Università degli Studi di Trento

Corso di Laurea Magistrale in Informatica

System Identification for Adaptive Software Systems - A Case Study

Final Thesis

Laureando/Graduant:
Genci Tallabaci

Relatore/Supervisor:
Prof. John Mylopoulos

28 Marzo 2012

Abstract

In software domain, a self-adaptive software system is able to modify its behavior at run-time, in order to respond to the user requirements changes, environmental changes, but also obsolete domain assumptions. Self-adaptive software changes behavior when it is not performing what it was intended for or possibly because its functionality/performance can be improved. Recently, Control Theory and feedback loops have been exploited in adaptive software system design. In this thesis, we propose an evaluation of the System Identification process for an existing system. This process makes use of a new class of requirements, namely Awareness Requirements, concerning other requirement's status at runtime. This approach is reasoning over requirements at runtime in order to provide adaptivity to a software system. By adopting System Identification approach it is possible to model, design and implement a self-adaptive system. In this experiment we evaluated System Identification approach with an available case study (an Automated Teller Machine (ATM)).

Acknowledgements

It would not have been possible to write this thesis without the help, guidance and support of the kind people around me. I am sincerily thankful to my supervisor, prof. John Mylopoulos, for his guidance, support and great patience at all times. Thanks to Vitor E.S. Souza for his good advices, support and for all the time dedicated to answer all my questions. Finally, I would like to thank my family and friends who supported me at all times.

Contents

Contents 4

1 Introduction 5
- 1.1 Motivation . . . 6
- 1.2 Objectives . . . 7
- 1.3 Methodology . . . 7
- 1.4 Organization of the thesis . . . 8

2 Research Baseline 9
- 2.1 Goal Oriented Requirements Engineering (GORE) . . . 9
- 2.2 Awareness Requirements . . . 11
- 2.3 System Identification . . . 15

3 Related Work 19

4 Case Study 25
- 4.1 Case Study Description . . . 25
- 4.2 ATM Goal Model . . . 30

5 Awareness Requirements for the ATM system 33
- 5.1 Formalizing Awareness Requirements . . . 43

6 System Identification 47
- 6.1 Goal Oriented Requirements Model . . . 47
- 6.2 Goal Model Parameters in ATM system . . . 49

7 Conclusions and Future Work 55

Bibliography 59

Chapter 1

Introduction

To design self-adaptive computational systems, one can take inspiration from nature [6]. Living organisms have an amazing ability to adapt to a changing environment, both in the short term (phenotypic plasticity) and in the longer term (Darwinian evolution). Indeed, most complex organisms exhibit traits that seem desirable in self-adaptive software, like system monitoring (senses, awareness); short-term changes in priorities (stress reactions, sleep); system reconfiguration (muscle growth, calluses); self repair (tissue healing); intrusion detection/elimination (immune systems); and maintaining state through transitions (gradual acclimation to a new environment). The complexity of software systems is continuously growing, making researchers focus on self-adaptivity as a way of decreasing complex system's management costs. Different approaches include feedback loops for designing adaptive systems. In this way, system's requirements satisfaction will be monitored and in case of failures or possible improvement in performance, some adaptive actions will be triggered.
Requirements Engineering as a branch of Software Engineering deals with the elicitation and analysis of system requirements. Requirements Engineering makes use of goal models to analyze and model what the stakeholders are interested in. In these models hard-goals represent the functional requirements while soft-goals represent the non-functional requirements. It is very important for the system to meet its requirements, but 'what if any requirement fails?' The system requirement's need to be monitored at run-time. In case any requirement is failing the system needs to adopt some actions in order to avoid such failures. Souza et al. [10] introduced a new class of requirements, called Awareness Requirements that leads to a feedback functionality, dealing with the run-time status of the system requirements. The Awareness Requirements represent a starting point for the implementation of an adaptivity process. Their evaluation at run-time provides the indicators of requirements convergence. In other words, it is pos-

sible to measure if the functional requirements are met or the non-functional requirements have been satisfied, or rather what is the satisfaction degree for the non-functional requirements. Moreover, the System Identification [9] process determines the equations that regulate the dynamic behavior of the system. In this phase, information on how to improve indicators in case of failure will be obtained. All this information will be represented as an extended and parameterized requirements model. Some of these parameters comes directly from the model, but some of them are identified during the System Identification process.A change in any parameter is considered an adaptation action to be applied to the target system. These parameters changes will be performed by the controller.
This thesis presents the evaluation of the System Identification process on an ATM (Automated Teller Machine) system, implemented by S. Liaskos. A requirements goal model is also provided along with the implemented software.

1.1 Motivation

The first step in the software development process is requirements analysis and specification. This process is critical to the success of the development process. It encompasses all the actions that are going to determine what the objectives/needs the software has to fulfill are. The designer cannot foresee every condition under which the system has to operate or even eventual requirements modification, possible failures or environmental change. High variability software can deliver the proper functionalities in multiple ways. If any of this ways is failing, the system can switch to an alternative way. For instance, if an Automated Teller Machine (ATM) system cannot successfully dispense cash on a customer's request it should print a compensation token that a customer can withdraw in any of the banks desks. Recently, researchers have been looking into self-adaptivity as a way of lowering management costs of complex systems. One possible way to do that is by exploiting high variability of a software system in order to deliver self-repair functionalities by means of re-configuration.
System identification process determines the rules that govern the dynamic behavior of the system. For repairing system behavior after any requirements failure or possible optimization the best reconfiguration must be selected. The chosen reconfiguration is going to minimally reconfigure the system from its current reconfiguration . Extending the ATM system into a self-adaptive ATM system with self-repair capabilities, it is going to support the failure's recovery and possible improvements in performance. Self-adaptation will positively contribute to the system robustness, but on the other hand it is not guaranteed that the new reconfiguration will positively contribute in terms of quality of service. The System Identification process is making use of a new class of requirements, AwReqs, concerning the requirements' status at run-time. In the case that any requirement is

failing, the equations generated during this phase will be used by the controller for conducting parameter tuning in order for the system to meet its purpose. The new adaptive ATM system will perform an assessment of failures at runtime and recover from such failures by changing its parameters.

1.2 Objectives

The objective of this thesis is to evaluate the proposal introduced by Souza et al. [9] using the Automated Teller Machine (ATM) case study. The ATM software should support a computerized banking network to fulfill its functionalities. An automatic teller machine accepts a cash car, interacts with the user, communicates with the bank computer to carry out the transaction, dispenses cash, prints receipts and so forth. During the evaluation of the Awareness Requirements, it will be possible to obtain information about what requirements are satisfied, denied or canceled. The System Identification process uses all the information gathered in the Awareness Requirements evaluation and will produce fundamental information, the equations that regulates the dynamic behavior of the system. These equations will be used by the controller in order to decide how to perform adaptive actions, by means of a parameter change. The objective of this thesis is to make use of the ATM requirements model along with the information gathered from the Awareness Requirements evaluation in order to determine the equations that regulates the dynamic behavior of the system. During the System Identification evaluation will be identified information on what adaptation actions are needed. This process is important in order to decide not only what should be the possible recovering actions but even more useful in order to prevent from failures.

1.3 Methodology

By applying a requirements engineering perspective in this thesis, we described the application of System Identification process on an existing system. The system considered in this laboratory demonstration is an Automated Teller Machine (ATM) simulation. By applying this approach to the ATM simulation it is possible to transform the exiting software system into an adaptive system with self-repair functionality. The system used in this demonstration is implemented by S. Liaskos. A requirements goal model is build up from the models provided from Liaskos and Wang. During System Identification it will be provided with a new set of parameters capturing different behaviors of the system. At the end of the process, the system requirements model will be extended and parameterized by a new set of parameters. Fundamental information about self repair capability of the system can be inferred by applying the notion of differential relations between such parameters and the indicators of requirements convergence. The

indicators of requirements convergence will be provided from the evaluation of Awareness Requirements at run-time. The feedback loop needs to compare the reference input (system requirements) against the measured output (measurable characteristic of the target system). In other words it will check whether the system requirements were met. Awareness Requirements (AwReqs) will be identified from the requirements goal model. The elicitation process uses the goal model by exploiting critical goals/tasks, metricizing soft-goals and also considering the goal refinement. The requirements should be written in patterns and then formalized using OCL_{TM} [7]. In this way they can be directly monitored by a framework and by using OCL_{TM} it is possible to talk about the requirements status at different time points. The process of conducting system adaptation is described in the following sections.

1.4 Organization of the thesis

The structure of the thesis : section 2 summarizes research baselines used in this proposal; section 3 compares the proposal to the related work; section 4 represents the original case study; section 5 describes the Awareness Requirements, describing also a formalization patterns; section 6 presents the System Identification process for the proposed case study and the last section presents the conclusions.

Chapter 2

Research Baseline

This section highlights the research background on Goal-Oriented Requirements Engineering (2.1), Awareness Requirements (2.2) and a systematic process for System Identification (2.3).

2.1 Goal Oriented Requirements Engineering (GORE)

Goals represents an important component in the requirements engineering (RE) process. They capture the various objectives a system has to achieve. Goal-Oriented requirements engineering uses goals for eliciting, specifying and analyzing requirements. Figure 2.1 shows an example for an Ambulance Dispatch System (ADS) goal model [9].

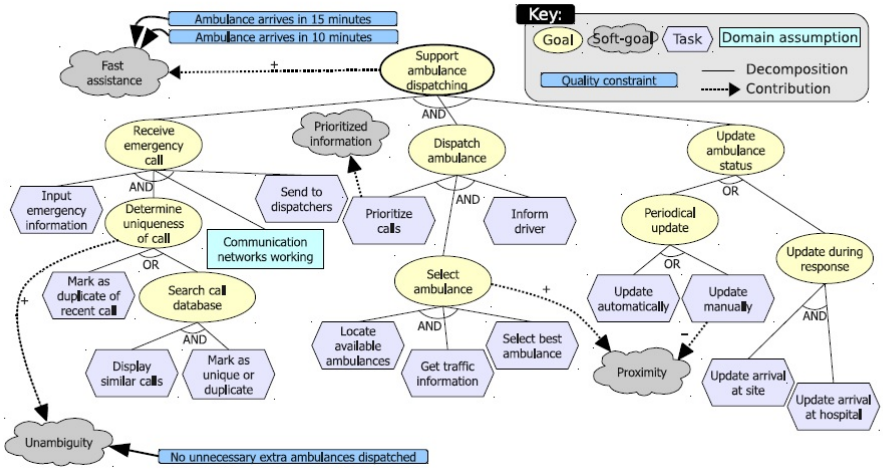

Figure 2.1: Goal Model for an Ambulance Dispatch System [3]

Goal-Oriented Requirements Engineering (GORE) [3] makes use of fundamental notions like, goals, softgoals, domain assumptions (DAs) and quality constraints (QCs). Goals can be functional (hard goals) dealing with the underlying services that the system is expected to deliver and non-functional (softgoals) refer to the expected quality of service, such as security, usability, and so forth. Goals can be refined into sub-goal that specify how the goals are fulfilled. If a goal G is AND decomposed in G1...Gn then all of the sub-goals must be satisfied in order for G to be satisfied.If a goal G is OR decomposed in G1..Gn then at least one of the sub-goals must be satisfied in order for goal G to be satisfied. For example, to receive an emergency call, one has to input its information, determine its uniqueness (are there other calls from the same emergency?) and send it to the dispatcher, all of this on the assumption that Communication networks [are] working. On the other hand, periodic update of an ambulances status can be performed either manually or automatically. The refinements continue until they reach a level of granularity such that for each task there is an actor able to perform it. Soft-goals represent non-functional requirements, meaning what is expected in terms of quality of service. Goals and tasks can contribute positively or negatively to the satisfaction of sub-goals, but for the soft-goal there is no clear-cut criteria for deciding whether a softgoal is satisfied or not. Softgoal satisfaction can be estimated by using qualitative contribution links that propagate either satisfaction or denial and will have four levels of contributions:
break (- -), hurt (-), help (+) and make (+ +).
E.g. selecting an ambulance using the software system contributes positively to the proximity of the ambulance to the emergency site, while using a manual ambulance status update, rather than automatic, contributes negatively to the satisfaction of the proximity softgoal. Quality Constraints (QCs) represent measurable criteria for evaluating whether a soft-goal is satisfied. In terms of refinement, a soft-goal can be refined into quality constraints, which are measurable metrics to evaluate how the system is achieving the soft-goal. In the example shown in Figure 2.1, unambiguity is measured by the number of times two ambulances are dispatched to the same location. On the other hand, fast assistance is refined into two QCs: ambulances arriving within 10 minutes to the emergency site and ambulances arriving within 15 minutes to the emergency site. Finally, Domain Assumptions (DAs) indicate states of the world, what is assumed to be true in order for the system to work. E.g. in the ADS system, it is assumed that communications networks (phone, internet connection, etc) are available and functional. If such assumption turn to be false, its parent goal Receive Emergency Call will not be satisfied.

2.2 Awareness Requirements

In Control Theory [1], a reference input represents the desired value of measured outputs. Figure 2.2 shows the various elements in a Control System.

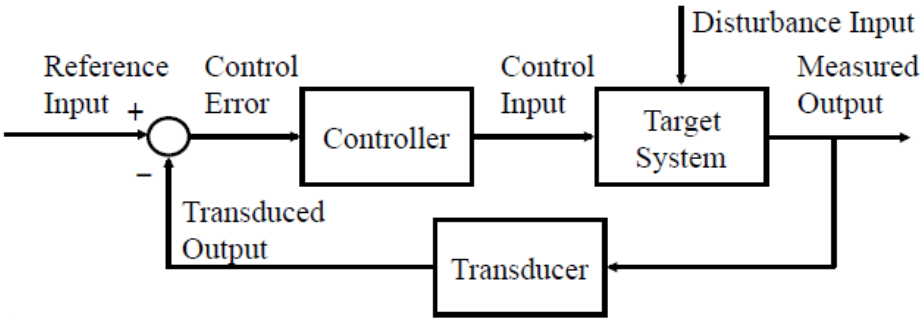

Figure 2.2: Elements of a Control System [1]

The measured output represents measurable characteristics of the system. Control input is manipulated in order to affect the output. The transducer translates the measured output in order to check if the measured outputs meet the reference input. The control error shows the difference between what is expected and what is measured on the system. If the output tells that the reference input is not achieved then the controller adjusts the control input in order to meet the measured output. At the requirements level, the reference input is represented by the system requirements, including not only basic requirements like goals, task, DAs but also AwReqs.

