Exploring Alternative Designs for Sociotechnical Systems

Fatma Başak Aydemir, Paolo Giorgini, John Mylopoulos
University of Trento
Trento, Italy
{aydemir, pg, jm}@disi.unitn.it

Fabiano Dalpiaz
Utrecht University
Netherlands
f.dalpiaz@uu.nl

***Abstract*—Sociotechnial systems (STSs) consist of a complex interplay of technical components, humans, and organizations. As other types of systems, STSs need to evolve in response to changing requirements and operational environments. Evolving STSs is a complex activity, which requires reconfiguration of technical components as well as rerouting of interactions among human and social actors. Moreover, reconfiguration has to respect participant autonomy, while coping with conflicting goals and noncooperation in identifying a configuration that minimizes changes relative to the current configuration.**

In this paper, we present a framework that supports design and evolution of STSs. The framework includes (i) the *DEST* language for modeling STSs as goal-oriented actors that interact via social commitments; (ii) techniques for building a network of interactions that fulfills participant requirements; and (iii) techniques for evolving an existing STS while minimizing change. We encode the design and evolution of STSs as an automated planning problem.

goal models; planning; sociotechnical systems; requirements engineering

I. Introduction

Exploring alternative solutions and constructing a plan to realize them is essential in times of crisis. This occurs in very different contexts: governments devise new policies to cope with economic crises, organizations re-engineer their processes to boost competitiveness in changing market conditions [1], software companies create new releases to improve customer satisfaction [2], etc.

Sociotechnical systems (STSs) are systems-of-systems in which technical, social, and organizational subsystems interact [?], [?]. Each of the systems has its own requirements—that often conflict with other systems' requirements— and the satisfaction of some of these requirements depends on the success of the interactions with other systems.

We encounter many STSs in our lives. A university is an STS that includes information systems, administrative staff, students, professors, etc. Air traffic management is an STS that involves flight controllers, pilots, radars, airliners, etc. Hospitals are STSs that include doctors, patients, nurses, local governments, visit reservation systems, patient information systems, and the like.

STSs are in constant evolution, for they need to cope with the different types of changes that can occur: the participating systems' requirements may change, new systems may join, some systems may leave, fail, or become untrustworthy [?], [?]. For example, in a university, professors may stop using an information system that they deem obsolete, the higher education ministry may require staff to check-in/out, the university board may impose a maximum time for students to obtain a degree, the dean may leave, etc.

In this paper, we support the design and evolution of STSs through the proposal of a planning framework that uses high-level requirements models of an STS to identify alternatives and to suggest a plan for implementing these alternatives in the STS. A plan is a sequence of actions through which participants requirements are satisfied [?] and Gans et al. [?] show that a design problem can be modeled as a planning problem. Our framework recommends the plans that preserve the stability of the STS by minimizing the cost for adopting/implementing the plan.

We represent the requirements of participating systems via goal models [3], [4]. Specifically, in order to capture the contractual relationships among STS participants, we adopt and extend the work on goals and social commitments by Chopra et al. [5].

The paper makes the following contributions:

- Introduces the *DEST* (*DE*signing *S*ociotechnical sys*T*ems) requirements modeling language to represent the actors in an STS, their goals, and their social interactions. The language also supports expressing the capabilities of an actor to achieve goals, goal conflicts, and other requirements such as priorities on goals and temporal ordering constraints.
- Formalizes the notion of an STS configuration and design plan, and shows how these concepts are useful for describing how to design an STS from scratch. A plan is a sequence of actions that leads to the establishment of a network of interactions among the subsystems of the STS that fulfills their requirements.
- Proposes a framework for supporting the evolution of an STS. The framework responds to the occurrence of triggering events in the STS (new requirements, exiting actors, etc.) by identifying a redesigned plan that reconfigures the STS to deal with the changed context.
- The problems of design and evolution are encoded in the PDDL 3.0 planning domain language—for off-the-shelf planners.

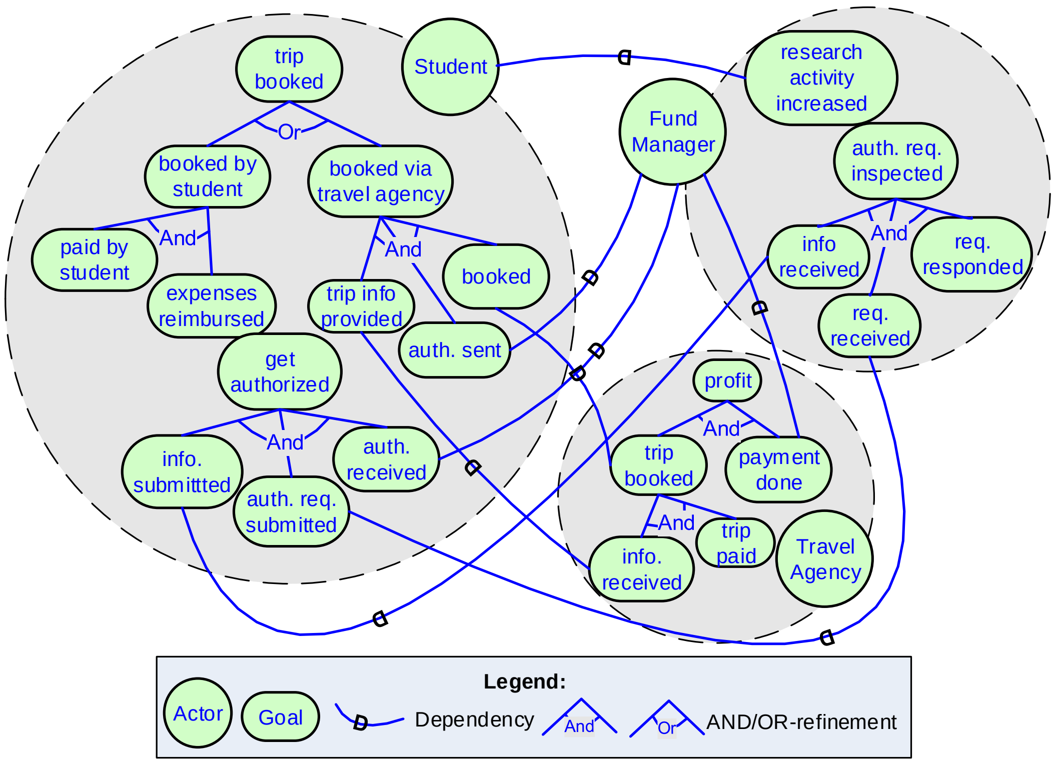

Fig. 1: Partial *i**/Tropos goal model for the travel authorization and reimbursement STS

- Presents a preliminary evaluation of our approach, both in terms of its visual scalability, and of the scalability of the planning mechanisms.

The paper is structured as follows. Section II presents our baseline on goal models, social commitments and planning. We introduce the *DEST* language to describe STSs in Section III. Section IV details how to build a sociotechnical system using *DEST*. Section V extends the framework of Section IV to support STS evolution. The details of implementation in PDDL is given in Section VI. Section VII presents a preliminary evaluation of our approach. Section VIII compares our work with the relevant literature. Finally, Section IX presents our conclusions and outlines future work.

II. Research Baseline

Our research baseline consists of two state-of-the-art modeling frameworks: *goal models* for representing requirements, and *commitments* for modeling social interactions.

