An Experiment on the Development of an Adaptive System based on the LAS-CAD — Incomplete Version[1]

Vítor E. Silva Souza
Department of Information Engineering and Computer Science — University of Trento
Via Sommarive 14, 38123, Trento, TN, Italy
vitorsouza@disi.unitn.it

March 20, 2012

[1]This is an incomplete version of this report, made available online at `http://disi.unitn.it/~vitorsouza/academia/a-cad-experiment/` so it could be cited by papers we have submitted that used this experiment as a running example [Souza *et al.*, 2012a,b]. Future versions will be published to the same URL.

Abstract

In our PhD research, we have been working on a control-theoretic approach for the development of adaptive systems with a Requirements Engineering perspective. The approach consists of three phases – *Awareness Requirements* elicitation, System Identification and *Evolution Requirements* elicitation — which are conducted in parallel with vanilla requirements activities to design a system that is able to adapt at runtime to undesirable situations. To facilitate this adaptation, the system models are done in a way that can be exploited at runtime by an adaptation framework.

In the context of this research, we have conducted an experiment in the modeling of an adaptive system based on the well-known London Ambulance Service Computer Aided Dispatch (LAS-CAD) case. This report presents the requirements elicited and modeled for an Adaptive Computer-aided Amblance Dispatch system (A-CAD) that has been designed using the aforementioned process.

Acknowledgements

This work has been partially supported by the ERC advanced grant 267856 "Lucretius: Foundations for Software Evolution" (unfolding during the period of April 2011 – March 2016) — http://www.lucretius.eu.

I would also like to acknowledge the contributions of my colleague Alexei Lapouchnian, professor William N. Robinson from Georgia State University (USA), and my advisor John Mylopoulos to some of the examples contained herein. Parts of these examples were developed while we were collaborating on the writing of technical papers related to this research.

Contents

Chapter 1

Introduction

For the past years, in the context of a PhD program, we have been working on a control-theoretic approach for the development of adaptive systems that adopts a Requirements Engineering (RE) perspective. The approach is based on state-of-the-art Goal-Oriented Requirements Engineering approaches and augments requirements models with information that is used at runtime for the operationalization of the system's adaptation.

An important aspect of any research proposal is validation. Roel Wieringa [2010] classifies many different forms of validation that can be conducted with respect to a research proposal, varying from a simple illustration, that helps the reader better understand the proposal, to a full case study, in which people outside of the research group use the proposed product/technique in the field in order to show that it produces the desired effects. In particular, Wieringa describes a "Lab Demo" (i.e. a "Laboratory Demonstration") as a technique used by the author on a realistic example in an artificial environment that shows that the technique could work in practice [Wieringa, 2010].

The purpose of this report is to document in detail all the steps taken in order to produce a lab demo as an initial validation of our approach for the development of adaptive systems. We believe that at this point of this research, a lab demonstration is enough to show the validity of the proposal and that further research on the topic could be pursued with possible application in practice once the research and its technological products have matured. We do not, however, think that further validation is unnecessary — deeper forms of validation, such as field experiments or case studies, are in our long-term research plans.

In order to use a realistic example, we based our lab demo in the case of the London Ambulance Service Computer Aided Despatch (LAS-CAD) System [swt, 1993]. This case study was presented at the 8^{th} International Workshop on Software Specification and Design [Finkelstein and Dowell, 1996] and became an exemplar in the Software Engineering community.

The adaptive system whose development is described in this report is based on the information available about the LAS-CAD system. From this point on, we refer to it as Adaptive Computer-aided Ambulance Dispatch, or A-CAD for short. Throughout this report its requirements models will be constructed iteratively and the final, complete version of the A-CAD models are provided in appendix A.

1.1 Papers related to this research

The focus of this technical report is the experiment conducted to validate the approach that is being developed in the context of our PhD program at the University of Trento. The approach itself has been presented in different publications and will not be further discussed here. After this section, we assume that the reader is familiar with it. The list below provides references to papers that are related to this research, summarizing each paper's contribution[1].

[1]PDF versions can be downloaded from the author's website: `http://disi.unitn.it/~vitorsouza/academia/`

- *Awareness Requirements for Self-Adaptive Socio-technical Systems* [Souza, 2010]: qualifying paper written as prerequisite for admission in the second year of the PhD program. It reviews the state-of-the-art on requirements for self-adaptive systems, presents the general idea of Awareness Requirements and discusses how the research in the context of the PhD course would be conducted around this proposal;
- *Awareness Requirements for Adaptive Systems* [Souza *et al.*, 2011b]: full paper published at the 6^{th} International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2011). The paper presents the baseline for our research, characterizes Awareness Requirements, exemplifies its use with a running example and validates the proposal using a requirements monitoring framework. Similar versions of this paper were published as technical reports DISI-10-049 (earlier version submitted to the 33^{rd} International Conference on Software Engineering — ICSE 2011, but not accepted for publication) [Souza *et al.*, 2010] and DISI-11-352 (extended version of the accepted SEAMS 2011 paper) [Souza *et al.*, 2011c];
- *System Identification for Adaptive Systems: a Requirements Engineering Perspective* [Souza *et al.*, 2011a]: full paper published at the 30^{th} International Conference on Conceptual Modeling (ER 2011). The paper presents a language and a systematic process for conducting System Identification, which, in Control Theory, is the process of quantifying the effects of control input in the measured output. The proposal adopts ideas from Qualitative Reasoning to cope with uncertainty and represent, in general terms, how changing a parameter in the target system affects its output;
- *From Awareness Requirements to Adaptive Systems: a Control-Theoretic Approach* [Souza and Mylopoulos, 2011]: short position paper published at the 2^{nd} International Workshop on requirements@run.time (RRT 2011). The paper argues for a control-theoretic, requirements-oriented view of adaptive systems and outlines our vision for this long-term research proposal for the development of adaptive systems;
- *Requirements-driven Qualitative Adaptation* [Souza *et al.*, 2012b]: paper submitted to a conference, currently under review. The paper describes a framework for qualitative adaptation based on the information modeled during System Identification, supporting different levels of precision of this qualitative information;
- *(Requirement) Evolution Requirements for Adaptive Systems* [Souza *et al.*, 2012a]: full paper accepted for publication at the 7^{th} International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2012, to appear). The paper introduces a new family of requirements, which allow analysts to specify adaptation strategies in terms of changes on the requirements model itself.

For a complete understanding of the approach we suggest the reader starts with the RRT 2011 position paper [Souza and Mylopoulos, 2011], which provides an overview of the proposal, and then go deeper into the Awareness Requirements proposal by reading the extended version of the SEAMS 2011 paper [Souza *et al.*, 2011c] followed by the ER 2011 paper on System Identification [Souza *et al.*, 2011a] and the two 2012 papers which detail alternatives when adapting the system [Souza *et al.*, 2012b,a]. The qualifying paper [Souza, 2010] is recommended only if the reader is interested in an overview of the state-of-the-art at the beginning of this research or the reasons that have motivated us to pursue it.

1.2 Report structure

This report has been divided in the following chapters:

§2: talks about the LAS-CAD system and presents requirements models for a computer-aided ambulance dispatch (CAD) system-to-be based on the description of the actual system as presented in the existing documentation about the case study. These models serve as the basis for the application of our approach for the development of adaptive systems;

§3: identifies Awareness Requirements in the CAD system-to-be, based on information about critical requirements and possible risks and hazards identified by analyzing the LAS-CAD case study. This constitutes one of the components of the adaptation requirements specification for the Adaptive CAD (A-CAD);

§4: presents the result of System Identification conducted on the models created thus far to recognize parameters that could be manipulated by the adaptation framework in order to adapt the target system, complementing the adaptation requirements specification of the A-CAD;

§5 complements the information elicited during System Identification with more precise details on the relation between system parameters and its outcome, specifying which adaptation algorithms should be used for which system failure by the adaptation framework;

§6: concludes the requirements elicitation phase with the A-CAD's Evolution Requirements, which represent precise stakeholder requirements for system adaptation (as opposed as the algorithms selected in the previous chapter), also associated to the possible system failures;

§7: provides an overview and some technical details about the framework that was implemented in order to conduct simulations that show that our approach could work in practice, i.e., the final product of the lab demo;

§8: summarizes our conclusions after conducting this experiment.

Furthermore, as mentioned before, appendix A shows the final, complete requirement models for the A-CAD.

Chapter 2

The Computer-aided Ambulance Dispatch System

The failure of the London Ambulance Service Computer-Aided Despatch (LAS-CAD) system in the fall of 1992 became a well known case study in the area of Software Engineering. Following the report on the inquiry published by the South West Thames Regional Health Authority [swt, 1993], papers on the subject were published in different communications, such as the proceedings of the 8^{th} International Workshop on Software Specification and Design (IWSSD) [Finkelstein and Dowell, 1996], the European Journal of Information Systems [Beynon-Davies, 1995], the Journal of the Brazilian Computer Society [Breitman *et al.*, 1999], ACM SIGSOFT Software Engineering Notes [Kramer and Wolf, 1996], amongst others.

Being a real system and having so much available information — due to its failure and subsequent inquiry — makes the LAS-CAD a good choice for validation of new research proposals. In effect, the focus of the discussions in the 8^{th} IWSSD was on which methods/techniques/tools should be applied in dealing with systems such as the LAS-CAD, and what research should be conducted to help in the development of such applications in the future [Kramer and Wolf, 1996]. Other examples of this use can be seen, for instance, in Letier's PhD thesis [2001] and You's masters dissertation [2004].

In particular, the LAS-CAD failure report [swt, 1993] states the following in paragraph 3024:

> *It should be said that in an ideal world it would be difficult to fault the concept of the design. It was ambitious but, if it could be achieved, there is little doubt that major efficiency gains could be made. However, its success would depend on the near 100% accuracy and reliability of the technology in its totality. Anything less could result in serious disruption to LAS operations.*

Thus, the high criticality of many of the components of the LAS-CAD make it a good case for adaptive systems, because self-adapting to failures — which invariably occur in a system that depends on near 100% reliability — is one way to avoid the aforementioned serious disruption to LAS operations. Take, for instance, the requirement of getting an ambulance to the scene of the incident as quickly as possible. In the case of the LAS, a set of standards (called ORCON) had been devised to indicate what percentage of ambulances should arrive in 3 minutes, 10 minutes and so on. There is no way to simply put that table into the system and guarantee that the standards will be followed [Kramer and Wolf, 1996]. Instead, adaptation actions can be taken whenever the system does not satisfy such requirements.

Note, however, that is not our intention to prove that the LAS would not have failed if it had been built as an adaptive system using our proposal. Many of the analyses conducted over the failure indicate that the procurement and the development processes were flawed, producing a bad quality system in general. Hence, if adaptation mechanisms had been developed to work with the LAS, there is no guarantee these would have been properly developed and have good quality and would therefore also be prone to failure. Our objectives here are to learn from the problems detected in the LAS in order to identify critical requirements and use those to develop a new system which would, in theory, be designed properly and have good quality in general.

As mentioned before, this report details the development of an Adaptive Computer-aided Ambulance Dispatch (A-CAD) based on the information available about the LAS-CAD system. In this chapter we present "vanilla" — i.e., non-adaptive — requirements for a CAD system-to-be, which will then be used as the basis for the application of our approach for the design and development of the A-CAD in the next chapters. We do this in three steps:

§2.1: provides an overview of the problem solved by the CAD, describing the problem domain, determining its scope and presenting general stakeholder requirements for a CAD system-to-be;

§2.2: complements the previous section by presenting goal models taken from a report on the experience of applying the $i^\star$ framework to the LAS-CAD system [You, 2001];

§2.3: based on the information from the previous sections, shows the system-level goal models for the CAD system-to-be, which are needed by our approach for the design of adaptive systems.