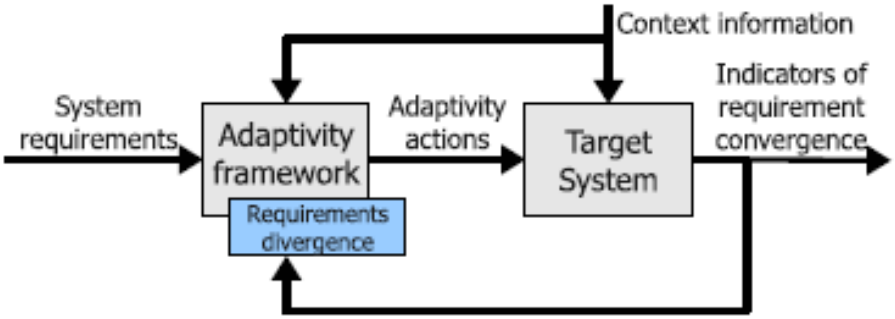

Figure 2.3: Adaptive system as control system [11]

Figure 2.3 shows a view of an adaptive system as a control system [11]. The measured output shows the requirements convergence, meaning the failure rate

of any requirement. If the output shows that the requirement has failed then the controller will perform some parameter change in order to bring the system to an acceptable state. The reference input is represented by the requirements, the measured output consists of indicators of requirements convergence, meaning that it is important to measure if functional requirements are met at runtime. AwReqs (Awareness Requirements) are introduced as a new class of requirements that talk about the success/failure of other requirements. AwReqs talk about the states requirements can assume during their execution at runtime. Figure 2.4 shows these states that can be assumed by goals, tasks, DAs, QCs and AwReqs themselves. When an actor starts to pursue a requirement, its result is yet Undecided. Eventually, the requirement will either have Succeeded, or Failed. For goals and tasks, there is also a Canceled state.

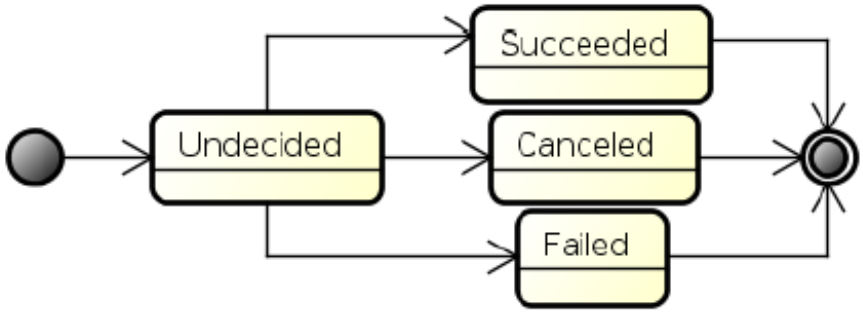

Figure 2.4: States of the requirements

Different types of AwReqs have been identified [10] such as **regular**, **aggregate**, **delta** and **trend** AwReqs.

- The simplest form of AwReq namely **regular AwReq** refers to requirements that should never fail. If a task needs to be executed, an instance of this task will be created and a 'never fail' constraint has to be checked. When a goal needs to be fulfilled, an instance of the goal is created and the constraint has to be checked. Instead, for the domain assumptions and quality constraints, their instances are created when their parent goal needs to be fulfilled. In this context their truth/falsity needs to be verified.

- **Aggregate AwReq** refers to instances of other another requirement and imposes constraints on their success/failure rate. Aggregate AwReq can also precisely tell the period of time to be considered. The verification frequency is an optional parameter. Some other aggregate AwReq can also specify the success/failure rate of the requirements in terms of min/max values allowed. Also, different requirements can be combined in order to

compare the success rates of requirements. This can be helpful at runtime when based on the desired property changes need to be performed in order to satisfy a goal/softgoal.

- **Delta AwReq** can be used to underline the acceptable thresholds for the requirement satisfaction, such as achievement time. Another pattern specifies that it is possible to talk not only about success/failure of requirements but also about the change of state, meaning that it is not important that a requirement will succeed/fail but in this case the requirement would not be in the Undecided state for more than a specified amount of time.

- **Trend AwReq** compares the success rates over periods of time in order to keep track of the evolving of the rates over time.

- **Meta AwReq** are AwReq that talk about other AwReqs. They are useful in system reconciliation actions. It is also possible to model meta-meta-AwReq referring to a meta-AwReq. Different levels of meta-AwReqs can be created by referring the requirements from the directly below level of requirements.

Like other types of requirements, AwReqs must be systematically elicited. Since they refer to the success/failure of other requirements, their elicitation takes place after the basic requirements have been elicited and the goal model have been constructed. There are several sources of AwReqs.

- One obvious source consists of the goals that are critical for the system-to-be to fulfill its purpose. If the aim is to create a robust and resilient system, then there have to be goals/tasks in the model that are to be achieved/executed at a consistently high level of success. Such a subset of critical goals can be identified in the process and AwReqs specifying the precise achievement rates that are required for these goals will be attached to them. This process can be viewed as the operationalization of high-level non-functional requirements (NFRs) such as Robustness, Dependability, etc.

- Similarly, AwReqs are applied to DAs that are critical for the system.

- Also AwReqs can be derived from softgoals. Quality Constraints (QCs) are identified in order to metricizes high-level softgoals. This way the system is able to quantitatively evaluate at runtime whether the quality requirements are met over large numbers of process instances and make appropriate adjustments if they are not.

- In Tropos and other variations of goal modeling notation, alternatives introduced by OR-decomposed goals are frequently evaluated with respect to

certain softgoals. The evaluations are qualitative and show whether alternatives contribute positively or negatively to softgoals. Softgoals are refined into QCs and the qualitative contribution links are removed. AwReqs can be used as a tool to make sure that good alternatives are still preferred over bad ones.

AwReqs have been introduced as requirements that refer to other requirements in terms of their success/failure. Before monitoring the requirements another important task is to formalize such requirements in order to facilitate the successive steps of the adaptivity process, like elicitation/analysis and also for improving communication between analysts and designers. Requirements can be specified in a variant of the Object Constraints Language (OCL), called OCL_{TM} - meaning OCL with temporal message logic. With OCL_{TM} it is possible to consider particular requirements in different periods of time. In order to formalize such requirements in OCL_{TM}, it is helpful to represent systems requirements in a UML class diagram. OCL_{TM} constraints refer to the class of requirements and their instances, attributes and methods.

2.3 System Identification

In Control Theory, when the measured system output is not matching the reference input (system requirements), the controller will perform some parameter change in order to bring the system in an acceptable state. *System identification* is the process that captures the effects of the control input on the measured system outputs. It aims to provide a systematic process for identifying the relations that regulate the behavior of the system. A common approach is to start from a partial knowledge of the system behavior and try to identify what are the relation holding between parameters and indicators, without going into details of what happens in the system. The system identification process proposed from Souza et al. [9] uses a Requirements Engineering (RE) perspective and assume that the requirement model of the system is available. The process is characterized by several steps and in order to pursue a more precise process,process steps can be applied iteratively. The input of the process is a requirements goal model and by applying the various steps will be possible to obtain more information on the system behavior. The first step of this process concerns with the **indicators identification**. The indicators represent the monitored values that can be used as feedback. Not all the possible variables will be monitored, but a subset of the most relevant ones has to be identified and after that monitored. Indicators are important values that provide information related to the performance of the adaptive system. The *AwReqs* represent requirements that refer to other requirements success/failure, and by stating the desired success rate for the goals/tasks and also the satisfaction criteria for the NFRs, they will be used as a starting point for the indicator identification in the system. The second step refers to the **parameters identification**. As mentioned before, in the requirements model the indicators are represented by the AwReqs. In this step all the possible variations that can affect the indicators will be explored. After that, these parameters can be exploited in order to improve the performance of the system, react to possible changes of requirements, react to possible failures and so forth. *Variation Points* (**VP**) and *Control Variables* (**CV**) represent the possible variations. These are the parameters that the system will monitor and manipulate. Also, an important phase of this step concerns parameter elicitation. In this phase, it can be helpful to search for possible reconfiguration for potential AwReqs failures. A **VP** can be identified by looking at the OR-decompositions in the goal model, which represent intentional variability in the system. Choosing a different path at a VP in order to fulfill a goal is a possible way for system re-configuration. **CVs** have an important role in system re-configuration. They can be applied to goals/tasks/domain assumptions and are used as abstractions over families of similiar goals. They represent other possible alternatives for goals, tasks or DAs, and parameters will be identified from such abstractions. After that, the requirements model will be enriched with these parameters. One of the most

important aspects on having CVs in the requirements model is represented by the possibility of having a large but slightly different alternatives represented in a compact fashion in the model. For the CV elicitation, the requirements engineer should pay particular attention during the goal model refinement. In summary, the system identification process needs as its input a requirement goal model and will provide as its output a set of qualitative differential equations. VPs and CVs will be attached to the model. The next step concerns the **differential relations identification**, in which the requirements engineer will capture every possible dependency between the indicators and the parameters. There are some heuristics that can be helpful with this task. In the case of VPs, softgoal contribution links capture this dependency. Another heuristic is to link indicators with the parameters that appear in the subtrees of the nodes to which the indicators are linked. The key idea is that if any parameter is linked to the goal G's subtree, the change in the parameter will also affect the subtree.

In order to underline how parameters affects the indicators, an indicator is defined as a function of some parameter.

$$indicator = f(parameter)$$

By using the derivative, it is possible to say how changes of parameters are propagated to indicators. If the derivative is positive, by increasing the value of the parameter the indicator will increase (e.g. success rate of some goal will increase), and vice versa by decreasing the parameters value, the success rate of any goal also will decrease.

Using the Leibnizs notation

$$\frac{\triangle indicator}{\triangle parameter} > 0$$

As evidenced by the fraction, there is an increase-increase relation if the derivative fraction (i.e. $\triangle x/\triangle y > 0$) is positive and the analogous decrease-decrease relation is also inferred in the case of a positive derivative fraction. The opposite decrease-increase relation can be inferred in the case of a negative derivative fraction. In order to improve writability a more simplified linearized notation is used. Because the parameters assume only discrete values, we used $\triangle x/\triangle y$ instead of *dx/dy*. The notification will be

$$\triangle\left(\langle indicator\rangle / parameter\right) > 0$$

Also the concept of *landmarks values* is used in order to specify an interval in which the relation between control variable and an indicators holds, meaning that out of this interval increasing the parameter will not increase the success rate of the desired goal.

$$\triangle\left(\langle indicator\rangle / parameter\right)[0, landmark\ value] > 0$$

The last step of the process concerns the **refinement** of the relations. Once the relations have been identified, the parameters that refer to the same indicator can be compared or combined. The requirements engineer sometimes may need to compare various relations and evaluate which of these strategies is the best reconfiguration. These adaptation strategies are stated by the notification explained before, what makes easier the evaluation of different alternatives. Also if more than one parameter affects positively some indicator, these parameters can also be combined. When the indicator/parameter relations need to be combined for the evaluation of the possible alternatives, it is very important to consider only relations concerning the current system reconfiguration. The system identification process can be applied iteratively. After each possible iteration the model will be enriched by an updated set of parameter-indicator relations.

Chapter 3

Related Work

A definition of Self Adaptive Software was provided in a DARPA Broad Agency Announcement on Self Adaptive Software (BAA-98-12) in December of 1997: *"Self Adaptive Software evaluates its own behavior and changes behavior when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible."* [4] In such software systems there are several ways of fulfilling their purpose and the effective changes will be done at runtime. The software should have some functionalities in order to evaluate its performance and behavior. It must be able to reconfigure its operations in order to improve its performance if needed. In [8], Salehie et al. proposed how the adaptation can/need to be applied. They introduced different approaches :

- *Static/Dynamic Decision-Making.* This approach deals with how the decision process can be constructed and modified. In the static way the decision process is hard-coded (e.g. like a decision tree) and modifying it requires recompilation for some of the systems components. In the dynamic decision making, policies and rules are defined externally so their modification can be performed during run-time to create/change behavior for both functional or non-functional requirements.

- *External/Internal Adaptation.* The adaptation approach is divided in two categories with respect to the adaptation mechanism and the application logic. The figure 3.1 describes both the internal/external approaches that can be adopted for Self-adaptive software systems.

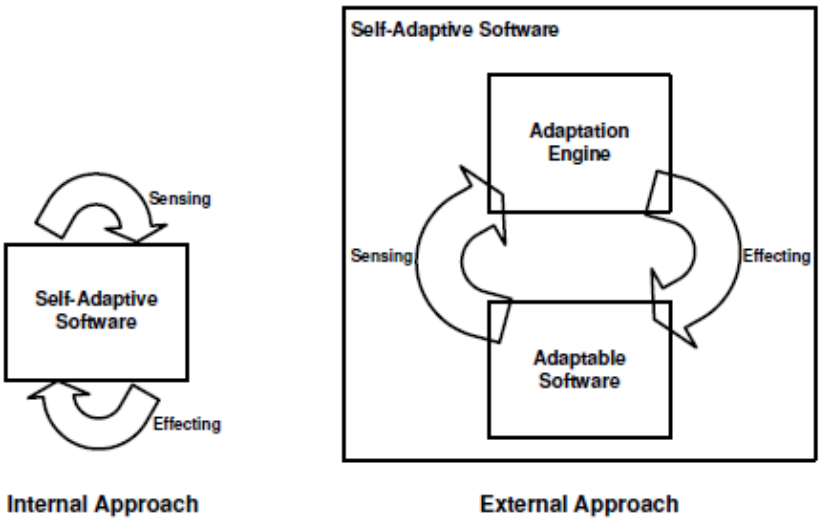

Figure 3.1: Internal and External Approach for Self-Adaptive Software System [8]

  - *Internal Approach* is based on the programming language features, such as conditional expressions, parameterizations and exceptions. This approach can be useful for local adaptation (e.g. exception handling) and needs global information about the system. It can be realized by extending existing programming language features or by introducing a new adaptation language.
  - *External Approach* uses an external adaptation engine containing adaptation processes. As described in the figure 3.1, the adaptive software system consists of an adaptation engine and an adaptable software. The external engine implements adaptation logic by using a policy engine and by the support of middleware. The advantage of the external approach consists on the fact that the adaptation engine can be reused and some adaptation processes can be applied also to various applications.

