"Socio-technical systems simulation and analysis: a goal-oriented framework"

by
Ruben Afonso Francos

Master Thesis in Software Engineering

June 1, 2010

Thesis Supervisors:

Prof. Paolo Giorgini
Universitá
degli Studi
di Trento

Prof. Oscar Pastor
Universidad
Politecnica
de Valencia

Abstract

One of the characteristics of the socio-technical systems (STS) is their dynamicity. By including factors traditionally considered external, as the social factor (the people involved), or elements from the environment, the system behaviour may vary in unsuspected ways depending on the circumstances.

In many situations STS present an important level of complexity inherent to them; either to its structure, composed of many elements, or to the existent relationships among them. This can lead even to new, unforeseen, emerging features.
Such complexity, coupled with the previously mentioned sensibility to changes, can lead to an unexpected behaviour of the system. This may result in the system ending in a state radically different, even undesired, from the initial state.

In goal-oriented systems, there are situations where unforeseen events can produce very dramatic alterations, making it to stop satisfying goals considered critical goals. Interesting are also the possible actions the system could perform to reach again a valid state, if possible. From the designer point of view, it would be useful to have a mean to analyze this problematic easily in order to obtain knowledge that can help in an hypothetical decision making period.

Having designer requirements in mind, in this work we provide a simulation framework of goal-oriented STS that allows their simulation under a set of predefined events, obtaining numerical results of defined metrics. Such results can be used as a part of the decision making support to the STS designer.

Hoping to provide a higher level of detail than ordinary goal models, in our proposed framework inner characteristics of goals can be described using a process notation. This brings more flexibility and accuracy to scenario modeling capabilities.

The solutions provided by our framework are valid processes that guaranteed the achievement of goals required by the actors of the modeled STS.
Such processes are expressed in a graphical process notation, allowing an easy visualization by the designer; it is also possible to simulate them in a simulation environment and obtain valuable information associated with predefined metrics.
As a output of cited simulations, our framework shows all obtained data through a user-friendly interface that simplifies the analysis.

Keywords

Socio-technical systems, goal-oriented modeling, simulation, BPMN, process modeling.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification.

Ruben Afonso Francos

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Software engineering is not an isolated activity but a part of a broader system engineering process. In a similar way, software systems are not solitary systems, they are a part of wider systems composed by human, social and organizational structure, and they inner relations.

Traditional technical systems approach have suffered of a reduced vision of the whole, where the human actors and the interactions between them were usually not counted.
Socio-technical systems (STS) is an approach to the design of these structures that tries to include technical systems adding also a social dimension, including human actors; after all they are the final entities that make use of the system and interact with it. Keeping this in mind, STS include organizational rules and operational processes regarding all such elements.

It is important to notice that having technical and social entities into a same system may leads to complex relationships, sometimes derived from the own interactions. One change in the structure may derive in changes at different levels of the system. In a greater scale, some subsystems are dependent on some other subsystems as well, so its behaviours and properties are indirectly connected. All this characteristics are translated in a certain grade of complexity, which it is also increased if we consider that STS can be designed having a great number of technical and social elements.

Participation of the people is one of the principles of design of STS[11]. The socio-technical approach is more focused on work group interactions than individual performance. Properly structured work groups, it is assumed, can provide incentives, assistance, and social support better than individual job design programs. Autonomous work groups, quality circles are popular examples of this perspective.
Groups are often given resources and responsibilities for defined areas and work as a team to identify and correct inefficiencies and work issues. Therefore the designer should have a holistic vision of the designed system, where every part works for achieving its own objective, but at the same time, they all are cooperating together as a one single entity to achieve a common goal.
As we have mentioned before, STS structure can lead to an important level of complexity. Although this has some disadvantages, it also provides some interesting features, as mentioned in [12], like the emergence of unexpected properties belonging to the system as a whole, but not found in the subsystems individually [4].

Having mentioned STS characteristics, one challenge at design time is to identify the involved actors and relationships among them, in order to fulfill all system goals.
Having this in mind appeared KAOS[9], a requirement elicitation technique that starts with stakeholder goals and derives functional requirements for a system-to-be and a set of leaf-goal assignments to external actors through a systematic, tool-supported process. KAOS is one of the most well known software engineering approaches that stresses the importance of explicitly representing and modelling organisational goals. In KAOS requirements specification consists of progressively refining high level goals until constraints, objects and operations that are assignable to individual agents are obtained. However, KAOS does not provide information about alternative solutions; this means the designer cannot be sure of the optimality of its result.

Afterwards, [3] proposes an approach inspired in Tropos, an agent-oriented software engineering methodology, which uses the i* modeling framework[10] to guide and support the system development process starting from requirements analysis to implementation. This approach constructs a space of assignment configurations by employing an off-the-shelf AI planner and does evaluation on these assignments to help identifying an optimal design. Particularly, the authors address the following problem: given a set of actors, goals, capabilities, and social dependencies, there is a tool generating alternative configurations which are goal-to-actor assignments and dependencies network among actors. The next step is to evaluate alternatives by assessing and comparing them with respect to a number of criteria provided by the designer.

Related to Tropos, [20] provides a simulation framework focused in having a good enough solution space and a quantitative evaluator assessing the assignment solutions; this way the designers of the software systems can make a decision about which assignment to choose per se. Such framework aims at generating goal-to-actor assignments and dependencies configuration (hereafter configuration or solution) and performs assessment on these solutions based on predefined or user criteria.

The Tropos methodology proposes starting the STS design identifying goals and actors for the modeled system. Such system is represented as one or more actors which participate in a strategic dependency model, along with other actors from the system's operational environment. In other words, the system comes into the picture as one or more actors who contribute to the fulfillment of stakeholder goals. In the next phase, the designer refines the goals and relations between the actors goals to provide the detail specification of the system. In particular, the designer constructs a goal model in which each of top goals is decomposed into AND- or OR-subgoals. Goals are decomposed from a very high abstract level to concrete tasks that can be assigned to human actors of software components. When being in charge of a goal, an actor can continue decomposing it, or fulfills it if it is a concrete task, or delegates it to another actor. It is possible to observe that these multiple choices in an actor behaviour means there will be many possible valid assignments satisfying top goals.

Due to the cited complexity, STS systems present a highly dynamic nature. They suffer changes along the time, in constantly evolution. Because of this characteristic we

are interested in STS capable of reconfiguration at runtime. The reconfiguration feature means that the STS tries to adapt to changes collecting and managing the information about the current state and then evaluating the load imposed on each actor based on his the capabilities. This allows to decide whether the system needs to be redesigned in response to such external (or internal) changes. If so, the STS replans the system structure in order to optimize the distribution of load imposed on actors.
This replan operation is a difficult decision: having multiple choices to readapt the system, one must be chosen. Picking the better alternative can be a very resource consuming operation, and the decision even must be taking analyzing candidates alternatives from more than one point of view.

In this work, we present a goal-oriented simulation framework to support the design of STS. Particularly we provide a modeling language and simulation techniques to study the behaviour of STS from a process point of view. As in [20], we use an AI-planner for searching through the assignments solutions space. We are interested in supporting the goal-oriented structured provided by Tropos adding also more expressiveness at goal level using a business process notation. This allows the obtention of assignment solutions expressed as business processes that can be simulated and compared automatically.

For the process description we choose a business process notation because of the flexibility provided by this kind of notation and also because of its ubiquity in the business analysis world. By this mean, we show how a goal-oriented methodology as Tropos can be complemented with a business process notation, as BPMN, seeing this way a possible use of such methodology in a real analysis scenario.

Contribution

The purpose of our work is to add an extra level of detail at goal level of STS models in order to provide more information about how goals are accomplished, obtaining solution processes that accomplish system goals.
This detail level is achieved adding extra process information to the model goals. More precisely, the contributions of our work are:

- Process information added to each goal of the goal model, allowing to describe AND-OR relations more accurately.
- Following the idea presented in [3], we use an AI-Planner, adding also information describing the process associated with the goals.
- Assignment solutions obtained are described as a process, allowing its execution.
- Simulation of solutions in a testing framework that allows also to define some events to see how the STS reacts.
- Presentation of simulation results in a user friendly interface.

Organization

This document is organized as follows

- *Chapter 1*: this chapter introduces our work.
- *Chapter 2*: provides preliminary information about the goal-oriented methodology used for describing the Socio Technical System, the Planning Domain Definition Language (PDDL) used for the problem description and the process notation adopted (Business Process Modeling Notation). Also we provide some background in process simulation.
- *Chapter 3* : presents the model used for containing the STS description.
- *Chapter 4*: gives an overview of the framework´s architecture. It shows the different modules compounding the framework and how intermediate output data from one module is used as input for another.
- *Chapter 5*: shows a detail description of how the model information is used to obtain the solution set of business processes and the simulation phase.
- *Chapter 6*: provides a description of the simulation and metrics used to evaluate the solution processes.
- *Chapter 7*: discuss the implementation of our framework and involved technologies.
- *Chapter 8*: explains the case study used to test our framework, giving a detail description of it and the defined goal model. Also discusses the found solutions and simulation results.
- *Chapter 9*: presents some conclusions of our and gives some points for further development.

Chapter 2

Background

Our proposed framework makes use of some technologies and methodologies that have important theoretical settings and a maturity enough to be considered reliable.

We have adopted a goal-oriented approach to model STS because the importance of goal-oriented techniques in requirement engineering. Tropos has been chosen because it reflects clearly the concepts of actor and goal and, beeing a modern methodology, it is flexible enough to model a great variety of scenarios. In the process notation part, we adopted a common business process notation as BPMN due to its easily understandable graphical set of elements.

In this chapter we describe briefly the main elements we have used to provide the needed functionalities to our work.

2.1 Goal-Oriented Requirement Engineering

The notion of goal is increasingly being used in requirements engineering (RE) methods and techniques today. Goals have been introduced into RE for a variety of reasons within different RE activities, and to achieve different objectives. Typically, the current system under consideration is analyzed in its organizational, operational and technical setting; problems are pointed out and opportunities are identified; high level goals are then identified and refined to address such problems and meet the opportunities; requirements are then elaborated to meet those goals.

A goal is an objective the system under consideration should achieve. Goal formulations thus refer to intended properties to be ensured; they are optional statements as opposed to indicative ones, and bounded by the subject matter. Goals may be formulated at different levels of abstraction, ranging from high-level, strategic concerns (such as "serve more requests" for a web server system) to low-level, technical concerns as 'response petition 5 on time' for the same example.

There are many reasons why goals are so important in the RE process.

- Achieving requirements completeness is a major RE concern. Goals provide a precise criterion for sufficient completeness of a requirements specification; the specification is complete with respect to a set of goals if all the goals can be proved

to be achieved from the specification and the properties known about the domain considered[22].

- Avoiding irrelevant requirements is another major RE concern. Goals provide a precise criterion for requirements pertinence; defining goals allows to identify characteristics of interest, but also to leave out those unnecessary or redundant.
- Explaining requirements to stakeholders is another important issue. A goal model can provide a intuitive perspective of a complex model, the goal metaphor adapts well to business scenarios. In particular, for business application systems, goals may be used to relate the software characteristics to organizational and business actors that are not used to deal with more technical model descriptions.
- Requirements engineers are faced with many alternatives to be considered during the requirements elaboration process. Alternative goal refinements provide the right level of abstraction at which decision makers can be involved for validating choices being made or suggesting other alternatives overlooked so far. Alternative goal refinements allow alternative system proposals to be explored.
- Goals drive the identification of requirements to support them; they have been shown to be important means for a systematic requirements elaboration process.

2.2 Tropos

Tropos [2] is an agent-oriented software development methodology founded on two key features:

- the notions of agent, goal, plan and various other knowledge level concepts are fundamental primitives used uniformly throughout the software development process.
- a crucial role is assigned to requirements analysis and specification when the system-to-be is analyzed with respect to its intended environment.

The Tropos methodology is intended to support all analysis and design activities in the software development process, from application domain analysis down to the system implementation. In particular, Tropos rests on the idea of building a model of the system-to-be and its environment, that is incrementally refined and extended, providing a common interface to various software development activities, as well as a basis for documentation and evolution of the software.

Models in Tropos are acquired as instances of a conceptual meta-model resting on the following concepts/relationships, as described in the cite document:

- **Actor**: which models an entity that has strategic goals and intentionality within the system or the organizational setting. An actor represents a physical, social or software agent as well as a role or position.

- **Goal**: representing actor's strategic interests.
- **Plan**: which represents, at an abstract level, a way of doing something.
- **Resource**: which represents a physical or an informational entity.
- **Dependency**: between two actors, which indicates that one actor depends, for some reason, on the other in order to attain some goal, execute some plan, or deliver a resource. The former actor is called the *depender*, while the latter is called the *dependee*. The object around which the dependency centers is called *dependum*. In general, by depending on another actor for a dependum, an actor is able to achieve goals that it would otherwise be unable to achieve on its own, or not as easily, or not as well. At the same time, the depender becomes vulnerable. If the dependee fails to deliver the dependum, the depender would be adversely affected in its ability to achieve its goals.
- **Capability**: which represents the ability of an actor of defining, choosing and executing a plan for the fulfillment of a goal, given certain world conditions and in presence of a specific event.
- **Belief**: which represents actor knowledge of the world.

2.3 PDDL

PDDL (Planning Domain Definition Language) is a standard encoding language for classical planning tasks. It has a syntax inspired by LISP. A PPDL planning task is composed by:

- *Objects*: things in the world that interests us.
- *Predicates*: properties of the objects that we are interested in; they have a binary value, being true or false.
- *Initial state*: the state of the world that we start in.
- *Goal specification*: the desired states we want to be true.
- *Actions/operators*: ways we have to change the state of the world where we are.

In PDDL, the planning tasks are divided into two files:

- *domain file*: where we keep the predicates and actions.
- *problem file*: file with the objects, initial state and goal specification.

The domain definition contains the domain predicates and operators (called actions in PDDL). It may also contain types, constants, static facts and many other things, but these are not supported by the majority of planners. The parameters used in predicate declarations (the :predicates part) have no other function than to specify the number of arguments that the predicate should have, i.e. the parameter names do not matter (as long as they are distinct). Predicates can have zero parameters (but in this case, the predicate name still has to be written within parenthesises).

The format of a (simple) domain definition is shown in Figure 2.1

```
(define (domain DOMAIN_NAME)
      (:requirements [:strips] [:equality] [:typing] [:adl])
      (:predicates (PREDICATE_1_NAME ?A1 ?A2 ... ?AN)
                   (PREDICATE_2_NAME ?A1 ?A2 ... ?AN)
  ...)

      (:action ACTION_1_NAME
      [:parameters (?P1 ?P2 ... ?PN)]
      [:precondition PRECOND_FORMULA]
      [:effect EFFECT_FORMULA]
      )
      (:action ACTION_2_NAME
      ...)
  ...)
```

Figure 2.1: Format of a generic PDDL file Domain definition

The problem definition contains the objects present in the problem instance, the initial state description and the goal. The initial state description (the :init section) is simply a list of all the ground atoms that are true in the initial state. All other atoms are by definition false. The goal description is a formula of the same form as an action precondition. All predicates used in the initial state and goal description should naturally be declared in the corresponding domain. In difference to action preconditions, however, the initial state and goal descriptions should be ground, meaning that all predicate arguments should be object or constant names rather than parameters. (An exception is quantified goals in ADL domains, where of course the quantified variables may be used within the scope of the quantifier. Note, however, the even some planners that claim to support ADL do not support quantifiers in goals.)

The format of a (simple) problem definition is shown in Figure 2.2

```
(define (problem PROBLEM_NAME)
  (:domain DOMAIN_NAME)
  (:objects OBJ1 OBJ2 ... OBJ_N)
  (:init ATOM1 ATOM2 ... ATOM_N)
  (:goal CONDITION_FORMULA)
  )
```

Figure 2.2: Format of a generic PDDL file Problem definition

As an example, Figure 2.3 shows a domain for a simple scenario:

in an industry, there is a robot arm that moves assembly pieces from one belt (named belt1) to another belt (named belt2). Initially, there is one piece in the first belt, also where the robot is. We want to move the piece to the second belt and move back the arm again to the first belt.

```
(define (domain robot-belt)
   (:predicates
            (belt ?b)
            (piece ?p)
            (at-belt ?p ?b)
            (robot-arm ?r)
            (robot-belt ?r ?b)
    )
    (:action take-piece
        :parameters (?belt1 ?belt2 ?piece ?r)
        :precondition (and (at-belt1 ?piece ?belt)
           (robot-belt ?r ?belt1)
        :effect (and (at-belt ?piece ?belt2) (robot-belt ?r ?belt2))
    )
    (:action restart-robot
        :parameters (?r ?belt1 ?belt2)
        :precondition (and (robot-belt ?r ?belt2))
        :effect (robot-belt ?r ?belt1)
    )
)
```

Figure 2.3: Example of PDDL domain description

And the corresponding problem is shown in Figure 2.4

```
(define (problem robot-scenario-simple)
      (:domain robot-belt)
      (:objects belt1 belt2 robot1 piece1)
      (:init (belt belt1)
             (belt belt2)
             (piece piece1)
             (robot-arm robot1)
             (at-belt piece1 belt1)
             (robot-belt robot1 belt1)
      (:goal (at-belt piece1 belt2)))
)
```

Figure 2.4: Example of PDDL problem description

PDDL was created in 1998 by Drew McDermott for the first International Planning Competition (IPC). After that many versions have appeared, introducing new features, but not all have survived.

After its creation, the next important version was PDDL 2.1, created in 2002 by Maria Fox and Derek long for IPC3. It included new features as metrics and the use of *declarative actions*, thus, actions having a duration over the time.
The next version, PDDL 2.2 came in 2004 by Stefan Edelkamp, Jörg Hoffman and Michale Littman for the next IPC congress, IPC 4. PDDL 2.2 introduced derived predicates and timed initial literals. Derived predicates are predicates that are not affected by any of the action available to the planner. Timed initial literals are facts that will become true or false at time points that are known in advance.