2.1 Overview

In this section we describe the domain, the scope and general stakeholder requirements for a CAD system-to-be. The information on this section will then be used as source for the elicitation of early and late requirements in the subsequent sections of this chapter. This information was taken from the aforementioned publications about the LAS-CAD case study [swt, 1993; Beynon-Davies, 1995; Breitman *et al.*, 1999; Finkelstein and Dowell, 1996; Kramer and Wolf, 1996], especially the system requirements description by Daniel Jackson in the appendix of [Kramer and Wolf, 1996]. Note that these descriptions and requirements are loosely based on the LAS-CAD and do not represent its actual requirements, because technical information about that system was not supplied by London authorities.

A CAD is basically a resource management system. Finite resources (ambulances, paramedics, drivers, equipments, etc.) have to be dispatched to locations of incidents happening unpredictably around a specific region (e.g., in the case of the LAS-CAD it is the city of London). The main goal of the CAD is to generate dispatching instructions regarding these resources in an optimized way in response to incidents. To do this, the system needs to keep track of each resource's status and interface with different human and software actors.

2.1.1 Domain Entities

The following list provides definitions and information on different elements pertaining to the CAD domain, mostly adapted from [Kramer and Wolf, 1996]. Italicized words refer to other entities which are also in the list:

- **Emergency Service**: a service provided by the public authorities of a specific *region* (e.g., the London Ambulance Service for the city of London) to the citizens of that *region* which consists in the dispatching of *ambulances* and their *crews* and *equipments* to *incidents*. The purpose of the CAD is to automate parts of this service;
- **Serviced Region**: the physical region (i.e. a set of *locations*) to which the *emergency service* may dispatch *ambulances* (can be a city, a state or province, etc.);
- **Call**: a telephone call to the *emergency service* (e.g., 999 in the UK, 911 in the USA, 190 in Brazil, 118 in Italy, etc.), identified by the caller's phone number and the starting time of the call;
- **Incident**: some kind of accident or emergency that triggers a *call* to the *emergency service* and requires assistance from one or more *ambulances* (e.g., a car accident that requires an ambulance to aid injured people). There is no precise definition of what is and what is not an incident, so each call is analyzed by the *emergency service*'s *staff* and may be dismissed as a non-emergency. An incident occurs at some

location and has a description (provided by the caller) and a status (to keep track if it is resolved). Incidents are associated with one or more *calls* (multiple calls about the same incident also have to be identified by the *staff*);

- **Location**: a physical location in the *serviced region*. A location is composed of an address and, optionally, more precise indications within the given address (e.g., the floor, in case of buildings);
- **Sector**: the *serviced region* is partitioned into sectors for the purpose of dispatching. Therefore a sector is also a set of *locations*, which is a subset of the *serviced region*. Each sector is associated to a list of preferred *hospitals* and *stations* to help the CAD generate optimized dispatching instructions. Sectors are likely to be physically contiguous, but this is not mandatory;
- **Station**: a place in which *ambulances* and *crews* wait for dispatching instructions. Each station has a *location* in the *serviced region* and they are likely to be spread around it for faster arrival at the *locations* of *incidents*;
- **Hospital**: public hospitals that have emergency rooms capable of receiving people brought by *ambulances* from the *locations* of the *incidents*. As *stations*, hospitals have their *locations* in the *serviced region* and are likely to be spread around it;
- **Ambulance**: any vehicle used by the *emergency service* to aid citizens in case of *incidents* (e.g., ambulance cars, emergency trucks, fire engines, motorcycle response units, helicopter, etc.). Vehicles are identified by their license plate numbers (or similar in case of helicopters), assigned to a station and can have their current (i.e., most recent) location registered in the system;
- **Equipment**: any device that is useful for aiding citizens in case of *incidents* (e.g., crash kits, stretcher, etc.). Equipments of different types are assigned to ambulances and are uniquely identified by an ID code;
- **Crew member**: human resources that work in *ambulances* and provide aid to citizens in case of *incidents* (e.g., drivers, paramedics, firemen, etc.). Crew members are usually referred to by the acronym EMT, which means Emergency Medical Technician. They are identified by their ID numbers and assigned to a specific ambulance;
- **Ambulance configuration**: refers to the type of vehicle, the roles of crew members and the present equipment in an ambulance. This information is important when dispatching, as some *incidents* might need a specific crew member or equipment for the aid to be successful. For example, one ambulance with two paramedics can be enough in a common car crash, but if the cars are on fire a fire truck and a firemen might also be needed at the *location*;
- **Staff**: people that work in the *emergency service*'s dispatching function, i.e., employees of the *emergency service* that are not part of a *crew* (e.g., telephone operators, resource allocators, dispatchers, etc.).

2.1.2 Scope

A real CAD system is very large and complex. For our experiments, we will focus on the core functions of a CAD software. We assume, therefore, that there are other systems which produce a series of events related to the ambulances managed by the CAD and which are monitored by the core CAD software to know which are available and where they are located (the dependencies between the CAD software and these other systems is shown in section 2.2).

Figure 2.1 shows the states an ambulance can assume during its life-cycle and the events that trigger the transitions. Below we describe these events, which are adapted from [Kramer and Wolf, 1996]. The CAD software is supposed to be aware of all such events.

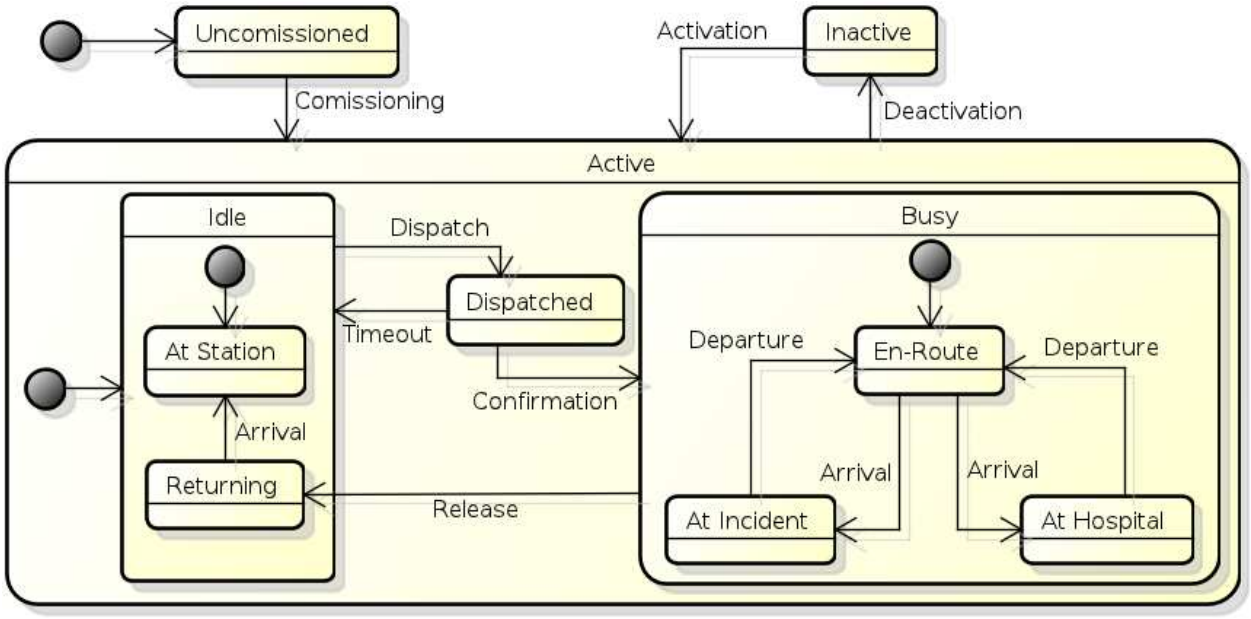

Figure 2.1: Statemachine diagram for Ambulances.

- **Creation**: ambulance has been registered within the system;
- **Comissioning**: ambulance has been assigned to a station. This assignment can change over time in case of need;
- **Activation and deactivation**: ambulances can be deactivated during certain periods of time (e.g., when they need to be repaired, refueled, etc.). Deactivated ambulances cannot be dispatched;
- **Arrival and departure**: ambulance has arrived or has left a given location. This location can belong to a station, a hospital or an incident;
- **En-route location**: periodical reporting of location during the mobilization of ambulances to a target location. This event is monitored only for ambulances that are active and outside their stations. Each ambulance in this condition is supposed to send location updates every 13 seconds;
- **Dispatch, timeout, confirmation and release**: when an ambulance is dispatched to an incident (by the core CAD software), it should be confirmed (by its crew) so it is considered engaged to resolving the incident. This has to occur in a timely fashion, otherwise the CAD will search for another ambulance to dispatch and the first one will go back to being idle. When the incident is resolved (e.g., injured people are dropped off at the hospital) the ambulance is released and becomes idle. Only idle ambulances can be dispatched.

Furthermore, events of **comissioning** and **deactivation** should also be monitored for crew members and equipment in order to know, at any given time, what is the configuration of each ambulance. For example, a crash kit could break and be sent to repair, leaving an ambulance without it; or an EMT could take a lunch break for one hour leaving his ambulance with one less crew member for a while.

Finally, some entities and situations are considered out of the scope of the CAD system. The following is a list of assumptions with respect to the requirements of the CAD software:

- **Caller information**: information about the caller and the phone used to report an emergency is added to the incident's report by the telephone operator for logging purposes only. The CAD software will

not consider this information when dispatching resources and there will be no support for identifying a thread of calls from the same person;

- **Incident category**: in real CAD systems, incidents are categorized by importance. For instance, the LAS has three main categories — A (red), B (amber) and C (green) — divided in two or three subcategories each [Reynolds, a]. Different categories can have different standards regarding levels of service, for example. We assume, however, that all calls are of the same category;
- **Treatment**: it is not the responibility of the CAD to follow the treatment of injured parties. In fact, the people affected by an incident are not monitored at all by the CAD, which expects only to receive a *release* event when ambulances are done with an incident. It is the responsibility of the dispatched crew to conclude when an incident is resolved and inform the CAD;
- **Dispatching to emergencies only**: ambulances only get allocated in the CAD in response to incidents. In case the service is provided by the public authorities, a separate system should manage these situations and deactivate ambulances whenever they get dispatched to non-emergencies;
- **Initial data is given**: the information required by the CAD to dispatch resources is assumed to be given: the limits of the serviced region, its division in sectors, location of hospitals and stations, list of preferred hospitals/stations for each sector, ambulances per station, ambulance crews and equipments, etc. In a real system, such information is presumably calculated and periodically modified after analyzing statistics on the amount and nature of incidents in each sector of the serviced region in the past.

2.1.3 Stakeholder Requirements

Given the above description of entities and the scope of the problem, the following is a list of general requirements for the CAD software:

Incident Response

GR-1. The system shall allow staff to register calls they receive from citizens;

GR-2. The system shall, whenever possible, detect the location of the caller and associate it with the call registry (public phones have associated locations, cell phones might be triangulated, etc.);

GR-3. The system shall allow staff to dismiss calls as non-emergencies;

GR-4. The system shall assist staff in identifying, through the information from the call, if it refers to an open incident in the system;

GR-5. The system shall allow staff to assign calls to open incidents as duplicates or create new incidents for calls;

GR-6. The system shall allow staff to indicate the number of ambulances needed and their respective configurations (e.g., ambulance with paramedics, fire truck and firemen, motorcycle response unit, etc.);

GR-7. The system shall allow staff to confirm the information related to new incidents, clearing them for dispatch by the system;

GR-8. The system shall, upon confirmation of an incident, determine the best ambulance to be dispatched to the incident's location, given the required configuration;

GR-9. The system shall inform stations of dispatched ambulances about the dispatching instructions, if the ambulance is in the station, or inform the ambulance itself, if it is not in the station;

GR-10. The system shall close incidents when all resources related to it are released (see GR-13);

GR-11. The system shall, in case of deactivation of an ambulance that is busy, determine the best ambulance to be dispatched in replacement of the one that has been deactivated, given the required configuration. GR-9 should follow accordingly;

GR-12. The system shall perform in such a way that at least 75% of the ambulances arrive within 8 minutes to the location of the incident once dispatching instructions have been sent (see GR-9). This constraint is based in the LAS-CAD standard for Category A calls [Reynolds, b].