- *Making/Achieving Adaptation.* The first strategy consists of engineering self-adaptivity into the system at the developing phase. The second one is to achieve self-adaptivity through adaptive learning. *Making* has an implied software engineering view to engineer adaptivity into the software system. *Achieving* has an implied artificial intelligence and adaptive learning view to achieve adaptive behavior.

Some proposals for adaptive system design use a goal-oriented approach for system requirement specification. KAOS has been used to specify adaptive system requirements. It also included runtime monitors of system requirements derived form the KAOS specification. Feather et al. [2] discussed adaptation semantics in adaptive systems and implicitly specified adaptation semantics with event-condition action rules. Their approach focused on monitoring requirements conditions that trigger adaptation. The KAOS approach also uses a graphical representation for specifying adaptation semantics. Zhang and Chang [14] use KAOS goal models to explicitly specify the adaptation semantics by introducing an extension to LTL called Adapt operator-extended LTL (A-LTL). They extended LTL with the adapt operator ($\xrightarrow{\Omega}$) in order to specify adaptation behavior. This operator is used as $A \xrightarrow{\Omega} B$, indicating that the adaptive software initially satisfies A and in a later state, it stops being obligated to satisfy A and starts to satisfy B. The A-LTL specifications of the adaptation semantics can be used in the specification of adaptive systems and can be checked for consistency, correctness, dynamic insertion of adaptive code into an adaptive system, etc. Zhang and Chang [14] identified three adaptation semantics:

1. *One-point adaptation.* The one-point adaptation is characterized by the system adapting from the source program to the target program at a specific point in execution. In this case the system adapts after it receives an adaptation request.

2. *Guided adaptation.* The guided adaptation is characterized by the system restricting the source program and retaining its source program behavior before a safe state is reached. The system then converts via an adaptation to the target program.

3. *Overlap adaptation.* The overlap adaptation is characterized by an overlap of the source and target programs. The system monitors the input and eventually stops the source program when appropriate, where the target program may have been executing before the source program stops executing.

Lapouchnian used a hybrid goal-oriented modeling technique [5] for goal modeling and system configuration. In his approach the system features are defined along with the possible adaptations and determine when to perform such adaptation actions.

Wang and Mylopoulos [13] proposed a framework that exploits high-variability in software systems for delivering self-adaptive capabilities through reconfiguration. The framework

1. monitors systems operations for failures
2. diagnoses root causes when failures occurs
3. identifies the actual failure when multiple diagnoses explain monitored data
4. generates a system reconfiguration for avoiding the failed operation

Goals and tasks are associated with preconditions, postconditions and monitoring switches. Preconditions and postconditions (effects) are propositional formulae in Conjunctive Normal Form (CNF). Monitoring switches are Boolean flags that can be switched on/off to indicate whether the corresponding requirement is monitored. Also they associate softgoals with three priority levels: high, medium or low. This prioritization is used during the reconfiguration selection, meaning that the reconfiguration that contributes most positively to softgoals of higher priorities is going to be selected. The framework contains *monitoring*, *diagnostic*, *reconfiguration* and *execution* components.

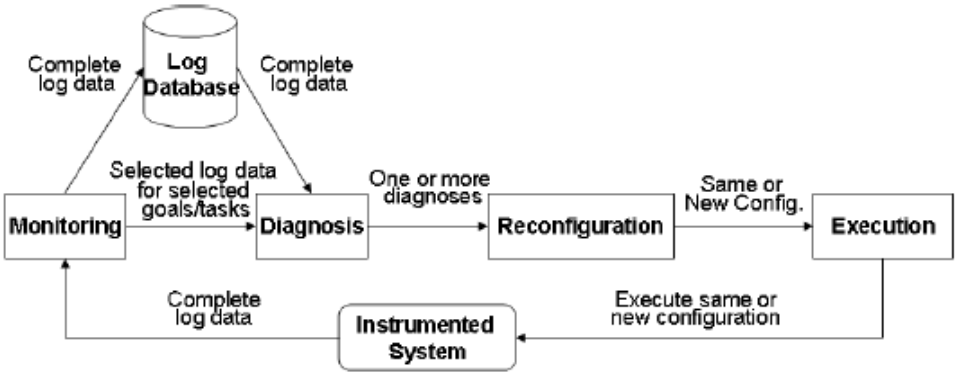

Figure 3.2: An Architectural View [13]

Figure 3.2 shows an overview of the architecture. The monitoring component monitors requirements at different levels of granularity, collects the complete log data and saves them to a Log Database. The selected data are passed to the diagnostic component. For reducing diagnostic overhead a subset of the complete log data are passed to the diagnostic component. This component analyzes the

information contained to infer if some requirement has failed. The diagnosing problem is transformed into a SAT problem by encoding goal model relations and log data into a propositional formula that is satisfied if and only if there is a diagnosis. If a denial is found, the component returns either a single diagnosis or multiple competing diagnoses each of them identifying different task/goal denials. Than the reconfiguration component computes the best reconfiguration that is free of failures. This component uses different algorithms, based on performance. It can use an algorithm for finding the most optimal global configuration if the performance is not important or can use an algorithm for fast failure recovery when performance matters. The appropriate configuration is after that passed to the execution component. If the configuration is the same as the behavior of the system, then no modifications are made. Otherwise, the component executes any necessary action to bring the system to a consistent state. In this approach, the basis for the monitoring, diagnosing and adaptation is represented by system requirements.

In the approach introduced by Souza et al. [9], techniques from Control Theory have been exploited in order to design systems that can use feedback loops as a adaptation tool. This approach is built from Requirements Engineering landscape, introducing a new class of requirements [10]. Such requirements lead to feedback loops, important for the adaptations strategy. *AwReqs*, the new class of requirements, refers to the runtime success/failure of other requirements. *Qualitative Reasoning* is used in this approach for introducing a systematic way of the identification of configuration parameters and qualitative relations among them. A change in each of these parameters represents a possible re-configuration to be adopted by the system for the achievement of the objectives.

Chapter 4

Case Study

The main objective for an ATM (Automated Teller Machine) is to support computerized banking network.

4.1 Case Study Description

The software controls a simulated ATM having a magnetic stripe reader for reading an ATM card, a keyboard and display for interaction with the customer, a slot for depositing envelopes, a dispenser for cash (in multiples of 20 euro), a printer for printing customer receipts, and a key-operated switch to allow an operator to start or stop the machine. The ATM also communicates with the bank's computer by using a communication link. The ATM will service one customer at a time. A customer will be required to insert an ATM card and enter a personal identification number (PIN) - both of which will be sent to the bank for validation as part of each transaction. The customer will then be able to perform one or more transactions. The card will be retained in the machine until the customer indicates that he/she desires no further transactions, at which point it will be returned - except as noted below.

Figure 4.1 shows the action **a13**: *select transaction*, in order for the goal **g12**: *conduct transaction* to be fulfilled the transaction type needs to be selected and then the action will be performed. Figure 4.2 shows the list of transactions that a customer can perform after authenticating himself by inserting a card and typing the personal PIN.

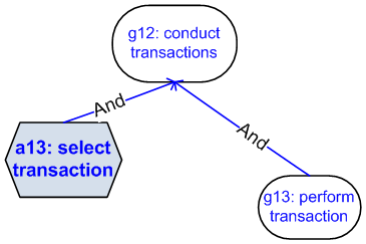

Figure 4.1: Conduct transaction

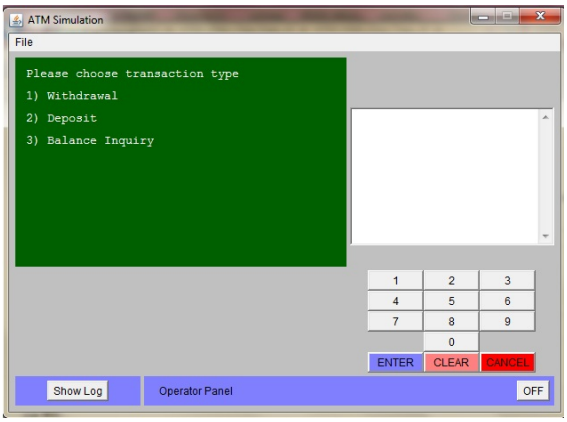

Figure 4.2: Select transaction type

The services provided to the customer are the following :

1. A customer must be able to make a cash withdrawal from any suitable account linked to the card, in multiples of 20.00 euro. Before the withdrawal an approval must be obtained from the bank by checking the customer's account. In order to fulfill **g14**: withdraw, first of all the system needs to *get customer withdrawal specifics*. It means that the system needs the information about the withdrawal account, from which to withdraw and it needs the customer to insert the desired withdrawal amount. Figure 4.8 shows the AND-decomposition focusing on the **a14**: *get withdrawal account* in the goal model. Figure 4.4 shows the account from which to withdraw.

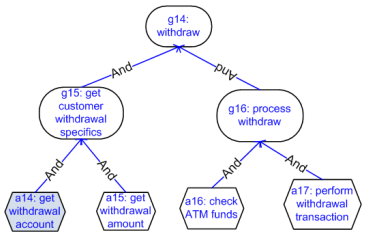

Figure 4.3: Get withdrawal account

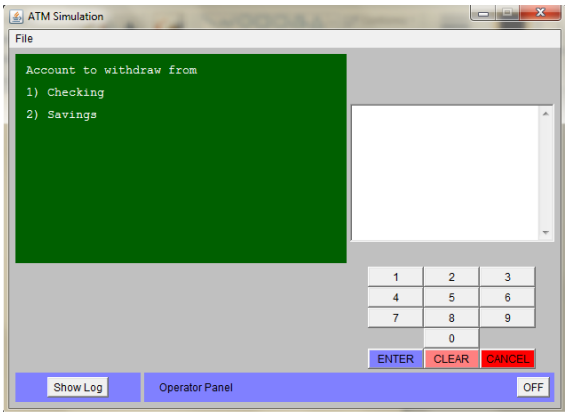

Figure 4.4: Select account to withdraw

Figure 4.5 shows the And-decomposition mentioned above by focusing on the **a15**: textitget withdrawal account and figure 4.6 gives a snapshot of how it is implemented in the system.

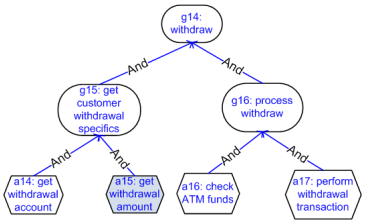

Figure 4.5: Get withdrawal account

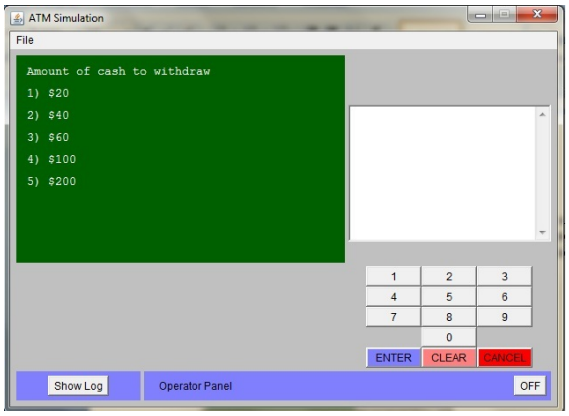

Figure 4.6: Select withdrawal amount

2. A customer must be able to make a deposit to any account linked to the card. The customer may use cash but also a check inserted in an envelope. The customer will enter the amount of the deposit into the ATM. The envelope will be removed from the machine by an operator that will proceed after that to a manual verification. Approval must be obtained from the bank before physically accepting the envelope. Figure 4.7 shows the task **a20** for the *account selection* in the goal model and the Figure 4.8 shows the account selection implementation for a deposit transaction.

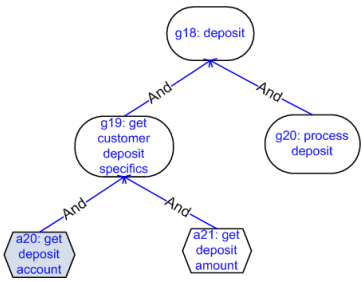

Figure 4.7: Get deposit account

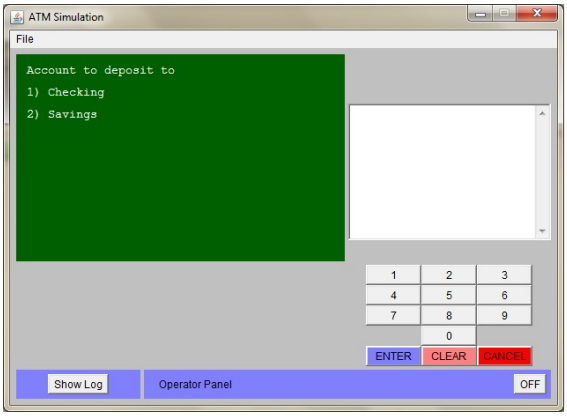

Figure 4.8: Insert deposit account

Figure 4.9, on the other hand shows the part of the model concerning the insertion of the amount to deposit and figure 4.10 gives a implementation snapshot of this action.