We illustrate our baseline and our proposal with the aid of the following case study concerning the travel authorization and expense reimbursement system at the University of Trento. We build our models on the basis of the outcomes of an interview with domain experts. Travel authorization and the reimbursement system of the university is an STS where the social actors including individuals and departments interact with each other through technological actors such as information systems.

The STS should satisfy the requirements of participating actors, and currently provides alternative ways of doing so, but needs to operate in the most cost–efficient way as the resources of the university are limited. Since the university encourages all of its personnel to participate in research activities, and there is an underlying bureaucracy for travel application, approval, and transaction processing, the social actors are not only academics, but also administrators, financial offices, department heads, research fund managers, travel agents, and travelers such as students, post-docs, professors, etc. Interactions among these actors are crucial for fulfilling their respective requirements. Design here amounts to establishing a network of interactions defined by commitments where requirements are fulfilled if all actors respect their obligations.

A simple version of the STS described above is as follows. There are three actors: (i) student, (ii) project fund manager, (iii) travel agent. These actors are heterogeneous in their requirements and their capabilities within the STS. For example, the student is concerned with the planning of a trip; on the other hand, the project fund manager aims to increase project research activity, while the travel agent wishes to make profit through the selling of tickets to the other two actors. As far as capabilities are concerned, the project fund manager is capable of making payments, and authorizing trips while the travel agent can book trips.

Goal-oriented requirements engineering [6] founded on the premise that stakeholder requirements can be modeled and analyzed as goals. We use the *i**/Tropos [3], [7] modeling language, which captures not only the goals assigned to a software system (as KAOS [4] does), but traces these goals back to the stakeholders (actors) that want them, and also represents social dependencies among those actors.

Fig. 1 shows a goal model in *i**/Tropos. *Actors* are visualized as circles, and an actor boundary is represented by a gray circle with dashed borders. *Goals* are ellipses linked to their AND/OR refinements via straight, solid lines. When an *actor* depends on another actor for the satisfaction of a goal, a dependency link is used to model this social relationship. For example, the student depends on the project fund manager to get authorization. Similarly, the project fund manager depends on the student for increased research output.

As shown in [5], [8], traditional goal models need to be extended to adequately represent STSs that consist of autonomous actors. In particular, social dependencies, as used in *i** and Tropos, do not capture the reciprocal nature of most social dependencies between a depender and a dependee.

To address this limitation, we rely on the notion of social commitment—denoted as *C(debtor, creditor, antecedent, consequent)*—, a contractual relation between two parties to bring out certain states of affairs [9]. The *debtor* commits to satisfy the *consequent* for the *creditor*, if the *antecedent* comes to hold.

Representing social interactions with commitments solves the problems with dependencies stated above, for commitments are created by the creditor when it communicates this to the debtor, and they are reciprocal by their own nature.

Consider the dependencies between student and fund manager in Fig. 1. The dependency of student for the authorization is directly related to the dependency of project fund manager for increased research output. Indeed, project fund manager gives the authorization if there is an increase in the research output by the student. In this case, a commitment would adequately capture the relationship between these two dependencies: when represented in the form *C(fund manager, student, research activity increased, authorization sent)*. A similar commitment exists between travel agent and student where the former relies on the latter for information related to the trip, and the latter depends on the former for booking the trip. The commitment *C(travel agency, student, info. received, trip booked)* not only shows that both actors agree to be involved in this contractual relation, but also explicitly states the relation between the two seemingly independent relationships, which cannot be captured in *i**/Tropos.

III. *DEST*: A Modeling Language for Designing Sociotechnical Systems

DEST (*DE*signing *S*ociotechnical sys*T*ems) is a modeling language for designing sociotechnical systems. *DEST* focuses on the social components and their interactions; we reify technical systems through the social entity (their developer, owner, or user) that is liable for the effects of their interactions with other entities. We model each component as an actor, an autonomous entity that aims to satisfy its own requirements either by its own means or through social interactions with other actors.

DEST is based on and extends (i) the *i** modeling framework, from which we take the concepts of *actor*, *goal*, and *refinement*; (ii) social *commitments* to represent the social interactions among the *actors*; and (iii) advanced requirements, relationships such as *priority* and *precedence* [10], [11]. Together, these elements support expressive modeling and reasoning for requirements and interactions during STS design and evolution.

Fig. 2 presents the meta-model of *DEST*. We explain the three fundamental concepts—goals, actors, and commitments—in Sections III-A–III-C. Fig. 3 illustrates the graphical syntax of *DEST*for the travel reimbursement STS.

A. Requirements

We represent stakeholder requirements as goals. In addition we introduce priority and precedence relationships between goals, so as to enable a fine-grained specification of the relative importance and urgency of the goals. Specifically, *DEST* supports the following elements to represent requirements:

- **Goal** models a desired state-of-affairs that an actor wants the STS-to-be to achieve. In an STS, the behavior of actors is determined by their respective goals.
- **Precedence**: a type of relationship between goals that specifies that one goal is to be satisfied/carried out before another. Precedence relationship represent actors' temporal constraints on the goals. For example, a project fund manager may require that a conference paper should be accepted to be published before its authors submit a travel authorization request.
- **Priority**: a type of a relationship between goals that indicates that one goal has higher priority than another. A student may prioritize spending its travel budget on plane tickets rather than on accommodation.
- **Conflict**: a goal may conflict with another, i.e., they cannot both be satisfied in the same STS configuration.
- **Refinement**: A goal may be AND/OR-refined to subgoals that are easier to fulfill than their parent. For example, a goal could be AND-refined into two subgoals for which there are actors capable of fulfilling them.
- **Capability**: the actors ability to achieve a goal, a category of goals or a state of affairs.
- **Goal Category**: each goal may belong in a category that indicates the type of capability needed to fulfill it. For example, in the travel reimbursement STS, a goal category could be "authorization", to aggregate all goals that can be fulfilled through authorization the requests. An actor that has the capability of satisfying a certain category, does not need to declare his capability for each goal that belongs to that category.

Fig. 3 describes a model built by using the concepts described above. For example, *'trip booked'* is a goal that is OR-refined into two other goals: *'booked by student' 'booked via travel agent'*, which implies that the child goals are easier than to satisfy the parent goal. In the model, a precedence relation is defined between the goals *'trip planned'* and *'paid by student'*, which states that the former should be satisfied before the latter. The goal *'expensive accommodation'* has priority over the goal *'expensive tickets'* so when taking actions in a plan