Resource Monitoring

GR-13. The system shall monitor for ambulance-related events (see §2.1.2) and keep the status of each ambulance up-to-date, including ambulance configuration;

GR-14. The system shall show accurate and up-to-date information about on-going incidents, including status, configuration and position of engaged ambulances;

GR-15. The system shall generate messages whenever ambulances arrive at the location of incidents, leave the location of incidents (to go to the hospital) and when they are released (incident resolved).

Exception Messages

GR-16. The system shall generate exception messages if the dispatching process does not conclude within 3 minutes. The process is considered concluded after the number of ambulances and their configurations have been assigned (see GR-6), the system has dispatched ambulances that fit the configuration (see GR-8 and GR-9) and all ambulances have confirmed the dispatch (see GR-13);

GR-17. The system shall generate exception messages if ambulances engaged to incidents are not released within 15 minutes of their confirmation (see GR-13) – in other words, incidents should be resolved within 15 minutes of dispatch;

GR-18. The system shall generate exception messages if ambulances seem to be going to the wrong direction with respect to the location they are supposed to go (see GR-13).

In the remainder of the chapter, the overview of the problem the CAD system intends to solve given by this section is regarded as information provided by the stakeholders and used as source for the requirements models presented in section 2.1.3. Next, we present $i^\star$ diagrams that illustrate the dependencies among actors involved in an ambulance dispatch system and the role of the CAD in this context.

2.2 An $i^\star$ Analysis of the LAS-CAD System

Complementing the information provided by the previous section, this section presents $i^\star$ [Yu *et al.*, 2011] diagrams that depict the dependencies among the actors involved in an ambulance dispatch activity before and after the development of a software system that automates some of its steps.

These diagrams were taken from the report "Experiences with applying the $i^\star$ framework to a real-life system" [You, 2001], an assignment delivered by Jane You for a Requirements Engineering course in the University of Toronto, Canada. The goal of the report was to evaluate the proficiency of $i^\star$ as a technique for RE by applying it to the LAS-CAD case. It uses mostly the same sources of information as this report (namely, [swt, 1993; Breitman *et al.*, 1999; Finkelstein and Dowell, 1996; Kramer and Wolf, 1996]).

Considering the manual system that was used by the LAS before the introduction of the CAD system, after an analysis of the report on the inquiry into the LAS-CAD failure [swt, 1993], You identified the involved actors and modeled the dependencies amongst them in the $i^\star$ strategic dependency diagram of figure 2.2 (page 13). In her diagram, *Service Consumer* represents citizens of the *serviced region*, whereas

Call Reviewer, *Resource Allocator*, *Radio Operator*, *Dispatcher* and *Operator at Ambulance Station* are all members of *staff*.

The way the manual dispatch process works, *service consumers* depend on *ambulance crews* to provide ambulance services when incidents occur. More specifically, they depend on *call takers* to take the information on the incident. On the other hand, *call takers* depend on *service consumers* to correctly report the incident.

Once incidents have been taken, they are described in an *AS form* and sent to review. Thus, *call reviewers* depend on *call takers* to effectively provide the incident information in this form. Analogously, the next member of the staff in the chain of activities, the *resource allocator*, depends on the *call reviewer* to effectively review the incident and pass along the reviewed information in the same *AS form*.

At this point, the job of the *resource allocator* is to select the best ambulances for the incident. To do this job well, they depend on *radio operators*, which are constantly in contact with ambulances via radio, to effectively provide status and location of ambulances. Thus, the chain of dependency reaches also the *ambulance crew*, which should effectively provide the *radio operator* with their status and location. All this is very important, as *ambulance crews* depend on *resource allocators* for optimized instructions, i.e., they do not want to travel long distances and far from their usual working section.

Mobilization instructions are then sent to *dispatchers*, which means they depend on *resource allocators* to effectively make decisions and provide these instructions. Then, the *dispatcher*'s job is to effectively send these instructions to the *radio operator* or the *operator at ambulance station*, which is represented by a dependency of these two actors on the *dispatcher*. The choice between *radio operator* or *operator at ambulance station* depends if the ambulance is on the streets or at the ambulance station, respectively.

Finally, it is the responsibility of either the *radio operator* or the *operator at ambulance station* to effectively communicate these instructions to the *ambulance crew* and we can also see that dependency depicted in the figure. The cycle completes with the dependency between *service consumer* and *ambulance crew*, mentioned at the beginning of this description.

The diagram also depicts the *LAS management* actor, whose dependencies basically indicate non-functional requirements (call answering time, emergency response time) of this non-computerized system. Obviously emergency services are not brought about for free and are performed in exchange for taxes paid by *service consumers*, but this dependency is not shown in the diagram for simplicity.

By analyzing this model, You concludes that the manual process is insufficient to meet time performance standards specified by ORCON to the LAS. Therefore, she proceeds to investigate alternative solutions to the problem. One that is of particular interest to our experiments is the use of a fully automated CAD system. Its $i^\star$ strategic dependency diagram is shown in figure 2.3 (page 14). Some of the acronyms in the figure might need explanation:

- **RIFS**: the *Radio InterFace System* is a sub-system of the *Radio System* used to communicate with ambulances;
- **AVLS**: the *Automatic Vehicle Location System* provides geographic coordinates of the current location of ambulances;
- **BT PSTN**: the *British Telecom Public Switched Telephone Network* is the means through which the *RIFS sub-system* sends mobilization instructions to *pritners at ambulance stations*;
- **MDT**: *Mobile Data Terminals* consist of devices installed in ambulances which are used by their *crews* to input status and location;
- **LAS/CAD**: as previously mentioned in this report, **LAS** means *London Ambulance Service* and **CAD** is *Computer-Aided Dispatch* (*Despatch* in British English).

Back to the scenario represented in the figure, the *call taker* inputs the incident information in the *CAD software*, which provides functionalities similar to the ones described by the general requirements in section 2.1.3, i.e., generates mobilization instructions to *ambulance crews*. The diagram also depicts some of the assumptions made in section 2.1.2, by representing dependencies between the *CAD software* and other elements. This is the information we are mostly interested here.

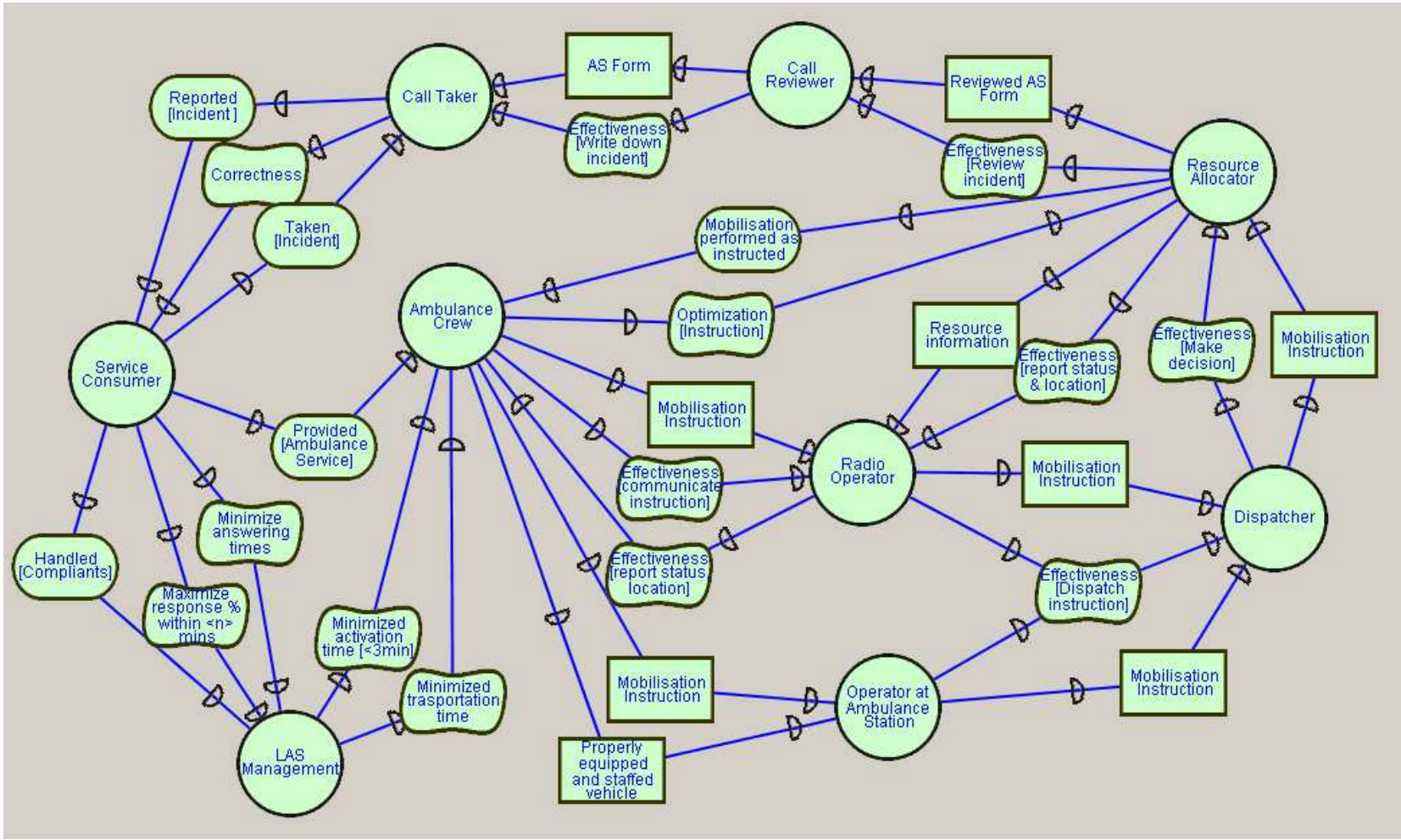

Figure 2.2: $i^\star$ Strategic Dependency model for the LAS staff, without the CAD system [You, 2001].