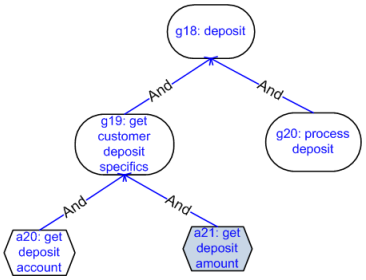

Figure 4.9: Get deposit amount

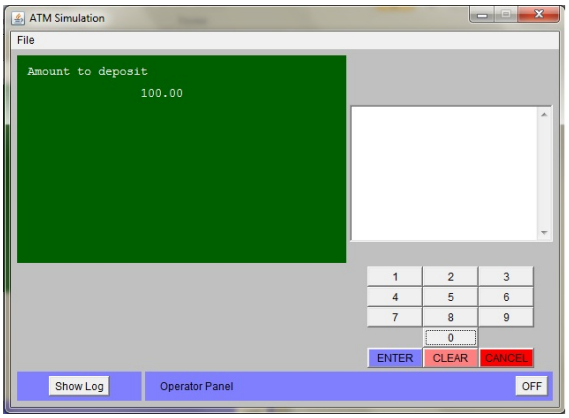

Figure 4.10: Insert deposit account

3. A customer also must be able to make a balance inquiry of any account linked to the card. Figure 4.11 shows the account selection action in the goal model and the figure 4.12 shows the inquiry selection action as it is implemented.

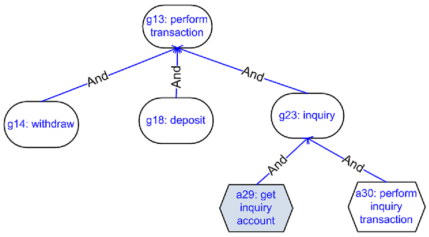

Figure 4.11: Balance Inquiry Account selection

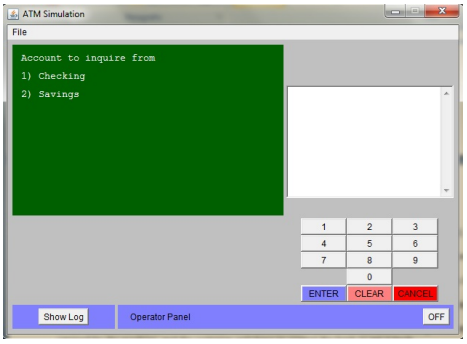

Figure 4.12: Inquiry account selection

Figure 4.13 gives a view of the task **a30**: *perform inquiry transaction* of the goal model. Again, figure 4.14 shows how the balance inquiry transaction is displayed at the system. The ATM will communicate each transaction to the bank and obtain verification that it was allowed by the bank. If the bank determines that the customer's PIN is invalid, the customer will be required to re-enter the PIN before a transaction can proceed. If the customer is unable to successfully enter the PIN after three tries, the card will be permanently retained by the machine, and the customer will have to

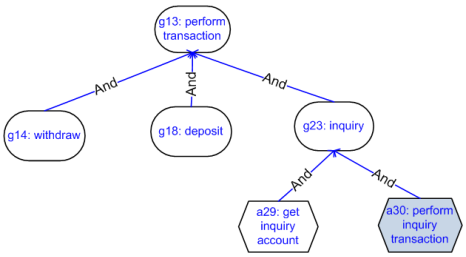

Figure 4.13: Perform inquiry transaction

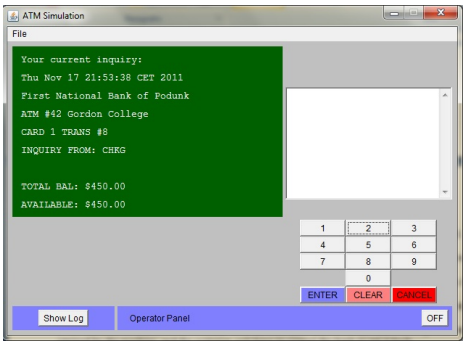

Figure 4.14: Balance Inquiry display

contact the bank to get it back. If a transaction fails for any reason other than an invalid PIN, the ATM will display an explanation of the problem, and will then ask the customer whether he/she wants to do another transaction. The ATM will provide the customer with a printed receipt for each successful transaction, showing the date, time, machine location, type of transaction, account(s), amount, and total and available balance(s) of the affected account. Figure 4.15 shows an example of a withdrawal receipt after a withdrawal transaction. The ATM will have a an operator panel

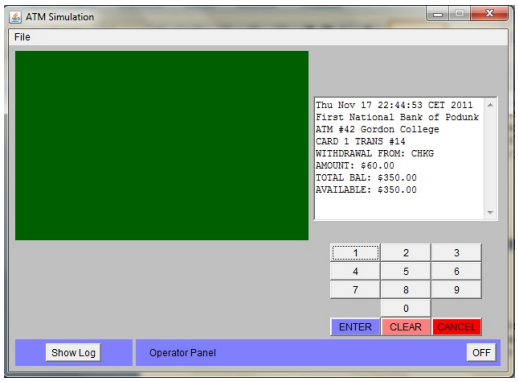

Figure 4.15: Withdrawal print receipt

with a key-operated switch (located on the "inside the bank" side) that will allow an operator to start and stop the servicing of customers. When the switch is moved to the "off" position, the machine will shut down, so that the operator may remove deposit envelopes and reload the machine with cash, blank receipts, etc. The operator will be required to verify and enter the total cash on hand before starting the system.

4.2 ATM Goal Model

The goal model is provided from the system developers. This section gives a brief description of the goal model. As showed in the figure 4.16, the main goal for the system is to **g1**- *provide ATM*. The main goal is AND-decomposed in three subgoals : **g2**-*operator starts ATM*, **g3**-*serve customers* and **g4**-*operator shutdown ATM*. Than **g4** is and AND-decomposed in task **a1**-*turn on ATM*, by using a switch located on the inside the bank side, in **g3**-*detect cash amount*, **a4**-*set up connection to bank* and in **a5**-*make ATM available*. The goal **g3** is OR-decomposed in **a2**-*use cash sensor*, where the operator uses a device for detecting the cash amount, and in **a3**-*use operator entry*, a manual verification performed by the operator. The softgoal **reduce operator workload** will have positive/negative contribution based on how the goal **g3** is fulfilled. The task **a2** helps its satisfaction but if the task **a3** is executed than it will hurt the softgoal satisfaction. On the other hand, the goal **g4**-*serve customers* is AND-decomposed in **g5**-*authenticate customer* and in **g11**-*conduct ATM session*. The goal **g5** will be AND-decomposed in **g6**-*get card info* and in the task **a9**-*check valid card*. The goal **g6** will be subsequently refined providing several alternatives for its achievement. As shown in figure 4.16, this part of goal model has a high rate of contribution links between goals and softgoals of the system. If the goal **g8**-*customer enters card number* is achieved, then the **efficiency** softgoal will be denied. Otherwise, if the task **a6**-*customer inserts card* is executed, then it will make the **efficiency** softgoal. On the other hand, if the task **a6**-*customer inserts card* is executed, it will make the **reduce customer workload** or if the goal **g8**-*customer enters card number* is achieved, the softgoal **reduce customer workload** will be denied. In terms of usability, if the task **a10**-*enter PIN from keypad* is executed, then it will help the **usability** softgoal, otherwise executing the task **a11**-*enter PIN from two-key keypad* will negatively contribute to the **usability** softgoal. Further on, the goal **g9**-*authenticate customers with PIN* is AND-decomposed into the goal **g10**-*get PIN* and in the task **a12**-*validate PIN*. If this task is executed, the softgoal **security** will be satisfied.

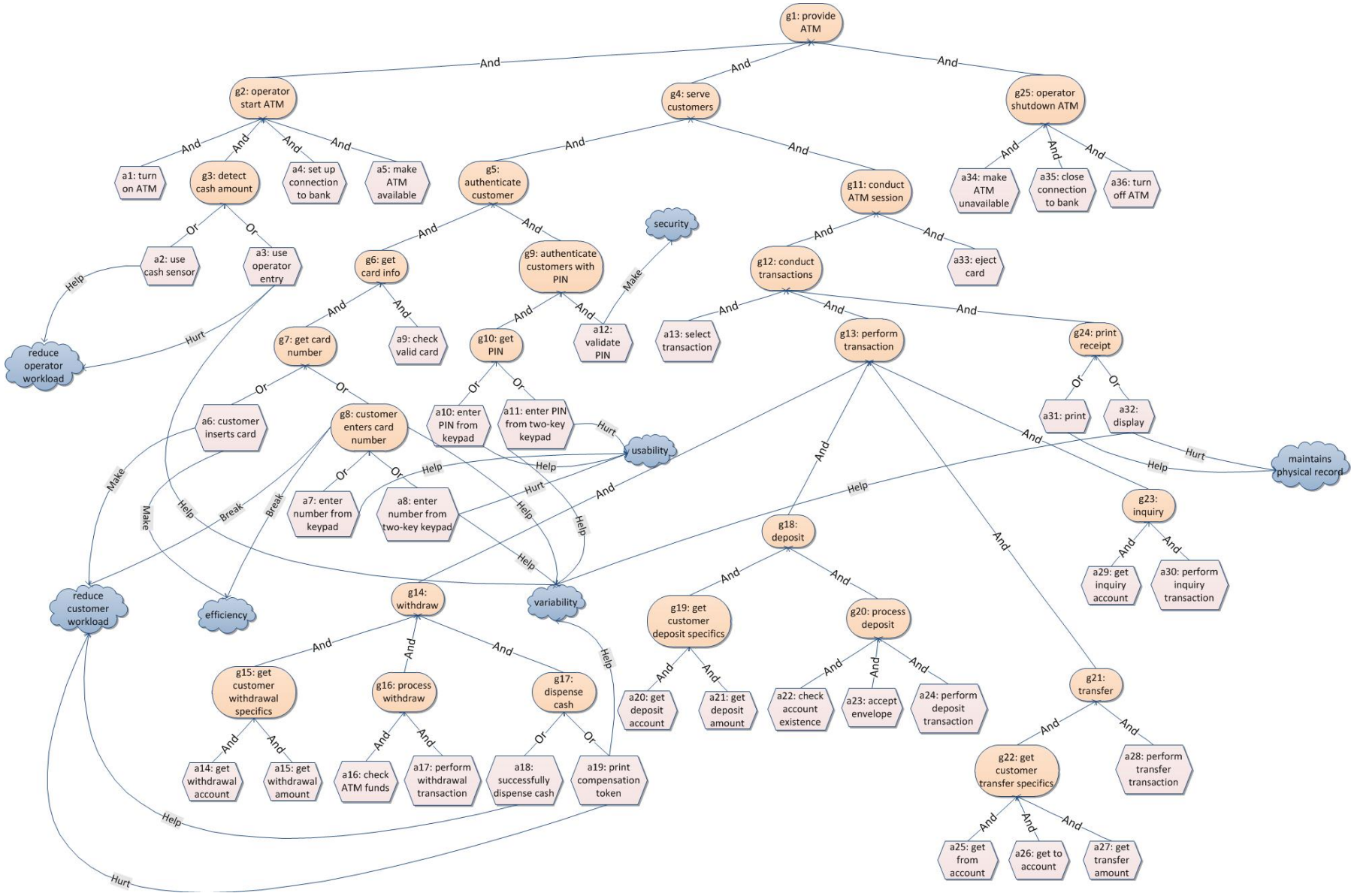

Figure 4.16: ATM goal model [13]

The goal **g11**-*conduct ATM section* is AND-decomposed in **g11**-*conduct transactions* and the task **a33**-*eject card*. At a lower level the goal **g12**-*conduct transactions* is AND-decomposed in task **a13**-*select transaction* and in **g13**-*perform transactions* and **g24**-*print receipt*. The goal **g13** will be refined into different alternatives based on the type of transaction the customer selected, such as withdraw, deposit or inquiry. The goal **g24**-*print receipt* is AND-decomposed in **a31**-*print* and in **a32**-*display*. If the task **a31**-*print* is executed, it will positively contribute the softgoal maintains physical records. But if the task **a32**-*display* is executed, it will hurt that softgoal. Finally, the goal **g25**-*operator shutdown ATM* is also AND-decomposed in task **a34**-*make ATM unavailable*, **a35**-*close connection to bank* and in task **a36**-*turn off ATM*.

Chapter 5

Awareness Requirements for the ATM system

AwReqs represents a new type of requirements, and as such they must be systematically elicited. Before their elicitation takes place the basic requirements must have been elicited and the requirements goal model must have been constructed. The model used in this experiment is a mixed model build up from the models provided from S. Liaskos and Wang [13] representing all the system's functional and non-functional requirements. First of all, domain assumptions and quality constraints in have been introduced into this model. *Connection to banking network available* has been introduced as a DA for the goal **g2**:*operator start ATM*. This assumption needs to be true for the goal to be achieved. On the other hand, in order to achieve the goal **g14**:*withdraw*, the DA *Cash dispenser never empty* needs to be true. Also constraints have been introduced in order to explicitly provide a satisfaction criteria for the system softgoals. For the softgoal *maintains physical record* the QC will be:*At least 95% of the requests are printed*. On the other hand, for the softgoal *reduce customer workload* the constraints will be:

- Customer performs withdrawal and deposit transactions within 3 minutes.
- Customer performs transfer and inquiry transactions within 6 minutess.

The first constraint refers to the *withdrawal* and *deposit* transactions, meaning that for the customer it will take no more than 3 minutes to successfully handle an withdrawal transaction. The second one refers to the *transfer* and *inquiry* transaction, meaning that for the customer such transactions can be successfully handled within 6 minutes. Figure 6.1 shows the extended ATM goal model, produced by introducing **DAs** and **QCs** into the original goal model.