2.4 Business Process Modeling Notation

The Business Process Management Initiative (BPMI)[23] has developed the standard Business Process Modeling Notation (BPMN).

The BPMN 1.0 specification was released to the public in May 2004. This specification represents more than two years of effort by the BPMI Notation Working Group. The primary goal of the BPMN effort was to provide a notation that is readily understandable by all business users, from the business analysts who create the initial drafts of the processes, to the technical developers responsible for implementing the technology that will perform those processes, and, finally, to the business people who will manage and monitor those processes.

In the original specification, BPMN is expected to support an internal model that enables the generation of executable BPEL4WS. Thus, BPMN would create a standardized bridge for the gap between the business process design and process implementation. However this is a difficult issue and, to this date, there is not a full transformation from

one model to another.

BPMN defines a Business Process Diagram (BPD), which is based on a flowcharting technique tailored for creating graphical models of business process operations. A Business Process Model, then, is a network of graphical objects, which are activities (i.e., work) and the flow controls that define their order of performance.

Historically, business process models developed by business people have been technically separated from the process representations required by systems designed to implement and execute those processes. Thus, there was a need to manually translate the original business process models to the execution models. Such translations are subject to errors and make it difficult for the process owners to understand the evolution and the performance of the processes they have developed. To help alleviate the modeling technical gap, a key goal in the effort to develop BPMN was to create a bridge from the business-oriented process modeling notation to IT-oriented execution languages that will implement the processes within a business process management system. *Figure 2.5* shows an example of a BPMN representation of a payment where a customer choose the payment method before receiving its products.

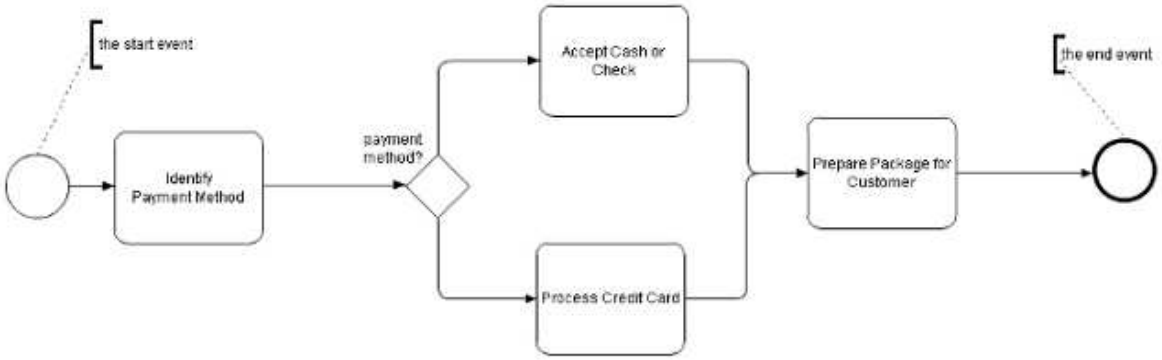


Figure 2.5: BPMN diagram of a payment by a customer

A BPD is made up of a set of graphical elements that enable the easy development of simple diagrams that will look familiar to most business analysts (e.g., a flowchart diagram). The elements were chosen to be distinguishable from each other and to utilize shapes that are familiar to most modelers. For example, activities are rectangles, and decisions are diamonds. It should be emphasized that one of the drivers for the development of BPMN is to create a simple mechanism for creating business process models, while at the same time being able to handle the complexity inherent to business processes.

The approach taken to handle these two conflicting requirements was to organize the graphical aspects of the notation into specific categories. This provides a small set of notation categories so that the reader of a BPD can easily recognize the basic types of elements and understand the diagram. Within the basic categories of elements, addi-

tional variation and information can be added to support the requirements for complexity without dramatically changing the basic look-and-feel of the diagram.

The modeling of business processes often starts with capturing high-level activities and then drilling down to lower levels of detail within separate diagrams. There may be multiple levels of diagrams, depending on the methodology used for model development. However, BPMN is independent of any specific process modeling methodology.

2.5 Business Process Simulation

Business Process Simulation field got more and more attention in past decades as a tool for improving the performance of companies. These types of simulation help in understanding, analyzing, and designing processes, and they provides quantitative estimates of the impact that a process design is likely to have on process performance and a quantitatively supported choice for the best design can be made [27].

Early insights on the business modeling problem were provided by the use of Petri nets in academical environments. Petri nets were originally developed in the 60'ies and the 70'ies, and they were soon recognised as being one of the most adequate and sound languages for description and analysis of synchronisation, communication and resource sharing between concurrent processes. However, attempts to use Petri nets in practice revealed two serious drawbacks. First of all, there were no data concepts and hence the models often became excessively large, because all data manipulation had to be represented directly into the net structure (i.e., by means of places and transitions). Secondly, there were no hierarchy concepts, and thus it was not possible to build a large model via a set of separate submodels with well-defined interfaces. [30].

The development of high-level Petri nets in the late 70'ies and hierarchical Petri nets in the late 80'ies removed these two serious problems. Coloured Petri Nets (also called CP-nets or CPN) is one of the two most well-known dialects of high-level Petri nets. CP-nets incorporate both data structuring and hierarchical decomposition - without compromising the qualities of the original Petri nets.

Despite of these difficulties, Petri nets were in use as a graphical modeling language for more than 40 years and mathematical analysis techniques allowed for analytical verification of many relevant properties of systems" behaviour. However, Petri nets nevertheless were seldom used in business applications. Even the few commercially available Petri net based software development tools exploited Petri net theory to a very limited extend only, maybe because there was an apparent gap between practical needs of business process management in industry and theoretical investigations of Petri nets in the academic sector [29].

Nowadays the most extended business process language is BPEL, short for Web Services Business Process Execution Language, born in 2002, and standardized by OASIS organization [28] as a standard execution language for specifying interactions with Web Services. With the diffusion of web services technologies the popularity of BPEL has

increased. Unfortunately, BPEL standard does not specify a standard graphical notation, as the OASIS technical committee decided it was out of scope. This has derived in a variety of graphical representations, as each software vendor, even using the BPEL language, has provided its own graphical notation.

BPEL was created with web services in mind and it has been used in a wide variety of sectors as a general purpose modeling language, even inside companies, sometimes pushing its semantic to the limit. For example, human interactions are not in the domain of BPEL. Despite wide acceptance of web services in distributed business applications, the absence of human interactions is a significant gap for many real-world business processes. This motivated the creation of an extension to fill this gap and to allow the representation of role-base human activities as well, named BPEL4People [31].
A range of commercial business process management applications capable of process simulation have been developed; they usually are distribute as complete business solutions, sometimes including advanced features as data mining knowledge extraction methods.

As said before, many of these tools use BPEL as execution language for the simulation engine, giving the designer a set of graphical elements to create diagrams. Trying to solve this lack of expressiveness and looking for a general purpose business notation, BPMN (Business Process Model and Notation) [14] was created and defined as standard.
The combination of BPMN graphical benefits with the BPEL execution capabilities at this moment have not showed all its potential yet; there is still a semantic gap between both and sometimes changes in a BPMN diagram cannot be reflected accurately in its equivalent in BPEL (and vice versa) [16]. This is an important issue in business software applications, where graphical editors are often used.
Recent works try to formalize *model-to-model* transformations between them [7], [8].

Despite this difficulties, BPMN is considered to have a greater expressiveness than BPEL, and it opens a new dimension in business process management, being flexible enough to model not only web services scenarios, but a wider set of them.

2.6 Example

In order to illustrate the kind of problem we are aiming to model we provide a theoretical example with some elements.
We show the characteristics of a simple STS, depicted in Figure 2.6 using Tropos notation to describe goals and actor´s capabilities. Specific details of the framework model will be discussed in following chapters. A more detailed case study based in real requirements is provided in Chapter 8.

The proposed STS shows the structure of a data center administration, having a main goal *G1*, the data center administration) decomposed in two goals, *G2:network administration*, and *G3:storage administration*, each one satisfied by any of the involved actors (*A1, A2, A3, A4, A5 and A8*).
The storage administration goal is subdivided by an AND-Decomposition in two goals

: *G8* for security testing and *G9*, performance testing operations. In a similar way, the network administration goal is decomposed in *G4*, if the network supervision is provided by a third private company, and *G5*, if the administration is carried on by the own company. In this case, the administration can be assigned to communications' department personnel, goal *G7*, or to systems' department employees, goal *G6*.

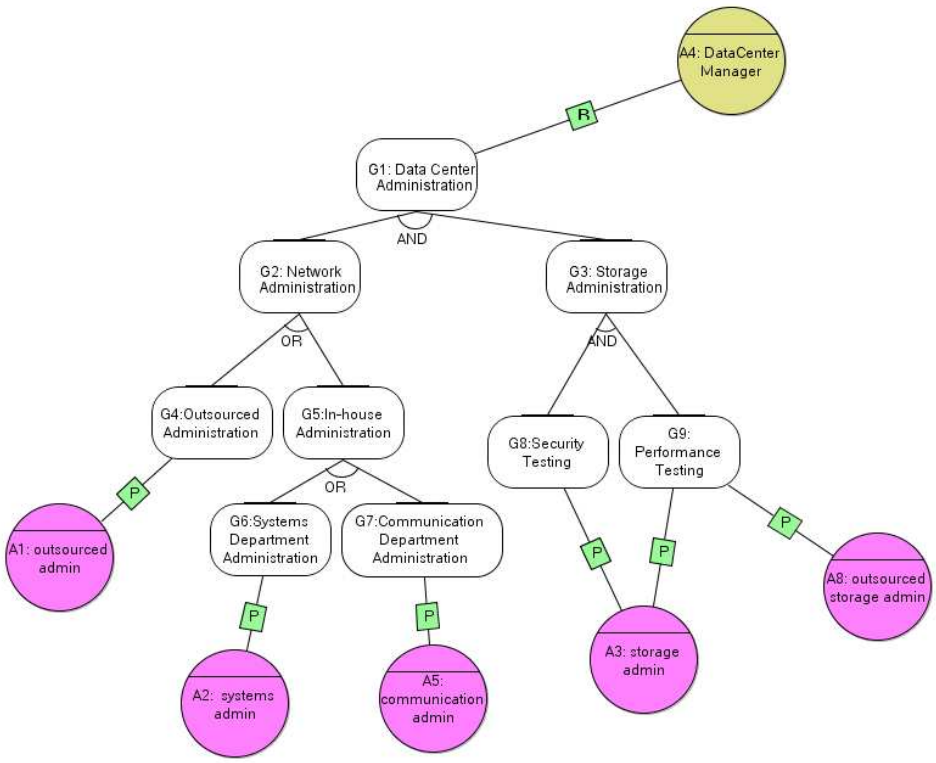


Figure 2.6: Example STS

Actor *A1* is a worker hired from an external company for network administration tasks; actor *A2* is an employee from system department who serves for the same purpose but he works for the own data center company instead of a private one. *A3* represents the storage systems administrator. Actor *A5* is a network admin assigned from the communications department. Actor *A8* is an outsourced administrator for storage's performance testing thus he has the capability of *G9*. The data center manager, actor *A4* does not provide any goal but request the accomplishment of top goal *G1*.

Each goal has one associated process (at least) describing its inner details. In Figure 2.7 we provide their diagrams. Shown processes match goals with the same number (i.e. process *P1* describes goal G1, P2 details G2, etc).

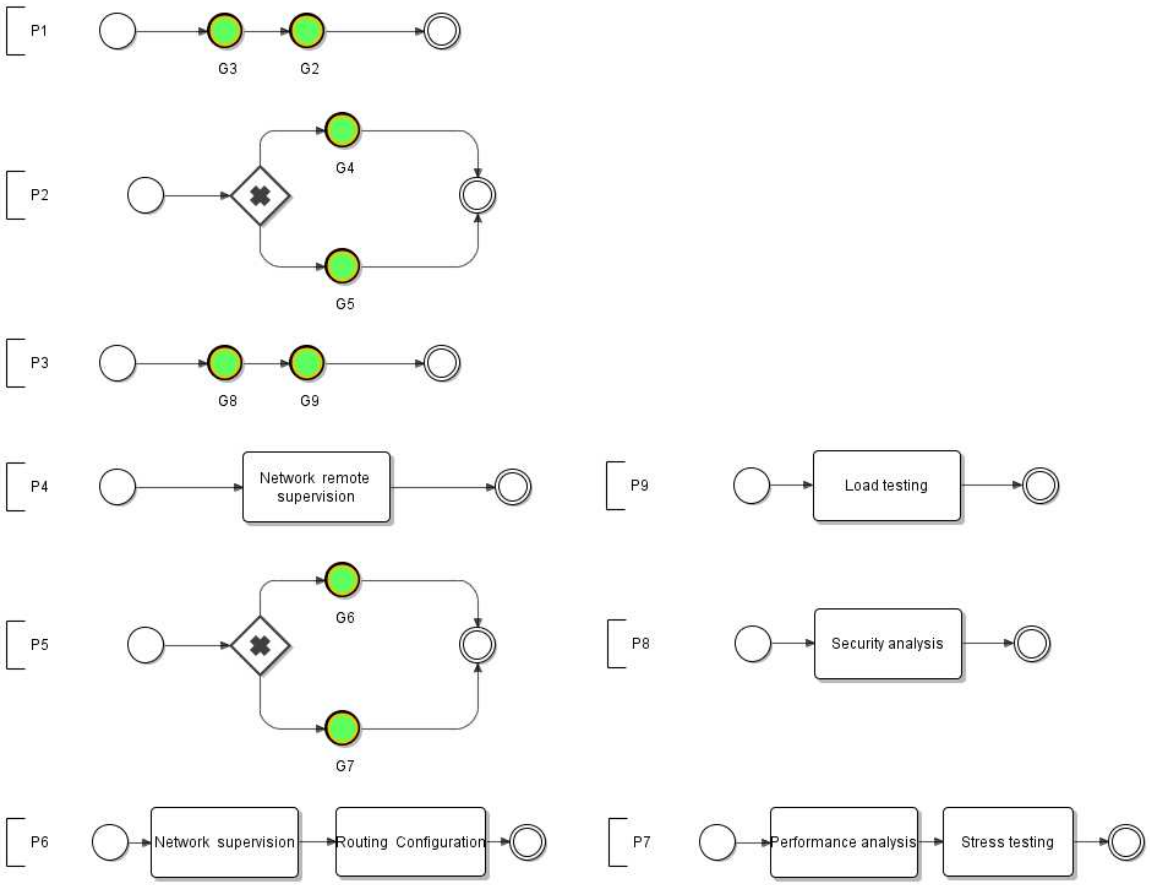


Figure 2.7: Example STS, associated processes

The first process,*P1*, describes how the AND-Decomposition is performed: first *G3* is satisfied and then *G2*. *P1* has one associated process, it means it can be decomposed only in one possible way. Goals *G2* and *G5* are decomposed in their own OR-Decompositions, translated in the addition of an *exclusive gateway* in their associated processes *P2* and *P5*. Process *P6*, associated to *G6:system department administration*, has two tasks in its process definition that must be executed to considered the goal as satisfied: network supervision and routing configuration supervision. Goal *G4: outsourced administration* is satisfied executing the task defined in *P4*, *network remote supervision*, that represents a remote supervision by an outsourced worker.
Processes associated with storage administration, *P8* and *P9* have self-describing tasks, showing some operations related to testing and security analysis.

Observing both pictures it is possible to deduce the number of solutions of this STS, i.e. number of ways of satisfying the requested goal G1: having two OR-Decompositions in the *G2* side of the goal model and an AND-Decomposition in the other part with three valid actor's combinations, we deduce there are *nine* possible assignments.

This information is all the input our framework needs to start the process of solution

generation (some values, as activities' cost take one specified by default).

2.7 Conclusion

Through this chapter we have described the key elements that serve as theoretical support of our framework. As noted before, there is a increasing interest in business process simulation as a tool to undertand complex system behaviours. Process theory by itself cannot model complex systems behaviour properly, it is needed a greater level of abstraction. This can be acchieved using a goal-oriented approach as we propose. Taking advantage of Tropos, we also benefit of a flexible methodology to design complex socio-technical systems.

Chapter 3

Organization Model

In this chapter we provide a description of the main model of the application, the Organization Model (OM), that holds all the description of the STS to model.
Specifically, the model keeps information regarding following elements:

- *Actor-Goal model*: contains the list of actors, their capabilities, and goals involved, including the dependencies among actors and the goal's decomposition. It allows also to specify process definitions for each goal.
  The structure used is based on the *Potential organizational model* described in [20], adding some extra features, as process definition capabilities.
- *Event model*: this model contains a list of events that the designer can specify. Such events may affect process activities during the simulation phase. Different types of effects are provided, described in Section 6.3.1.
- *Evaluation model*: contains some constant values related to the cost associated to the activities. Information regarding defined evaluation metrics can be found in the Chapter 6.
- *Solution model*: solutions obtained by the AI planner (discussed in Section 7.1) are parsed and stored into this model for keeping the obtained information about assessment into the Organization Model.

3.1 Actor-goal Model

An **Actor** is an autonomous entity that performs activities to achieve goals. An actor is capable of satisfying goals, as expressed by *can_satisfy* relationship. Also, it may request another actor to satisfy certain goal, as represented by **Delegate** element.

A **Process** is any task performed within a company or organization. In BPMN a Process is depicted as a network of objects and the controls that sequence them. We extend this definition, including Goals and other Processes into such network, setting respectively *hitBy* and *uses* relationships.

The **Goal** class represents stakeholder objectives (or requirements) for the system and its operating environment. A Goal may be decomposed in a set of subgoals, as indicated by the **Decomposition** element, being this of AND or OR type.