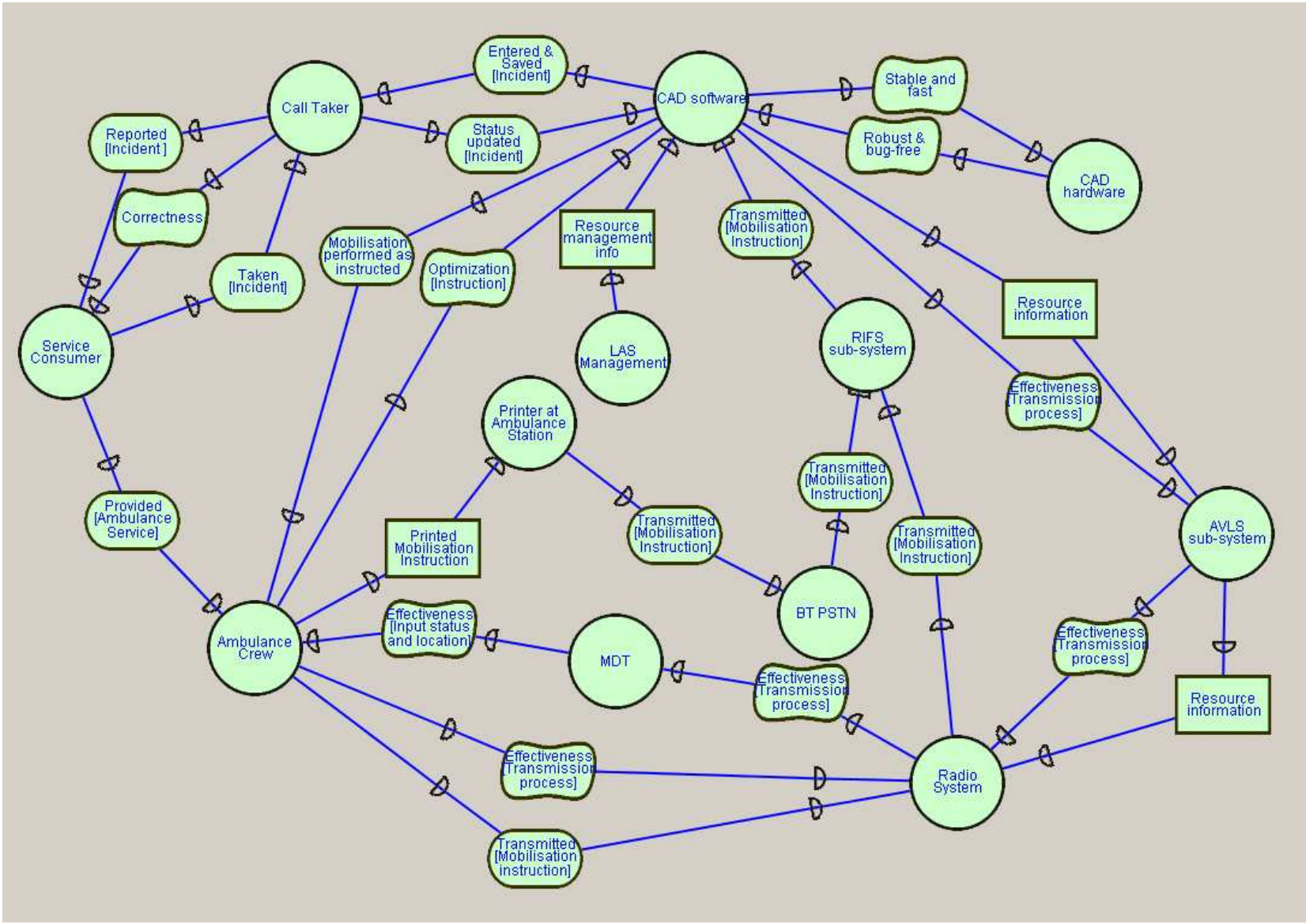

Figure 2.3: $i^\star$ Strategic Dependency model for a fully automated CAD system [You, 2001].

The following list ellaborates on these dependencies, showing how they were handled in the fully automated LAS-CAD scenario according to You's analysis and how they could be implemented in our CAD experiment, complementing the information previously shown in section 2.1.2:

- **Ambulance (de)activation, confirmation and release**: we assume the CAD is aware (i.e., gets notified, monitors for, or any other means of knowing these events) when an ambulance gets deactivated (e.g., for repairs), reactivated, when the ambulance's crew confirms they will respond to an incident and when they are done with this response and are, thus, released.
  - In figure 2.3, these notifications are done by the *ambulance crew* through the *MDT*, which depends on the *radio system* for transmission;
  - For our experiments, we assume ambulances have *MDTs* installed and that they are capable of communicating with the *CAD software* through a transmission medium to be chosen during the design stage of the software development process (e.g., GSM network).
- **Ambulance location**: ambulances are supposed to inform their location to the CAD at periodic intervals.
  - For the LAS-CAD this is done by the *AVLS sub-system* which, like the *MDT*, depends on the *radio system* for transmission;
  - For our CAD system-to-be we assume ambulances are also equiped with GPS devices that inform the *MDTs* about their location. The *MDTs*, then, should transmit this data back to the *CAD software* the same way as (de)activation, confirmation and release events are transmitted.
- **Comissioning and deactivation of ambulance resources**: in order to dispatch ambulances with the proper configuration to incidents it is important for the *CAD software* to know when crew members and equipment get comissioned to any given ambulance (i.e., they become part of the ambulance configuration) and when they get (de)activated (e.g., crew members can take time off work, equipment can go to repair).
  - You's $i^\star$ analysis did not consider this as a dependency, which probably means this is directly handled by the *CAD software* or it is not considered at all (maybe ambulances in London always have the same configuration);
  - For our CAD system-to-be this would be yet another functionality present in the ambulance's *MDTs*. Crew members could check in/out of work and equipment could be (de)activated directly at the ambulance, which would transmit the data back to the *CAD software* in the same fashion as before.
- **Support functions**: section 2.1.2 also mentions assumptions on the initial data about the existing domain entities being given and events of ambulance creation and comissioning (to a station) being received. While figure 2.3 does not contain dependencies to represent these assumptions either, in our experiments we assume the existence of a *CAD support software* component that allows the emergency service's managers to input the initial data and further create and comission ambulances;
- **Serviced region's map**: many of the CAD functions, such as displaying the location of ambulances and identifying if they are going on the right direction, depend on a up-to-date map of the serviced region's streets and buildings. The report on the inquiry into the LAS [swt, 1993] lists "gazetteer and mapping software" as part of the LAS-CAD system. In our experiments we also assume the existence of such software.

Using the information from this and the previous section as stakeholder requirements for a CAD system-to-be, the next section presents goal models for this system. Such models are required by our proposal for the design of adaptive systems and are extended according to our approach in the following chapters.

2.3 Goal Models for the CAD

Based on the requirements that have been elicited in the previous sections, along with complimentary information about the activity of computer-aided dispatch, this section finally presents the goal model that will be used as basis for the development of the Adaptive CAD (A-CAD) software. Figure 2.4 (page 17) shows the system-level goal model for the CAD system-to-be.

This model is loosely based on the $i^\star$ strategic rationale model [Yu *et al.*, 2011] and uses the same concepts as many Goal-Oriented Requirements Engineering (GORE) approaches: goals, softgoals, quality constraints (QCs) and domain assumptions (DAs) [Jureta *et al.*, 2008]. However, note that we represent AND/OR *refinement* relations, avoiding the term *decomposition* as it usually carries a part-whole semantic which would constrain its use among elements of the same kind[1] (i.e., goal to goal, task to task, etc.). A refinement relation, on the other hand, can be applied between a goal and a task or a goal and a domain assumption and indicate how to satisfy the parent element: the goal is satisfied if all (AND refinement) or any (OR refinement) of its children are satisfied. In their turns, tasks are satisfied if they are executed successfully and domain assumptions are satisfied if they hold (the affirmation is true) while the user is pursuing its parent goal.

The model, thus, represents the requirements of the CAD software in a hierarchical goal-based structure of satisfiability with obvious Boolean semantics. For example, we have previously mentioned in this report that the main objective of a CAD system is to *generate optimized dispatching instructions*. To satisfy this goal, the CAD software must satisfy all of its subgoals, namely: *call taking*, *resource identification*, *resource mobilization* and *incident update*. Domain assumptions, such as those described in section 2.2, are also shown in the model and should be true in order to satisfy their parent goals. For instance, it is assumed that *resource data* and the *gazetteer* (map data) are up-to-date and provided by their respective support systems in order for the CAD software to satisfy its main objective.

Furthermore, goals and tasks in the figure were modeled using different border thickness to indicate those that are initiated by the staff member — thin border — and those that are pursued automatically by the CAD system itself — thick border. For instance, a staff member should open the appropriate form in the system to *register the call*. However, once the address of the call has been registered, the CAD system can autonomously *search for duplicates*.

In the following paragraphs, we describe the goal model of figure 2.4, associating its elements with the requirements that were described back in section 2.1.3. The requirements IDs are shown between square brackets (e.g., [GR-1]). Not by chance, these requirements are associated with the tasks in the model, as they represent a sequence of steps an actor (human or system) can perform to fulfill them. Moreover, all tasks in the model are associated with a requirement, showing that there are no tasks without purpose here.

Call taking, an activity performed mostly by staff members, consists on responding calls to the emergency service (task performed outside the system and, thus, not shown in the model), *registering* them in the CAD system, *confirming* that they are indeed emergency calls [GR-3] and *assigning* them to an incident. During registration, the system should try to *detect the caller's location* [GR-2] to help staff expedite the activity of *inputting emergency information* [GR-1]. Analogously, the system should *search for duplicates* [GR-4] to help staff decide if they should *create new incident or assign as duplicate* to an existing one [GR-5].

Once a call has been taken and the incident registered, *resource identification* and *mobilization* are conducted for each incident. The former, performed by staff, consists on *specifying the configuration of ambulances* [GR-6] — i.e., indicate how many ambulances should be dispatched to the incident's location and what kinds of resources (human and equipment) are needed — and *confirming the incident* [GR-7] for dispatch by the system. *Resource mobilization* is then conducted by the system itself, *determining the best ambulance* [GR-8] from those available and based on the provided configuration and *informing stations / ambulances* about dispatch instructions [GR-9].

While *call taking* should be achieved for each call and *resource identification* and *mobilization* achieved for each incident, *incident update* is a goal that should be constantly maintained by the system. Categorization

[1] One could argue that it makes no sense to consider a task or a domain assumption a part of a goal. In effect, we have received such criticism in the past, in more than one occasion.

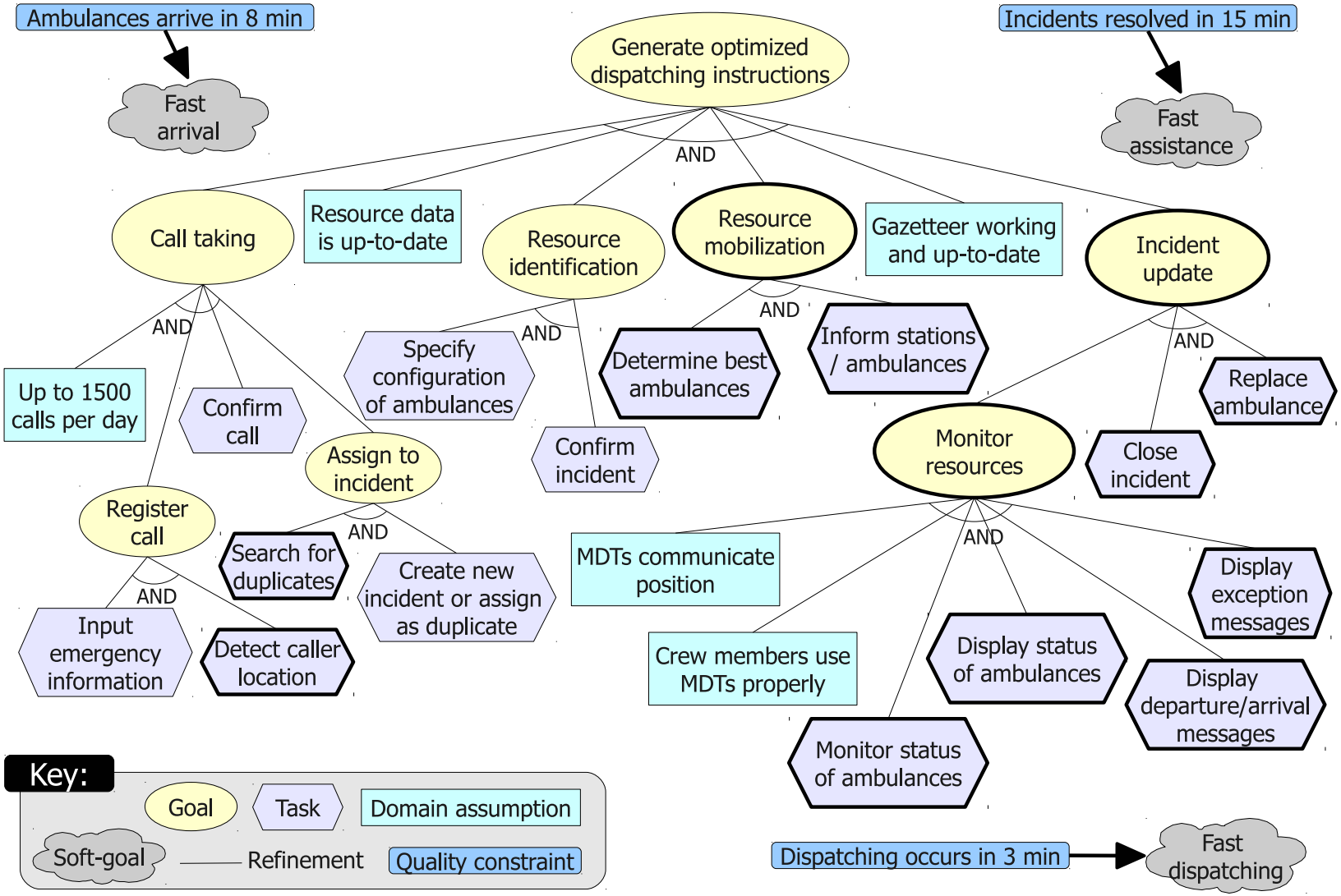

Figure 2.4: System-level goals for the CAD system-to-be.

of goals into **achieve** and **maintain** goals have been proposed in previous works in the area of agents and multi-agent systems [Dastani *et al.*, 2006; Morandini *et al.*, 2009]. This means that the CAD system should attempt to satisfy this goal sub-tree periodically, at every t units of time (t to be specified during design).