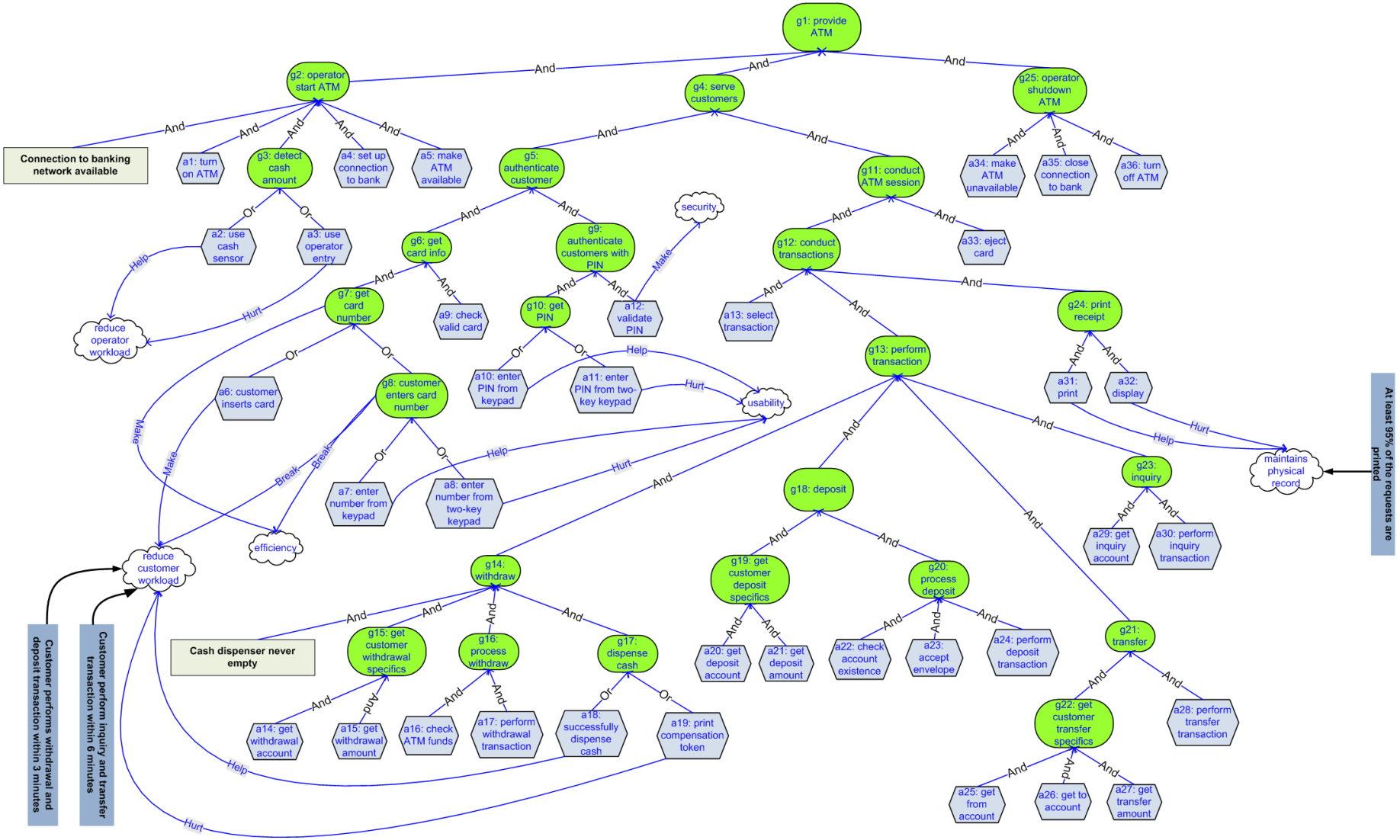

Figure 5.1: Extended ATM Goal Model

The first step during the AwReqs elicitation is about the identification of the goals/tasks that are critical for the system in order to achieve its purpose. Also, critical domain assumptions and quality constraints are identified. The goals/tasks identified are the requirements that need to have a consistently high success rate. The goal **g4**: *serve customers*, represents a high-level goal and for this requirement the failures must be very minimal. The failures for this requirement should be at most two, during each month. The goal **g3**: textitdetect cash amount, and the task **a4**: *set up connection to bank* represent two critical requirements because many activities will depend on their level of success. Also, *connection to banking network available* and *cash dispenser never empty* are critical domain assumptions for the system. Also, the task **a6**: *customer inserts card* and the task **a31**: *print* represent two critical tasks for the system. AwReqs can be derived from softgoals. For the softgoal *reduce customer workload* two QCs there have been introduced and to each of them the required success rate is attached. The first QC specifies that *Customer performs transaction within 3 minutes* refers to the withdrawal and deposit transaction. The AwReq specifies that if this is the case, then the success rate of such constraint must be satisfied in 90% of the cases. The second QC states that *Customer performs transaction within 6 minutes*, referring to the transfer transaction. Such transactions are more complex than the ones mentioned above, so that the success rate for this QC is required to be at 60%. In this way, it can be evaluated at runtime whether such requirements are met over large number of system executions and if it is not the case to make the related adjustment. The figure 5.2, gives a taste of the graphical representation of the AwReqs introduced. Most of the requirements introduced up to now are **regular** AwReqs.

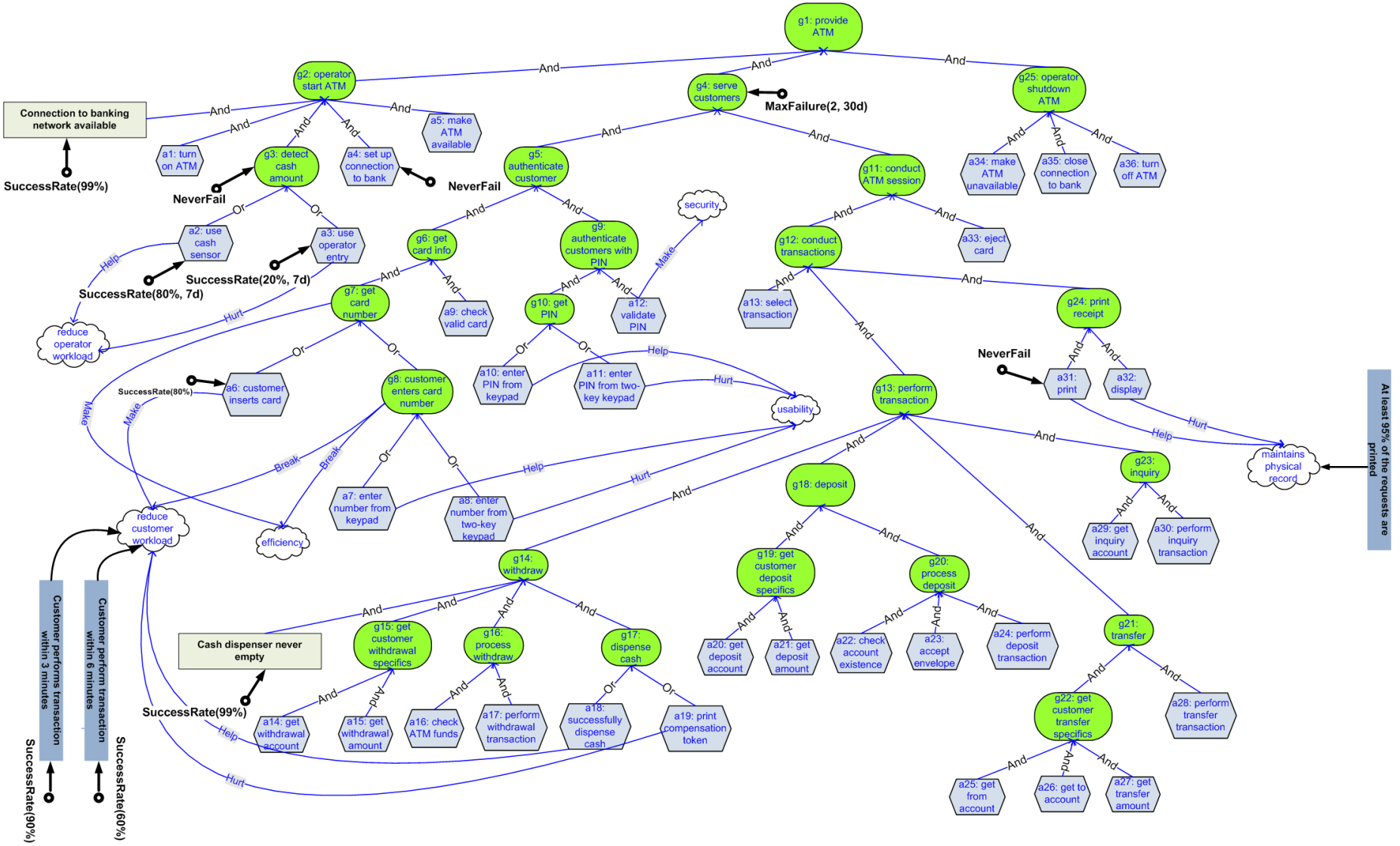

Figure 5.2: Graphical representation of some AwReqs

Other AwReqs have been elicited from the OR-decomposed goals. Some of the AwReqs compare the success count of two requirements. The goal **g7**:*get card number*, the OR decomposition shows that it can be achieved by the task **a6**: *customer inserts card* or by achieving goal **g8**:*customer enters card number*. The AwReq introduced here specifies that during the alternative selection the success rate of selecting the task **a6** will be 100 times higher than of the goal **g8** for achieving the high-level goal *get card number*. In order to specify the desired property for the alternative selection, other AwReqs have also been identified when the goal **g8**:*customers enters card number* needs to be fulfilled. In this case, it is specified that the task **a7**:*enter number from keypad* must have the success count of 50 times more than the task **a8** that asks the customer to *insert card number from two-key keypad*. The same success count is specified during the fulfillment of the goal **g10**:*get PIN*. The success count of the task **a10**:*enter PIN from keypad* must be 50 times more than of task **a11**:*enter PIN from two-key keypad*. Comparing success counts specifies that "good" alternatives are preferred over those alternatives that negatively contribute the non-functional requirements. Goal **g3**:*detect cash amount* can be achieved by executing the task **a2**:*use cash sensor* or by executing the task **a3**:*use operator entry*. If the cash sensor is used, the softgoal *reduce operator workload* will have a positive contribution, unlike if the task **a3** is performed. In that case this softgoal will have a negative contribution. The task **a13**:*select transaction* represents a particular task, when the customer have to decide the task to perform. A delta AwReqs talks about the change of state for this requirement. It must be at the **Undecided** state for at most 30 seconds, after that it has to switch to a **Succeeded** (transaction is selected) or **Failed**(user sections ended) state. Once the transaction is selected (task **a13** is successfully executed), new AwReqs that specify acceptable thresholds for the fulfillment of subsequent requirements in terms of elapsed time are introduced. So, after the transaction is selected, the goal **g17**:*dispense cash* and the task **a24**:*perform deposit transaction* should be executed within 3 minutes. On the other hand, the task **a28**:*perform transfer transaction* should be executed within 6 minutes from the successful transaction selection. Figure 5.3 shows a graphical representation of the AwReqs elicited in the ATM goal model along with other elements, such as DAs and QCs. AwReqs are represented as circles with arrows pointing at the requirement they refer. If the AwReq is comparing success rates/counts the first parameter is omitted, as the AwReq is pointing to it.

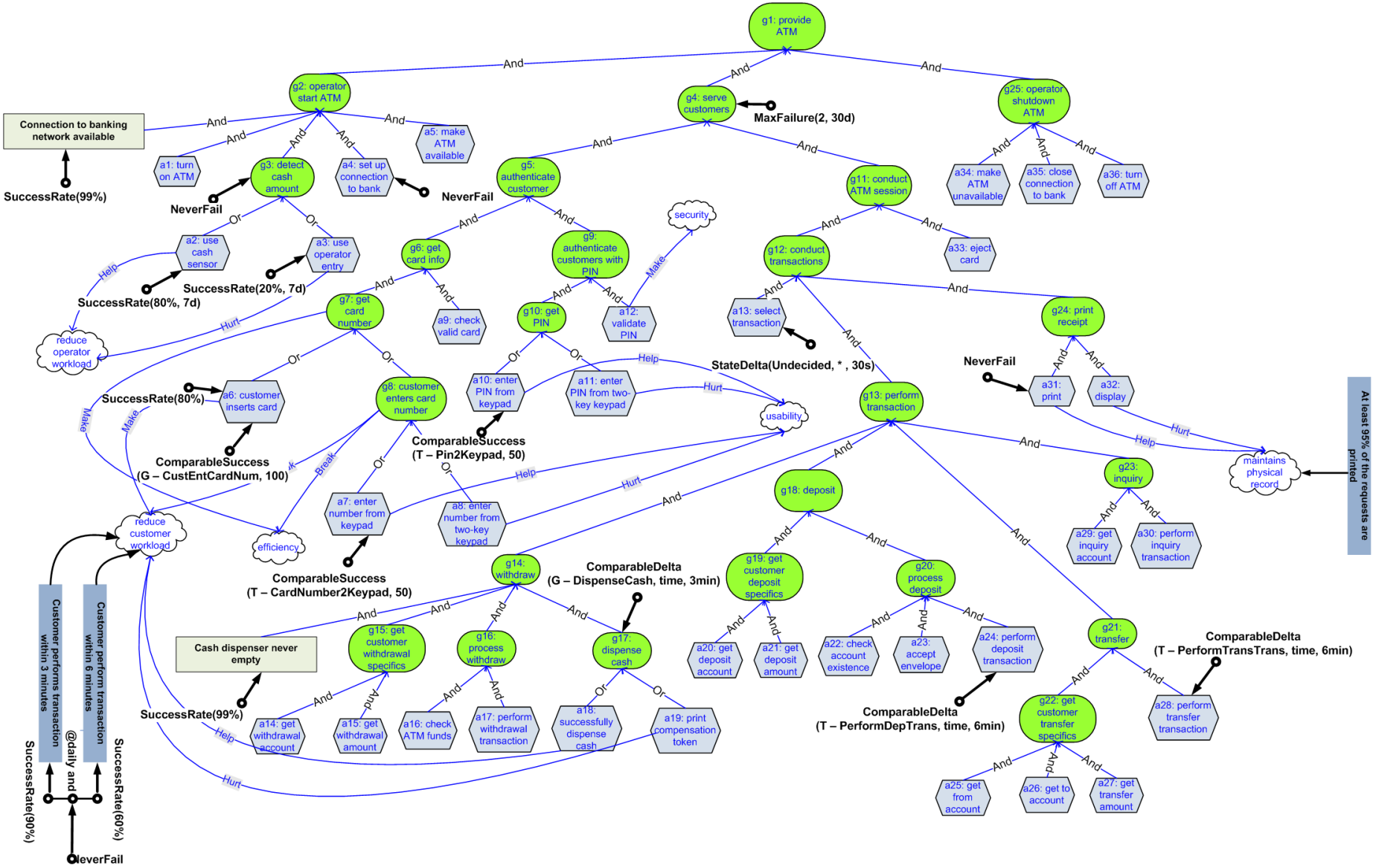

Figure 5.3: Graphical representation of AwReqs elicited in the ATM goal-model

Table 5.2 shows the AwReqs identified during the ATM system analysis. Each is provided with a brief description along with the type and pattern used for their specification. The table refers to the Figure 5.3, page 28.