Finally, an **Activity** is an atomic element that is included within a Process. An Activity is used when the work in the Process is not broken down to a finer level of detail. Generally, an end-user and/or an application are used to perform the Activity when it is executed. The relation between a Process and an Activity is realized on *contains* relationship.

Figure 3.1 shows the class diagram for the Actor-Goal Model.

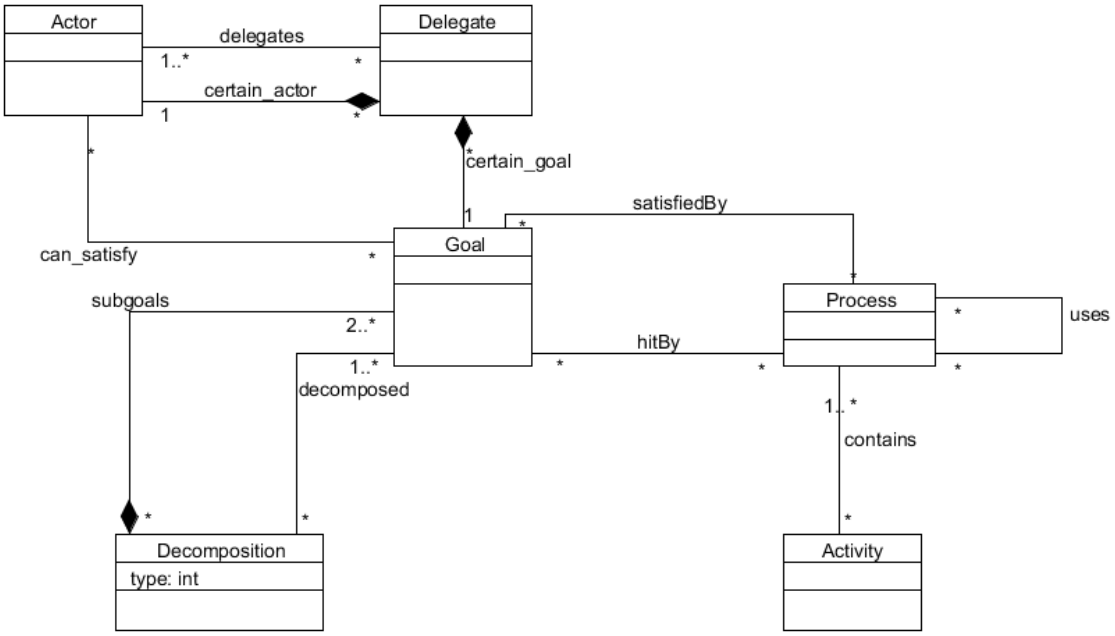


Figure 3.1: Actor-Goal Model class diagram

We associate also following properties to model elements:

- **Executability**: an **Activity** or a **Process** are said to be *executable* at a certain instant if they can be carried out by an actor and their preconditions are met.
- **Satisfiability**: a **Goal G** is *satisfiable* if exists at least one *executable* Process P_k associated with G.

The Executability property ensures that the template generation process (described in 4.2) ends once it is started, detecting possible loops among process references before such process starts.
In a similar way, the Satisfiability property allows to check whether a goal has a valid process associated, which is a mandatory requirement for taking a goal into account in the solution process generation.

As a result of these properties, we obtain some derived model:

- *The executable property definition related to Process is recursive.*

  A process definition may contain reference to Activities, other Processes and Goals. In order to avoid this loop there must be at least one process that contains only Activities in its definition.

- *A process may satisfy a Goal OR just use such Goal in its process definition.*

  The relation between a Goal G and a Process P can be of two clearly differentiated types: P satisfies G (therefore P is executable); or in the other hand, P uses G in its own process definition. Both are realized in the model diagram through, respectively, *satisfiedBy* and *hitsBy* relations.

If we take the cited STS example 2.6 from the previous chapter, all provided information regarding actors (and their capabilities), goals, goal decompositions and process would be contained in this model.

3.2 Event Model

The OM allows also the designer to define events affecting the simulation of the solution process through the Event Model. Such events can be grouped together in a **EventSet** group for easier handling and they can be enabled or disabled, as the Event sets, allowing to insert them into the model but keeping them inactive until needed. This feature also provides the designer a fast workflow to run simulations using a group of events and easily switch to another group, analyzing both results.

All events have a *Name* label, for attaching a representative, more descriptive name to them. Using the proposed Event Model is easy to add new specialized event types. As shown in *Figure 3.2*, in our work we have provided two of them, but the model is flexible enough to allow adding new ones without too much effort.

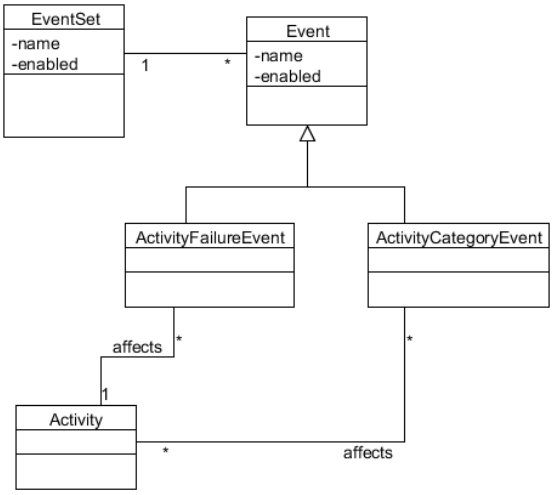


Figure 3.2: Event Model class diagram

The provided event subclass, *ActivityFailureEvent* is intended to affect one task of a certain process. It reflects a situation of a failure in the task where it cannot be accomplished by the associated actor, halting therefore the involved process. The other provided event, *ActivityCategoryEvent* creates an event category to group all indicated tasks. It models a failure event affecting all tasks in the category. If a task is contained in the enabled category, the process will be stopped.

Both event specializations assume that the events are constant on time, this is, when enabled, they will happen every time their associated tasks are executed in a simulation.

Once again, taking as example the previous defined STS from Section 2.6. Processes defined in 2.7 have defined tasks. If we wanted to represent a fail in the task *t21* of the process *P21*, we would define an event of type *ActivityFailureEvent*. In a similar way, if we need to group tasks (maybe because both have similar characteristics) as again *task21* and the task *t3* from process P3, affected both by a same failure, we would create a event of type *ActivityCategoryEvent* including such tasks.

3.3 Evaluation model

This model contains a list of constant values related to simulation metrics, as shown in Table 3.1.

Using such values we calculate the cost of simulated solutions process; described later in Chapter 6.

Name	Default Value
cost	1
replan	4

Table 3.1: Constant values from Event Model

The *cost* value refers to the default assigned cost of every activity. This value can be provided by the designer through the means described in Chapter 7, but in case it is not, a default value is used instead. This default value can be edited as well.
The *replan* value is the default cost value assigned to the replanning operation. When the simulator interrupts a process because an event affecting an activity has been detected, it tries to resume from the stopped activity, reusing the *executed* activities, and it chooses another solution from the solution set. Such resuming operation has a cost, and although it is assigned one by default, it may be changed by the designer if needed.

3.4 Solution Model

The last model we described is the Solution Model (SM), showed in Figure 3.3. It main goal is to store the solution assessments provided by the planner.

Such solutions are parsed and stored into the Solution Model; they will be used later to extract assessment information. The SM stores not just the solution in PDDL format, but each one of its steps performed by the planner, action by action (the format of these actions is described in Chapter 5.2).

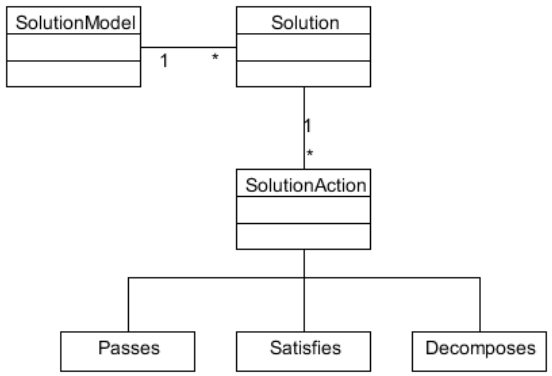


Figure 3.3: Solution Model class diagram

Besides each solution is also stored the solution process associated that has been generated by the Solution Generator (detailed in Section 4.3).

The *Action* class and its specialization classes represent the *action* instructions provided by the planner as solution assessment. They are explained in depth in Chapter 5.

Taking the example from Figure 2.6, shown previoulsy in Chapter 2; the STS has four possible solutions; if we gave the description of such system to the Solution Generator as input, the found assessments solutions would be stored in this model. The next table 3.2 shows the actions forming one of the possible solutions.

Action
AND_DECOMPOSES (A4, G1, G2, G3, P1)
PASSES (A4, A1, G2)
SATISFIES (A1, G2, P21)
PASSES (A4, A3, G3)
SATISFIES (A3, G3, P3)

Table 3.2: One possible Solution from the example STS

3.5 Conclusion

Along this chapter we have seen the description of the compounding parts of the Organization Model. It is the main model in our framework as it contains all valuable information, including the STS description provided by the designer through the model editor and all results obtained by the different modules of our framework. Inside the Organization Model, the most important submodel is the Actor-Goal model because it keeps an important part of the whole system information and is the center of the interaction of the designer with the framework.

Having a central element as the Organization Model, the information required by each module is easily accessible when it is needed, and the generated results can be stored and reused by other components, as the output of the AI Planner, which is parsed and keep in the Solution Model and used later by the Solution Manager.

Chapter 4

Framework Architecture

In this chapter we provide a description of the architecture of our framework.
The steps performed since the designer defines its STS through the editor until solution processes are obtained are not performed by one single module but a chain of them.

The different modules carry out a part of all involved operations, obtaining their input as a data flow from the previous module, then performing certain operations, and finally providing an output *to feed* next modules. As an example, the planner takes care of the assessment problem, providing the Solution Manager with crucial information to calculate the final solution.

The Figure 4.1 shows a diagram of the architecture of our framework. We provide a description of each part; some of then will be discussed on next chapters. All represented edges between modules are important, as all of them contribute in one way or another.
The execution starts when the designer ends the STS definition, and then continues automatically until the Solution Manager provides, if no errors were present, a set of solution processes.
All the steps are automatized and do not require any designer intervention further than defining the number of solutions and the maximum search time in the planner options (even these values could be omitted and the framework would use default ones).

If finally solution processes are found, it is possible to simulate them. This is done in the Solution Simulator module, which also allows to specify events altering such simulation. In any case, the simulation is performed without any other intervention as well. Metric values obtained from the simulation may be analyzed in the Simulation Visualizer.

One relevant detail is that, once calculated, resulting solution processes are stored in the Solution Manager; this means several events may be defined and different cost values tested on the simulator without having to recalculate previous phases (as the planner).

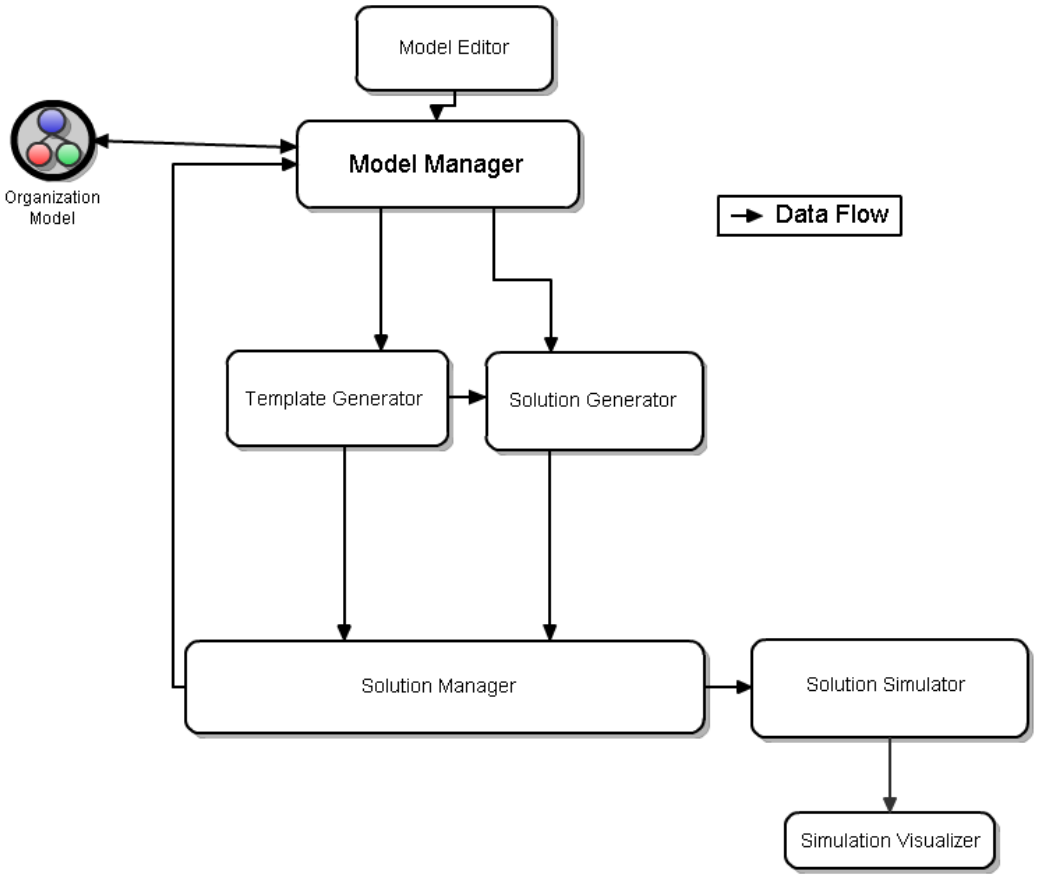


Figure 4.1: Architecture of the proposed framework

The described architecture consists of:

- *Model Manager*: it is the main element of our framework, holding the Organization Model (OM) information, the event model and the cost model.
- *Model Editor*: is an editor that allows the manipulation of the data contained in the OM. It provides the GUI facilities expected from a visual editor.
- *Template Generator*: it builds a template diagram from the process information contained in the OM. In a *template diagram* all information related to goals hierarchy is reflected into the process information, obtaining a hierarchical diagram of processes as the final result. Description of this module is extended in Chapter 5.
- *Solution Generator*: is the module in charge of transforming the model information from the OM into a valid PDDL description file used as input of the AI planner. This module runs the planner with the specified parameters, looking forward to

finding as many solutions as indicated. It usually has a slow performance due to the assessment searching, which is a resource consuming task.

- *Solution Simulator*: obtained solution processes can be simulated in this module, obtaining relevant information concerning execution costs.
- *Simulation Visualizer*: provides a facility to show the simulation results to the user through a GUI consisting on charts and diagrams.

4.1 Model Manager & Editor

The Model Manager keeps the information from the Organization Model. This information is used later by other parts of the framework as the Template Generator and Solution Generator.

It is also deeply integrated with the Model Editor, that provides the graphical appearance of the manager, allowing the user to manipulate the information through a user-friendly interface. Model Manager also is used as main axis of our application because all functionalities provided by the Solution Manager and the visualization utilities can be reached from it.

The Organization Model (OM) is the data model that organizes the data structure necessary for other parts of the framework. The OM contains the information about the organizational model including list of goals, process information associted to them, actors, and all the dependency relationships among actors and goals.

The description of the Organization Model is provided in Chapter 3.

4.2 Template Generator Module

The Template Generator Module (TGM) is the module in charge of validating the process information associated with the goals. It also checks for cyclic relationships, both direct and indirect.

Analyzing the process information associated to each goal, this module generates a process diagram, called *template diagram* editable by the user, that reflects the global goal hierarchy of the Organization Model.

This diagram will be then analyzed by the Solution Manager to generate the solution processes associated with each assessment solution; this way we could consider the Template Generator as the first step in the solution process generation.

It is important to note that the creation of the template is *static*: the template only reflects the hierarchical nature of the defined Organization Model, and it does not depends of the solution assessments.

The Template Generator is composed of three submodules, which are depicted in Figure 4.2.

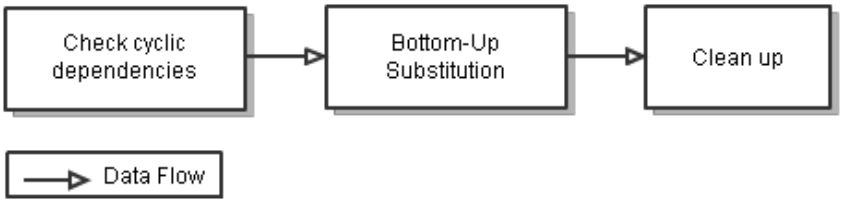


Figure 4.2: Architecture of the Template Generator Module

- *Cyclic Dependencies*: the first step of the process is to analyze the process information associated to goals to verify that no cyclic dependencies are present. This checking is performed starting from the leaf goals of the hierarchy and ending at the top goals of the hierarchy, following a classical bottom-up strategy.
  A *cyclic dependency* would exist if one goal had in its associated process a reference to at least one of its parent goals.
  This cycle would prevent the template template from being created correctly, therefore a validation mechanism to check this situation is mandatory.

- *Bottom-Up Substitution*: this is the core operation of the TGM. If no cycles are present among the goal references, the TGM starts a bottom-up substitution where parent goal references in each process diagram are substituted by the template obtained in its child goals.

- *Clean Up*: in this last operation the resulting template is cleaned up in order to obtain at the end a clean diagram suitable for being edited and analyzed easily by the designer. Performed operations include removal of redundant incoming and outgoing edges, and elimination of orphan intermediate events, among others.