To satisfy *incident update*, then, the CAD system should *monitor resources*, *close incidents* [GR-10] when the ambulances are released and *replace ambulances* [GR-11] that break down during service. *Monitoring resources* consists on *monitoring the status of ambulances* [GR-13] — including all events described in section 2.1.2 — and displaying *the status of ambulances* [GR-14], *departure/arrival messages* [GR-15] and eventual *exception messages* [GR-16, GR-17, GR-18].

For resource monitoring to work, the CAD system depends on a couple of assumptions being true. First, *MDTs [should] communicate position* of busy (engaged) ambulances at regular intervals of time (at every 13 seconds, as specified in section 2.1.2). Second, it is assumed that *crew members use MDTs properly* to notify about events in the ambulance statechart (also see section 2.1.2) that cannot be triggered automatically by the ambulance's position, namely: comissioning, activation, deactivation, confirmation and release. Position and status of ambulances are needed in order to calculate the best ambulance to be assigned at any given time.

Finally, figure 2.4 also shows three softgoals and their respective quality constraints (QCs) that refer to time-related requirements that have been elicited from the different publications about the LAS-CAD case study:

- Dispatching, the process that starts when a call is responded and ends when ambulances acknowledge the dispatching instructions, should be done in up to 3 minutes [GR-16];
- Once ambulances have acknowledged dispatching instructions, they should arrive at the incident's location in up to 8 minutes [GR-12];
- The total time of assistance, which starts when an ambulance acknowledges dispatching instructions and ends when they are released from the incident, should not take more than 15 minutes [GR-17].

It is important to note that the goal model of figure 2.4 does not use $i^\star$ contribution links to indicate goals that contribute positively or negatively to softgoals. While contribution links are very useful in many situations (e.g., deciding the best alternatives in OR-refinements using the NFR framework [Chung *et al.*, 1999]), they are not required thus far by our approach and, therefore, have been omitted. In effect, we have modeled the system requirements so far with no variability whatsoever: there are no OR-refinements in figure 2.4! This has been done on purpose to keep the model simpler at this stage and variability will be added to the requirements later during our approach.

In the next chapter, *Awareness Requirements* for the CAD system are elicited, modeled and formalized to indicate what are the things that should be monitored by the feedback loop in order to adapt to failures. In the sequence, chapter 4 shows the result of *System Identification* on the very same model, which will extend the model of figure 2.4 with OR-refinements to add variability.

Chapter 3

Awareness Requirements for the A-CAD

In the previous chapter, we have presented "vanilla" requirements for a Computer-aided Ambulance Dispatch (CAD) system. By "vanilla" we mean the requirements of the system-to-be that are not related to its desired adaptation capabilities. In other words, so far we have modeled the requirements of a CAD system that cannot adapt to any failures.

In this chapter, we start applying our approach for the development of adaptive systems to the CAD with the objective of developing an Adaptive CAD, or A-CAD. We start with the identification of *Awareness Requirements* (*AwReqs*) [Souza *et al.*, 2011b], which will later indicate to an adaptation framework what to monitor for in order to adapt [Souza and Mylopoulos, 2011].

It is important to point out, however, that we do not prescribe any specific order in the design of adaptive systems with respect to the activities proposed by our work. In some cases systems might have already been modeled with high variability in mind while in others it might make sense to perform System Identification before anything else. In this particular experiment, we started from a requirements model for the CAD system with no variability and proceeded to identifying *AwReqs* and, later on, performing System Identification (chapter 4), selection of adaptation algorithms (chapter 5) and further elicitation of *Evolution Requirements* (section 6). New *AwReqs* will be presented later in these chapter, showing that the approach can also be applied in an iterative fashion.

Therefore, in this chapter, we: analyze — again using the available publications on the LAS-CAD case as source [swt, 1993; Beynon-Davies, 1995; Breitman *et al.*, 1999; Finkelstein and Dowell, 1996; Kramer and Wolf, 1996] — what are some possible situations to which a CAD software might have to adapt (§3.1); then we model and formalize *AwReqs* to some of these situations as part of our experiment (§3.2).

3.1 Situations that Require Adaptation

The main aspect of the LAS-CAD system analyzed by the aforementioned publications [swt, 1993; Beynon-Davies, 1995; Breitman *et al.*, 1999; Finkelstein and Dowell, 1996; Kramer and Wolf, 1996] is its failure. In [Kramer and Wolf, 1996], for example, the focus of the analysis was to point out methods/techniques/tools that should be applied in dealing with systems such as the LAS-CAD, and what research should be conducted to help in the development of such applications in the future in order to avoid its failures.

Our approach in this report is to use self-adaptation to avoid the problems caused by failures such as those that afflicted the LAS-CAD. This chapter identifies requirements and assumptions that are critical to the success of the system in order to, in a later step, attach to them certain adaptation actions that would be taken whenever the system does not satisfy such requirements.

The list below contains some CAD-related failures which were considered as possible causes for the LAS-CAD demise:

- **Misusage**: lack of cooperation from staff and crew, ranging from wilful misusage to direct sabotage of the system; staff/crew members unfamiliar with the system or improperly trained to use it. This could cause crew members to use different ambulances or equipment than those specified in the dispatching instructions, crew members not pressing the appropriate buttons to confirm/release the dispatch, etc.;
- **Transmission problems**: delays or corruption of data during transmission from ambulances to the central CAD software caused by excess load on the communication infrastructure, interference with other equipment, bad coverage by the communication network in some areas (black spots), etc.;
- **Unreliable software**: errors or incorrect information produced by any of the softwares associated with the CAD system (see section 2.2);
- **Unfamiliar territory**: dispatching of crews to parts of the serviced region they were not familiar with, which also made them drive longer to go back to the station at the end of the shift. Can cause discontentment, which triggers misusage; and longer times to resolve the incident, which could trigger exception messages;
- **Stale ambulance information**: caused by transmission problems and/or system misusage can cause the system to generate dispatching instructions which are not optimal, causing other problems such as sending crews to unfamiliar territory;
- **MDT problems**: mobile data terminals that lock up, are not readable or malfunction due to poor installation or maintenance can cause transmission problems, misusage or stale information;
- **Slow response speed**: ambulances take too long to arrive due to other problems that were already cited. This could cause citizens to call the emergency service again, increasing the number of calls. This could also cause a flood of exception messages;
- **Flood of calls**: an average amount of calls is expected everyday, but for some reason this number can significantly increase at any given day (e.g., the LAS worked with an average of 1300-1600 emergency calls and received more than 1900 calls at the day of the failure);
- **Flood of exception messages**: exception messages should be generated when dispatching does not finish in 3 minutes [GR-16], ambulances are not released in 15 minutes [GR-17] or go the wrong way to the incident's location [GR-18]. Other errors, such as transmission problems, misusage and MDT problems could cause a flood of exceptions which hinder the work of the staff.

In the next section, *AwReqs* are modeled in order to identify, through monitoring of the CAD system, some of these problems.

3.2 Awareness Requirements

The first step to tackling the problems identified in the previous subsection is to be aware when they happen. We have thus identified 12 *AwReqs* for the A-CAD, covering most of these problems. It is important to note, however, that this list is not meant to be exhaustive. The purpose of this experiment is to demonstrate that *AwReqs* can help avoiding a complete system failure by adapting to some of the situations that contributed to the LAS-CAD demise. To develop an Adaptive CAD that would be used in practice in a big city like London would most certainly require a lot more effort and elicit many other *AwReqs* in the process.

Figure 3.1 (page 21) shows the goal model for the CAD previouly presented in figure 2.4 (page 17), with added *AwReqs*. Moreover, a new task — *Get feedback*, under goal *Resource mobilization* — was also added to cope with the *unfamiliar territory* problem, as will be discussed next. Table 3.1 (page 22) summarizes the elicited *AwReqs* and shows, for each of them, a short description, the CAD problem from which they originated and the pattern that represents them, as proposed in [Souza *et al.*, 2011b].

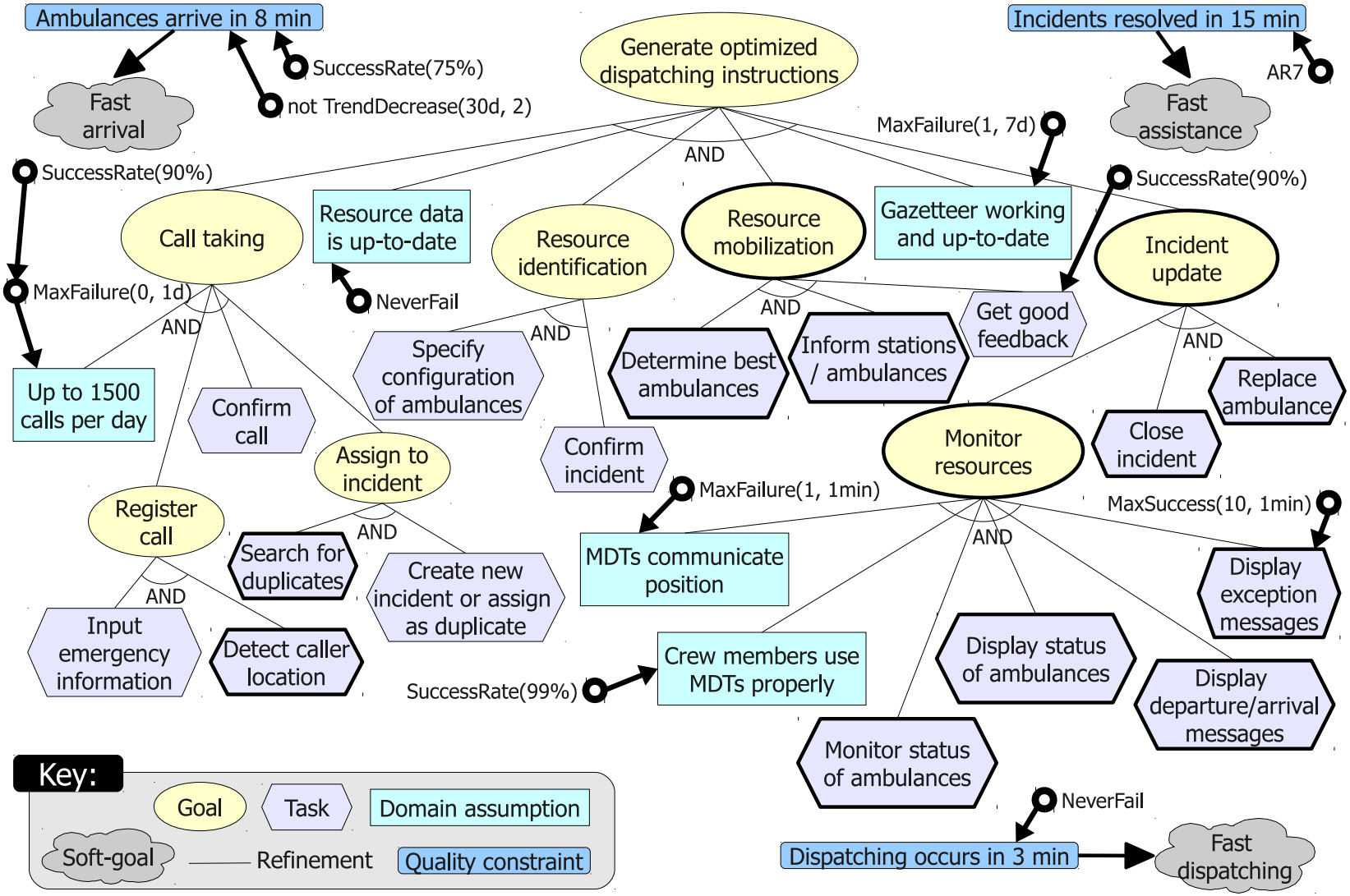

Figure 3.1: Goal model for the A-CAD system-to-be, with the elicited *AwReqs*.