Id	Description	Type	Pattern
AR1	*Detect cash amount* (**g3**) should never fail	Regular	NeverFail(T - DetectCashAm)
AR2	*Set up connection to bank* (**a4**) should never fail	Regular	NeverFail(T - SetUpConnect)
AR3	*Print Receipt* (**g24**) should never fail	Regular	NeverFail(T - PrintReceipt)
AR4	*Connection to banking network* should have a "high" success rate	Aggregate	SuccessRate(D - ComBankNet, 99%)
AR5	*Cash dispenser never be empty* should have a "high" success rate	Aggregate	SuccessRate(D - CashDispens, 99%)
AR6	*Enter pin from keypad* (a**10**) should succeed "much more" times than *enter pin from two-key keypad* (**a11**)	Aggregate	ComparableSuccess(T - PinKeypad, T - Pin2Keypad,50)
AR7	*Enter card number from keypad* (**a7**) should succeed "much more" times than *enter card number from two-key keypad* (**a8**)	Aggregate	ComparableSuccess(T - PinKeypad, T - Pin2Keypad,50)
AR8	*Reduce customer workload* less than 3 minutes should succeed at 90% of the time, *Reduce customer workload* less than 6 minutes should succeed at 60% of the time, measured daily	Aggregate	@daily SuccessRate(Q - CustWork3mins, 90%) and SuccessRate(Q - CustWork6mins,60%)
AR9	*Serve customers* (**g4**) should fail at most twice in a month	Aggregate	MaxFailure(G - ServeCustomer, 2, 30d)
AR10	*Use cash sensor* (**a2**) should have a "high" success rate over one week periods	Aggregate	SuccessRate(T - UseCashSensor, 80%, 7d)

Table 5.1: AwReqs identified for the ATM system, their types and patterns

AR1, **AR2**, **AR3** are the simplest form of AwReqs, the requirements to which they refer should never fail.
AR1 considers every instance of the referred requirement. Such instance is created for every execution and the constraint "never fail" should be checked for every instance.
AR2 considers the task *Set Up Connection to Bank* (**a4**). It is a critical task for the system. For each instance of this task, when created the constraint "never fail" should also be checked.
AR3 considers the goal *Print Receipt* (**g24**), that is Or-decomposed in two tasks: *Print* (**a31**) and *Display* (**a32**). Once the customer selects to print out the receipt, this task should satisfy the constraint "never fail".
AR4 and **AR5** are AwReqs applied to DAs (domain assumptions) that are critical for the system. **AR4** considers the domain assumption *Connection to Banking Network*. It demands the referred DA to be true 99% of the time the critical goals, such as *Serve Customer*(**g4**), etc. are attempted, . **AR5** considers the domain assumption, *cash dispenser should not be empty* should have a "very high" success rate. It demands the referred DA to be true 99% of the time the critical goals/tasks, such as *Dispense Cash* (**g17**), etc. are attempted.
AR6 and **AR7** compare success rates of two requirements. They refer to alternative selection when deciding how to fulfill the goals: *Get PIN* (**g10**) and *Customers Enters Card Number* (**g8**). *Get PIN* **g10** is OR-decomposed into the tasks: *Enter Pin from Keypad* (**a10**) and *Enter Pin From Two-Key Keypad* (**a11**). The goal *Customers Enters Card Number* is OR-decomposed into the tasks: *Enter Number From Keypad* (**a7**) and *Enter Number From Two-key Keypad* (**a8**). These OR-decompositions have contribution link to the softgoal *Usability*, so different success rates are desired for the alternatives.
AR8 combines different requirements, integrating two QCs with different target values. The customer workload in most of the cases should rise up to max 3 minutes for transactions like *Withdraw* (**g14**) and should rise up to max 6 minutes for more complex transactions like *Deposit* (**g18**) or *Transfer* (**g21**).
AR9 refers to a critical goal (**g4**). In this case, goal **g4** can fail at most two times in one month.
AR10, **AR11**, **AR12** and **AR13** refer at the properties needed to meet in the case of an alternative selection at runtime. To fulfill goal *Detect Cash Amount* (**g3**) even the action *Use Cash Sensor* (**a2**) or *Use Operator Entry* (**a3**) can be selected. This AwReq declares that the requirement **a2** should have a "high" success rate, measured in one week.

Id	Description	Type	Pattern
AR11	*Use operator entry* (**a3**) should have a "low" success rate over one week periods	Aggregate	SuccessRate(T - UseOperEntry, 20%, 7d)
AR12	Customer inserts card (**a6**) should have a "high" success rate	Aggregate	@SuccessRate(T - CustInsCard, 80%)
AR13	Customer inserts card (**a6**) will succeed "much more" times than Customers enters card number (**g8**)	Aggregate	ComparableSuccess(T - CustInsCard, G - CustEntCardNum, 100)
AR14	Select transaction (**a13**) should be decided within 30 seconds	Delta	StateDelta(T - SelectTransaction, Undecided, *, 30sec)
AR15	If transaction = withdraw dispense cash (**g17**) should be successfully executed within 3 min after select transaction (**a13**) is successfully executed, for the same user section	Delta	ComparableDelta(G - DispenseCash, T - SelectTransaction, time, 3min)
AR16	If transaction = deposit Perform deposit transaction (**a24**) should be successfully executed within 3 minutes after a successful execution of select transaction (**a13**), for the same user section	Delta	ComparableDelta(T - PerformDepTrans, T - SelectTransaction, time, 3min)
AR17	If transaction = transfer Perform transfer transaction (**a28**) should be successfully executed within 6 minutes after a successful execution of select transaction (**13**), for the same user section	Delta	ComparableDelta(T - PerformTransTrans, T - SelectTransaction, time, 6min)
AR18	**AR8** should never fail	Meta	NeverFail(AR8)

Table 5.2: AwReqs identified for the ATM system, their types and patterns

Instead, in **AR11** states the desired success rate for the requirement *Use Operator Entry* (**a3**). It should have a "low" success rate measured weekly.
AR12 states the desired success rate for the requirement (**a6**). In the achievement of the goal *Get Card Number* (**g7**) it is declared that the requirement **a6** should have a "high" success rate.
AR13 shows the comparison between the success rates of two requirements. This could be helpful when deciding at runtime an alternative selection. It declares that the requirement **a6** should succeed "much more" times than the requirement *Customer Enters Card Number* (**g8**).
AR14 as a delta AwReq specifies acceptable thresholds for the fulfillment of the requirement, in this case the task *Select Transaction* (**a13**). It specifies that the task **a13** should be satisfied (successfully finish execution) within 30 seconds.
AR15, **AR16** and **AR17** are delta AwReqs that talks about the state of the requirements.
AR15 shows that the requirement *Successfully Dispense Cash* (**a18**) should leave the state **Undecided** within 3 minutes after *Select Transaction* (**a13**) is satisfied (successfully executed).
Also the AwReqs **AR16** states that the requirement *Perform Deposit Transaction* (**a24**), should leave the state **Undecided** within 3 minutes after *Select Transaction* (**a13**) is satisfied (successfully executed).
AR17 refers to the maximum thresholds for a requirement *Perform Transfer Transaction* (**a28**) to leave the **Undecided** state, in other words the most important issue is that the requirements should leave this state within 6 minutes.
AR18 is a meta AwReq that refers to another AwReqs. It demands that the AwReq **AR8** should never fail.

5.1 Formalizing Awareness Requirements

As expressed before AwReqs are just requirements that refer to the success or failure of other requirements. The language selected to formalize such requirements is OCL_{TM} [10]. With OCL_{TM}, requirements are considered as first class citizens and make it possible to talk about status of particular requirements at different time points. The approach uses:

- design time requirements (modeled as a goal model and AwReqs elicited previously)
- run-time instance requirements, as such as various transactions or requests

Each requirement of the system is represented by a UML class extending the appropriate class of the diagram. The first letter of each class name indicates which element is extended (G for Goal, T for Task and so on). In Figure 5.4 the requirements are represented in a UML class diagram. This represents an important step before formalizing the requirements in OCL_{TM} language because OCL_{TM} constraints refer to classes and their instances, attributes and methods. This classes are only abstract representations of the elements of the system goal model and they will be part of a monitoring framework in the future.

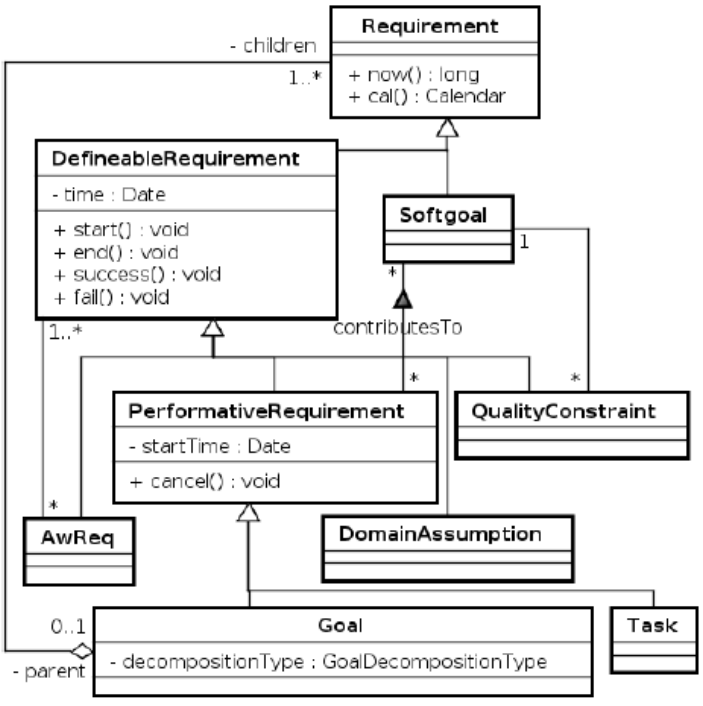

Figure 5.4: Requirements Class Model [10]

Figure 5.5 shows the formalization of the *AwReqs*. In this section, *AwReqs* are represented in a formal language, which will be important during the monitoring process.

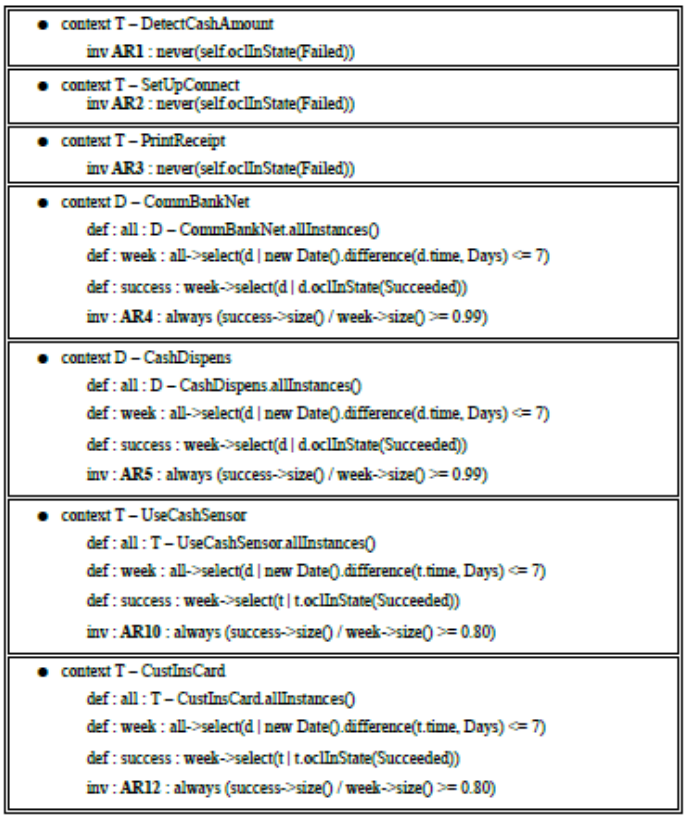

Figure 5.5: AwReqs formalization in OCL_{TM}

The invariants specify that instances of **AR1** :**T - DetectCashAmount**, **AR2** :**T - SetUpConnect**, **AR3** :**T - PrintReceipt** should never be in the Failed state, so for example, *DetectCashAmount* should never fail or *SetUpConnect* should never fail and so forth. Aggregate AwReqs place some constraints over collection of instances. In **AR4**, all the instances of the domain assumption *Connection to Banking Network*, **D - CommBankNet**, are obtained in a set. Then the subset of the instances that succeeded will be retrieved among all the instances. This result is obtained by using the **select()** operation. In the end, the set of the succeeded instances is compared with the set of all instances in order to declare that 99% of the instances are *always* successful. The date comparison is used in order to show that the evaluation of the success rate is performed considering only the past week. In **AR5**, all the instances of the domain assumption *Cash Dispenser*, **D - CashDispens** are obtained in a set. Then the

subset of the instances that succeeded will be retrieved among all the instances. This operation is obtained by using the **select()** operation. In the end, the set of the succeeded instances is compared with the set of all instances in order to declare that 99% of the instances are *always* successful. The date comparison is used in order to show that the evaluation of the success rate is performed considering only the past week. Also, for **AR10** and **AR12** the same operations are applied to the respective instances of the tasks. **AR14** is not about the success/failure of any requirement but on the fact that the task *Select Transaction* cannot be in the *Undecided* state for more than 30 Seconds. The formalization shows that eventually the tasks *Select Transaction* should not be in the *Undecided* state and the difference between the start and the end time should not be more than 30 seconds. **AR15**, **AR16**, **AR17** are delta AwReqs that talk about not only the success of the requirements, but also the time elapsed to finish their execution. So, from the main set of instances the *select()* operation singles out the instances of **G - DispenseCash** in the case of **AR15**, the instances of **T - PerformDepTrans** in the case of **AR16**, and **T - PerformTransTrans** in the case of **AR17**. The invariant states that the related set should have exactly one element that will be both successful and finish its execution within 3 (**AR15**) or 6 (**AR16**, **AR17**) minutes of **T Select Transaction**'s end time. Figure 5.6 shows the formalization of the delta AwReqs elicited in the system. Delta AwReqs specify invariants not over sets of instances, but over single instances of the requirements.