The **Bottom-Up substitution** is performed until the processes associated to the top goals (those having no parent) are also transformed. The top template, associated with the top goals of the model is called *main template*, because it includes all model goals. Every intermediate template generated along the hierarchical tree is stored as a separated template. This is done for performance purposes, but also because in further steps of the process, the Solution Generator Module (SGM) will make use of them to determinate whether processes associated to a goal are *executable* or not (see Chapter 3.1).

All steps compounding the Template Generator are performed automatically without being necessary any designer intervention.

A more in-depth description of the Template Genetor module is provided in Section 5.1

4.3 Solution Generator

The Solution Generator accesses the information in the OM and generates the solutions space. From the OM, the Solution Generator tries to generate solutions by assigning goals to appropriate actors according to their capabilities. Since a goal can be provided by many actors, there are many ways to do the assignment. Therefore, it is impossible to search into the solution space one by one and perform the evaluation to find which one is the best.

The mission of the Solution Generator is to extract a subgroup of solutions from the whole search space, maybe not the optimal ones, but approximately.
Instead of building the generator from scratch, we employ a general AI planner that takes an input describing the planning problem specified following the format of the Problem Domain Definition Language (PDDL) (see Chapter 2.3 for further information).

The Solution Generator runs the AI planner externally, and when it has find the specified numbers of solutions (or search time is expired, whatever happens first) it parses the solutions and insert them into the OM. All this is performed automatically, without any kind of intervention of the designer.
Figure 4.3 shows the architecture of the Solution Generator.

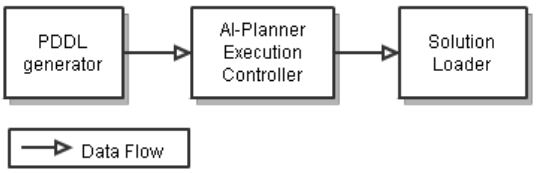


Figure 4.3: Architecture of the Solution Generator

- *PDDL Generator*: extracts the information from the OM and generates a PDDL file. It also access to the Template Generator module to check that each goal has associated at least one valid process.
- *AI-Planner Execution Controller*: initializes the AI planner with the generated PDDL in the previous step. Being the planner executed outside of the framework as an external process, it is necessary to controls its status. The planner is executed until it arrives to the specified number of solutions, or the search time expires. Generally this is a time consuming task that is executed for a relative long period of time, depending of the number of solutions to be found and the complexity of the problem.
- *Solution Loader*: it parses the obtained files from the planner and save the information into the OM. It is important to notice that the Solution Loader parses all the information from the solutions, action by action. This means all information provided by the parser will be available in further steps if needed.

4.4 Solution Manager

The Solution Manager Module (SMM) receives as input the template obtained by the Template Generator Model (TGM), and the assessment solutions found by the Solution Generator.
Having both, the SMM generates a process associated to each solution, called *solution process* because in a simulation environment, the execution of this type of process guarantees all goals referenced in the corresponding assessment solution are achieved.

Basically, the SMM chops the main template removing the references to unused goals. Then proceeds to substitute all goal references with the content of the associated processes to such goal, following the assessments of the planner. Figure 4.4 shows the architecture of the SMM.

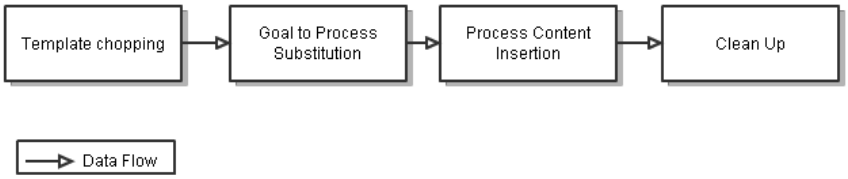


Figure 4.4: Architecture of the Solution Manager Module

All this steps will be applied to each assessment solution found by the Solution Generator Module (SGM).

- *Template chopping*: in this stage, a list of referenced goals is obtained from the solution, and a copy of the main template is created. Each goal reference from the template that is not used in the solution is deleted, therefore remaining only those useful for the solution process.

- *Goal to Process Substitution*: the next step is, goal by goal, transforming each goal reference to the corresponding process reference. It is not allow to reference a goal having no process associated (this situation is detected previously by the Template Generator).
  The Figure 4.5 shows an example of this transformation.

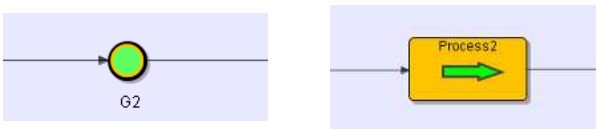


Figure 4.5: Simple transform of a goal reference to its associated process

- *Process Content Insertion*: in this step, references to processes are translated into real content, removing them and inserting into the diagram the content of each process diagram. This operation is done bottom-up, starting from the leaf goals of the hierarchy and ending in the top ones, keeping this way the consistency and ensuring at the end there is not any process reference left.

- *Clean Up*: as last step, some clean up operations are performed on the result solution process, as removing useless intermediate states and gateways.

All cited transformations of the main template are performed automatically, without any further user intervention.

4.5 Solution Simulator & Visualizer

This module takes the selected solution process from the Solution Simulator Module (SSM) and simulates it.

The result then is displayed to the designer through a graphical interface using charts and tables.

The figure 4.6 shows the architecture of the SSM.

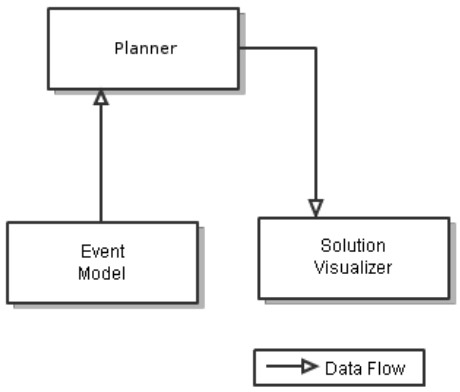


Figure 4.6: Architecture of the Solution Simulator

- *Solution Planner (splanner)*: it is the core of the SSM. Given a solution it simulates the associated solution process.
  If the simulation is taking events into account, the splanner checks if any of them triggers an interruption of the execution. In such case, it calculates what part of the process has been executed so far and base on this decides which other solution to choose (replanning), analyzing the involved costs. All this actions are logged for further analysis.
  It is also possible to run simulations without events; the splanner will just execute

the solution process from beginning to end without interruptions and metric will be calculated as well.

- *Event Model*: The simulation may be done without events or using the Event Model contained in the Organization Model, therefore taking into account events defined by the designer.

- *Solution Visualizer*: it provides a graphical representation of the results obtained in the simulation. If the simulation involved events, all triggered events, replanning actions and chosen solutions are showed to the designer in a tree view representation.

4.6 Conclusion

Our proposed framework has a high scalable modular-oriented architecture; as can be seen in this chapter, there are many modules having each one its own tasks to perform. This allows adding new features without producing a high impact in the rest of elements, and details of a module (for example, the AI planner) may be changed to adapt the framework to diverse requirements without redesigning the whole architecture.

The framework can be divided in two parts, attending to the type of results the provide. We have a more *static* part, formed by the Template Generator and the Solution Generator: their outputs, template and assessment solutions respectively, are expected to be generated and stored for further use. Their obtention is time consuming, so the designer usually will calculate them once, when the STS is properly defined, and then proceed to the simulation phase, which is considered to be more *dynamic* because it is expected to be executed many times using different conditions (of metrics values and defined events).

Chapter 5

Solution Process Generation and Simulation

Once the designer has completed the goal model definition and all involved goals and assigned actors have been specified, it is necessary to add the process description to model goals. This information, joined with the solution assessments provided by the AI planner, will be used as guideline to create a set of solution processes.

Following chapter provides a detail view of the solution obtention. In the first part it shows how, taking all information associated to the goals of the model, it is possible to create a general diagram, that we have called *maintemplate*, which is nothing but a process where all activities are goal references. We also describe how process information is attached to the goal model.

In the second part we describe the information provided to the planner, the format used for defining PDDL files and the type of information it provides as solution.

In the third part of the chapter, we give details about steps taken to *clean* the obtained template, using as filter the assessments obtained by the AI planner. Described steps will be performed for each one of the assessments.

Finally in the last section we discuss the simulation of the solution processes and the considerations involved in the defined metrics. The introduction of events into the simulation phase and how they alter the considered metrics is cover as well.

5.1 Template Generation

One important characteristic of our goal model is that each goal has associated at least one process definition.

Definitions attached to leaf goals are different of the associated with intermediate goals. A typical leaf goal will have a representation formed by a list of connected activities, events and more business process elements (depending of the supported level of BPMN features).

It is important to notice that a leaf goal should not have a goal reference directed to one of its parent goals. In such case, the cycle checking algorithm would stop the template generation and no template would be generated.

An intermediate goal, on the other hand, typically will have same cited BPMN elements

but mixed with goal references.
It is also possible to have plain process references inside intermediate goals process definition; we have added this feature for allowing more flexibility but it use is discourageous because it lacks some semantical meaning. Processes are expected to be associated to goals; a goal reference means its associated process are executed, thus satisfying such goal. Having only a process reference does not provide enough information about achieved goals.

Goal references may be in any place of the diagram where an BPMN Activity element is expected.
If we want to keep the semantical expressiveness between the goal model and the generated template, each intermediate goal must have at least one process containing a goal reference to each one of its child goals.

Figure 5.1 shows a part of the previously proposed Example 2.6, where the top goal is represented through two child goals by means of an AND-Decomposition. In this type of decomposition, a goal is decomposed in two child goals; both must be satisfied to considered its parent goal satisfied as well. The provided process in the Figure corresponds to goal *G1*. If the goal were a AND-Decomposition having many child elements, the process would be similar but having one goal reference for each one.
Between processes could be placed more BPMN elements as events, other activities, etc.

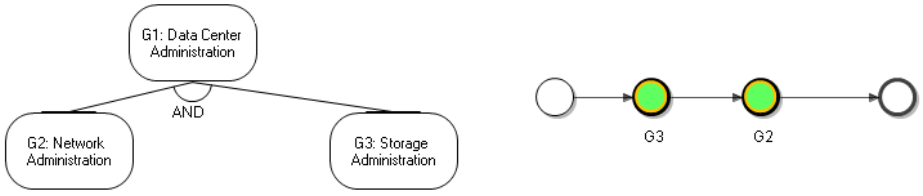


Figure 5.1: AND Decomposition with two child goals

Note that goal references could be in any order (in last example *G3* is performed before *G2*), depending of the requirements of the designer. This provides a mechanism to specify the order of the decomposition, thus adding an extra level of detail. Also it is possible to specify whether an AND Decomposition is done sequentially or concurrently. Figure 5.2 shows the same last example but considering if it were a parallel AND-Decomposition.

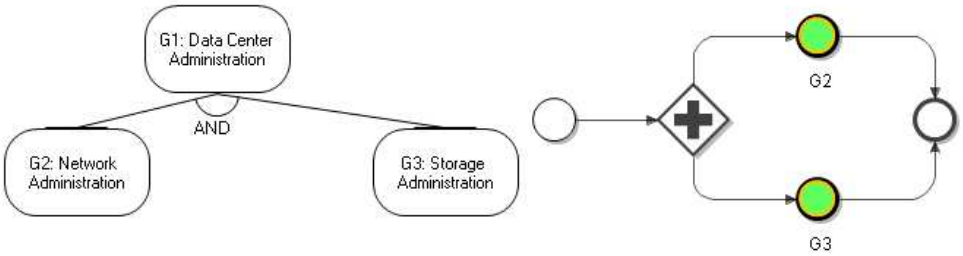


Figure 5.2: Parallel Gateway in an AND Decomposition

The parallelism is indicated using the BPMN Activity called Parallel Gateway; it represents a parallel execution flow starting where the gateway is located. A three-edges output gateway is also supported; it could be used for a three goals AND-Decomposition. Once more, the designer may set the goals in the order more convenient for his scenario.

Up to this point we have described how to represent an AND Decomposition.
Now we show how to do the same operation with an OR-Decomposition; in this type of decomposition, a satisfied child goal is enough to satisfy the parent. Trying to keep the coherence of the goal model with the process representation, an OR-Decomposition is considered strictly OR. The business process associated element is the BPMN Exclusive gateway, where possible alternatives are mutually exclusive.
Figure 5.3 shows an example of such decomposition, using as reference a part of the proposed Example 2.6, the description of the goal *G2*. The depicted process corresponds to goal G2. As in the case of an AND Decomposition, it is possible to define many child goals.

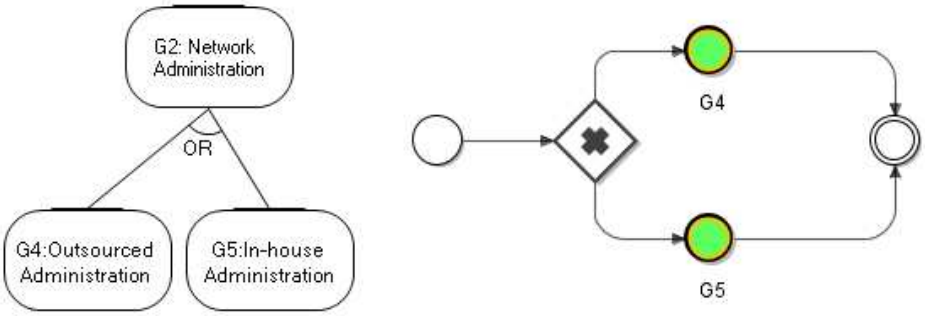


Figure 5.3: OR Decomposition

Once all goals have an associated process and they are described properly, the construction of the main template can starts. It consists of a iterative bottom-up *substitution* process that, starting from the leaf goals, creates a template for each goal of the goal model using as reference all *process diagrams* created by the designer in previous steps. For the leaf goals, no template is created if it do not contain any goal reference in its

associated process. Adding goal references to leaf goals must be done with caution of not creating a cyclic dependency: the referenced goal must not be a parent (direct or indirect) of the goal.

If the goal is an intermediate goal, with some kind of decomposition, a template is created substituting the goal references of its process with the templates of referenced goals. The time require for this task depends of the dimension of the tree and the number of goal references; for performance purposes each intermediate template is stored. This permits to reuse a template in case it is referenced more than once and also is used by the Solution Generator to determine whether a goal is *executable* or not.
When this substitution is finished, all stored templates will contain goal references pointing to leaf goals.

Figure 5.4 shows the transformation of part of the Example 2.6. It shows the top goal *G1* with two levels of child goals; processes associated to both involved parent goals, G1 and G3 are shown as well. The result of the bottom up transformation results in the substitution of the goal G3 for the goals G8 and G9.

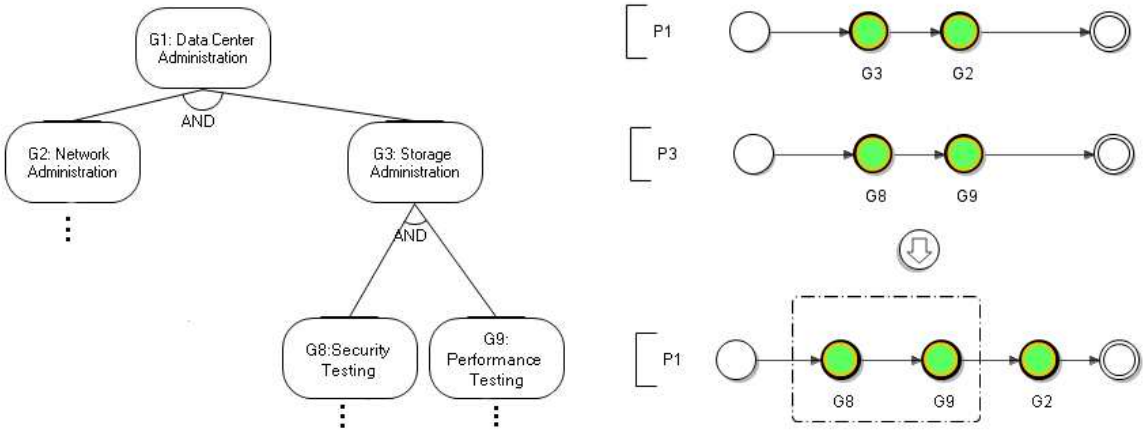


Figure 5.4: Bottom up transformation of an AND Decomposition

Now we discuss another possible scenario, this time when OR-Decompositions are present. The Figure 5.5 shows the scenario having two of such goals, part of the Example 2.6.

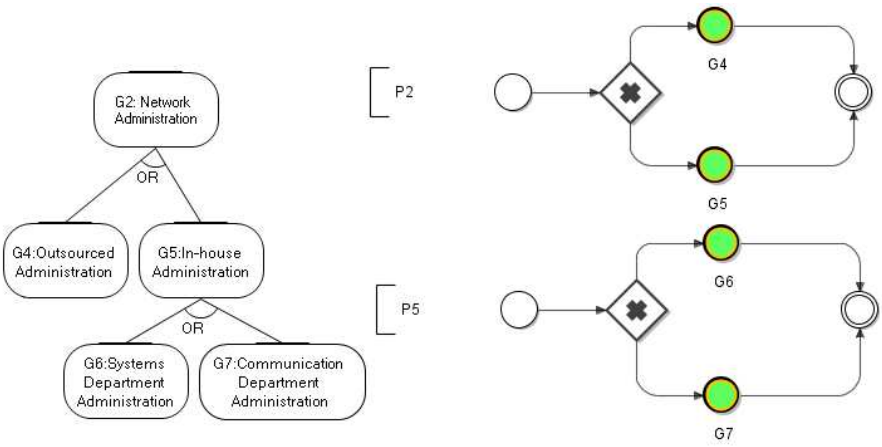


Figure 5.5: Hierarchy in a scenario with an OR-Decomposition goal

And the Figure 5.6 shows the resulting template diagram, with two inserted OR gateways.
Notice that one OR gateway could be simplified, for sure only one of the goals will be used, but the Solution Manager will remove unnecessary goal references. This removal operation is faster than perform complex simplifications that involved semantic analysis at this point.