Id	Description	Aware of	AwReq Pattern
AR1	DA *Up to 1500 calls received per day* should not fail at any given day.	Flood of calls	MaxFailure(D_MaxCalls, 0, 1d)
AR2	AwReq AR1 should succeed 90% of the time considering month periods.	Flood of calls	SuccessRate(AR1, 90%, 1m)
AR3	QC *Ambulances arrive in 8 min* should have 75% success rate.	ORCON	SuccessRate(Q_AmbArriv, 75%)
AR4	The success rate of QC *Ambulances arrive in 8 min* should not decrease 2 months in a row.	ORCON	not TrendDecrease(Q_AmbArriv, 30d, 2)
AR5	DA *Resource data is up-to-date* should always be true.	Unreliable software	NeverFail(D_DataUpd)
AR6	DA *Gazetteer working and up-to-date* should not be false more than once per week.	Unreliable software	MaxFailure(D_GazetUpd, 1, 7d)
AR7	Task *Monitor status of ambulances* should be successfully executed with status *released* within 12 minutes of the successful execution of task *Inform stations / ambulances*, for the same incident.	Slow response	–
AR8	DA *MDTs communicate position* should not be false more than once per minute	Trans-mission	MaxFailure(D_MDTPos, 1, 1min)
AR9	DA *Crew members use MDTs properly* should be true 99% of the time.	Misusage	SuccessRate(D_MDTUse, 99%)
AR10	Task *Display exception messages* should successfully execute no more than 10 times per minute.	Flood of messages	MaxSuccess(T_Except, 10, 1min)
AR11	QC *Dispatching occurs in 3 min* should never fail.	Slow response	NeverFail(Q_Dispatch)
AR12	Task *Get good feedback* should succeed 90% of the time.	Unfamiliar territory	SuccessRate(T_Feedback,90%)

Table 3.1: Summary of the *AwReqs* elicited for the A-CAD.

In the following paragraphs, we justify the elicitation of each *AwReq*, explaining the rationale for its elicitation based on the CAD problems listed in the previous section.

Flood of calls: the proposed solution for the CAD represented earlier by the goal model of figure 2.4 assumes that up to 1500 calls are received per day. If much more calls than that are received in any given day, something must be done so this flood of calls does not hinder the whole system, hence *AwReqs AR1* and *AR2* were elicited. The former indicates the domain assumption *Up to 1500 calls received per day* should not fail at any given day and could trigger adaptation actions to deal with a flood of calls in a particular day. The latter, by its turn, says that *AwReq AR1* should succeed 90% of the time considering month periods. This meta-*AwReq* raises awareness to the possibility that the average number of calls per day is raising and the system should evolve to normally support a bigger number of daily calls.

It is interesting to note that an aggregate *AwReq* `MaxFailure(D_MaxCalls, 0, 1d)` was used instead of a simple `NeverFail(D_MaxCalls)`. The reason for this is the following: failure of the former is registered once for the given period (1 day), whereas the latter is checked for every instance of the domain assumption verification, which would most likely be implemented at every call. Having the meta-*AwReq* applied to the aggregate *AwReq* conveys the intended meaning of *AR2*: in 90% of the days in a month, the number of calls did not overcome the 1500 threshold. If *AR1* were not aggregate, *AR2*'s percentage would be applied to the number of calls, not the number of days!

ORCON standard: this is not one of the problems listed in section 3.1, but a standard the LAS is supposed to follow, which we have mentioned in the beginning of section 2. Based on [Reynolds, b], we have elicited general requirement GR-12, which says that 75% of the ambulances should arrive within 8 minutes to the location of the incident. That is precisely what *AwReq AR3* imposes over the quality constraint *Ambulances arrive in 8 min*. Furthermore, *AwReq AR4* alerts staff about a decreasing trend in the success rate of the quality constraint, which could allow management to fix the causes of this problem before it goes lower the threshold imposed by ORCON.

Unreliable software: the CAD system depends on other software to work properly and if these are not reliable, problems are bound to arise. The standard CAD goal model (figure 2.4) thus assumes that the support system that provides data about resources and the gazetteer that provides maps of the serviced region are working properly. An *AwReq* was modeled for each of these systems: *AR5* imposes a *never fail* constraint on *Resource data is up-to-date*, whereas *AR6* tolerates one failure per week for the gazetteer.

Slow response: we divide the response of the ambulance service in two parts: dispatching, done by the staff at the central, and resolution, done by the crews in their ambulances. A constraint on the first part is depicted in the CAD model by quality constraint *Dispatching occurs in 3 min* and to indicate the criticality of this constraint, *AwReq AR11* indicates the constraint should never fail. For the second part, delta *AwReq AR7* was added to the A-CAD goal model. This *AwReq* does not have a pattern, as its definition is too specific to fit into one. It prescribes that, for each incident, the time between the ambulance or station being informed about the incident and the ambulance being released from the same incident should be no longer than 12 minutes. Counting the 3 minutes of dispatching, that gives a total of 15 minutes for incident response, as prescribed by quality constraint *Incidents resolved in 15 min*.

Transmission problems: the CAD goal model of figure 2.4 includes the domain assumption *MDTs communicate position*, because current position of each ambulance is essential to a proper ambulance dispatch. *AwReq AR8* establishes, then, that this assumption can fail at most once per minute.

Misusage: for the CAD to work properly, it is also assumed that *Crew members use MDTs properly*. The criticality of this domain assumption is the reason for *AwReq AR9*, which prescribes a 99% success rate for it.

Flood of messages: task *Display exception messages* adds to the CAD the capability of alerting the staff in case of different problems in the ambulance service. To cope with a possible flood of such alerts that hinders staff work, an *AwReq* was added to the amount of time this tasks succeeds in its execution. *AR10* indicates that the task should succeed at most 10 times per minute.

Unfamiliar territory: to be aware if ambulance crews are operating outside of their usual sector, a new task was added to the goal model of the CAD. *Get good feedback*, under goal *Resource mobilization*, succeeds if the crew indicates that the incident was correctly dispatched to them. Then, *AwReq AR12* establishes a

90% success rate for this task, which would alert management if more than 10% of the incidents were judged to be badly dispatched.

We can see in table 3.1 that almost half of the *AwReqs* elicited for the A-CAD impose constraints on domain assumptions being true, which denotes the importance of adapting to changes in the environment in which the A-CAD operates. Checking if a domain assumption is true, however, may not be a trivial thing. Therefore, the following lists specifies how each of the domain assumptions should be checked:

- **Up to 1500 calls received per day:** this is the simplest assumption to be checked, as it refers to calls, which is one of the domain entities of the CAD. There are many ways of keeping the count of how many calls there have been during each 24 hour period (e.g., a query on a database of calls);
- **Resource data is up-to-date:** this assumption is deemed false if any crew or staff member reports inconsistencies between the information shown by the system and reality;
- **Gazetteer working and up-to-date:** this is checked in the same fashion as the previous assumption (data is up-to-date), plus it should be verified that the gazetteer system responds whenever it is queried;
- **MDTs communicate position:** the CAD should check that all busy (engaged) ambulances report their position at every 13 seconds;
- **Crew members use MDTs properly:** the MDT should detect and warn the CAD when things are done in violation of the proper protocol. For instance, an ambulance should not leave the station without confirmation (an incident has been assigned to it) or deactivation (for repair, etc.).

Finally, in order to avoid possible ambiguity from reading the *AwReqs*' descriptions in table 3.1, each *AwReq* has been formalized in OCL_{TM}, as proposed in [Souza *et al.*, 2011b]. This formalization, shown in listing 3.1, is intended to be a formal documentation of each *AwReq* for the developers and later will have to be adapted to work with the monitoring framework (this is also discussed in [Souza *et al.*, 2011b]). Last but not least, the formalization assumes the existance of a class `DateUtil` which is capable of comparing a given timestamp to the current one or with another timestamp.

Listing 3.1: Formalization of the A-CAD's *AwReqs* in OCL_{TM}.

```
1  package acad
2
3  -- AwReq AR1: domain assumption 'Up to 1500 calls received per day' should always
       be true.
4  context D_MaxCalls
5    inv AR1: never(self.oclInState(Failed))
6
7  -- AwReq AR2: AwReq 'AR1' should succeed 95% of the time considering month periods
       .
8  context AR1
9    def: all : Set = AR1.allInstances()
10   def: month : Set = all->select(x | DateUtil.diff(x.time, DateUtil.DAYS) <= 30)
11   def: monthSuccess : Set = month->select(x | x.oclInState(Succeeded))
12   inv AR2: always(monthSuccess->size() / month->size() >= 0.95)
13
14 -- AwReq AR3: quality constraint 'Ambulances arrive in 8 min' should have 75%
       success rate.
15 context Q_AmbArriv
16   def: all : Set = Q_AmbArriv.allInstances()
17   def: success : Set = all->select(x | x.oclInState(Succeeded))
18   inv AR3: always(success->size() / all->size() >= 0.75)
19
20 -- AwReq AR4: the success rate of quality constraint 'Ambulances arrive in 8 min'
       should not decrease 2 months in a row.
21 context Q_AmbArriv
22   def: all : Set = Q_AmbArriv.allInstances()
23   def: m1 : Set = all->select(x | DateUtil.diff(x.time, DateUtil.DAYS) <= 30)
24   def: m2 : Set = all->select(x | (DateUtil.diff(x.time, DateUtil.DAYS) > 30) and
         (DateUtil.diff(x.time, DateUtil.DAYS) <= 60))
```

```
25    def: m3 : Set = all->select(x | (DateUtil.diff(x.time, DateUtil.DAYS) > 60) and
          (DateUtil.diff(x.time, DateUtil.DAYS) <= 90))
26    def: success1 : Set = m1->select(x | x.oclInState(Succeeded))
27    def: success2 : Set = m2->select(x | x.oclInState(Succeeded))
28    def: success3 : Set = m3->select(x | x.oclInState(Succeeded))
29    def: rate1 : Double = success1->size() / m1->size()
30    def: rate2 : Double = success2->size() / m2->size()
31    def: rate3 : Double = success3->size() / m3->size()
32    inv AR4: never((rate1 < rate2) and (rate2 < rate3))
33
34  -- AwReq AR5: domain assumption 'Resource data is up-to-date' should always be
        true.
35  context D_DataUpd
36    inv AR5: never(self.oclInState(Failed))
37
38  -- AwReq AR6: domain assumption 'Gazetteer working and up-to-date' should not be
        false more than once per week.
39  context D_GazetUpd
40    def: all : Set = D_GazetUpd.allInstances()
41    def: week : Set = all->select(x | DateUtil.diff(x.time, DateUtil.DAYS) <= 7)
42    def: weekFail : Set = week->select(x | x.oclInState(Failed))
43    inv AR6: always(weekFail.size() <= 1)
44
45  -- AwReq AR7: task 'Monitor status of ambulances' should be successfully executed
        with status 'released' within 12 minutes of the successful execution of task '
        Inform stations/ambulances', for the same incident.
46  context T_MonitorStatus
47    def: related : Set = T-InformAmbs.allInstances()->select(x | x.argument("
          incident") = self.argument("incident"))
48    inv AR7: eventually(self.argument("status") = "released") and never(related->
          exists(x | DateUtil.diff(x.time, self.time, DateUtils.MINUTES) > 12))
49
50  -- AwReq AR8: domain assumption 'MDTs communicate position' should not be false
        more than once per minute.
51  context D_MDTPos
52    def: all : Set = D_MDTPos.allInstances()
53    def: minute : Set = all->select(x | DateUtil.diff(x.time, DateUtil.SECONDS) <=
          60)
54    def: minuteFail : Set = minute->select(x | x.oclInState(Failed))
55    inv AR8: always(minuteFail.size() <= 1)
56
57  -- AwReq AR9: domain assumption 'Crew members use MDTs properly' should be true
        99% of the time.
58  context D_MDTUse
59    def: all : Set = D_MDTUse.allInstances()
60    def: success : Set = all->select(x | x.oclInState(Succeeded))
61    inv AR9: always(success->size() / all->size() >= 0.99)
62
63  -- AwReq AR10: task 'Display exception messages' should successfully execute no
        more than 10 times per minute.
64  context T_Except
65    def: all : Set = T_Except.allInstances()
66    def: minute : Set = all->select(x | DateUtil.diff(x.time, DateUtil.SECONDS) <=
          60)
67    def: minuteSuccess : Set = minute->select(x | x.oclInState(Succeeded))
68    inv AR10: always(minuteSuccess.size() <= 10)
69
70  -- AwReq AR11: quality constraint 'Dispatching occurs in 3 min' should never fail.
71  context Q_Dispatch
72    inv AR11: never(self.oclInState(Failed))
73
74  -- AwReq AR12: task 'Get good feedback' should succeed 90% of the time.
75  context T_Feedback
76    def: all : Set = T_Feedback.allInstances()
77    def: success : Set = all->select(x | x.oclInState(Succeeded))
78    inv AR12: always(success->size() / all->size() >= 0.9)
79  endpackage
```