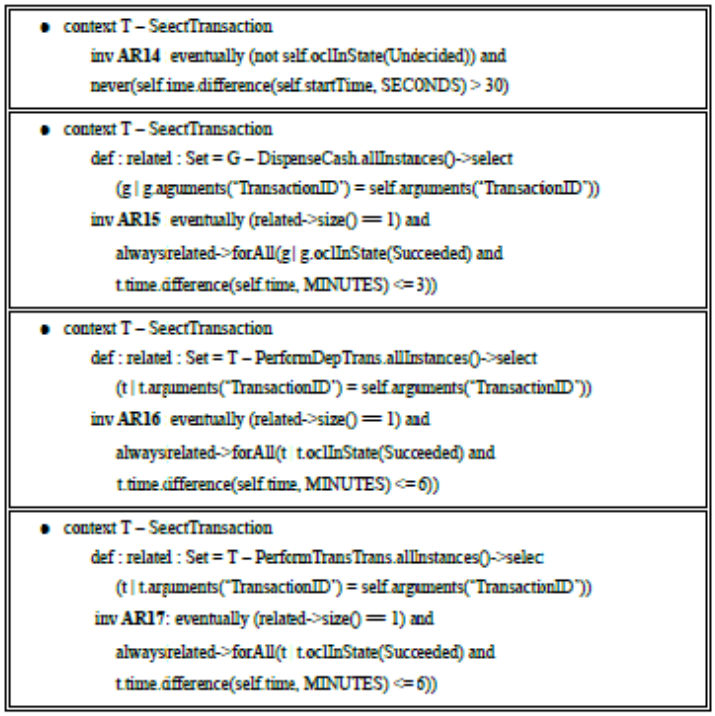

Figure 5.6: AwReqs formalization in OCL_{TM}

Chapter 6

System Identification

This section describes the system identification process. It introduces some new notations, like differential relations between parameters and indicators.

6.1 Goal Oriented Requirements Model

The ATM system requirements model is built as a goal-oriented requirements model, in terms of goals, softgoals, tasks, quality constraints (QCs) and domain assumptions (DAs). The main goal of the system, *Provide ATM*, which is AND-decomposed in **g2**-*Operator Start ATM*, **g4**-*Serve Customers* and **g25**-*Operator Shutdown ATM*. All of the sub-goals have to be executed in order to fulfill the main goal. Also, **g2** is AND-decomposed in the task **a1**-*turn on ATM*, and the goal **g3**-*Detect Cash Amount* that is OR-decomposed in two tasks **a2**:*Use Cash Sensor*, denoting that the cash amount is not detected by any operator but is performed by some device, and **a3**:*Use Operator Entry*, meaning that there has to be any operator detecting the cash amount, in order for the ATM to start. These two tasks contributes to the fulfillment of the *reduce operator workload* softgoal. The task **a2** helps the fulfillment of the softgoal. On the other hand the task **a3** hurts the softgoal's satisfaction. The DA *Connection to Banking Network Available* must always be true in order for the operator to *Start ATM*. Also, in order to perform *Withdraw* operation, *Cash Dispenser Never Empty* should be true. The softgoal *Reduce Customer Workload* is one of the most important softgoals, so in order to metricize this softgoal the clear-cut criteria for it will be "transaction length to be between 3 and 6 minutes for the same user section". For the softgoal *Maintains Physical Record*, the criteria will be "at least 95% of the requests are printed". Figure 6.1 shows a goal model enriched with the domain assumptions and quality constraints identified at the ATM model.

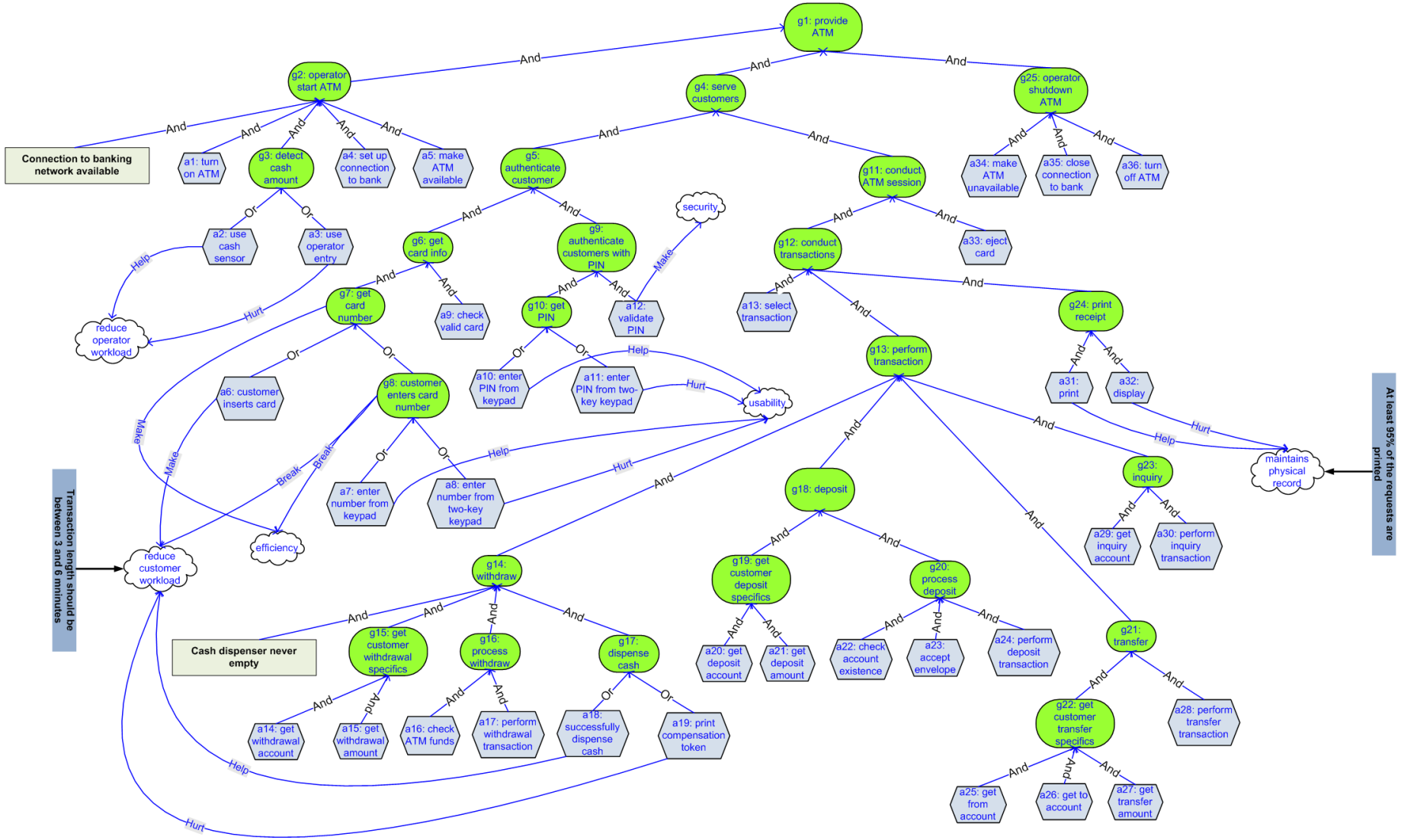

Figure 6.1: ATM Goal Model

6.2 Goal Model Parameters in ATM system

Another important aspect of the process refers to the identification *variation points* (VPs), *control variables* (CVs) and *indicators* in the ATM requirements model. The indicators will be monitored while the VPs and CVs can be controlled. The monitored and controlled parameters represent a way to implement reconciliation for adaptive systems at runtime. These parameters are introduced much earlier in the development process, at the level of requirements.
Variation Points (VPs) and *Control Variables* (CVs) relate to *indicators*. The OR-decompositions in goal models represents the **Variation Points**. Choosing one path instead of another at a VP represents the binding of intentional variability in the system. This variability allows for system re-configuration and reconciliation between system requirements and runtime behavior of the system. In the ATM system, five VPs have been identified and named **VP1**, **VP2**, **VP3**, **VP4** and **VP5**.
At **VP1**, when selecting the task to fulfill the goal *detect cash amount*, choosing the task *use operator entry* instead of *use cash sensor*, the operator workload will increase, so the softgoal *reduce operator workload* will be hurt.
At **VP2**, choosing *asking the customer to insert the card* over *asking the customer to input the card number*, the customer workload will increase. Swapping between the paths at VP2 provides a lot of information. So, if the customer inserts the card, his workload will be reduced but if the customer is asked to input the card number, it will influence the system requirements much more, like the *efficiency* softgoal will be denied, such as the customer workload will increase, so *Reduce Customer Workload* will be hurt. All these kinds of information are very important when deciding how to reconfigure the system in order to fulfill specific requirements.
At **VP3**, during the authentication, if the customer is asked to *enter the PIN from the keypad* instead of *enter PIN from two-key keypad*, the *usability* softgoal will be affected. In the first case, it will have a positive contribution and in the second case it will be negatively affected.
At **VP4**, during the selection of the way the customer will enter the card number, if the path where the *customer enters the card number from keypad* is chosen then the *usability* softgoal will have a positive contribution. Otherwise if the customer is asked to *enter the card number from a two-key keypad* then the *usability* softgoal will be hurt.
Finally, at **VP5**, in order to fulfill the goal *dispense cash*, when the task *successfully dispense cash* is selected over *print compensation token* task, the customer workload will be reduced. All this information will be helpful when selecting to adapt the system, exploiting the fact that different choices in different VPs have different contributions to NFRs of the system (*reduce customer workload*, *reduce operator workload*, increase *efficiency* or increase *usability*). Based on the

satisfaction of a desired NFR, the proper choice in any VP will be selected to reconcile the system behavior and the system requirements.

Another powerful mechanism for system (re)configuration is represented by the **Control Variables**. Control Variables are derived from families of related but slightly different goal/task or DAs alternatives. System behavior is influenced by the set of Control Variables, so its execution, success rate or quality of service can be different for different values of CVs. Control Variables are part of the system input and can be applied to the goals, tasks or domain assumptions. They represent abstractions over goals or domain model fragment. The benefits of having Control Variables include the ability to represent large number of model variations in a compact way as well as the ability to concisely analyze how changes in CV values affect the system success rate or quality of service (e.g. when Serving Customers, Starting ATM, etc.).
Starting from one of the high level goals, *Serve Customers*, the **NOA** (*Number of Operators Available*) is the CV that affects the success rate of this goal. This CV has to do with the number of operators available for providing help or explanations about how to use the ATM system. The NOA can have 3 values depending on the customer attendance rate. So, from 8:30 am to 10:30 am and from 2:30 pm to 4:30 pm, there is only one operator available, but from 10:30 am to 12:30 am there will be 2 operators available. Once the customer asks at the front office for any help, the operator will give all the relevant instructions. Out of these time intervals there are no operators available. So the success rate of Serve Customer can be defined over this control variable. The simplified linearized notation can describe how the changes of this CV affect the success rate of the goal Serve Customers.

$$\triangle\left(\langle \textit{success rate of Serve Customers}\rangle / NOA\right) > 0$$

This relation shows how the value of the CV affects the success rate of the goal: by increasing the number of operators the success rate increases. If there are two operators available instead of only one, the success rate of the goal will grow, and so forth. Adding more operators it will help the success rate of Serve Customers to increase. As shown by the goal model, in order to fulfill the goal *Customers Enters Card Number*, the customer can enter the card number from keypad (task **a7**) or enter the card number from two-key keypad. A CV that can affect the success rate of this goal is **NNIR** (*Number of New Information Required*). In some cases, we can also have a customer that for some reason enters an incorrect card number several times or it can also be the case that someone can steal the card number of any customer. In order to have a more secure and complete customer identification, additional information may be required. This information can be : *date of birth*, *ID card number* and an answer to some *secrete*

question. So, the success rate of the goal *Customer Enters Card Number* can be defined over this control variable. Using simplified linearized notation we can describe how the changes of this CV affect the success rate of the goal Serve Customers.

$$\triangle\left(\langle \textit{success rate of Customers Enter Card Number}\rangle / NNIR\right) > 0$$

So, more information is asked from the customer in addition to the card number, the greater the success rate of the goal is. Obviously, we cannot ask for to much additional information, otherwise it increases the risk of having unsatisfied clients.

One of the domain assumptions in order for the ATM system to be operative is *Cash Dispenser Never Empty*. Although there will be some different situations, like days of the week, weekends or holidays. Based on the cash needed to fulfill client's needs the cash dispenser should be loaded. Another CV is identified, **AAM** (*Available Amount of Money*). This variable will have different values on different days. Normally, the cash dispenser is loaded with a specific amount of money, but if it is a holiday or a weekend, it must be charged three/four times the "normal" daily amount. The value of this CV will affect the value of the goal *Withdraw*, which is a critical one. By increasing the value of the CV for specific situations, the success rate of the goal will grow. Using simplified linearized notation we can describe how the changes of this CV affects the success rate of the goal Serve Customers.

$$\triangle\left(\langle \textit{success rate of Withdraw}\rangle / AAM\right) > 0$$

In order to fulfill the goal *Customer Withdrawal Specifics*, the task **a14** (*get withdrawal account*) and the task **a15** (*get withdrawal amount*) must be executed. The withdrawal amount will not be greater than a limit value specified by the bank. But, using the operations (withdrawal, transfer, and so forth) a customer performs in different ATMs (belonging to other banks) the account details are not always updated just in time. Then some of the account policy is not followed ("*broken accounts*"), meaning the daily limit is overlapped, and this should not be allowed. Based on the number of the "*broken accounts*" the bank will consider the possibility of decreasing the daily limit. The **VDL** (*Value of Daily Limit*) value will affect the success rate of the task **a15**, *get withdrawal amount*. But, in this case by decreasing the value of the control variable the success rate of the task **a15** will increase. The *Value of Daily Limit* can also be increased in the case there is no "*broken accounts*" in the last quater/half of year. Using simplified linearized notation we can describe how the changes of this CV affects the success rate of the goal Serve Customers, if the value of the daily limit is decreasing.

$$\triangle\left(\langle \textit{success rate of Get Withrawal Amount}\rangle / VDL\right) > 0$$

Using simplified linearized notation we can describe how the changes of this CV affects the success rate of the goal Serve Customers.

$$\triangle\left(\langle \textit{success rate of Get Withrawal Amount}\rangle \,/\, VDL\right) < 0$$

Another control variable is identified in order to improve some of the NFRs of the system. Before the system ejects the card, the customer will be asked if he/she can answer to some questions about the ATM system. The control variable **CF** (*Customers Feedback*) represent the number of transactions elapsed between the surveys. So, the success rate of goal *Serve Customers* (one of the top-level goals) will depend on the values of the **CF** control variable. These quantitative answers need to be translated into how the system must be improved to better satisfy the clients. The questions will be:

1. Did it take too long to perform an operation?
2. How did You find the money denomination during the withdrawal?
3. What is Your opinion on the operator's support?