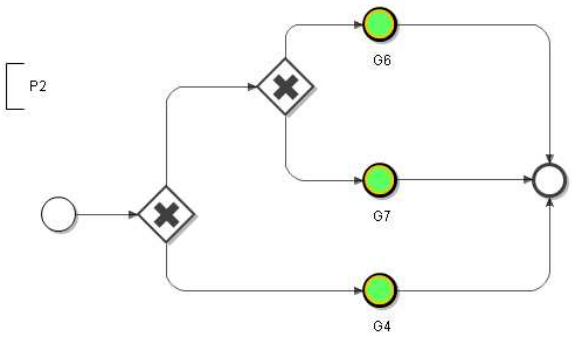


Figure 5.6: Resulting diagram from a bottom-up transformation of an OR-Decomposition goal

The template generation ends when the bottom-up substitution arrives to the top goal, where a template of the whole goal model will be finally generated (the *main template*). This diagram can be edited graphically so the designer can have a global overview of the goal model in a graphical repreentation.

The last remaining step is to clean the template diagram. There may be remaining content from intermediate steps that should be removed to provide a cleaner diagram.

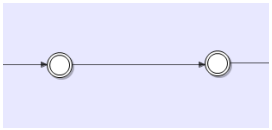


Figure 5.7: Example of intermediate states, candidates to removal

Figure 5.7 shows two typical intermediate events that after some insertion operations become unnecessary (insertions produce artificial intermediate states between process diagrams, and sometimes they became useless due to diagrams' reorganization).

As a result of all steps detailed in this section we obtain a main template describing goal model structure; next Figure 5.8 shows the template for the Example 2.6.

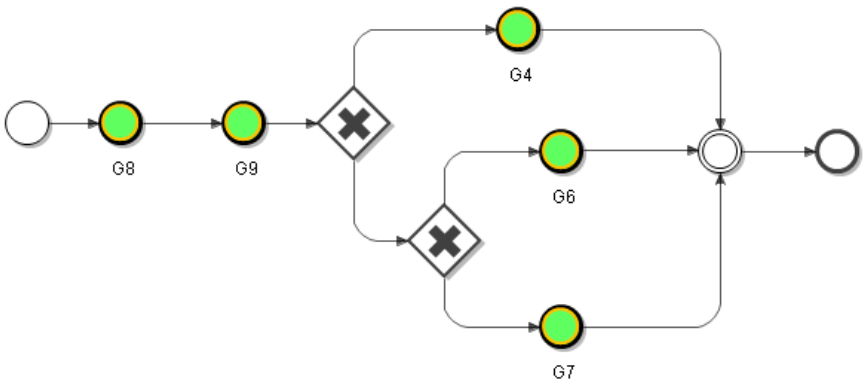


Figure 5.8: Example of intermediate states, candidates to removal

5.2 Organization level planning

The organization level planning problem we have to resolve is how to assign goals to the roles defined in the organization level, assigning at the same time to each goal one of its associated process. Once obtained all results, we are specially interested in the set of predicates declaring an actor fulfills a goal using a certain process (planner action *Satisfies*, described in following sections).

Our framework uses some predicate descriptions previously defined in [1], and inherits background from [20].

5.2.1 Formalized planner input

To formalize the input to the AI planner we use some predicates taking variables of following defined types: goal, actor, agent, role and process. All of them except *process* correspond to concepts of Tropos modeling notation. Due to our level of abstraction

we do not need to make any distinction between actor, agent and role, thus we only use *actor* type.

Process type is added to represent goal-associated process defined in our Organization Model. We adopt predicates from [1], used successfully for a similar formalization in [20]. Table 5.1 shows a list of implemented predicates. Two of them were provided to fulfill our own requirements: *satisfied_by* and *executable*.

Predicates *and_subgoal* and *or_subgoal* contains information describing goals decompositions. It is important to note that the details of such decompositions, i.e. the order how child goals are decomposed, is not provided here: it is contained in the template described in the previous step.

Requires reflects a goal requiring other goal to be satisfied before. It indicates that a goal depends of another goal to be satisfied.

The property associated with a goal of being satisfied by a certain process is reflected in the introduction of *satisfied_by* predicate; a goal may have more than one to represents that it has many associated processes.

Actors also have properties as *request*: an actor request certain goal to be accomplished. If an actor can satisfy a goal then *can_provide* is used to represent it.

The relationship of an actor delegating to another actor the fulfillment of a certain goal is specified through *can_depend_on_g* predicates.

Goal property predicates
and_subgoal (g, g_1, g_2, . . . , g_n: goal)
or_subgoal (g, g_1, g_2, . . . , g_n: goal)
require (g_1, g_2: goal)
satisfied_by (g: goal, p: process)
Actor property predicates
request (a: actor, g: goal)
can_provide (a: actor, g: goal)
can_depend_on_g (a_1, a_2 : actor, g: goal)
Process property predicates
executable (p: process)

Table 5.1: PDDL Predicates

Predicate *executable* indicates a process that is marked as executable (see Executability definition in 3.1). Executability of defined processes is guaranteed by the success of the cycle-detection algorithm in the Template Generator Module, performed before the PDDL description file is generated. This ensures designer has provided valid information in the process associated to a certain goal.

At this level, given a goal we say it is *executable* if a template diagram has been generated for it and it does not contain any goal reference. This fact reveals all goal references from the process description have been correctly substituted for its child goals, having these a valid template as well (thus we could say is propagated bottomup).

Using all this predicates the problem is expressed as a PDDL description. The planner will try to find a valid set of solutions, as many as specified, without exceeding the maximum search time, specified previously by the designer.

5.2.2 Planning problem

The planning problem the AI planner is faced up is to find the list of needed actions to achieve a certain goal; starting from an initial state of the represented world, find the valid set of actions to arrival a certain state of the world. The set of available actions are described in Table 5.2, including the preconditions and effects associated to each one.

AND_Decomposes (a, g, g_1, g_2,. . . , g_n, p)	
precondition:	request(a, g) ∧ executable(p) and_decompose(g, g_1,. . . , g_n, p)
effect:	¬request(a, g) request(a, g_1) . . . request(a, g_n)
OR_Decomposes (a, g, g_1, g_2,. . . ,g_n, p)	
precondition:	request(a, g) ∧ executable(p) or_decompose(g, g_1,. . . , g_n, p)
effect:	not request (a, g) ∧ request(a, g_1), . . . , ∨ request (a, g_n)
Satisfies (a, g, p)	
	precondition: request(a, g) ∧ can_provide(a, g) ∧ ¬satisfied(g, p) ∧ executable(p)
effect:	¬request(a, g) ∧ satisfied(g)
Passes: (a_1, a_2, g))	
precondition:	request(a_1, g) ∧ can_depend_on(a_1, a_2) ∨ can_depend_on_g(a_1, a_2, g))
effect:	¬request(a_1, g) ∧ request(a_2, g)

Table 5.2: PDDL Predicates

The actions *AND_Decomposes* and *OR_Decomposes* are triggered when an actor a requests a goal g and it is decomposed into a list n of subgoals, being the order of the decomposition provided in the process p.
Passes action consists in an action where an actor a_1 delegates a goal g to another actor, a_2.
The last action, *Satisfies*, takes place when an actor a satisfies a goal g described using the process p.

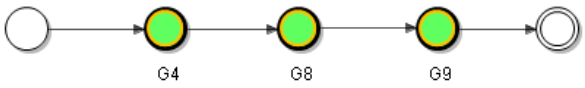
5.3 Solution Process Obtainment

Solution Manager module is the module of our framework in charge of taking the solutions obtained by the AI planner and perform the necessary substitutions on the main template of the model to obtain the solution processes.
This operation is performed automatically and starts a chain of substitutions that ends providing the designer a group of valid solutions expressed as business processes.

Given a solution from the planner, the Solution Manager extracts all actions of type *Satisfies* (described in Table 5.2) to get the assigned actor and process of each goal.
Using such this information, the template is transformed removing all goal references not cited in the planner assessments. This leaves the template populated only with the goals referenced in the solution.

The next step is to substitute each goal reference for its template diagram, which has been generated by the Template Generator in a previous phase.
This substitution phase is guaranteed to finish thanks to the cycle detection algorithm used previously in the template generation (see Section 5.1).
If the process has reached this point without errors, after the substitution all remaining goal references are pointing to leaf goals; as we described in Section 5.1, stored templates only contain leaf goal references.
These are then substituted for the process reference associated by the planner.

We consider again the previous Example 2.6. Assuming the planner has provided the solution including **G4 as a goal assigned** to a certain actor, after processing the main template with the planner assessments we obtain the process describe in the next Figure. We illustrates steps describing how the process diagram evolves through substitutions until it arrives to the final solution process.



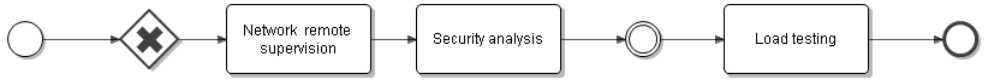
(a)

(b)

(a) diagram after removing unnecessary goal references, following planner assessments.(b) goal to process substitution, following same assessments.

The remaining step is to substitute process references with the content of the processes, taking care of keeping coherence between the inserted content and the location where it is represented in the diagram (adding business intermediate activities or readapting sequence edges if necessary). To ensuring the resulting process is clear enough to be

easily understood by the designer, some cleaning operations are performed at the end. Following we illustrate corresponding process substitution in the previous Figure.



(c)

(d)

(c) process content insertion, (d) process after clean up.

The cleaning operation helps to clarify and remove such remaining elements from transformations. This step, while making the solution process clearer, also helps to make easier the simulation process, removing unnecessary elements that could confuse the simulator.

5.4 Conclusion

In this chapter we have analyzed the whole solution obtention procedure, from the template generation to the final solution processes; it is also completely automatic thus it does not require any designer intervention. As we said in the conclusion of the previous chapter, this part it is not considered to be used so often as the simulator module because the solution obtention requires an important amount time, specially the AI planner.

The Solution Manager is the point of the process where the benefits of the goal-oriented methodology and the process notation converges. As we have described before, assignments provided by the planner in its output indicate the processes to be substituted in the main template, therefore creating the set of solution processes.

Being described using a business process notation, solution processes can be easily analyzed by the designer; this fact is taking specially into account by our framework, doing clean up operations after each important manipulation of process diagrams.

The simulation phase allows the designer to evaluate the results in a plain simulation or in a more complex environment defining customized events. This provides great flexibility to our framework, the designer can model his STS and then study their behaviour under certain circumstances. Our framework is intended to serve as a first approach, more complexity could be added to the simulation part, specially to the replanning decision part, a field the research community has studied for many years.

Chapter 6

Simulation

All the effort invested in the processes of generating the template and obtaining solutions from the planner would be in vain without a mechanism to evaluate the candidate solutions and extract some information about the system behaviour.

Our framework provides a simulation environment where it is possible to run the obtained solution process, showing the output by means of tables and charts.
Being our work focused on business process, shown results are centered on process metrics more than in actor and goals assessments.

We provide a metric on process activities that is used to evaluate process quality. It is possible to change such metric values on each activity without having to recalculate process solutions; this means solutions can be obtained once and simulated many times testing different metric configurations. Our framework provides also a workbench where events affecting the simulation can be defined easily. Including events give an idea of the response of the different solution processes to unexpected situations. All results from simulations are available to the designer, including replanning costs, events occurrence and alternative plans found.

This chapter provides details of the simulations, which metrics are used and how they are calculated.

6.1 Process representation: Timeline

Resulting solution processes are represented using BPMN, a business process notation mainly focused in graphical interpretation. BPMN diagrams are easily understood by a user with minimal business notation knowledge. However, this notation is not execution oriented and some previous steps must be done before simulating a BPMN diagram properly.

In our work we have used a data structure called *Timeline* that simplifies the simulation process, event definition and allows additional information extraction, as event's failure time or process's total cost. When a simulation starts (with or without events), a Timeline is obtained from each involved solution process. Then, the simulation is performed using the Timeline as reference. Note that Timeline structures are dynamically generated every time a simulation is carried out and then destroyed, i.e. they are not stored anywhere.

The Timeline structure is a proposed data structure that represents activities of a business diagram as a sorted list of *instants*. Each instant has a list of activities associated. The position where they are placed depends of their associated cost (see next Section 6.2), which gives us a kind of *time* sequence describing the order of simulation of such activities.

Figure 6.1 shows the Timeline equivalent to a possible solution process of the example 2.6. This example consists only of two activities; in case of having more they would be converted in a similar way.

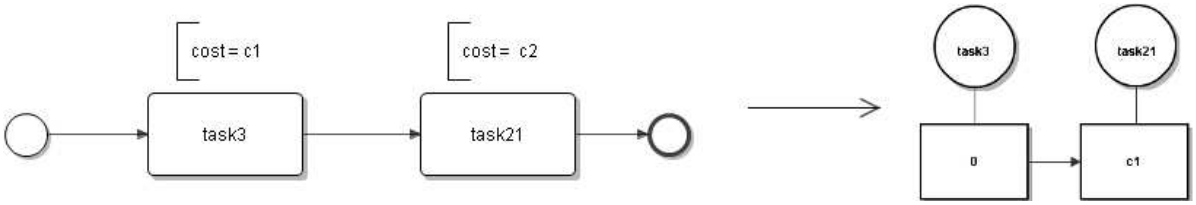


Figure 6.1: Timeline representation of a two-activities process

Using a Timeline structure we transform the process expressed in a flow-oriented style to a more time-oriented representation that suits better with the simulation phase. The first activity starts at instant zero because it is the first one. The total duration of the process would be the last represented instant (in this case *c1* plus the duration of the last group of activities, in our example, *c2*), thus is, *c1+c2*. We do not take into account the rest of elements of the business notation as *event* items or intermediate *gateways*.

Last figure depicted how a Timeline is obtained from a simple sequencial process. The process notation also allows parallel flow, this is, execution of diverse elements at the same time. Timeline structure represents concurrent behaviour grouping together, in the same position of the Timeline, those activities that must be executed in a parallel way.

Figure 6.2 shows activities from Figure 6.1 but assuming they are executed parallely, this is, they start to be executed at the same instant of time, but each of them may ends at a different one.

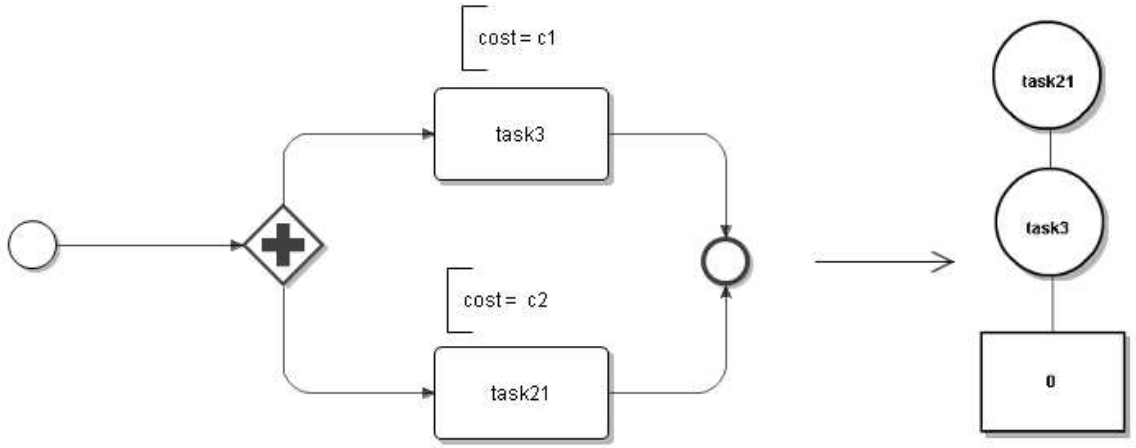


Figure 6.2: Timeline representation of a two-activities parallel process

In our approach, the duration of a group of parallel activities is the maximum value of their durations. This is done to keep the consistency with the adopted process notation. Once a group of activities has started the simulation, the next group of activities will be simulated only when the longest activity has been successfully simulated. Note that this forces to have durations strictly greater than zero. In case of having two parallel activities with the same duration, they will be considered to finish at the same instant of time.

In previous examples we have described simple examples of how Timeline data structures are generated to be accurated with represented processes. More complex processes, having a combination of parallel and sequence activities, can be used as well.

Figure 6.3 shows a complex process having some sequence and parallel activities together. Below is shown also its associated Timeline. Assuming each activity has a cost c_i, where $c_1 > c_2$ and $c_3 > c_4$ (this means the first group in the Timeline has a duration of c_1 and the second, c_3). The Figure shows an intermediate event activity that, as stated in the specification of BPMN, it is supposed to serve for synchronization purposes. However it could be ommited and the meaning would remain the same: the synchronization would take place before starting the next activity.