The *AwReqs* presented in this chapter allow a monitoring infrastructure to be aware of situations in

which the system would have to adapt in order to avoid overall failures. The next chapter presents the result of System Identification [Souza *et al.*, 2011a] on the A-CAD, which recognizes system parameters that can be changed and how this change affects the success of the system in meeting its requirements. With these results available, adaptation algorithms are selected to deal with the failure of some of the elicited *AwReqs* in chapter 5, whereas some other *AwReqs* can only be handled by Evolution Requirements (chapter 6).

Chapter 4

System Identification for the A-CAD

The first step in a feedback-loop-based adaptation cycle is to monitor for the conditions to which the system has to adapt. The *Awareness Requirements* (*AwReqs*) elicited in the previous chapter represent a solution to that step. Then, once the system (or a framework around the system) knows a situation requiring adaptation has taken place, there are many different ways of performing the actual adaptation.

One possibility is to reconfigure the system, i.e., change one or more system parameters that could help reconcile the system with its requirements. There exist previous works on using parameters for run-time reconciliation (e.g., [Feather *et al.*, 1998]) and exploring variability at requirements (i.e., OR-refinements in the goal model) for system reconfiguration (e.g., [Lapouchnian *et al.*, 2007; Wang and Mylopoulos, 2009]). However, goal models lack an important piece of information when performing such activities: how do changes on system parameters (including OR-refinements) affect the system output and its ability to meet its requirements?

In [Souza *et al.*, 2011a] we propose a language and a systematic process to conduct *System Identification* for adaptive systems. This activity is commonly performed for control systems with the purpose of answering the previous question. We have conducted *System Identification* for the A-CAD and, in this chapter, we provide the results for the last three of the four steps of the proposed systematic process: identify parameters (§4.1), identify differential relations (§4.2) and refine relations (§4.3).

The first step of the process — identify indicators — was conducted in the previous chapter, where *AwReqs* were elicited and added to the model. We use *AwReqs* as indicators in our approach in order to focus on the important values that the adaptive system should strive to achieve. Therefore, in this chapter we start with goal model G (which has been shown in figure 3.1) and the set of indicators I (the *AwReqs* that have been listed in table 3.1) as inputs. At the end of the process, we should have a parametrized specification of the system behavior $S = \{G, I, P, R(I, P)\}$, where P is the set of parameters, and $R(I, P)$ is the set of relations between indicators and parameters.

4.1 Parameters

As noted at the end of chapter 2, the goal model that has been elicited for the A-CAD has no variability whatsoever. We can see, in effect, that neither figure 2.4 nor figure 3.1 (same model with added *AwReqs*) have any *variation points* (i.e., OR-refinements) or *control variables*. Therefore, at this step of the process, system parameters could not be identified, but, instead, they actually had to be elicited.

We have, thus, analyzed the *AwReqs* elicited in the previous chapter (the set of indicators I) and tried to come up with possible variability scenarios that could help in case of *AwReq* failure. During this process, goal model G has been changed to accommodate eight new parameters: control variables *NoC*, *NoSM* and *LoA* and variation points *VP1* through *VP5*. For presentation reasons, we show the new goal model in four different figures, each of which focusing on a section of the model that has been changed to introduce new system parameters. Furthermore, the name of the *AwReqs* shown in these figures has been added next to

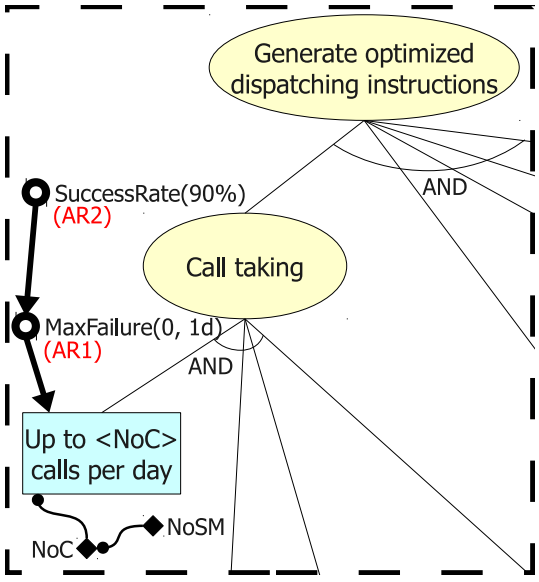

Figure 4.1: New parameters *NoC* (maximum *Number of Calls* that can be handled daily) and *NoSM* (*Number of Staff Members* working on the present day).

their graphical representation for an easier reference.

4.1.1 *NoC* and *NoSM*

Figure 4.1 shows control variables *NoC* — maximum *Number of Calls* that can be handled daily — and *NoSM* — *Number of Staff Members* working on the present day. The domain assumption *Up to 1500 calls per day* has also been changed and now reads *Up to* $\langle NoC \rangle$ *calls per day*, meaning the assumption is checked against the *NoC* parameter and is no longer fixed at 1500 calls.

However, parameter *NoC* is a special kind of parameter that cannot be set directly. Instead, it is declared as a direct function of another parameter, namely *NoSM*. The maximum number of calls the service can take in a day is then calculated based on the number of staff members working on that specific day. Hence, by changing the number of staff members on duty one can affect positively or negatively the success rate of the domain assumption, affecting, thus, indicators (*AwReqs*) *AR1* and *AR2*.

This case is particularly interesting because it is one kind of chain reaction among parameters. This is a particular kind, in which a parameter (e.g., *NoC*) can be defined by a precise function of another (e.g., *NoSM*). Other types of chain reaction could also be qualitative. For instance, we might want to denote that a specific parameter (e.g., number of servers) affects the performance of the system, which in turn affects client satisfaction. In this case, we cannot define one parameter as a function of the other, but just indicate there is a qualitative relation between them. Chain reactions were not analyzed in depth in [Souza *et al.*, 2011a] and, therefore, can be subject of a future technical paper.

4.1.2 *LoA* and *VP1*

Figure 4.2 shows control variable *LoA* — *Level of Automation* of the dispatch procedure — and variation point *VP1*. As noted in the figure, *LoA* is an enumerated parameter that can assume one of three values: *manual* (dispatch will be done completely manually by staff members, communicating with ambulances and stations via radio), *automatic with confirmation* (dispatch orders are suggested by the A-CAD but are sent only if a staff member confirms that is indeed the best choice) or *automatic* (the A-CAD autonomously generates dispatch orders and send them to ambulances/stations).

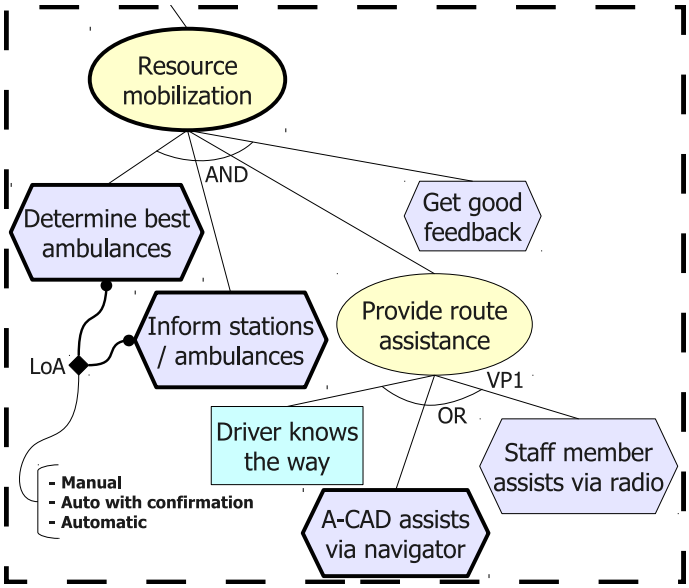

Figure 4.2: New parameters *LoA* (*Level of Automation*) and *VP1*.

Changing the value of this parameter can affect indicators *AR3*, *AR4*, *AR9* and *AR12*. The rationale behind this effect is that switching to a more manual process helps solve problems that are too complicated for the A-CAD's reasoning capabilities. The interaction between the staff member in charge of the dispatch and crew members in ambulances and stations can make sure the crew agrees with the dispatching instructions (increasing the success rate of *Get good feedback*), allows for the staff member to assist the crew about the use of the MDT (increasing the success rate of *Crew members use MDTs properly*) and ultimately aid in achieving the ORCON standards (higher success rates for *Ambulances arrive in 8 min*). Obviously the benefits do not come for free: the more manual the process is, the more time each staff member spends on each incident, which allows them to take less calls a day and makes dispatching more time-consuming.

The parameter *VP1*, by its turn, was elicited to provide alternatives for improving indicator *AR7*, which talks about the time crews take to resolve an incident once the dispatching information has been received by them. One way the A-CAD can help in this matter is to provide route assistance to ambulance drivers, so they can reach the incident's location and, whenever needed, take injured people to the hospital as fast as possible. Therefore the goal *Provide route assistance* has been added to *Resource mobilization*'s AND-refinement. This new sub-goal can be satisfied in three different ways: (a) assuming that the *Driver knows the way* and, thus, doing nothing; (b) having the *A-CAD assist via navigator*; or (c) having the *Staff member assist via radio*. Again, here there is a trade-off between how personalized this assistance is and how much time it takes from staff members.

Indicator *AR7* could also be affected by changes in *LoA*. Once again, having a more direct communication between staff member and ambulance crew can help determine the best way to reach the incident's location and resolve it.