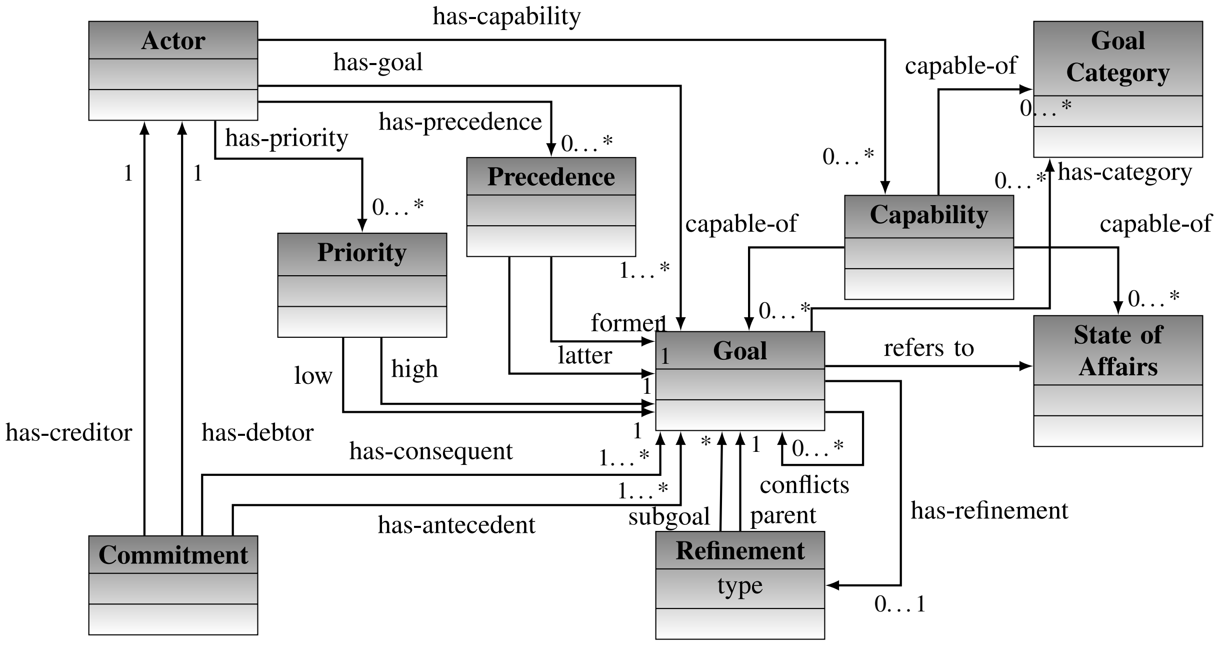

Fig. 2: Meta-model of the *DEST* modeling language

towards satisfying the goals, the former goal has a higher priority. There is a conflict relationship defined between the *'cash payment'* and the *'expensive tickets'* goals, therefore only one of them could be satisfied by the STS.

B. Actors

An actor is an autonomous component of an STS. An actor could be a human such as a student or a project fund manager, a social entity such as an airline or a hotel, or a technical system such as the information system used within a department. The following relationships relate actors to their requirements:

- **has-goal**: this denotes that an actor has a requirement to be satisfied by the STS. In *DEST*, requirements consist of goals and relationships such as precedence, priority, and conflict. A solution to the design problem should respect these relationships and satisfy the goals of actors.
- **is-capable-of**: an actor is capable of (fulfilling) a goal if it can do so on its own, without any help from other actors. I an actor has some goals that she is not capable of satisfying, the actor must interact with other actors to get the goal satisfied by some other actor who is capable of satisfying the goal. The actor who has the capability to satisfy the goal may ask for the satisfaction of some of her own goals, as often in case of real life, which initiates a reciprocal social interaction between actors with various capabilities. There may be alternative solutions for an STS where an actor still interacts with others to get a goal satisfy of which she is capable. For example, a travel agent may choose to use some intermediary travel company for bookings due to its lower cost although she has the capability of booking tickets and hotel rooms.
- **is-capable-of-category**: an actor is capable of satisfying any goal that belongs in a particular category.

There are three actors in the model presented in Fig.3 : student, project fund manager, and travel agent. Each actor has its requirements represented as goals, priorities, and precedences within his actor boundary. Capabilities such as 'booking' and 'payment done' are shown in cloud shaped nodes within the boundaries of the actor who is capable of them.

C. Commitments

A commitment represents a contractual relation between two actors as they socially interact. Two actors get engaged in a commitment whenever their capabilities are insufficient for satisfying their requirements, or when interacting with others is more convenient than exercising an internal capability. A commitment is defined in terms of four relationships:

- **has-debtor**: the actor who is responsible for satisfying the consequent of the commitment. A debtor participates in the commitment because there is at least one goal listed in the antecedent that he wants to achieve. The debtor should be capable of satisfying all goals in the consequent.
- **has-creditor**: the actor who is the beneficiary of the commitment. The creditor of the commitment is interested in the satisfaction of at least one goal listed in the consequent. The creditor should ensure that antecedent goals are satisfied, but need not be the one to do so.
- **has-antecedent**: a set of goals whose satisfaction is a precondition for the commitment to be fulfilled.
- **has-consequent**: a set of goals that the debtor is obliged to fulfill.

Fig. 3 includes three commitments represented as split rectangles. For example, the commitment between the travel agent