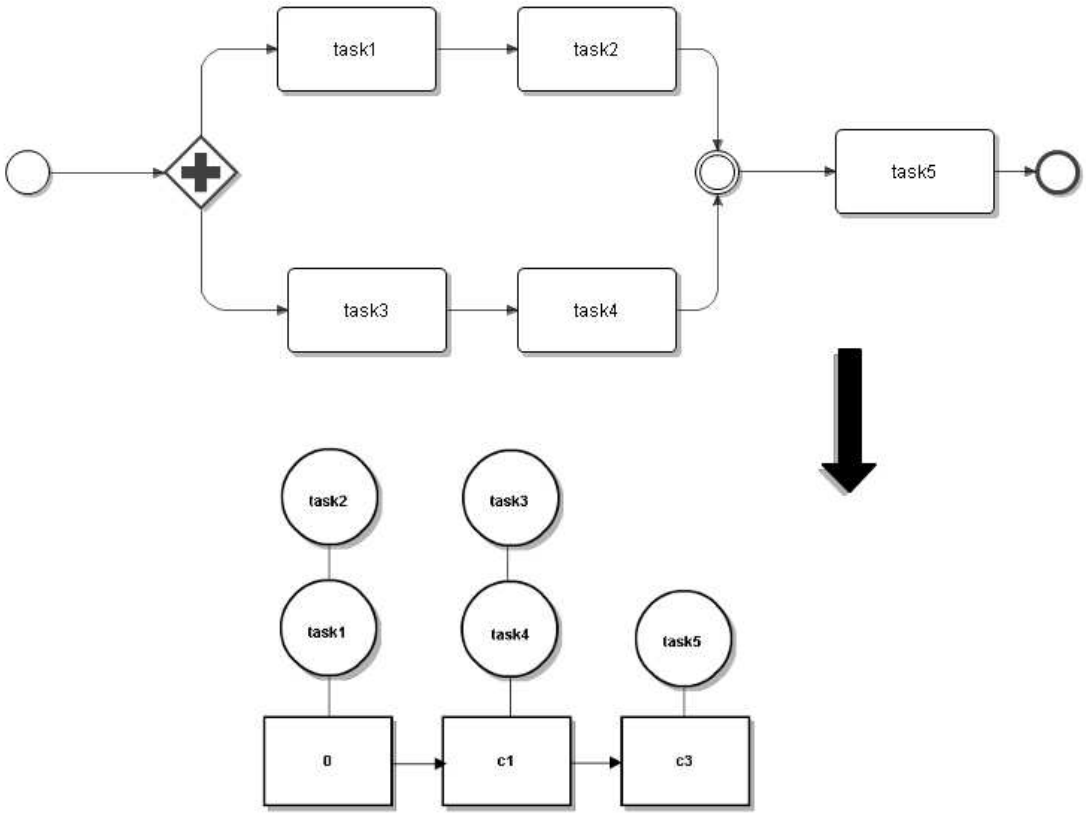


Figure 6.3: Timeline representation of a process

The benefits of adopting Timeline structures are noticed in the simulation phase. An event interrupting a certain activity can be easily located in the time sequence just obtaining the instant where such activity is placed. Resuming operations and, specially, processes comparison are easier with this kind of representation; checking whether two processes are similar is a matter of comparing their activities' lists.

6.2 Evaluation Metric

We define a metric in order to set a mechanism to evaluate the quality of found solution processes.
It must be a representative value that gives us a mean to compare such solutions. considering a process simulation time is in the order of milliseconds, the metric should be relatively easy to obtain as well.

One possible measure could be the simulation time required for each solution; however it is not different enough from one solution to another to consider it valid. Another possible alternative could be the length of the solution: that one having less tasks would

be considered better. This could be valid if all task had the same importance, but usually not all tasks contribute the same to the solution.
Being interested on process tasks, and taking into account that usually each task has its own characteristics, one possible metric is to evaluate the cost of a process taking into account the cost of its tasks. This way each task of the solution process has an associated cost value. In case it is not provided, a default value taken from the Evaluator Model is assumed (see section 3.2 on Chapter 3 for a description).

We define the value C_{task} as the cost associated with a certain task of a process. This value is shared by all the instances of the task that exist in the framework; it means that a task being referenced in many solution process will have always the same value in all its occurrences. This allows to keep the task cost coherent in all solutions, insuring conclusions extracted from the results are consistent. Our framework provides also a mechanism to modify such task costs globally.
Given a process p with n tasks, we define its cost value as

$$Cost_p = \sum_{i=1}^{n} C_{task}(i)$$

Figure 6.4: Cost for simple simulation

This means for evaluating the quality of a solution process we only take into account the cost of its tasks. This value is representative only for *plain* simulations, simulations where not events are introduced. Metrics when process may be interrupted will be covered later.
We do not consider a cost for *event* elements of the process diagram, neither for process gateways (both are special elements of the process element notation).

6.3 Simulation with events

Having a metric as the previously defined and a set of candidate solutions it is possible to simulate them and decide which one is better in terms of global cost.

However, it still leaves open the question of which one has a better response facing to unexpected events. Because of this our framework also provides an environment for defining events and study how different solution processes behave.
This introduces new factors in the simulation to keep metric accurate. For example, it is necessary to set a cost for replanning operations.
Every time an event takes place and interrupts a process, there is a cost associated to the operation of choosing another solution to restart from (if this is possible); even may happens that, instead of just restarting from the beginning of the chosen solution, the simulation algorithm reuses part of the stopped solution, thus reducing part of the global cost.

As can be seen, including events adds some new dynamics that must be taken into account when the final cost of the solution is calculated.

In this section we describes the simulation when events are present, what types of events can be used and how defined metric is modified.

6.3.1 Event types

We initially provide two possible types of events to be used by the designer. However, our framework is flexible enough to permit adding more of them easily.
Events can be defined and then set as enabled or disabled. This allows to keep them stored and test a variety of scenarios switching easily between different configurations.

- ActivityFailure Event: it is an event affecting a certain task. It affects all instances of the task in all solutions.
- ActivityFamilyFailure Event: it defines a *family* of affected events: a group of affected tasks grouped together.

These events may happen in no predefined order. Also it is possible to define as many of them as needed.

When an event is activated (because the current task in the process simulator matches that one defined for the event), the process is stopped in the affected task. It means all precedent executed tasks can be reused if a valid process to restart from is found. This leads us to another factor to take into account in the event simulation: the replanning.

6.3.2 Replanning

A replanning operation takes place when an event stops the simulation of a solution process and it is necessary to decide which alternative solution use in order to continue the simulation.

In the best case, a valid enough candidate to reuse part of the interrupted solution is found. In this case we talk of replanning with resuming: the impact of the replanning in the total cost will be reduced as it is possible to reuse what has been done.
In the other hand, if the best solution candidate forces to restart from the beginning because no task can be reused, the total cost is more affected by the cancelled initial process.
There are many strategies and heuristics developed around replanning decision. We have taken a simple approach based on the speed. When a replanning is needed, our simulator tries with the rest of solutions. This way, at the end we have a description of all possible alternatives of replanning.

6.3.3 Solution cost

Taking into account replanning cost, now we define the cost associated to the simulation of a solution process when an event takes place. It is a modification of Formula 6.4

adapting it to replanning and resuming operations.
Given,

- two candidate solutions *s1*, the initial solution, and *s2*, the alternative, with *n1* and *n2* tasks respectively.
- an event E1 affecting *s1* at instant of time *M*,
- r_{s2} the resuming position in *s2*, that is, the greatest position where to resume from (0 in the worst case, if no replanning can be performed).

$$cost_{solution} = \sum_{i=1}^{M} C_{task_{s1}}(i) + cost_{replanning} + \sum_{j=r}^{n_2} C_{task_{s2}}(j)$$

Figure 6.5: Cost for event simulation

The replanning cost takes a default value set in the Event Model that may be changed by the designer.

6.4 Conclusion

The simulation of processes is a complex task that starts selecting a proper process representation.

BPMN, the process notation selected for representing our solution processes provides a graphical set that simplifies its description and provides a friendly GUI for data handling. However from a simulation-oriented point of view the process could be represented in a better way if its behaviour were described in a time scale. Timeline data representation allows an easy manipulation of processes information in a simulation environment and it is flexible enough to support future additions of more BPMN diagrams elements (for example, intermediate events and messages).

Process simulation without event is straightforward if we compare it with all decisions involved in the simulation involving events. Our framework provides two types of events to be defined but it is flexible enough to allow new ones. The strategies proposed nowadays for the replanning decision are many and diverse, we have implemented a simple one as reference.

A simulation environment is incomplete if it does not measure any characteristic. We have described a simple metric (cost associated to each activity) that permits the comparison of processes quality.

Chapter 7

Implementation

In this chapter we provide a brief description of the involved technologies and details regarding the implementation of our framework.

We have used Java as the main programming language due to its performance and its wide use by the research community. The developed implementation is based on Eclipse Modelling Framework (EMF) technologies. The processes handling and representation is done with Business Process Model and Notation (BPMN).
The AI planner chosen among all available alternatives is a general purpose planner, LPG-td. It was a winner in IPC competition, providing the required flexibility for modelling a variety of scenarios and it was used in related works as well [20].

7.1 Eclipse Modeling Framework

The Eclipse Modeling Framework (EMF) [18] is a modeling framework that provides model-driven development inside the Eclipse framework, being able to automatically generate code from a structured model description.
The main strength of EMF is it model facility code generation, that allows to work with a metamodel editable through a graphical interface. Once the model is completed, EMF generates code automatically for it, optionally even also a model editor. This code can also be manipulated and edited by the developer, remaining unmodified in sucesives code generations.
It also provides tools for automatic testing and editor creation. One point of interest is its framework development environment for Rich Client Application (RCP) that allows generating a fully Eclipse-independent application, that can be deployed easily on any Java-compliant system.

Asides the EMF, the Eclipse Framework also provides the Graphical Modelling Framework (GMF) [19]. More modeling tools can be installed thanks to an extensive repository where search for plugins and download them directly to the platform.

7.2 AI Planner

Following the orientation of [20] we have used LPG-td (Local search for Planning Graphs) [24], a fully automated planning solver that supports PDDL 2.2 specification [25].
LPG-td is very configurable, providing many options to specify the type of solution search, as well as the number of solutions and maximum time of search, among others.

Probe of the performance of LPG-td is the fact that it is used actively in the IPC (International Planning Competition) events held every year.

7.3 Business Process Modeling Notation

The notation used for the representation of processes is the Business Process Model and Notation (BPMN) [14] which is a relatively new notation in the business field developed by the Business Process Management Initiative (BPMI).

Using BPMN it is possible to fill the gap between the business process design and process implementation. Thanks to the Eclipse framework facilities, our framework makes use of BPMN through the BPMN plugin for Eclipse developed by Intalios [15], now added as a default plugin in the Eclipse framework[17].
Being an open source plugin, it has been possible for us to modify and adapt some features in order to suit our requirements (as including goal and processes references graphical elements). Specifically, the BPMN plugin for Eclipse integrates seamlessly into the EMF and GMF Eclipse environments, allowing the creation and manipulation of BPMN diagrams programatically.

In our implementation, BPMN processes are saved in two different files, one containing the graphical information relative to the BPMN graphical representation (with .bpmn_diagram extension), and other with BPMN elements (having .bpmn extension). The process manipulation by code, programatically, is performed always on the second file. However, all graphical changes made using the graphical interface are stored in the first one.

7.4 BPGoal Application

As implementation of all exposed in this document we have build a simulation framework named BPGoal, based on the editor implemented in [20].

BPGoal is a framework for editing and simulating STS systems that allows the designer to develop them using a goal-oriented modeling approach. Goals inner details can be described using BPMN, a common business process notation.
Our framework obtains processes based on the possible assessments that fulfill the described goal model. These solution processes then can be executed in a simulation workbench, under a set of test events defined previously by the designer.

This provides the designer with a decision making support framework, where a STS can be tested in order to understand better its behaviour under certain circumstances.

BPGoal is a completely autonomous and highly extensible Eclipse RCP application that can be expanded easily using plugins developed using the same Eclipse Plugin Framework.

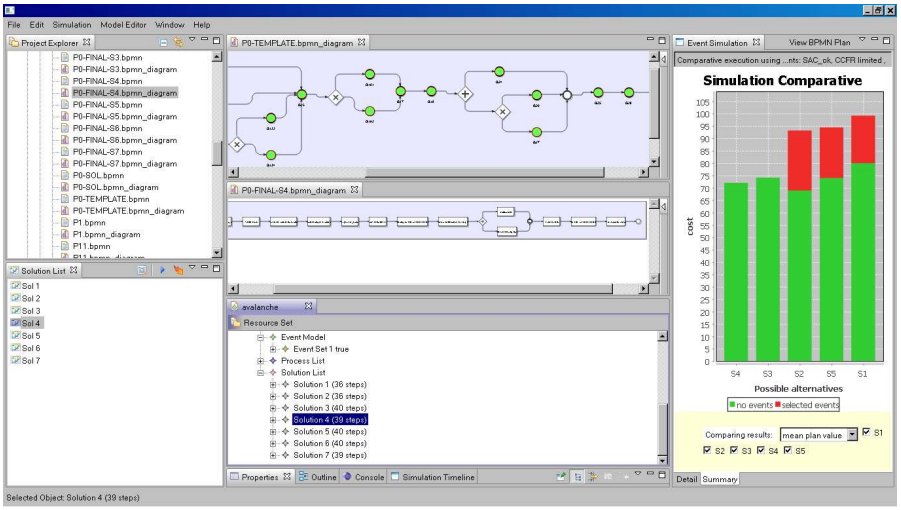


Figure 7.1: BPGoal framework, showing process diagram and statistics

BPGoal provides a content navigator where the designer can create it STS model and organize all relevant files through folders. Our application does not set any limit regarding the internal structure of the project, and it allows the creation of as many models and process diagrams as needed.

A typical workflow using BPGoal features would be,

1. create or edit a STS using the STS editor. With the same editor, attach BPMN information to intermediate goals of the model and describe the leaf goals details through a BPMN process diagram.
2. launch the planner after setting searching options, as maximum time of search time and number of solutions to be found.
3. edit the obtained solution processes for evaluation purposes.
4. create a set of events of interest.
5. simulate the found solutions using the events, analyzing the results to decide which one fits better the requirements; define new events and execute the simulation again with the new configuration.

In the next section we provide a brief description of the main parts of the application and the tasks they are used for.

7.4.1 OM Editor

The Organizational Model editor (OM editor) is a full editor environment for STS models. It provides the expected features found in any modern editor such *copy* and *paste*, loading and saving files in the project, etc.
It allows to manipulate the defined elements in the STS as actors and goals through a tree-styled interface, as depicted in Figure 7.2.

7.4.2 Solution List

The Solution List is automatically generated by the Solution Manager. It is display at the left side of the main window and shows a list with all found solution processes, as shown in Figure 7.2. It allows to select which solutions use for the simulation phase. This is an important decision when running simulations with events: depending of the available alternative solutions the total cost of the simulation may vary.
Selecting one of the solutions is also possible to view the associated process in the process viewer.

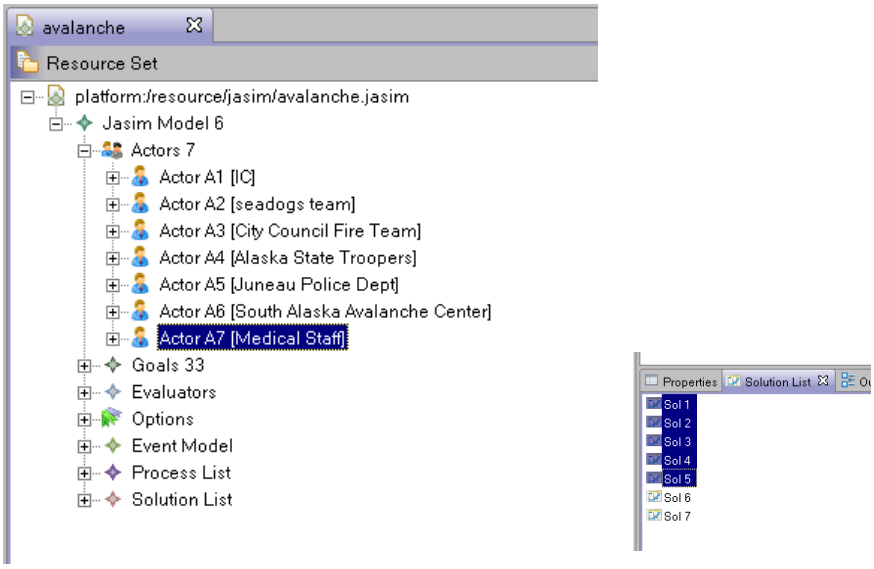


Figure 7.2: OM Editor and Solution List

7.4.3 Simulation results

When the simulation is completed, results are displayed in the Simulation window (Figure 7.3). It is composed by two parts: the Detail tab and the Summary tab.

The simulation algorithm starts the simulation with each selected simulation and if events are triggered, it tries to replan selecting another solution to resume. This means at the end we will obtain as many solutions as selected, and associated to each one a list of alternative execution plans with all possible combinations of replanning.

The Detail tab shows the simulation results organized by solutions, and the mean value for each one of them. Information is displayed in a tree-styled interface, where solution nodes may be unfolded to reveal all details of each replanning. Provided information includes details as triggered events, instant of interruption, replanning costs and intermediate cost values.
All this information is available for a detailed analysis of the execution.

The Summary tab shows the information provided in the Detail tab (Figure 7.3) but in a brief way by means of a bars diagram. Solutions to be displayed may be activated or removed through a check-box to allow a cleaner interface. Summary tab gives two possible visualizations:

- Min cost value: solutions are sorted by its minimum plan cost, in ascending order. The green bar marks the normal cost of the solution if events were not involved. The red bar indicates the cost of the execution plan with lower cost. A solution with only a green bar means it was not affected by events.

- Detail plan values: this visualization shows all plan costs associated with a solution, in ascending order. All non interesting solutions may be disabled by the check boxes, leaving only the desired results.
  A non-interrupted solution will display here only one bar (because it has only one execution plan).

In the top margin of the Simulation window there is also information indicating which events were used in the simulation.

