration mechanism**, which aims at making a socio-technical system self-configuring, and proposes a multi-agent architecture for its implementation.

The proposed reconfiguration mechanism

- collects and manages the information about the system;
- evaluates both the system state (e.g. the overall workload), and the local utilities of each agent to decide whether the system needs to be redesigned in response to external or internal changes;
- and, if the above evaluation shows that the reconfiguration is needed, replans the system structure in order to optimize it with respect to the evaluation criteria of interest.

The notification about the change is obtained either from the inside of the system or from the environment. Each system agent is obliged to communicate to some central point if it committed to, or achieved a goal. Four types of triggering events are supported, namely, the situations when a new agent enters the system, or the existing one leaves, when a new system goal is introduced, or one of the old ones is satisfied. However, due to the flexibility of the PDDL representation, it is possible to extend the formalization to support the changes in the agents' capabilities and commitments, failures when achieving goals, etc.

This framework can be applied to the organization of firemen rescue teams, where different alternatives are available (truck type, fire fighting approach, firemen equipment) and several agents are involved. In this scenario, finding the optimal solution is fundamental, and the centralized planning-base approach is the best choice.

Talos [25] is an architectural approach to self-reconfiguration based on a **decentralized reconfiguration mechanism**, where self-reconfiguration is seen from the perspective of each agent/component. Three different subsystems are the core of the self-reconfiguration process each agent performs:

- *Monitor*: the agent should continuously monitor both its internal state and the location where it is running (similarly to what happens for the centralized approach). The internal state is evaluated verifying the status of the agent's goals, detecting new goals, failures, and fulfillments. A mailbox is exploited to figure out the incoming requests from other agents, who want to interact to achieve their own goals. The external context is monitored receiving events from the set of artifacts which can be seen or are used by the agent.
- *Diagnose*: monitored events are linked to the goal model by traceability links, triggering new top-level goals and notifying failures or achievements. Diagnosis provides different levels of detail depending on the chosen goal monitoring granularity; in Talos we exploit a variant of Wang's goal monitoring switches [26]: the closer the monitoring switches are to the plans, the more detailed diagnosis we can obtain. A particular kind of diagnosis is related to the enactment of dependencies, where other agents provide information concerning dependency requests (e.g., refuse, contract, accept).
- *Compensate*: after detecting and diagnosing a failure, the following step consists of taking a countermeasure to this failure. We propose the execution of two sub-tasks to properly carry out this activity:
  - A compensation plan should be executed to "undo" the effects of the failed plan. An important information from diagnosis is to understand which action of the plan failed, or if the plan failed to achieve the goal though it terminated correctly.
  - A self-reconfiguration process is enacted to choose another strategy to achieve the goal of which a failure event was generated. A variant of goal analysis is used to perform this step.

The decentralized solution is suitable in the fire fighting scenario, as well. In the scene described in Figure 1 the out-of-order bell inhibits the best overall alerting strategy (playing a sound alarm), and this failure requires a prompt local reconfiguration (e.g., alerting the fire warden). Involving a central control unit would produce delays and could result in a failure, especially if the fire damaged the physical network enabling the communication with the central unit.

V. Conclusion and future work

In this paper, we have presented and analyzed a number of extensions to the Tropos methodology to support the development and the runtime operation of a complex class of modern systems, namely socio-technical systems. The design and runtime emergent properties of these systems present a lot of challenges for developers and there is a clear need for innovative engineering tools and techniques.

We have presented two design-time modeling and reasoning techniques, focused on location properties of an STS and risk analysis, respectively. Also, the problem of runtime reconfiguration of a socio-technical structure was addresses in the paper with two approaches, centralized and decentralized, suitable each for different application areas.

As future work, we believe it will be important to further elaborate and better integrate the techniques presented in this paper. Particularly, we would like to work on the implementation of an integrated CASE tool for the development of STSs.

VI. Acknowledgements

This work has been partially funded by EU Commission, through the SENSORIA, SERENITY, and MASTER projects, and by the PRIN program of MIUR under the MEnSA project.

References

[1] F. Emery, "Characteristics of socio-technical systems," *London: Tavistock*, 1959.

[2] F. Emery and E. Trist, "Socio-technical systems," *Management Science, Models and Techniques*, vol. 2, pp. 83–97, 1960.

[3] I. Sommerville, *Software Engineering*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2006, ch. Socio-Technical Systems.