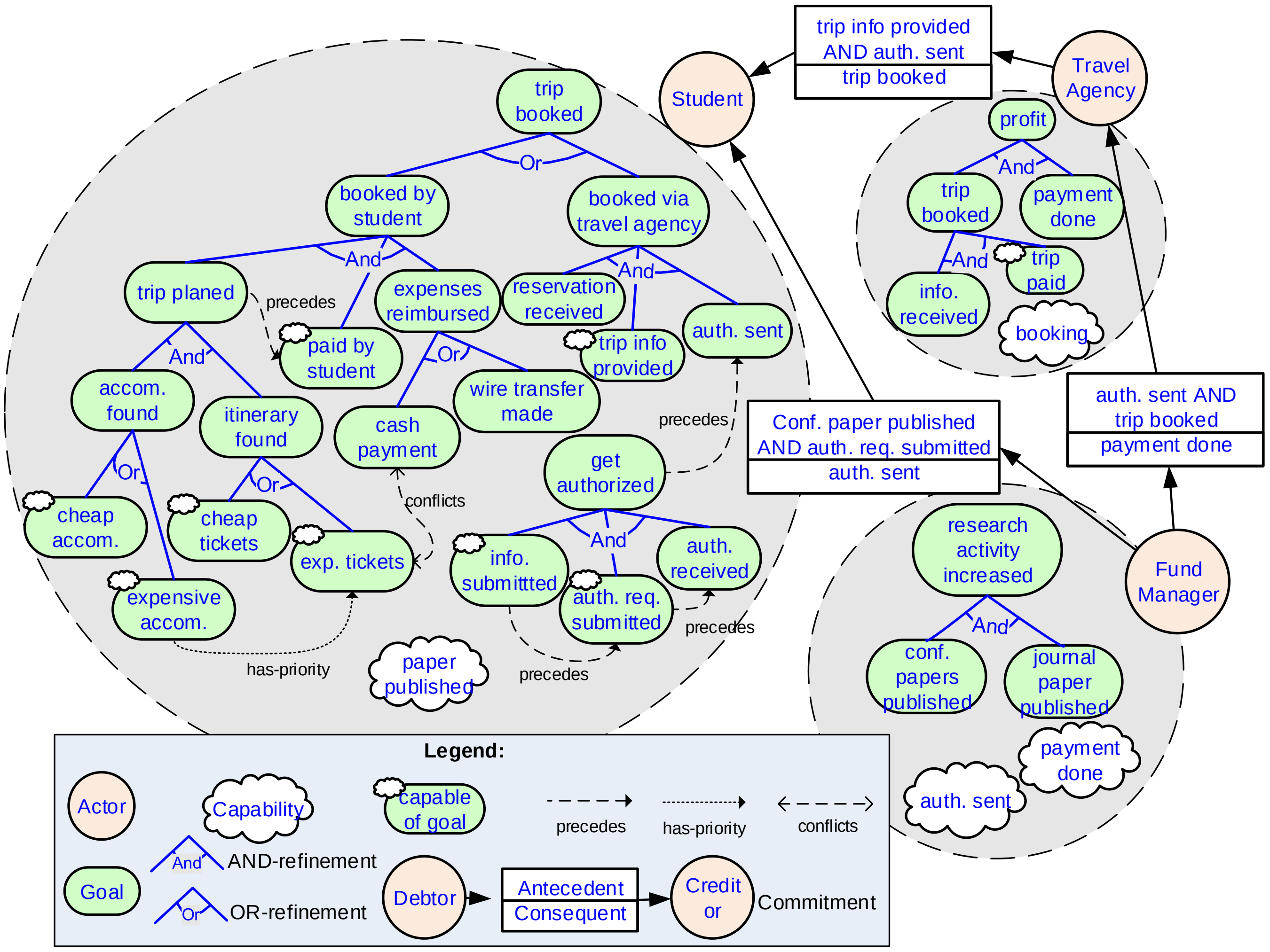

Fig. 3: *DEST* model of the travel reimbursement sociotechnical system

and student, where travel agent is the debtor that commits to book the trip if the student provides needed information about the trip and sends authorization for it.

IV. Designing a STS

The modeling language introduced in Section III helps abstracting the requirements of actors in STS, relationships among these requirements, actor capabilities and social interactions between actors that help fulfilling the requirements. Analysis of this model consists of exploring the space of alternatives for fulfilling requirements: actor root goals, while respecting precedence, priority, and conflict relationships. Alternative solutions may follow different paths to satisfy the root goals. Differences in a path includes following different OR-refinement branches, using an actor's own capabilities to satisfy leaf goals, and activating one of the commitments described in the model. An STS deploys one of the alternative solutions at run time as its *configuration*.

Definition 1 (Configuration)**.** *A configuration* Cfg *of a DEST model* M *is a 3-tuple* $\langle \mathsf{G}, \mathsf{Cap}, \mathsf{Com}\rangle$*, where* G *is the set of (sub-)goals that the actors in the STS are adopting,* Cap *is the set of capabilities that the actors are exploiting, and* Com *is the set of commitments that the actors have established.* G*,* Cap*, and* Com *are subsets of the set of goals, capabilities, and commitments in* M*, respectively.*

A configuration is *empty* at the beginning. Actors state their goals, their capabilities and relationships among their goals as well possible refinements and commitments. A *valid* configuration includes the selected set of commitments and refinements that satisfies the top-level goals of the actors while respecting the specified relationships among goals.

We now present the design process whereby a STS is designed using *DEST*:

Definition 2 (Design process)**.** *Let* M *be a DEST model that represents the space of alternatives of an STS, and let* Cfg *be a valid configuration of* M*. A design process* $\mathsf{D}_{\mathsf{Cfg},\mathsf{M}}$ *is a list of actions* $(\mathsf{Act}_1, \ldots, \mathsf{Act}_n)$ *that, if correctly executed starting from an empty configuration, leads to the configuration* Cfg*.*

The actions that construct a design process correspond to the execution of one of the following *action types* on an element of a *DEST* model:

- **Adopt goal**: an actor can adopt a new goal when he doesn't have the goal, and the goal is not already satisfied. As a result of this action, the actor intends to achieve (*has*) the goal.
- **Satisfy goal via capability**: an actor can satisfy a goal if *i*) the actor has the goal, *ii*) the goal has not been satisfied, *iii*) no conflicting goals are satisfied, *iv*) all goals that shall precede it are satisfied, and *v*) the actor is either capable of satisfying the goal itself or the category of the goal. As a result, the goal becomes satisfied, and the actor does no longer have the goal.
- **Refine goal**: an actor may AND/OR-refine a goal if *i*) the actor has the goal, *ii*) the goal has not been satisfied, and *iii*) there is a refinement for that goal in the actor's goal model. As a result of the action, the actor does no longer have the parent goal, but he has the subgoals.
- **Create commitment**: An actor may create a commitment playing the role of debtor if there is at least one goal in the antecedent that the actor wants to satisfy.
- **Accept commitment**: An actor accepts a commitment as creditor if the actor wants to satisfy at least one of the consequent goals. As a result, the actor drops its goal(s) listed in the consequent, and adds the goals in the antecedent.
- **Detach commitment**: The debtor of the commitment detaches it when all goals in the antecedent are satisfied. Such condition binds the debtor to satisfy the consequent, so he adopts the goals listed in the consequent.
- **Discharge commitment**: The creditor of the commitment discharges the commitment when all the goals in the consequent are satisfied.

All these *action types* defined above are used at design time and executed on goals and commitments. Actors may *adopt* new goals during the design process. After adopting a goal, an actor may either satisfy the goal, or further *refine* it into other goals. If capable, the actor may choose to *satisfy the goal via its own capabilities*. If the actor is not capable of satisfying the goal or does not prefer to do so due to some constraints such as budget, the actor *creates a commitment* with another actor to get its goal satisfied. Creating a commitment is to activate the respective commitment described in the model and a declaration to the creditor of the commitment to satisfy the goals listed in the consequent of the commitment one of which may be adopted by the creditor. In return, the debtor actor expects the goals listed in the antecedent to be satisfied. If the creditor is interested in the commitment, that is, if there are some of its goals listed in the consequent of the commitment, the creditor *accepts* the commitment. Once the goals in the antecedent are satisfied, the commitment becomes *detached*. When the debtor actor satisfies the goals in the consequent the commitment is *discharged*. The list of actions that operate on commitments do not include two commitment operations that are defined in [9]: canceling a commitment and violating a commitment. We deliberately omit these operations as they are deviations from the successful execution of commitments, which may occur at run run-time as exceptions but are not used at design time.

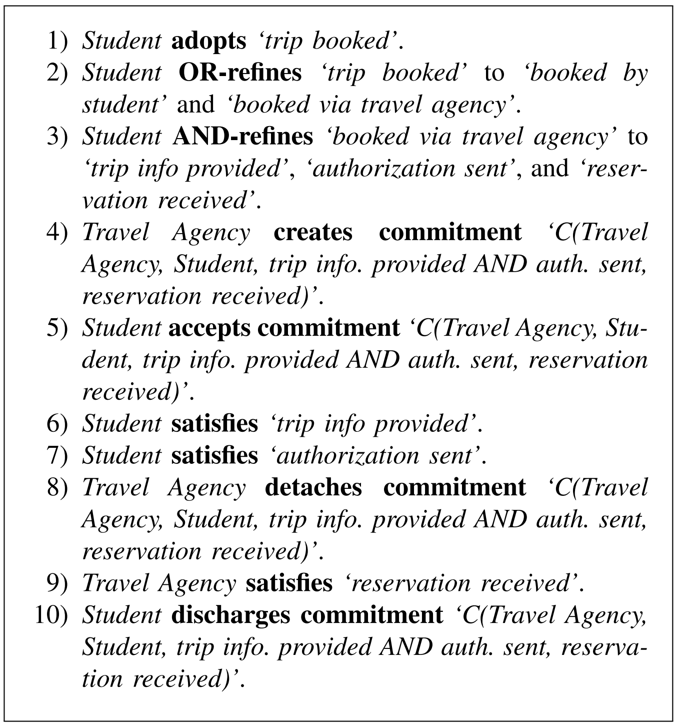
1) *Student* **adopts** *'trip booked'*.
2) *Student* **OR-refines** *'trip booked'* to *'booked by student'* and *'booked via travel agency'*.
3) *Student* **AND-refines** *'booked via travel agency'* to *'trip info provided'*, *'authorization sent'*, and *'reservation received'*.
4) *Travel Agency* **creates commitment** *'C(Travel Agency, Student, trip info. provided AND auth. sent, reservation received)'*.
5) *Student* **accepts commitment** *'C(Travel Agency, Student, trip info. provided AND auth. sent, reservation received)'*.
6) *Student* **satisfies** *'trip info provided'*.
7) *Student* **satisfies** *'authorization sent'*.
8) *Travel Agency* **detaches commitment** *'C(Travel Agency, Student, trip info. provided AND auth. sent, reservation received)'*.
9) *Travel Agency* **satisfies** *'reservation received'*.
10) *Student* **discharges commitment** *'C(Travel Agency, Student, trip info. provided AND auth. sent, reservation received)'*.