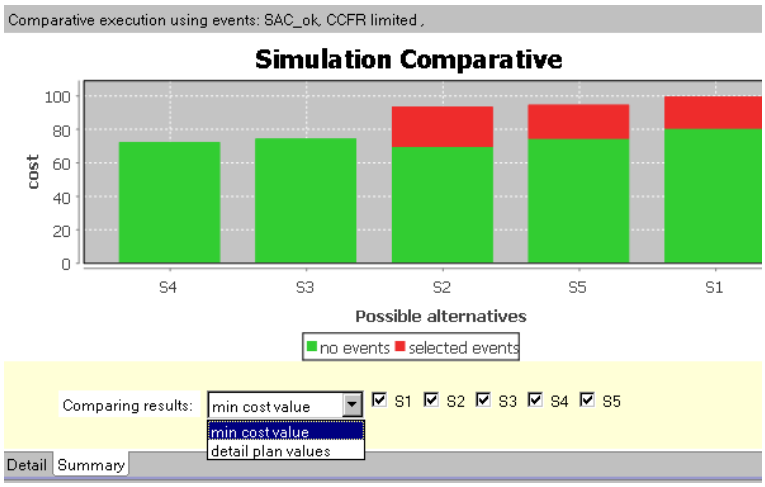


Figure 7.3: Event Simulation result

Chapter 8

Case Study

In this chapter we show an example case where our framework is used to define an STS based in an emergency management situation. In the Chapter 2 we provided a theoretical example of an elementary STS to clarify some previous sections but now we discuss how to model a real scenario.

Instead of setting a hypothetic scene we have chosen a real description describing real actors and goals. The chosen scenario is **Avalanche Crisis Management**, a real urban avalanche response plan based in the original document [21], from the City and Borough of Juneau (CBJ), Alaska.
It assigns available resources and defines a management structure to response to an emergency situation caused by an avalanche. The response plan is flexible enough to analyze response capabilities in a wide range of avalanche situations, from open space events to such happening in a urban environment.
As stated in the beginning of the document,

The plan identifies departments, agencies and individuals that are directly responsible for emergency response and critical support services. It provides a management structure for coordinating and deploying essential resources following that of the nationally recognized Incident Command System.

Using the original document as reference we have defined a goal-oriented model trying to capture the essence of the organizational structure.
The document provides a sequence of events that has been used for setting the order of the goal assessments sequence, the list of involved actors has been extracted almost literally from the original.

Also, while trying to be accurate with the document, some assumptions have been made in order to obtain a well defined problem model from the provided information.
More specifically, our framework defines process and activities related to them and the original, being a descriptive document, does not provide this kind of descriptions. We have added some process details where it was not specified, in some cases adding more complexity to our model than stated in the original.

8.1 Scenario description

The incident response starts with an avalanche event. Initially, Intervention chief (IC)) starts the operation with traffic and crowd control, delegated operations to the Police in first place or to Alaska State Troopers (AST) if necessary.
The next step in the procedure is to decide if the avalanche zone is secure enough from secondary avalanches, requesting the intervention of South Alask Avalanche Center (SAC) specialists. They evaluate on the ground the risks of the incident area giving the agree if there is no danger.

On the other hand, if the area is still insecure, they may decide to wait a certain amount of time or use explosives if necessary for clearing the surrounding slopes. This dangerous task may be perform by them or with the help of AST units.
After the SAC green light, if there are buildings involved in the incident, the IC requests the checking of structural hazards to the engineers group of the City Council Fire Team (CCFR). Once these two approvals are obtained, the IC gives the order of setting an operational area in the affected zone, creating a delimited area, setting up an on-site hospital and the landing zone for the helicopters.

These tasks are performed by CCFR or AST, depending of the availability.
When the security of the searchers is guaranteed, starts the recover victim phase with the help of the CCFR for obtaining information about the possible victims location, the use of specialized search dogs (Seadogs) for trails following and collaboration of available medical units in charge of assisting and transporting victims to the hospital.
The incident is considered as resolved when all victims have been found and evacuated.

In our proposed scenario, the two main goals to accomplish are to ensure the searchers safety and to recover victims as soon as possible, in this order.

Satisfying these goals means the incidence is resolved. Each one is decomposed in other subgoals until arrive to the *leaf* goals, which are not decomposed but assigned to one or more actors, who have the capability of achieving the goal. Such actors are shown in Table 8.1. For the safety concerns, when a incident response starts, the main tasks are safety related operations, as crowd control, zone security analysis, and setting an operation area for resources deploying.

The victim recovery goal is also guaranteed being decomposed in other goals as finding the victims and victims evacuation. In this case two goals (G26, *Ask familiars*, and G27 *Ask survivors*) has been represented as concurrent in the process diagram associated to its parent goal (G28).

Goals can be accomplished by a certain actor or in some cases by many of them, providing this a wider solution search space. The main goal G0, is requested by the actor A1, *Incident Chief*, (IC) who delegates associated subgoals to trusted actors, starting this way the goal assignment process.
Actors and its corresponding capabilities are included in the Appendix, 10.3.

Name	Abbreviation	Description
A1	IC	Incident Chief
A2	Seadogs	Seadogs rescue team
A3	CCFR	City Council Fire Team
A4	AST	Alaska State Troopers
A5	JPD	Juneau Police Department
A6	SAC	South Alaska Avalanche Center
A7	MS	Medical Staff

Table 8.1: Actors involved

Many of the defined goals have a precedence constraint, this is, they cannot be performed until another certain goal is satisfied. This is mandatory because some goals can be achieved only when certain conditions are met. For example, goal G2 (*victims recovery*) can be satisfied only when the security of the searchers is guaranteed; also G2 (*victims evacuation*) can only be accomplished, obviously, after victims have been found (G21, *find victims* goal). Adding these precedences goals will be achieved following an explicit order, which is also common in these types of scenarios.

The Table 10.1 in the Appendix provides a description of each goal.

Using a classical goal-oriented modelling approach, it would be necessary to add explicitely such precedence information to the goal model, this way the planner would take it into account, generating the solution assessments in the correct order. However, our framework uses also process information attached to every goal where it is indicated the order of the decompositions. This means the resulting solution processes from the template transformation will execute the goals in the correct order. Having these mechanism, parent goals as G0 can determine easily the order of precendence of its child goals. For example, the first goal G0 has the process information shown in the Figure 8.1 where the order of the AND decomposition is specified to accomplish first G1 and then G2.

Figure 8.1: Process information associated to G0

The total of goals in our scenario is 37, being 20 child goals and 17 intermediate goals. Each one has its own process description associated, therefore obtaining 37 process definitions. In these definitions there is a mean value of 2 goal references (max value of 3 for both decomposition types, AND e OR).

8.2 Results

Attending only to the goal structure represented there are 48 different solutions. But taking into account also the number of processes associated to goals, the total number of solutions grows to 86 (there is a goal, G31, with two possible associated processes).

Despite this, because of the complexity of the task, the used planner gets six different solutions for the proposed scenario in the maximum given time (1 min). This obtained solutions are different each time the planner is executed because the search algorithm starts with a random seed. This makes saving found solutions even more important.

Table 8.2 shows the found solutions. Results are shown in rows, Goals columns represent the different goals assessments provided by the parser.

Sol	**Goals**													
sol1	G31	G4	G14	G15	G161	G163	G164	G18	G26	G24	G25	G28	G29	
sol2	G31	G4	G14	G15	G162	G163	G164	G18	G26	G24	G25	G28	G29	
sol3	G31	G4	G14	G15	G162	G163	G164	G18	G27	G24	G25	G28	G29	
sol4	G32	G4	G135	G133	G15	G162	G163	G164	G18	G26	G24	G25	G28	G29
sol5	G31	G4	G135	G133	G15	G162	G163	G164	G18	G26	G24	G25	G28	G29
sol6	G31	G4	G135	G133	G15	G162	G163	G164	G18	G26	G24	G25	G28	G29

Table 8.2: Candidate solutions

Once the set of solutions has been obtained, the next step is to start the simulation and analyze their quality.

Initially we perform the simulation with the events disabled, therefore the solutions are executed completely without interruptions or replannings. As evaluator of the results we use the cost value of the solution process (see Chapter 5).

Table 8.3 shows the resulting cost detailed in Cost column and the the number of steps performed by the plannet to obtain each solution. The details of the activities' assigned costs are provided in the Appendix, Table 10.5.

Sol	**Steps**	**Cost value**	**Sol**	**Steps**	**Cost value**
sol1	38	68	sol4	42	84
sol2	38	69	sol5	43	88
sol3	38	67	sol6	42	93

Table 8.3: Simulation Cost of each solution, without events

As can be seen in the results, solution #3 has the lowest cost of all the candidates. This means in a ideal situation, without unexpected events, this would be our choice. It is also one of the *shortest* solutions, with 13 goals satisfied (solutions #4, #5 y #6 have one more goal). Analyzing each associated process we observe that the lower cost value of is due to the goal G11 (*SAC aproval*); here the solution #3 has assigned the

goal G14 (*SAC gives ok*), while others accomplish G13 (*Use of explosives*), with higher child goals costs.
The difference between #3 and solution #1 is the associated process in G31; last one uses process P31, with a higher cost than P312, used in solution #3.

Now we proceed to analyze the behaviour of found solutions in a simulation environment where some unexpected events happen, altering the expected process flow.
Using the capabilities provided by our framework we define two new events, described in Table 8.4. The first one, *SAC_ok*, represents a failure in the SAC decision regarding its capability to decide successfully whether a zone is secure from secondary avalanches or not. The second event added, *AST unavailable* affects not one but two activities, representing the hypothetical situation of a limitation in the services provided by AST group, in this case being unable to delimit an operational area and to perform the survivors information retrieving.

Event	**Involved process activity**	**Affected goals**
SAC_ok	SAC gives OK	G14
AST unavailable	Delimit area,Ask survivors	G133, G27

Table 8.4: Defined events of the simulation

Using these events and the solutions shown in Table 8.3 as input, our framework tries to find a valid execution plan, simulating the processes and replanning when necessary. Results of the simulation are shown in Table 8.5; this information can also be analyzed graphically, the output of our framework provides this features by means of charts. Having six solutions, if one of them is not valid due to the defined events, there are many replanning alternatives when starting from each solution.

Solution	**Found plans**	**Minimum Plan Cost**	**Mean Plan Cost**
Sol1	2	92	94
Sol2	2	92	94
Sol3	1	90	90
Sol4	-	-	-
Sol5	1	88	88
Sol6	1	93	93

Table 8.5: Results of the simulation with defined events

As can be seen in the results, solution #5 have only one plan associated and also the lowest cost value. This means its obtained plans (in this case just one) are less affected by events. In this case, it means #5 is not affected at all, because its cost value is the same with or without events. Solutions #1 and #2 increased its costs because of the events (without them, they have the lowest values of the solution set). It interested to observe that solution #4 does not provide any plan: it is because execution cannot be

resumed when its execution is stopped. For further analysis, at the end of this document we provide the solution process S1 (10.1) and S5 (10.2).

Given our set of events solution #5 seems to be the best solution to take, having a good performance with defined events (in fact, it is not altered).

8.3 Scalability

The scalability and performance of our framework depends of each of its parts and the type of task they perform. The analysis implies dividing the whole simulation in two parts. Following we describe each of them step by step.

The first stage, considered as a *static* part because it is done once and then reused, is the phase composed by the template and solution generations: these steps do not need to be repeated again every time a solution simulation is performed.

The second stage, more *dynamic*, is the simulation of found solutions, optionally involving also events simulation. The performance of this part is important because simulations could be repeated many times, having an important number of candidates solutions and events, and is supposed to be the part of the simulator more stressed by the designer.

As first step of the framework workflow, when the designer has finished the description of the goal model, the Model Manager checks the consistency of the goal model, i.e. making sure there are not leaf goals without associated actors. This is a fast operation, without significant impact on performance.

The next stage is the template generation from the process diagrams of each goal, operation performed by the Template Generator module. This task could become more time consuming in large models because the module checks for cyclic references among goals, operation resulting in a bottom-up checking algorithm along the whole goal model tree.

As final step, the template generation involves the analysis of all process diagrams present in the goal model, generating a template for each intermediate goal, resulting at the end in the main template generation. Because of the involved number of disk access, intermediate templates are saved and reused when possible. The time invested in this stage is high, so we decide to test the template generator performance increasing the number of goal references in the processes which template is generated from.

Results in Table 8.6 were obtained using our scenario goals.

References	Cycles checking	Template generation
2	100	963
4	934	3368
6	5362	19546

Table 8.6: Time invested in the template generation (ms)

It seems that the number of goal references in the process diagrams may be a factor to take into account in scalability matters. Its growth increases significantly the amount of time invested in the template generation.
These results show that the best performance is reached when we have a small number of goal references in the process diagrams. The number of such references depends of the number of child goals of the goal, therefore cited value will be low is most cases.

The next part of our framework to be evaluated is the Solution Generator.

Once templates are generated, the Solution Generator uses the intermediate templates to calculate the **executability** of involved goals; this operation is fast and it does not represent a problem to the scalability. Such information then is translated into the PDDL definition file generated from our scenario. This file will be used to feed the chosen AI planner.

When the planner ends, result solutions (if any) are parsed back to the model. They will be used later to generate the solution processes through the Solution Manager. In order to illustrate the performance of these steps in Table 8.7 we provide some results.

Time required by the planner serves only as orientation: it uses a different random seed every time it is executed so discrete search times are not representative of its performance. A more detailed study of the scalability of the chosen AI planner (LPG-Td) can be found in [3].

The time needed for the solution search depends of the selected planner. Being a goal-oriented problem with many actors and goal decompositions it may be high.

Solution	Planner	Planner solution parsing	Solution process generation
1	9373	6	102
2	9000	6	109
3	9300	7	150
4	9780	9	186
5	9000	9	240
6	9200	9	290

Table 8.7: Time required for the solution generation (ms)

As we have indicated, planner time is not representative so we will leave it outside our analysis.
The second column of the table, the parsing time needed for parse the solution from PDDL format to our model, reveals that the cost of this operation is irrelevant; its value remains almost constant while increasing the number of solutions to parse. The last column reflects the time needed by the Solution Manager to *clean* the main template with each solution in order to obtain its process representation. This operation is not time consuming even having more than one solution to generate.

Finally we arrive to the Solution Simulator, the module in charge of simulating solution processes.
In this stage, processes to be simulated dont have any conditional task to evaluate. This

means that simulation without events is performed with no decisions to be made, like replanning branches or interruptions.
Again, we carry out a test. Taking as example a process with 12 activities, we increase the number of them to view how required simulation time increases, first without events. Results are shown in Table 8.8.

Activities	**Time**
12	15
36	21
48	32
84	63

Table 8.8: Time required for a process simulation (ms)

The advantage of all performed transformations along the framework is visible in this phase of the workflow. Thanks to assessment solutions joined with a process representation, simulations can be performed in a fast way.

When the simulation requires replanning, the decision of which other solution should be taken is made just taking one of the candidate solutions and trying to execute it (avoiding those already discarded). This, although could lead to picking the wrong alternative, minimizes the time dedicated to this task.

Chapter 9

Conclusions and Future Work

The STS approach, taking into account the human dimension and relationships between technical systems and involved actors, allows a flexible modelling of complex software systems.
Nowadays the human factor is even more important than ever; we are used to be surrounded by software systems in our workplaces but also in our everyday life, and the design of such systems must recognize the importance of users and the potential impact of their behaviour in the software system.

Choosing a goal-oriented methodology as we have done in our framework provides us with a robust tool resource for modelling STS systems. It allows the designer to focus on important features of the model, as system goals and user interactions.
Goal-oriented methodologies have been studied and analyzed for years, having therefore a consolidated formal base. The solid theoretical background allows to evaluate system features in a consistent way, even finding emergent characteristics not guessed at a first sight (i.e. performance and scalability analysis).
This also contributes to expand the use of such methodologies in automatic environment, as our framework for example, where information contained in the goal model feeds an autonomous AI planner and generates solutions using a business process notation, all without intervention of the designer.

In this work we have complemented descriptive capabilities of a goal methodology with the benefits of process theory used to describe inner details of the goal systems. This approach provides some important benefits in the simulation phase. Using BPMN, a modern business notation widely extended in business environments, resulting diagrams can be represented in a clear way, easily understood by users with minimal business notation background.

One point of special interest seems to be the performance of the selected planner, responsible of searching valid assessments among all possible combinations.
Thanks to the PDDL format translation phase and the modular approach used in our framework, it is possible to change the planner with one that fits better our requirements. This leaves the door open to future planner with better performance in goal-oriented scenarios.

From a general point of view, starting from the first step of the workflow, we have gone from a goal-oriented scenario, harder to understand for a non technical user, to a

result represented as a set of solutions expressed in a more friendly notation that can be also executed and simulated.

The combination of the goal-oriented approach with the inherented benefits of a process notation in terms of understanding and simulation capabilities.

Event Module provides the final user a simple and straight forward way of defining events, analyzing the scenario through a graphical and intuitive tool. These advantages make working with goal-oriented models an easier task, and over all, they allow to study the system behaviour against possible events scenarios, giving valuable information hardly guessed without using a simulator framework as this we present.

Using the Eclipse platform, and the framework it provides for the fast development of applications, has contributed to a graphical implementation faster than using traditional methods.
Being a highly modular platform, we have been able to set up a customized development environment quickly.

The simulation framework proposed in this work is based on modern, well known technologies, as Java, Eclipse and the BPMN notation are. This guaranteed our work to be extended easily, keeping it in the line of nowadays software development trends.

Future revisions of our framework will provide a graphical editor for the goal model edition, and also a better support of the whole set of features of the BPMN notation. Our implementation, given its experimental state, only uses a reduced set of them. Regarding the Event module, some more event definition types will be added, looking forward a more event-detailed solution simulation platform.

Bibliography

[1] Volha Bryl and Paolo Giorgini. Automated design of socio-technical systems: From organizational structure to instance level design. In Proc. of 23rd IEEE/ACM International Conference on Automated Software Engineering, 2008.

[2] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. Tropos: An agent-oriented software development methodology. Autonomous Agents and Multi-Agent Systems, 8(3):203ï¿½236, 2004.