And the answer will be selected among four possible choices :

1. Bad
2. Sufficient
3. Good
4. Excellent

Based on the answers of the customers, some aspects of the system may be improved in order to positively affect the customer satisfaction. So, the success rate of the goal *Serve Customer* will be affected given the changes based on the customer feedback, but also the customer's happiness level will be affected. If the number of elapsed transactions is too small then maybe the same customer performing, for example, more than 3/4 transaction, needs to reply twice at the survey questions and this will negatively affect his mood. Using simplified linearized notation we can describe how the changes of this CV affects the success rate of the goal *Serve Customer* and also will not make customers unhappy because of wasting time on surveys.

$$\triangle\left(\langle \textit{success rate of Serve Customers}\rangle \,/\, CF\right) < 0$$

Figure 6.2 gives a taste of the graphical representation of the parameters introduced so far in the goal model.

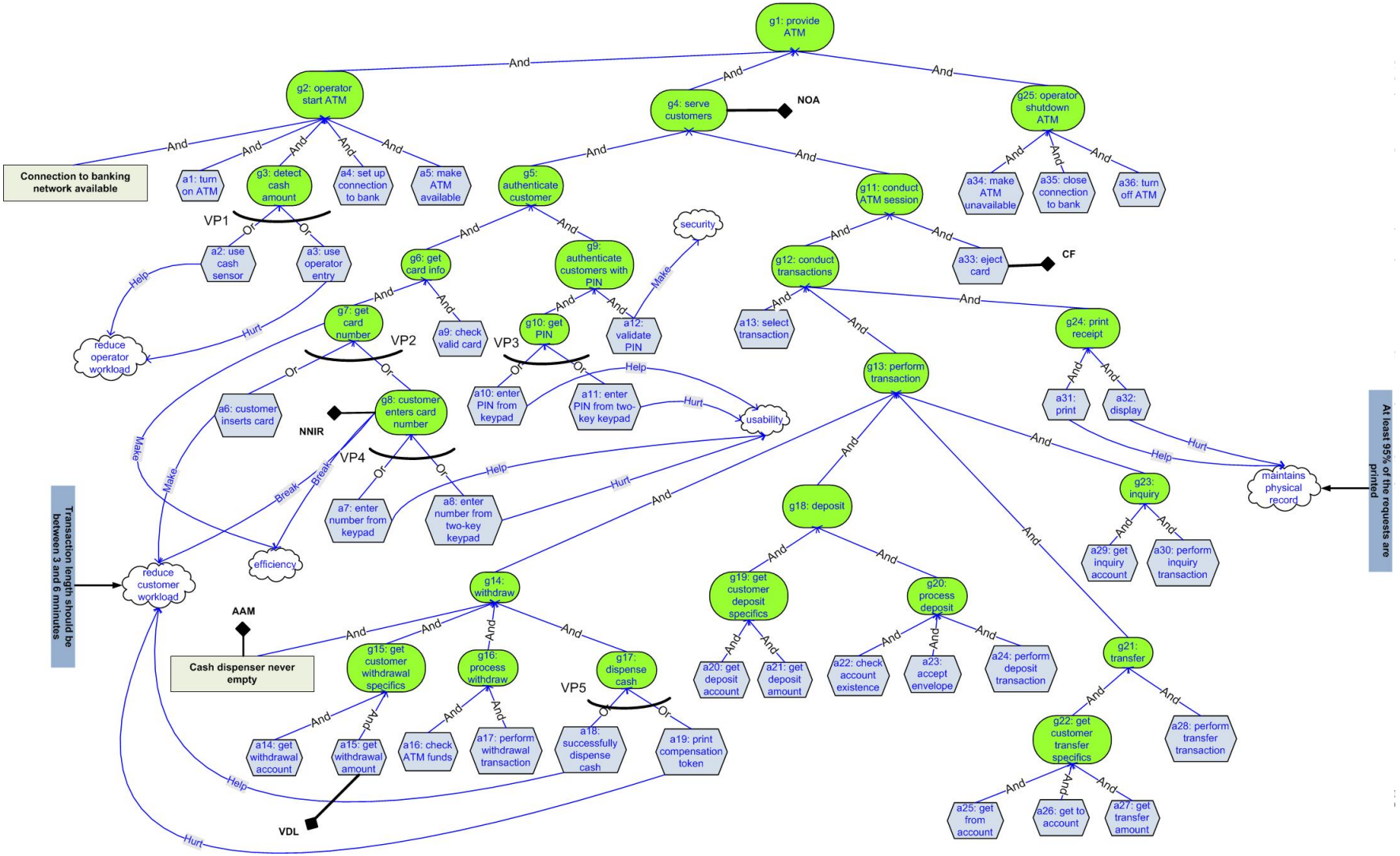

Figure 6.2: A graphical representation of parameters

Chapter 7

Conclusions and Future Work

This thesis presented an evaluation of **System Identification** process on an *Automated Teller Machine* (**ATM**) as a systematic approach conducted in modeling parameter-indicator relations based on the available information. Concerning the stated objectives, we have performed the following main steps.

1. **indicators identification**. During this phase a deeper analysis was performed on the system requirements model. So, we focused on the most relevant part of the requirements which can lead to self-adaptivity capability. In the applied approach the **AwReqs** represent the starting point for the indicators identification. Going deeper allows a better understanding of the system requirements, what can be the critical goals/tasks and domain assumptions in terms of variables' monitoring. This process did not require too much cognitive effort, but on the other hand it was time-consuming.

2. **identification of parameters**. During this step we identified possible variations in the goal model, such as *Variation Points* and *Control Variables*. The modification of these parameters could lead to a system recovery from possible failures or to improvements of the system performance. The *variation points* (**VPs**) are easy to locate in the model. They are represented by the OR-decompositions in the model, which can be easily recognised. Instead, the *control variables* (**CVs**) are part of the system input. They are not easy to compute as they represent abstractions over goals/-domain fragments. CVs can allow to represent variation in the model and the modification of such parameters could also affect the indicators. The CVs elicitation is taking into account the AwReqs, such that for each parameter's change the related AwReq will be positively affected. The stated task was the most complicated stage of the experiment. In order to come

up with these parameters a better knowledge on the application domain is needed and the derived abstractions should be linked to the AwReqs.

3. **identification of qualitative differential relations**. At this step, for every indicator we had to find the parameter that can affect such indicator. Some heuristics were used, such as contribution links to the softgoals (in the case of VPs), capturing parameter to some indicator sub-tree, and so forth. In this phase *Qualitative Reasoning* ideas were also used as we were dealing with partial knowledge of the system. We used a simplified linearized notation to describe the binding between parameters and indicators.

To give some examples, in case of the indicator related to the AwReq which refers to *Detect Cash Amount* goal the possible variation is represented by a Variation Point that either can be performed by the task *Use Cash Sensor* or by the task *Use Operator Entry*. The controller in this case will change the value of such VP based on the contribution links each of them has to the softgoal *Reduce Operator Workload*. Also in the case of the indicator related to the AwReq that refers to the domain assumption *Cash Dispenser Never Empty* the controller can change the value of the VP from *successfully dispense cash* to *print compensation token* if such indicator has failed. Considering the goal *Serve Customers* the parameter that affects this indicator is the *Number of Operators Available*. Such indicator should fail at most twice in a month, so in order for this requirement to converge the controller can change its value by selecting from the set [0, 1, 2]. By increasing the number of operators, different behaviors of the system are represented, every variation has a different impact on the goal *Serve Customers*.

The System Identification represents a sistematic process of requirements analysis. At this stage we are going not only to accommodate what are the features and the purposes of the system, but also to extend the requirements model with new concepts from Control Theory, with the aim to represent different situations/behaviors of the system still in a compact way (by means of VPs and CVs). If we think of representing such behaviors of the system without these new concepts, probably different requirements models would have been necessary in order to represent all the possible situations. So, it allows to describe the system's behavior concisely as well as exhaustively. Then the adaptation action could be performed by the controller. In this way the system identification process contributes not only to recovery from failures and system performance improvement, but also positively contributes to the system robustness. During the reconfiguration process controller will also take into account the softgoal satisfiability, as far as specific clear-cut criteria were imposed in the model. This experiment showed the possibility of decreasing the maintanance costs of the ATM system. As far as future benefits, dealing with adaptivity in early stages of the software process can reduce maintenance costs for the project, although

one should make sure these savings are greater than the cost of doing System Identification in the first place.
During the experiment we realized that there is no CASE Tool which can support the carious diagram creation or even to convert the AwReqs patterns into OCL_{TM} representation. What can also be useful is a method that can help to estimate if the eventual parameter change is going to allow us to achieve the desired objective.
Based on the stated qualitative relations the controller will perform a parameter change in order to affect positevely the related indicator. Even though the relation can describe the impact of the parameter change on the specific indicator, it cannot predict it in a quantitive way. However by introducing the "landmark" concept, we can predict that out of some range of values the parameter change will not affect the indicator any more.
In future works another step could be added to the evaluation of this process, e.g. such as the *relations refinement*, meaning the comparison or the combination of the initial equations.
Despite the experiment showed good results, more experiments are needed in order to estimate the approach above. Possible future work can use the information provided from the evaluation of *Awareness Requirements* for monitoring the system's requirements at runtime. Given the indication whether the system requirements have been met, it can be possible to identify how to improve **indicators** (in case of failure) by modifying the **parameters** introduced in the **System Identification** process to provide *adaptivity actions* that will be taken by the controller during reconciliation. An adaptivity framework can be used to perform a parameter tuning and reason about possible adaptation strategies to adopt in case of failures.
Some possible adaptation strategies can be:

- **reconfigure**, by identifying possible parameters that can be modified in order to meet requirements congruence (given any requirement failure)

- **abandon**, if it is the case that there is no possible parameter to be tuned or tuning some parameter can bring to conflicting effects on more than one indicators.

- **evolution** [12], by modifying the requirements model in a specific way with the aim of adapting it to some expected situation. Based on some requirements performance triggers the model can be transformed.

Other possible future work can take into account also context information, how properties of the environment can affect the system requirements. The relation parameter-indicator can also depend on environmental properties. It can be also interesting in future to consider not only reactive actions (adaptation actions to

adopt after failures occurred) but also to consider proactive reasoning (probabilistic reasoning on failures that can appear on the software system)

Bibliography

[1] T. Abdelzaher, Y. Diao, J.L. Hellerstein, C. Lu, and X. Zhu. Introduction to control theory and its application to computing systems. *Performance Modeling and Engineering*, pages 185–215, 2008. [cited at p. 11]

[2] M.S. Feather, S. Fickas, A. Van Lamsweerde, and C. Ponsard. Reconciling system requirements and runtime behavior. In *Proceedings of the 9th international workshop on Software specification and design*, page 50. IEEE Computer Society, 1998. [cited at p. 21]

[3] I.J. Jureta, J. Mylopoulos, and S. Faulkner. Revisiting the core ontology and problem in requirements engineering. In *International Requirements Engineering, 2008. RE'08. 16th IEEE*, pages 71–80. IEEE, 2008. [cited at p. 9, 10]

[4] R. Laddaga and P. Robertson. Self adaptive software: A position paper. In *SELF-STAR: International Workshop on Self-* Properties in Complex Information Systems*, volume 31. Citeseer, 2004. [cited at p. 19]

[5] A. Lapouchnian, S. Liaskos, J. Mylopoulos, and Y. Yu. Towards requirements-driven autonomic systems design. In *ACM SIGSOFT Software Engineering Notes*, volume 30, pages 1–7. ACM, 2005. [cited at p. 22]

[6] P.K. McKinley, B.H.C. Cheng, and C.A. Ofria. Applying digital evolution to the development of self-adaptive uls systems. In *Software Technologies for Ultra-Large-Scale Systems, 2007. ULS'07. International Workshop on*, page 2. IEEE, 2007. [cited at p. 5]

[7] W. Robinson. Extended ocl for goal monitoring. *Electronic Communications of the EASST*, 9(0), 2007. [cited at p. 8]

[8] M. Salehie and L. Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 4(2):14, 2009. [cited at p. 19, 20]

[9] V. Silva Souza, A. Lapouchnian, and J. Mylopoulos. System identification for adaptive software systems: A requirements engineering perspective. *Conceptual Modeling–ER 2011*, pages 346–361, 2011. [cited at p. 6, 7, 9, 15, 23]

[10] V.E. Silva Souza, A. Lapouchnian, W.N. Robinson, and J. Mylopoulos. Awareness requirements for adaptive systems. 2010. [cited at p. 5, 12, 23, 43]

[11] V.E.S. Souza and J. Mylopoulos. From awareness requirements to adaptive systems: a control-theoretic approach. In *Requirements@ Run. Time (RE@ RunTime), 2011 2nd International Workshop on*, pages 9–15. IEEE, 2011. [cited at p. 11]

[12] Lapouchnian A. Souza V. E. S. and J. Mylopoulos. Evolution requirements for adaptive systems. 2012. [cited at p. 57]

[13] Y. Wang and J. Mylopoulos. Self-repair through reconfiguration: A requirements engineering approach. In *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, pages 257–268. IEEE Computer Society, 2009. [cited at p. 22, 31, 33]

[14] J. Zhang and B.H.C. Cheng. Using temporal logic to specify adaptive program semantics. *Journal of Systems and Software*, 79(10):1361–1369, 2006. [cited at p. 21]