Fig. 4: A partial design plan for the model described in Fig. 3

The output of the design process is a set of actions of whose execution leads to a valid configuration for the STS. The precedes relation imposes constraints on the satisfaction order of goals; thus, the order of the actions in a design process is relevant and should be used as a guideline for the implementation of the STS.

Fig. **??** we provide a partial enactment of the design process for our example STS in Fig. 3. The plan sets up the interaction between the *student* and the *travel agency*. In this scenario, the *travel agency* wants to make profit by selling trips, and needs to receive necessary information for the trip as well as reimbursement for the trip (hotel and tickets). The *travel agency* is capable of making payments and booking trips. In the following steps, first, the *student* adopts and refines a goal (Steps 1-3). Then the *travel agency* creates a commitment which potentially matches its goals and the capabilities to those of the *student*'s (Step 4). The *student* accepts the commitment (Step 5), and the two successfully A possible sequence of run-time interaction that instantiates the design plan is also shown (Steps 6-10). Among these steps, detaching (Step 8) and discharging a commitment (Step 10) are necessary for one actor to acknowledge that the other actor has, indeed, satisfied the antecedent (Step 8) and the consequent (Step 10).

It is generally the case that there are alternative ways to satisfy requirements for a STS. This happens when a goal is OR-refined, when several actors have capabilities for the same goal, and when multiple commitments are possible for fulfilling a goal.

The problem of identifying configurations that fulfill a given set of requirements can be reduced to an automated planning problem [12], where a tool is employed to search the encoding of a *DEST* model for feasible design plans that respect the constraints. The planner returns a list of actions (a plan) to be followed at run time that results in the satisfaction of the requirements of the actors.

In order to identify the best plan, among many possible ones, we add to each action a *cost*, which corresponds to the effort for an actor to execute that action. There are multiple ways of expressing a cost: either a cost can be assigned to each action (e.g., to adopt(g_1)), or a standard value is assigned to action types, or simply, all actions are assumed to have unit cost. The best plan is the one that has minimal cost, among all possible plans. An alternative way to quantify the results of the actions is to add a utility to each action, in this case the best plan maximizes the total utility. How to assign these are beyond the scope of this paper, one of the existing methods from the literature could be applied for this task. Karlsson and Ryan [**?**] adopts a cost based approach for prioritizing requirements. Regan et al. [**?**] provide a method on eliciting reward information and use regret-reduction to choose suitable solutions. Their approach could be broadened to utility elicitation. In Section VI, we show how we map the identification of the best plan into an input for an off-the-shelf planner.

V. Supporting Evolution for STSs

The framework for STS design that we provide in Section IV addresses the problem of devising a plan starting from scratch. However, in most cases, STS designers are confronted with the evolution of an existing system.

While an STS can evolve in an unmanaged manner (when the actors change their relationships autonomously), we are interested here in managed evolution, i.e., when the designer is in charge of devising an alternative configuration and a plan to reach it. We call this activity an *evolution episode*.

An episode is triggered by the occurrence of one or more events. There are many types of triggers; here, we consider only those that affect the elements of *DEST*:

- **Changed capabilities**: an actor may lose a capability that he needs to satisfy either his own goals, or the goals he adopted through a commitment. In the former case, the actor fails to satisfy his own requirements and needs to establish a commitment with another actor to get his requirement satisfied. In the latter case, the actor either violates the commitment of which he is the debtor, and fails to satisfy the consequent, thereby harming another actor, or he cancels a commitment of which he is the creditor and disappoints the debtor actor who relies on him to satisfy the antecedent.
- **Changed commitments**: new commitments can be created, introducing new opportunities for fulfilling the requirements, or existing commitments may be withdrawn, threatening the actors that rely on those commitments.
- **Unreliable actor**: an actor may not respect its commitments, either intentionally (i.e., maliciously), or due to other exogenous reasons, such as external regulations, lack of resources, etc. Such an actor does not satisfy the consequent of a commitment even though the antecedent is satisfied by the creditor or he does satisfy the antecedent although he agrees to take part in the commitment, therefore stalls the debtor.
- **Dropping a requirement**: an actor may not respect his commitments if he drops a requirements, thereby losing his interest in the interaction.
- **New requirements**: actors may have new requirements (goal, priority, precedence) which cannot be satisfied through the existing plan.
- **Joining/leaving actors**: the effect may be either beneficial or harmful. A new actor, along with its capabilities and potential commitments, provides new alternatives, but also introduces new requirements to satisfy. A leaving actor does harm other actors, which may be relying on his commitments. However, quitting also means that there are fewer requirements to satisfy.

Whenever one of these events makes the current configuration invalid, an evolution plan has to be identified, that brings the STS towards a valid configuration.

Definition 3 (Evolution plan)**.** *Given a DEST model* M*, and an invalid configuration* C *for* M*, an evolution plan is a list of additions and removals of actions (those in Section IV), such that the correct rollback of removals and the correct execution of additions conducts to a valid configuration* C' *for* M.

In order to preserve the stability of the STS, we would like the plan to lead to the valid configuration that is the closest to the current one. We measure closeness in terms of the cost of the evolution plan: the lower the cost, the closer the configuration. To do so, we associate with every addition/removal action a cost:

- **Remove** an action that exists in the previous plan but fails the bring out the desired effect due to one of the evolution triggers. Removing an action changes the currently implemented STS, so it has its own cost. As a result of removing an action, all of its effects are rolled back.
- **Add** one of the actions listed in Section IV either to compensate for the rolled back effects of a removed action, or to find a solution for a new requirement. For simplicity, we assume that the cost for adding an action in the evolution phase corresponds to the cost of adding it in the design phase.

Consider the following scenario as an example of the evolution in our example STS. The *travel agency* leaves the STS, therefore the current configuration becomes invalid for the rest of the actors. Any new configuration will have to remove the elements that relate with the *travel agency*, such as its commitment to the *student*. In particular, the additions in Steps 4–10 described in Section IV have to be removed. The only alternative for the *student* is to follow the other branch in the goal model for *'booked by student'*. Since he does not have the capability for the reimbursement, he needs a commitment from the *fund manager*. The actions that are related to existing commitments are removed (starting from the creation and onward) and a new commitment is created that has both the authorization and the reimbursement in its consequent. Then, the *student* accepts the new commitment and the interaction continues.