[4] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos, "Tropos: An agent-oriented software development methodology," *Autonomous Agents and Multi-Agent Systems*, vol. 8, no. 3, pp. 203–236, 2004.

[5] A. Rao and M. Georgeff, "An abstract architecture for rational agents," *Proceedings of Knowledge Representation and Reasoning (KR&R-92)*, pp. 439–449, 1992.

[6] ——, "Bdi agents: From theory to practice," *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pp. 312–319, 1995.

[7] R. Ali, F. Dalpiaz, and P. Giorgini, "Location-based variability for mobile information systems," *Proceedings of the 20th International Conference on Advanced Information Systems Engineering (CAiSE'08)*, 2008.

[8] ——, "Modeling and analyzing variability for mobile information systems," *Proceedings of the 4th Ubiquitous Web Systems and Intelligence Workshop (UWSI 2008)*, 2008.

[9] ——, "Location-based software modeling and analysis: Tropos-based approach," *Proceedings of the 27th International Conference on Conceptual Modeling (ER 2008)*, 2008.

[10] S. Liaskos, S. McIlraith, and J. Mylopoulos, "Representing and reasoning with preference requirements using goals," Tech. rep. CSRG-542, Computer Science Department, University of Toronto, Tech. Rep., 2006.

[11] B. Schneier, *Beyond Fear: Thinking Sensibly about Security in an Uncertain World*. Springer, 2003.

[12] Y. Asnar and P. Giorgini, "Modelling Risk and Identifying Countermeasures in Organizations," in *Proc. of CRITIS'06*, ser. Lecture Notes in Computer Science, vol. 4347. Springer, 2006, pp. 55–66.

[13] Y. Asnar, R. Moretti, M. Sebastianis, and N. Zannone, "Risk as Dependability Metrics for the Evaluation of Business Solutions: A Model-driven Approach," in *Proc. of ARES'08*. IEEE Press, 2008.

[14] Y. Asnar, P. Giorgini, F. Massacci, and N. Zannone, "From Trust to Dependability through Risk Analysis," in *Proc. of ARES'07*. IEEE Press, 2007.

[15] V. Bryl, P. Giorgini, and J. Mylopoulos, "Designing Cooperative IS: Exploring and Evaluating Alternatives," in *CoopIS'06*, 2006, pp. 533–550.

[16] D. S. Weld, "Recent Advances in AI Planning," *AI Magazine*, vol. 20, no. 2, pp. 93–123, 1999.

[17] S. Edelkamp and J. Hoffmann, "PDDL2.2: The Language for the Classical Part of the 4th International Planning Competition," University of Freiburg, Tech. Rep. 195, 2004.

[18] LPG Homepage, "LPG-td Planner," http://zeus.ing.unibs.it/lpg/.

[19] V. Bryl, F. Massacci, J. Mylopoulos, and N. Zannone, "Designing Security Requirements Models Through Planning," in *CAiSE'06*, 2006, pp. 33–47.

[20] P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone, "Modeling Security Requirements Through Ownership, Permission and Delegation," in *Proc. of RE'05*. IEEE Press, 2005, pp. 167–176.

[21] Y. Asnar, V. Bryl, and P. Giorgini, "Using risk analysis to evaluate design alternatives." in *AOSE*, ser. Lecture Notes in Computer Science, L. Padgham and F. Zambonelli, Eds., vol. 4405. Springer, 2006, pp. 140–155.

[22] A. van Lamsweerde, "Divergent views in goal-driven requirements engineering," *Foundations of Software Engineering*, pp. 252–256, 1996.

[23] M. Feather, S. Fickas, A. Van Lamsweerde, and C. Ponsard, "Reconciling system requirements and runtime behavior," *Proceedings of the 9th International Workshop on Software Specification and Design*, pp. 50–59, 1998.

[24] V. Bryl and P. Giorgini, "Self-Configuring Socio-Technical Systems: Redesign at Runtime," in *SOAS'06*, 2006.

[25] F. Dalpiaz, P. Giorgini, and J. Mylopoulos, "Talos: an architecture for self-reconfiguration," DISI-08-026, Tech. Rep., 2008.

[26] Y. Wang, S. McIlraith, Y. Yu, and J. Mylopoulos, "An automated approach to monitoring and diagnosing requirements," *Proceedings of the 22nd IEEE/ACM international conference on Automated software engineering*, pp. 293–302, 2007.