[3] Volha Bryl, Paolo Giorgini, and John Mylopoulos. Designing sociotechnical systems: from stakeholder goals to social networks. Requir. Eng., 14(1):47-70, 2009.

[4] F Dalpiaz, R Ali, Y Asnar, V Bryl and P Giorgini. Applying Tropos to Socio-Technical System Design and Runtime Configuration. In Proceeding of the Italian workshop Dagli OGGETTI agli AGENTI (WOA08), 2008.

[5] P. Giorgini and M. Kolp and J. Mylopoulos and M. Pistore. The Tropos Methodology: an overview. Methodologies And Software Engineering For Agent Systems. Kluwer Academic Publishers, 2004.

[6] Volha Bryl and Paolo Giorgini. Self-configuring socio-technical systems: Redesign at runtime. International Transactions on Systems Science and Applications, 2(1):31-40, 2006.

[7] Pau Giner, Victoria Torres, Vicente Pelechano. Bridging the Gap between BPMN and WS-BPEL: M2M Transformations in Practice. Department of Information Systems and Computation, Technical University of Valencia, Valencia, Spain.

[8] Victoria Torres, Pau Giner and Vicente Pelechano. From BPMN to BPEL4People: A MDE Approach. IV Jornadas Científico-Técnicas en Servicios Web y SOA, 29-41, 2008,

[9] Rajdeep K. Dash, Nicholas R. Jennings, and David C. Parkes. Computational mechanism design: A call to arms. IEEE Intelligent Systems, 18(6):40 - 47, 2003.

[10] Eric Siu-Kwong Yu. Modelling strategic relationships for process reengineering. PhD thesis, Toronto, Ont., Canada, Canada, 1996.

[11] Eli Berniker. Some Principles of Sociotechnical Systems Analysis and Design. School of Business Administration Pacific Lutheran University Tacoma, Washington, October 1992.

[12] Ian Sommerville. Lecture 2: Critical Systems Engineering, from course Critical Systems Engineering. St Andrews University, Scotland, 2010.

[13] Axel Van Lamsweerde. Divergent views in goal-driven requirements engineering. In Joint Proceedings of the Sigsoft 96 Workshops, Specifications
96 , ACM, pages 252.256. Press, 1996.

[14] BPMN. Business Process Model and Notation v 1.2.
http://www.omg.org/spec/BPMN/1.2.

[15] BPMN for Eclipse. Intalios Inc.
http://www.eclipse.org/bpmn

[16] Matthias Weidlich, Gero Decker, Alexander Grokopf, and Mathias Weske. BPEL to BPMN: The Myth of a Straight-Forward Mapping. Hasso Plattner Institute, Potsdam, Germany.

[17] Eclipse Galileo. Eclipse foundation.
http://www.eclipse.org/galileo

[18] Eclipse. Eclipse Framework.
http://www.eclipse.org

[19] Eclipse GMF. Eclipse Graphical Modeling Framework. http://www.eclipse.org/gmf

[20] Minh Sang Tran Le. A Simulation Framework for Self-Reconfigurable Socio-Technical Systems. European Master in Informatics. University of Trento, Trento, October 2009.

[21] City and Borough of Juneau. Urban Avalanche Response Plan. Juneau, Alaska, February 2004.

[22] K. Yue, "What Does It Mean to Say that a Specification is Complete?", Proc. IWSSD-4, Fourth International Workshop on Software Specification and Design, Monterey, 1987

[23] Business Process Management Initiative (BPMI.org).
http://www.bpmi.org

[24] LPG Home page. Lpg-td planner. http://zeus.ing.unibs.it/lpg/.

[25] Stefan Edelkamp and Jorg Hoffmann. Pddl2.2: The language for the classical part of the 4th international planning competition. Technical Report 195, January 2004.

[26] Thomas Davenport. Process Innovation: Reengineering work through information technology. Harvard Business School Press, Boston 1993.

[27] M.H. Jansen-Vullers and M. Netjes. Business Process Simulation - A Tool Survey. Department of Technology Management, Eindhoven University of Technology, Eindhoven, The Netherlands, 2006.

[28] Web Services Business Process Execution Language. http://www.oasis-open.org

[29] Jorg Desel, Andreas Oberweis, Wolfgang Reisig, Grzegorz Rozenberg. Petri Nets and Business Process Management. Seminar at Schloss Dagstuhl, Wadern, Germany, 6-10 July 1998.

[30] S. Kumanan and O.V. Krishnaiah Chetty. Estimating product development cycle time using Petri nets. Springer, London, 2005.

[31] WS-BPEL Extension for People (BPEL4People). http://www.oracle.com/technology/tech/standards/bpel4people/index.html

Chapter 10

Appendix

A. Goal Description

Name	Description	Name	Description
G0	Resolve incidence	G29	Transport victims to hospital
G1	Keep searchers safe	G28	Victims on site assistance
G2	Recover victims	G25	Extract victims
G3	Crowd Control	G24	Follow trails
G4	Traffic Control	G27	Ask survivors
G31	Police Crowd Control	G26	Ask familiars
G32	AST Crowd Control	G23	Get victims ubication info
G7	Declare avalanche secure zone	G22	Evacuate victims
G8	Set operations area	G21	Find victims
G11	SAC aproval	G162	AST delimits area
G15	Engineers aproval	G161	CCFR delimits area
G12	SAC recommends to wait	G18	Setup Helicopters landing zone
G13	SAC recommends use of explosives	G17	Setup on site hospital
G14	SAC evaluation ok	G16	Delimit operations area
G131	SAC evaluate points	G15	Engineers aproval
G132	Explosives deployment	G134	CCFR deployes explosives
G133	SAC deployes explosives		

Table 10.1: Goal Description

B. Goal Decomposition

Goal	Decomposition	Type	Name	Description	Type
G0	G1, G2	AND	G6	G3,G4	AND
G1	G6, G7, G8	AND	G132	G133, G134	OR
G2	G21, G22	AND	G16	G161, G162	OR
G3	G31, G32	OR	G13	G131, G132	AND
G23	G26, G27	OR	G11	G12, G13, G14	OR
G7	G11, G15	AND	G22	G28, G29	AND
G8	G16, G17, G18	AND	G21	G23, G24, G25	AND

Table 10.2: Goal Decomposition

C. Actor Capability

Actor	Capability
A2	G24, G25
A3	G136, G15, G134, G161
A4	G18, G32, G162, G26, G27, G163, G164
A5	G31, G4
A6	G12, G135, G133, G14
A7	G28, G29

Table 10.3: Actor Capability

D. Associated Processes

Goal	Process	Goal	Process	Goal	Process
G0	P0	G29	P29	G133	P133
G1	P1	G28	P28	G132	P132
G2	P2	G25	P25	G134	P134
G3	P3	G24	P24	G15	P15
G4	P4	G27	P27	G14	P14
G31	P31, P312	G26	P26	G16	P16
G32	P32	G23	P23	G13	P13
G7	P7	G22	P22	G17	P17
G8	P8	G21	P21	G18	P18
G11	P11	G162	P162	G15	P15
G161	P161	G12	P12		

Table 10.4: Process Associated to goals

E. Activities' costs

Activity	Cost	Activity	Cost
AST Setup	4	CCFR delimits area	4
Inform press & media	5	Police cordons the area	5
Request family for info	12	Transports logistics	6
SAC deployes explosives	16	On site medical assistance	8
Crowd control by AST	18	Police dissuades bystanders	2
Follow trails	8	Setup Helicopters Landing Zone	6
Search for trails	5	Traffic rerouting	6
GIS analysis for explosives deployment	4	Report Final Evaluation	2
Tranport to hospital	10	Extract victims	10
Clear hospital routes	4	JPD coordination	6
IC coordination	2	Ask survivors	7
Set emergency units route	5	Inform the public	2
Hospital coordination	6	AST delimits area	9
SAC waits	9	CCFR deployes explosives	6
Engineers aproval	6	SAC ok	2
SAC evaluates points for explosives	17	CCFR evaluates points for explosives	6
Police switches off traffic lights	4	Police setups on site HQ	4

Table 10.5: Case study: activities' costs

F. Case Study solution processes

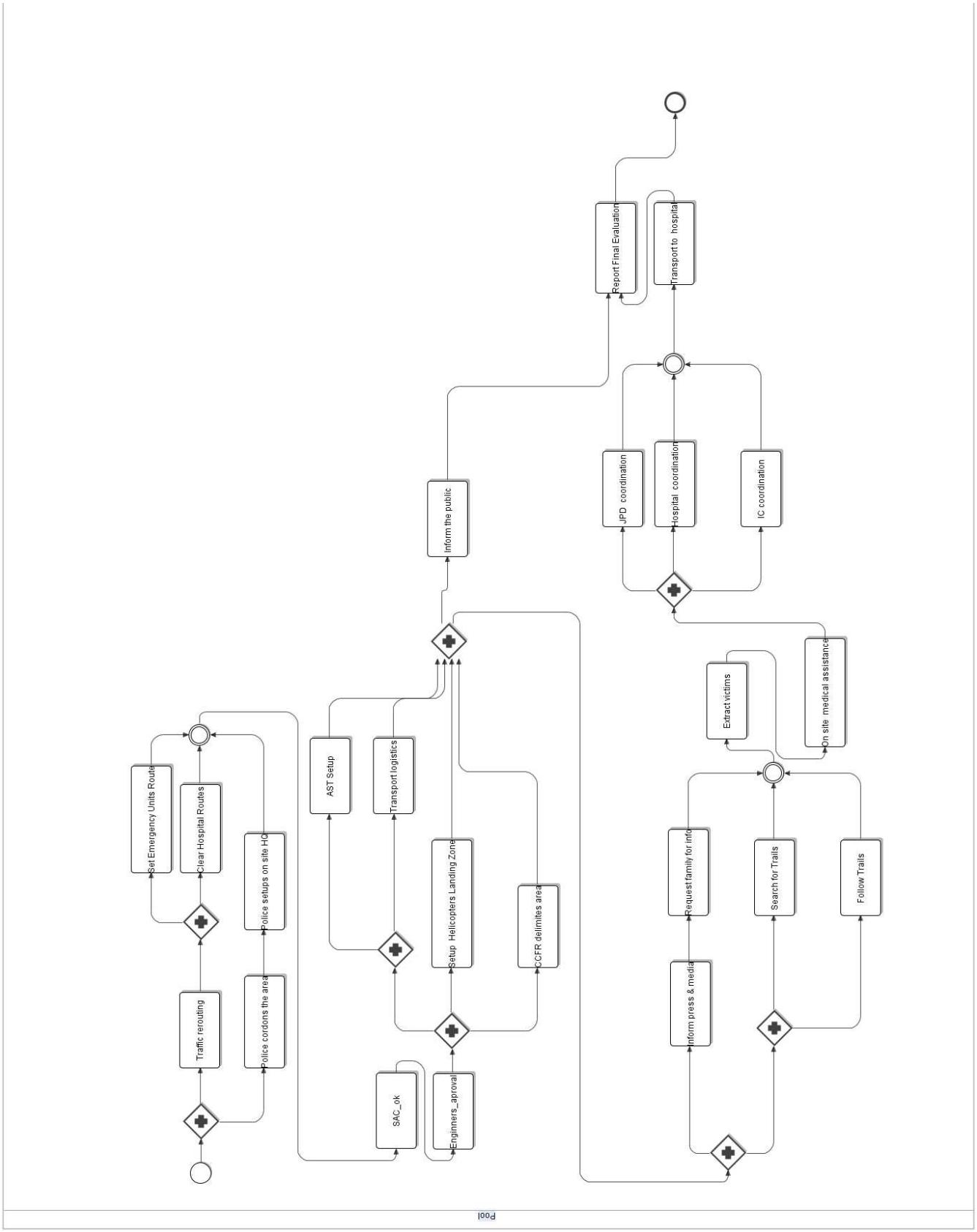


Figure 10.1: Case Study Solution S1

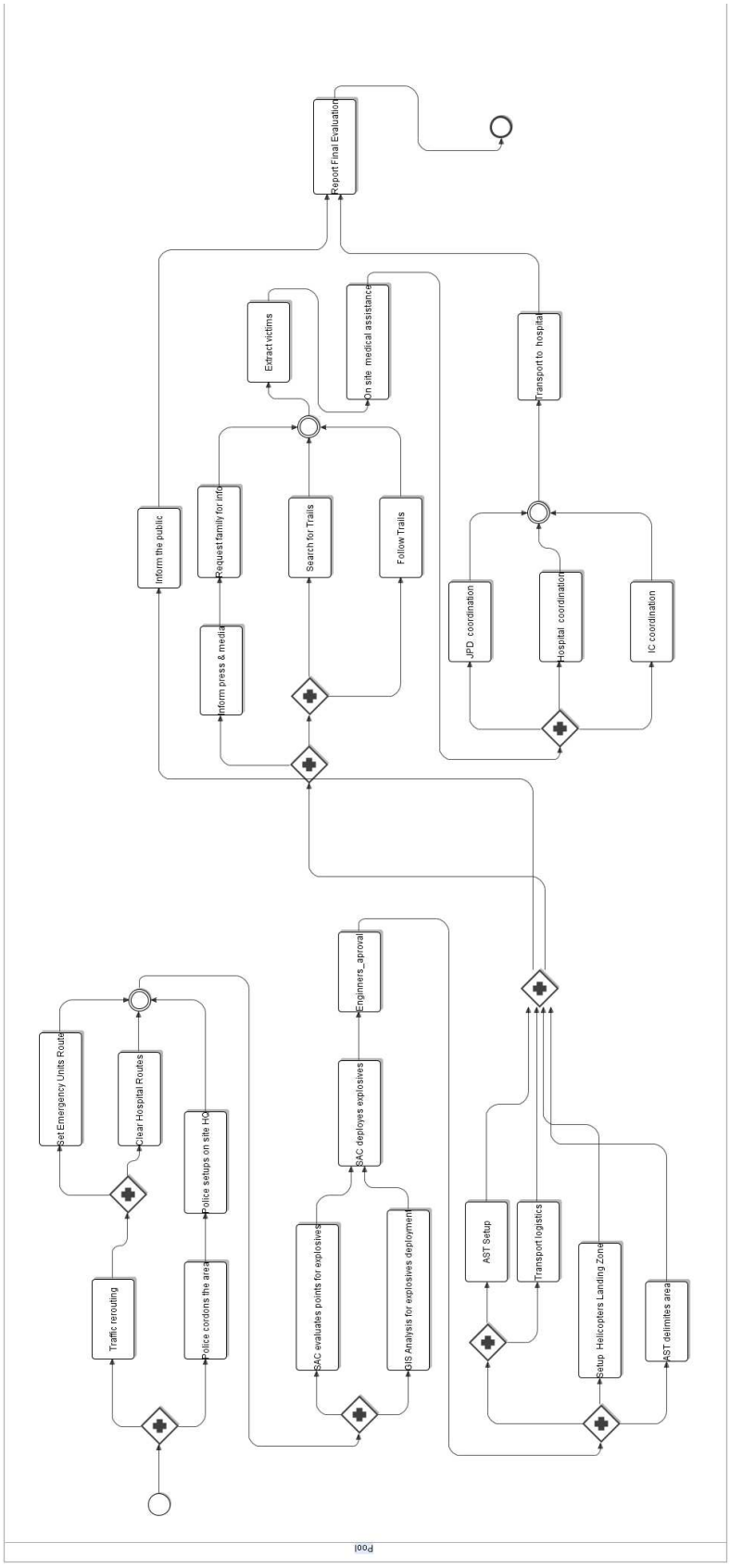


Figure 10.2: Case Study Solution S4 (selected solution)

G. Case Study Goal Model

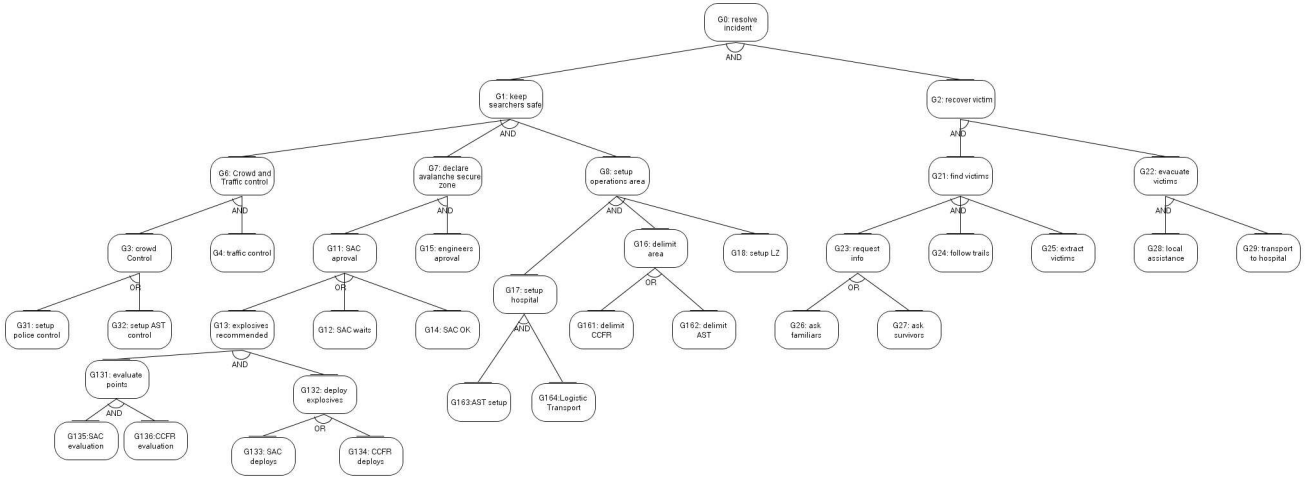


Figure 10.3: Scenario Goal Model Diagram