An interesting case is where the *fund manager* gains the capability of directly paying for accommodations and tickets, and the *student* changes the refinement of his root goal accordingly. Furthermore, he states that he prefers this option to others, that is, *'paid by student'* and *'booked via travel agency'*, thus, the current configuration does not respect the preference of the *student*. However, without the commitment with the *student*, there is no way for the *travel agency* to satisfy his requirements, so the plan is unchanged and discards the preference of the *student*. However, noting that there is no solution that accommodates both the requirement of the *travel agency* and the preference of the *student*, a designer may choose to intervene and remove the *travel agency* from the STS. Then, a new plan is constructed as in the previous example. Another possibility is that even though the plan has not changed, the *student* does not interact with the *travel agency* at run time, hence forces him out of the STS.

VI. Implementation

The field of AI planning has found a number of applications in various areas such as multiagent systems and robotics. A planning problem includes the initial state of the world, the desired of the world, and the possible actions that can be taken throughout the plan [12]. Using the *DEST* language described in Section III and the actions in Section IV, we encode the problem of building a network of interactions for the actors in an STS to a planning problem that satisfies actor requirements while minimizing cost. We encode the problem using the Planning Domain Definition Language (PDDL) [**?**], the de-facto standard input format for planners. We use PDDL v3.0 [13], which supports preferences and soft constraints that we need to implement the *has-priority* relation.

An of-the-shelf PDDL planner has two inputs: the domain description and the problem definition. We map the *DEST* concepts defined in Fig. 2 into PDDL constructs in the domain description file. Mappings of the actions in Section IV are also added to the domain description file. This file can be re-used for various problem definitions, each consisting of the model (instances of the concepts), the initial state of the STS, metrics for the plan, and the instances of the requirements.

In the domain description file, we state the requirements for the planner, which are features of PDDL that the planner provides. For our purposes, we use `typing` to abbreviate the type declarations for the multiple objects of the same type, `adl` for using disjunctions, quantifiers in preconditions, and conditional effects, `:constraints` for the trajectory constraints and `numeric-fluents` to model costs. Among the classes in Section III we map *Actor*, *Goal*, *Goal Category*, and *Commitment* classes into object types in PDDL, so `goal(trip-booked)` is translated that *trip-booked* represents an instance of the *goal* class in *DEST*. Since there is a one-to-one correspondence between *DEST* and PDDL for these four classes, we directly map their relationships into namesake PDDL predicates. For example, a *has-goal* relation between *actor* `a` and *goal* `g` is mapped to `has-goal(?a - actor, ?g - goal)`. Note that we provide a template in the domain description file, and the question marks mean that these objects are variables for that predicate.

The other classes (*refinement*, *conflicts*, *precedence*, and *has-priority*) are mapped through their relations with the four classes mentioned above into PDDL, rather than introducing namesake object types. We implement `and-ref`, `or-ref`, and `conflicts` to capture *refined-to* relation (specifying the refinement type) and *has-conflict* relation, respectively. For the *priority* class, we combine *high-* and *low-priority* relations into `has-priority`. Similarly, we define the `precedes` predicate for the *precedence* classes. Other than these predicates, we define auxiliary predicates to check the state of a goal or a commitment, such as `is-satisfied` for a goal and `is-created` for a commitment. The full list of the predicates,together with the object types and orders are given in the first two columns of Table I.

In our implementation of the actions and the *DEST* language, in order to find an optimal plan, we use cost as a metric. The individual efforts performed by each actor are aggregated via in the `actor-cost` fluent. When an actor performs an action, the actor's cost increases by that action's cost. The cost of satisfying a goal by a particular action is kept in `has-goal-cost`. Finally, the overall cost of the plan is kept in `total-cost`. Those values are initiated in the problem definition files before the planning starts. All the fluents that we use are listed in the third column of Table I. Various cost aggregation functions or other metrics could be defined and implemented in the PDDL language and we leave exploring the details of plan optimization as a future work. The implementation of the priority relation is also part of the future work where the priority violence is possible but not desired.

We define a PDDL action for each action in Section IV. Also, to determine the satisfaction of a goal through AND/OR-refinement, we implement the corresponding actions that satisfies the goals instead of relying on the PDDL's derived axioms due to performance issues. A PDDL action has three components: parameters, precondition, and effect. Below we provide the PDDL code for the `satisfy-goal` action: both preconditions (the actor has the goal, etc.) and effects (the goal becomes satisfied, etc.) correspond to those described in Section IV. Finally, the costs are increased by *i*) a variable cost for satisfying the goal `g` by the actor `a`, and *ii*) a fixed cost for satisfying a goal. The second part of the implementation is the problem definition, where the instances of the objects and relations are defined and the initial values of the fluents are assigned, such as `(:init (=(total-cost) 0 ))`.

TABLE I: List of PDDL Predicates and Fluents

Predicates	Predicates	Fluents
(is-satisfied ?g - goal)	(and-ref ?g ?g1 - goal)	(actor-cost ?a - actor)
(or-ref ?g ?g1 - goal)	(has-goal ?a - actor ?g - goal)	(has-goal-cost ?a - actor ?g - goal)
(precedes ?g ?g1 - goal)	(has-priority ?g ?g1 - goal)	
(has-category ?g ?gc - gcat)	(conflicts ?g ?g1 - goal)	(total-cost)
(is-capable ?a - actor ?g - goal)	(is-capable-cat ?a - actor ?g - goal)	
(has-debtor ?c - comm ?a - actor)	(has-creditor ?c - comm ?a - actor)	
(has-ant ?c - comm ?g - goal)	(has-cons ?c - comm ?g - goal)	
(is-created ?c - comm)	(is-discharged ?c - comm)	
(is-detached ?c - comm)		

```
;Actor satisfies a goal
(:action satisfy-goal
 :parameters(?a - actor ?g - goal)
 :precondition(and (has-goal ?a ?g)
   (not (is-satisfied ?g))
   (or (is-capable ?a ?g)
    (exists(?cat - gcat)
     (and (has-category ?g ?gcat)
     (is-capable-cat ?a ?gcat))))
   (not(exist(?cg - goal)
       (and (or (conflicts?g ?cg)
            (conflicts ?cg ?g))
            (is-satisfied ?cg))))
   (forall(?pc - goal)
       (imply(precedes ?a ?pc ?g)
       (is-satisfied ?pc))))
 :effect (and (is-satisfied ?g)
       (not (has-goal ?a ?g))
       ((total-cost) += (has-goal-cost ?a
           ?g) + <num>)
       ((actor-cost ?a) += (has-goal-cost
           ?a ?g) + <num>)))
```

Fig. 5: PDDL action for an actor satisfying the goal of which it is capable

Also, the goals that are adopted by the *actors* are stated in the `(:goal ... )` section to tell the planner to satisfy these particular *goals*. In this implementation we opt for a simple metric, the minimal total cost, which is encoded as `(:metric minimize (total-cost))`.

In order to handle evolution (Section V), we implement remove-actions: for each action `a` implemented, remove-action-`a` takes back the effects of the action `a` (except the cost). Moreover, in the case of evolution, the initial state is a definition of the current configuration (as opposed to the empty state for the initial design case).

VII. Preliminary Evaluation

Visual scalability. Fig. 3 illustrates our running travel reimbursement STS example in *DEST*. We deliberately choose a syntax similar to that of *i**/Tropos to foster adoption from experts in the area of goal-oriented requirements engineering. The visual notation of *DEST* follows the design principles provided in [14]. **Semiotic clarity.** There is a one-to-one correspondence between the symbols used in the visual notation and their referent concepts: each symbol is only used to represent a single concept from the language and each language concept is represent by only one symbol.

Concepts, such as actor, goal, capability and commitments are highly distinguishable as their respective symbols have clearly different shapes from different shape families. **Perceptual discriminability**. Goals belong in the oval family, whereas actors are from the circles. We introduced the new node shape for commitment, which is represented by split rectangles so the shape does not only belong to a different shape family from other symbols used in *DEST* visual syntax but it also distinctly different from the rectangle node used in *i** which represent resources since it is split in two. The other introduced node is the cloud-shaped node that corresponds to 'can-satisfy' relationship. By using shapes from distinct shape families we ensure the visual distance between the node symbols are at least one.

To further increase the visual distance, positional cues are used. Goal and capability nodes are placed within the boundaries of an actor, actor name nodes are placed on the actor boundary and the commitments are placed between the actors. Since the goal and capability symbols are spatially close to each other, we use color as the secondary dimension for the visual distance. As light green is traditionally used for the goal nodes, we use white for the capability nodes. We depart from the *i** syntax for the actor names and color them in blush to increase the visual distance from goal nodes both of which are light green in the *i** syntax.

Complexity management. We omit representation of goal categories not to overload the visual syntax. Furthermore, in order to distinguish the goals that an actor is capable of satisfying himself, we place small cloud shape nodes on them. So, with the shape, size, color, horizontal and vertical positions being different, the nodes have a visual distance of five from each other.

The visual distance of the edges used in the notation is also greater than one. Edges to and from commitments are solid lines with an arrowhead filled with black. Since the direction of the edges naturally conveys the meaning for the debtor and the creditor we do not use textual labels on these edges to keep the notation light for the human eye. We follow the traditional representation for the refinements, which is solid blue lines, and the type label is positioned close to the parent node. Among the edges that represent the precedes, has-priority and conflicts, the edge that represents conflicts relations differentiates from the other since it is *i* bi-directional dashed *ii* has simple arrowhead, has-priority edge has a dotted line whereas precedes edge has the dashed style. So at least a visual distance of two is ensured among the edge types with the variables texture (line style), shape (arrowhead), and color. Textual labels also ensure that the three edge types are distinguishable from each other.

Overall, symbols used in the *DEST* visual notation have at least visual distance one from the other symbols. In terms of shape, color, and position the notation stays well within the boundaries of the cognitive capacities, that is number of perceptible chunks as there are 3 relative positions with respect to the actor boundaries (in, out, on) and 2 with respect to the refinements (label is closer to the parent and at the intersection of the edges) so a total of 5 where the maximum value is 10 to 15. Also there are 4 colors used in the notation where the maximum capacity for the color variable is seven to 10.

Scalability with respect to the size of the problem. To test the scalability of our approach with respect to the size of the problem, we manually create a *DEST* model that consists of three actors, four commitments, two OR-refinements, five AND-refinements, one precedence, one conflict and 20 goals. We then automatically generate 30 test models by replicating the original model. So the final model consists of 90 actors, 600 goals, 120 commitments, 60 OR-refinements, 150 AND-refinements, 30 precedence and 30 conflicts. We use sgplan version 5.22[1] as our choice of of-the-shelf planner and run our test files together with our domain file which includes our domain declaration in PDDL. The experiment is run on an Ubuntu 12.04 virtual machine with 2 GB memory hosted on a Mac OSX with 4 GB memory and a 2.5 GHz Intel Core i5 processor. The results of the experiments are summarized in Fig.4 where y-axis shows the time in seconds to find a solution (plan) and x-axis shows the number of replicas in the test model. The first phase to find a solution is to read the domain description and problem files and to construct the model. The time spent for the first phase is indicated as ‘parsing time’ and shown by bright green line with cross in Fig.4. Parsing time increases rapidly beyond 1000 model elements. The second phase is the planning phase in which the planner search for solutions in the search space, that is, the constructed model in the first phase. Planning time is shown by the blue line with squares in Fig.4. The planning time constitutes a less significant of the total time spent. Parsing time has a more rapid increase than the planning time. The reported values highly depends on the performance of the chosen planner.

The results are promising, for we have artificially synthesized extra-large models that are way bigger than those that are typically created by modelers. However, more work is required to test scalability when increasing other dimensions of the model, such as complexity and connectivity.

[1]http://wah.cse.cuhk.edu.hk/wah/programs/SGPlan/

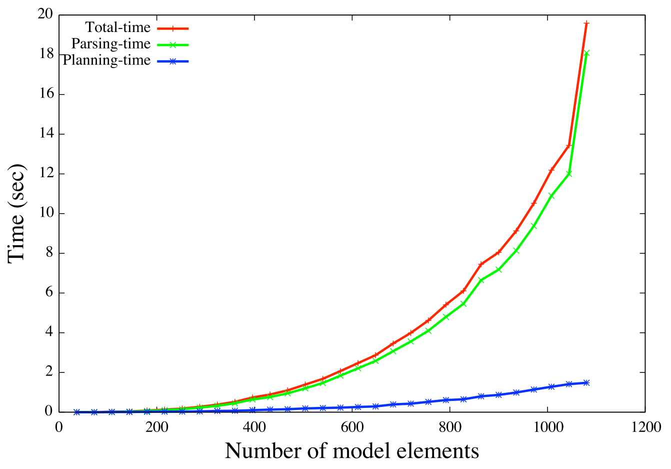

Fig. 6: The results of the scalability experiments w.r.t model size

VIII. Related Work

Goal-based requirements engineering has been widely applied for sociotechnical system research [**?**]. KAOS [4] is a goal-oriented approach that also provides reasoning but does not represent actors explicitly. Tropos [7] software development framework that is based on the *i** modeling framework [3] associates actors with their respective goals and represent social interactions as dependencies. Precedence and priority relationships do not exist in these frameworks. Several analysis techniques are devised including interactive analysis and forward and backwards analysis on the models [**?**].

The benefits of integrating commitments into goal models are discussed in [5], [8]. [8] proposes a shift in goal-oriented requirements engineering which we follow and use as a basis for our research. [5] focuses on a different problem than ours—whether a specific agent’s goals are supported. Thus, the interaction network is given as input specified in a commitment-based protocol. We focus on building an interaction network given the requirements and the capabilities of the actors to satisfy their requirements.

Bryl et al. [15] support the design of STSs by analyzing actor capabilities and exploring a search space of actor dependency networks. In that work, a social interaction between two actors is represented via dependencies, which we replace in our work with the more expressive concept of a commitment. The authors focus on evaluation of alternative plans and devise several metrics for the evaluation. We focus on a trigger-based evolution of an STS. We use a minimal cost metric to choose the optimal plan among all alternative solutions, and support a richer set of requirements types.

Günay et al. [16] provide a run-time approach for an agent within a multiagent system to generate a commitment-based protocol that satisfies the goals of the agent, and rank the protocols using a trust/risk metric. We consider more expressive types of requirements, and our outcome is not simply a protocol, but also suggests actor internal actions (adopt a goal, use capability, etc.).

Jureta et al. [17] discuss the need for specifying priorities in requirements models. We adopt their idea and include priority specification in our meta-model. Techne [18] is a requirements modeling language that includes priorities. Ernst et al. [19] use Techne to identify alternative solutions to the requirements problem. A similar approach is taken by Liaskos et al. [20] (using traditional goal models), that implement optional goals and priorities in PDDL. However, these approaches are inadequate for modeling the requirements of an STS, as they lack the concepts of actor and interactions. As for the implementation of the priorities, we leave it as a future work to adopt the elaborate implementation of preferences and priorities presented in [20].

Business process management offers well-developed techniques for supporting evolution [21]. In this field, the problem is that of identifying and implementing variants of a business process, starting from an initial configuration. While we use a radically different modeling language, the usage of case-based reasoning to drive evolution based on previous history [22] is an interesting research direction for our framework.

IX. Conclusions

We have presented a framework that supports exploring alternative plans for building and evolving a sociotechnical system. Our framework is model-driven, and uses our proposed *DEST* language for representing actor requirements in an STS and the space of alternative designs.

In addition to introducing *DEST*, we have proposed techniques for building a network of interaction that fulfills participant requirements from scratch, and also to re-design such a network in case of evolution. For implementation, we have encoded plan generation to a problem a native language of an automated planner.

DEST visual notation has similar limitations in terms of readability as the other goal-oriented approaches do. As the *DEST* models become bigger, it is hard for humans to read the model as a whole. A dedicated editor with different types of views as in [**?**] could be a solution to this problem. Eliciting cost values for single actions is a challenge for practical purposes, yet even more elaborate cost schemes, and the effect of actions on other actions are other future challenges that are need to be addressed. Finally, to obtain the quantitative effect of additions and removals during the evolution is another challenge to overcome. Trying to satisfice the requirements as suggested in [**?**] may help decreasing the complexity.

This paper opens the door to several future research directions. Firstly, for the empirical validation of our framework (we are currently conducting a project in collaboration with an Italian airport). Secondly, we are conducting a study of the scalability of our approach. Thirdly, we are looking at the usage of historical information to guide the evolution of an STS using case-based reasoning techniques. Finally, we are working on and the development of a design environment for STSs founded on the *DEST* proposal.

Acknowledgments

This research was partially supported by the ERC advanced grant 267856 'Lucretius: Foundations for Software Evolution'.

References

[1] V. Grover, J. T. C. Teng, and K. D. Fiedler, "Information technology enabled business process redesign: an integrated planning framework," *Omega*, vol. 21, no. 4, pp. 433–447, 1993.

[2] M. M. Lehman and J. F. Ramil, "Rules and tools for software evolution planning and management," *Ann. Softw. Eng.*, vol. 11, no. 1, pp. 15–44, 2001.

[3] E. S.-K. Yu, "Modelling strategic relationships for process reengineering," Ph.D. dissertation, 1996.

[4] A. Dardenne, A. Van Lamsweerde, and S. Fickas, "Goal-directed requirements acquisition," *Sci. Comput. Program.*, vol. 20, no. 1, pp. 3–50, 1993.

[5] A. K. Chopra, F. Dalpiaz, P. Giorgini, and J. Mylopoulos, "Modeling and Reasoning about Service-Oriented Applications via Goals and Commitments," in *Proc. CAiSE*, ser. LNCS, vol. 6051. Springer, 2010, pp. 113–128.

[6] A. Van Lamsweerde, "Goal-oriented requirements engineering: A guided tour," in *Proc. RE*. IEEE, 2001, pp. 249–262.

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

[8] A. K. Chopra, J. Mylopoulos, F. Dalpiaz, P. Giorgini, and M. P. Singh, "Requirements as Goals and Commitments too," in *Intentional Perspectives Inf. Syst. Eng.*, 2010, pp. 137–153.

[9] M. P. Singh, "An ontology for commitments in multiagent systems," *Artif. Intell. Law*, vol. 7, no. 1, pp. 97–113, 1999.

[10] I. J. Jureta, A. Borgida, N. A. Ernst, and J. Mylopoulos, "Techne: Towards a New Generation of Requirements Modeling Languages with Goals, Preferences, and Inconsistency Handling," *Proc. 18th IEEE Int. Requir. Eng. Conf. (RE 2010)*, pp. 115–124, Sep. 2010.

[11] S. Liaskos, S. a. McIlraith, S. Sohrabi, and J. Mylopoulos, "Integrating Preferences into Goal Models for Requirements Engineering," *Proc. 18th IEEE Int. Requir. Eng. Conf. (RE 2010)*, pp. 135–144, Sep. 2010.

[12] D. S. Weld, "Recent advances in {AI} planning," *AI Mag.*, vol. 20, no. 2, p. 93, 1999.

[13] A. Gerevini and D. Long, "Plan constraints and preferences in {PDDL3}: The Language of the Fifth International Planning Competition," *Tech. Report, Dep. Electron. Autom. Univ. Brescia, Italy*, 2005.

[14] D. L. Moody, "The "Physics" of Notations: Towards a Scientific Basis for Constructing Visual Notations in Software Engineering," *Softw. Eng. IEEE Trans.*, vol. 35, no. 5, pp. 756–778, Nov. 2009.

[15] V. Bryl, P. Giorgini, and J. Mylopoulos, "Designing socio-technical systems: from stakeholder goals to social networks," *Requir. Eng.*, vol. 14, no. 1, pp. 47–70, 2009.

[16] A. Günay, M. Winikoff, and P. Yolum, "Generating and ranking commitment protocols," in *Proc. AAMAS*, 2013, pp. 1323–1324.

[17] I. J. Jureta, J. Mylopoulos, and S. Faulkner, "Revisiting the core ontology and problem in requirements engineering," in *Proc. RE*, 2008, pp. 71–80.

[18] I. J. Jureta, A. Borgida, N. A. Ernst, and J. Mylopoulos, "Techne: Towards a new generation of requirements modeling languages with goals, preferences, and inconsistency handling," in *Proc. RE*, 2010, pp. 115–124.

[19] N. A. Ernst, A. Borgida, and I. Jureta, "Finding incremental solutions for evolving requirements," in *Proc. RE*, 2011, pp. 15–24.

[20] S. Liaskos, S. A. McIlraith, S. Sohrabi, and J. Mylopoulos, "Integrating preferences into goal models for requirements engineering," in *Proc. RE*, 2010, pp. 135–144.

[21] A. Hallerbach, T. Bauer, and M. Reichert, "Capturing variability in business process models: the Provop approach," *J. Softw. Maint. Evol. Res. Pract.*, vol. 22, no. 6-7, pp. 519–546, 2010.

[22] B. Weber, S. Rinderle, W. Wild, and M. Reichert, "{CCBR}–driven business process evolution," in *Case-Based Reason. Res. Dev.*, 2005, pp. 610–624.