4.1.3 *VP2* and *VP3*

Figure 4.3 shows variation points *VP2* and *VP3*, which have been elicited along with a new subtree of the main goal of the system in order to include an alternative to the gazetteer for map provision, therefore affecting indicator *AR6*. The goal *Obtain map information* was added to the model where the domain assumption *Gazetteer working and up-to-date* used to be, making the assumption one of its children in OR-refinement *VP2*.

The other child — goal *Obtain map info manually* — is, in effect, the alternative to using the gazetteer

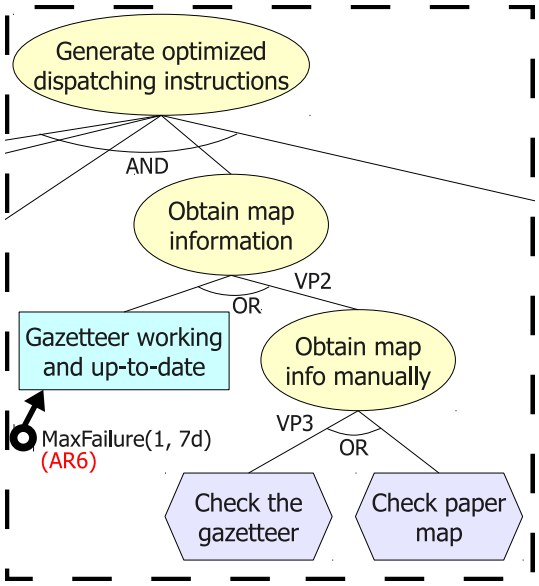

Figure 4.3: New parameters *VP2* and *VP3*.

automatically. When this alternative is selected, a staff member is supposed to check the map to determine the exact location of the incident and the best ambulances to be dispatched. This goal is further refined into two tasks in *VP3*: the staff member can either *Check the gazetteer* itself or, in extreme cases, *Check paper map*.

Like in previous parameters, the alternatives range from highly automated to highly manual, providing a trade-off between avoiding software mistakes and the time taken by staff members for each dispatch.

4.1.4 *VP4* and *VP5*

Finally, figure 4.4 shows variation points *VP4* and *VP5*. In the former, a new goal — *Update position of engaged ambulances* — has replaced domain assumption *MDTs communicate position*, making it one of its children in an OR-refinement. The other child is task *Crew updates position via radio*, which consists on a manual fallback for when MDTs are not working properly, thus affecting indicator *AR8*. Radio contact between crew and staff also allows crew members to avoid using the MDT altogether, passing all information directly via voice. Therefore, this parameter also affects indicator *AR9*.

Parameter *VP5* provides a simple solution to the flood of exception messages, monitored by indicator *AR10*: add messages to a message queue instead of showing them directly. To this end, the task *Display exception messages* has been replaced by a homonymous goal, which is now its parent, having on the other side of the OR-refinement the task *Add to message queue*.

4.2 Differential Relations

In the previous step of the systematic System Identification process, we have identified parameters of the system that can be modified at runtime in order to improve some of the system's indicators that were elicited earlier. In this step, we formalize the qualitative effect between parameters and indicators through differential relations presented in table 4.1.

For the relations that refer to enumerated control variable *LoA* to make any sense, it is required that a total order of the parameter's enumerated values be provided. This order shall be as follows: $\langle manual \rangle \prec \langle auto\ with\ confirmation \rangle \prec \langle automatic \rangle$. Variation points assume their default order, i.e., ascending from

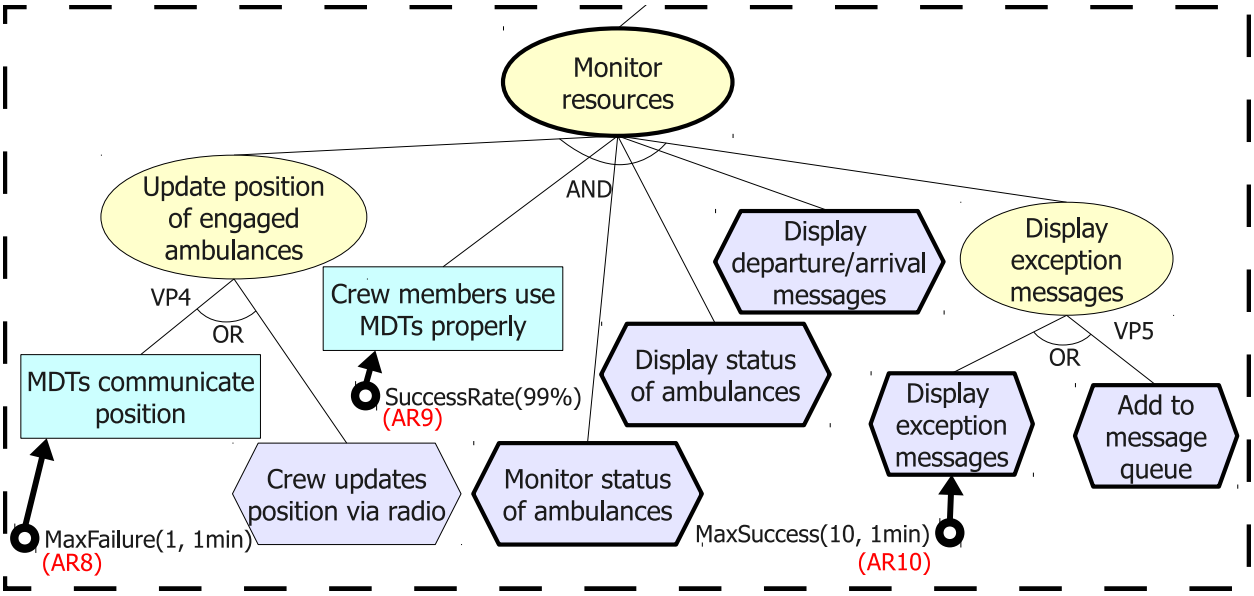

Figure 4.4: New parameters *VP4* and *VP5*.

$$\Delta(AR1/NoSM)[0, maxSM] > 0 \quad (4.1)$$
$$\Delta(AR2/NoSM)[0, maxSM] > 0 \quad (4.2)$$
$$\Delta(AR3/LoA) < 0 \quad (4.3)$$
$$\Delta(AR4/LoA) < 0 \quad (4.4)$$
$$\Delta(AR9/LoA) < 0 \quad (4.5)$$
$$\Delta(AR11/LoA) > 0 \quad (4.6)$$
$$\Delta(AR12/LoA) < 0 \quad (4.7)$$
$$\Delta(AR3/VP1) > 0 \quad (4.8)$$
$$\Delta(AR4/VP1) > 0 \quad (4.9)$$
$$\Delta(AR7/VP1) > 0 \Delta(AR11/VP1) < 0 \quad (4.10)$$
$$(4.11)$$

$$\Delta(AR6/VP2) > 0 \quad (4.12)$$
$$\Delta(AR11/VP2) < 0 \quad (4.13)$$
$$\Delta(AR12/VP2) > 0 \quad (4.14)$$
$$\Delta(AR6/VP3) > 0 \quad (4.15)$$
$$\Delta(AR11/VP3) < 0 \quad (4.16)$$
$$\Delta(AR12/VP3) > 0 \quad (4.17)$$
$$\Delta(AR8/VP4) > 0 \quad (4.18)$$
$$\Delta(AR9/VP4) > 0 \quad (4.19)$$
$$\Delta(AR11/VP4) < 0 \quad (4.20)$$
$$\Delta(AR10/VP5) > 0 \quad (4.21)$$

Table 4.1: Differential relations added to the model during System Identification.

left to right according to their position in the model.

Most of the effects formalized by the relations were discussed in the previous step because they motivated the very elicitation of the parameters. However, at this step of the process each of the elicited parameters were again analyzed and compared to each system indicator to make sure all effects were identified and modeled. This analysis resulted in the identification of the following new relations:

- All parameters, with the exception of *VP5* and *NoSM*, have an effect on indicator *AR11*, which says that quality constraint *Dispatching occurs in 3 min* should never fail. A higher level of automation (*LoA*) improves it (cf. 4.6), whereas choosing to do tasks manually with the involvement of a staff member (*LoA* and *VP1* through *VP4*) has a negative effect on it (cf. 4.6, 4.10, 4.13, 4.16, 4.20);
- Variation points *VP2* and *VP3* also have an effect on indicator *AR12*, which states that task *Get good feedback* (for the dispatch choice) should succeed 90% of the time (cf. 4.14, 4.17). The rationale is that obtaining map information manually may help in the process of choosing the best ambulance to dispatch;
- Parameter *VP1*, which indicates the kind of route assistance to give ambulance drivers, also affects indicators *AR3* and *AR4*, which refer to quality constraint *Ambulances arrive in 8 min* (cf. 4.8, 4.9). Providing route assistance may help satisfy ORCON standards.

Another activity of this step is the identification of landmark values for numeric control variables, that establish intervals in which the identified relations can be applied. The only applicable numeric parameter is *NoSM* and all of its relations (cf. 4.1, 4.2, **??**) are valid in the interval $[0, maxSM]$, $maxSM$ being a qualitative value that represents the maximum number of staff members the ambulance service infrastructure can hold. *NoC* is also numeric, but it is not applicable as cannot be directly modified (it is a function of *NoSM*). Hence, no differential relation or landmark value were identified for it.

4.2.1 Trade-offs

Given the relations in table 4.1 and assuming each of the parameters has been assigned an initial value it is possible to use the information of how parameters affect indicators at runtime to change their values whenever there is a system failure, i.e., when there are indications of requirements divergence. This change, however, may require some kind of trade-off analysis at runtime. For instance, as stated before, choosing to do dispatching tasks manually (*LoA* and *VP1* through *VP4*) might improve several different indicators, but at the cost of having a negative impact over *AR11*.

A careful analysis of these relations, however, will indicate that there are some indicators missing in our model of the A-CAD. After all, *AR11* is the only indicator that receives a negative impact from some of the parameter changes and this impact can be remedied by increasing *NoSM*. Therefore, why not setting everything to manual and increasing the number of staff members to the maximum? Also, if switching *VP5* to *Add to message queue* solves the flood of messages problem, why not use it exclusively?

The answer to these questions relies on some implicit quality indicators, i.e., non-functional requirements that have not been explicitly elicited. Clearly, increasing the number of staff members also increases the cost of the overall system, whereas the use of a message queue might be avoided unless strictly necessary because of user friendliness concerns. For the purposes of this experiment, we assume the existance of the following stakeholder requirements:

- We should aim for *Low cost* (softgoal). In particular, stakeholders would like *Montlhy cost below* €$\langle MaxCost \rangle$ (quality constraint), where $MaxCost$ is a qualitative variable representing the maximum amount of money that should be spent for the ambulance service at any given month. This requirement should never fail;
- The A-CAD should have *User friendly GUIs* (softgoal). In particular, it should be the case that *Staff members see messages in* $\langle S \rangle$ *secs*, where S is a qualitative variable representing the maximum amount

$$\Delta\left(AR13/NoSM\right) < 0 \tag{4.22}$$
$$\Delta\left(AR14/VP5\right) < 0 \tag{4.23}$$

Table 4.2: Differential relations for the newly elicited indicators *AR13* and *AR14*.

of seconds between message generation and message display. For this requirement, stakeholders would like it to fail no more than $NoSM$ per week, meaning that at most there should be, in average, one failure per staff member working on the ambulance service.

Figure 4.5 (page 34) shows the complete goal model for the A-CAD after System Identification, including all parameters and the new requirements that have just been elicited. Given the new indicators (*AR13* and *AR14*), new differential relations were also identified and are displayed in table 4.2.

Finally, it is important to note that the resulting model is much simplified if compared with a real ambulance dispatch system. Taking the London Ambulance System as an example, there were probably many other softgoals and quality constraints to be elicited from the stakeholders, leading to more indicators (*AwReqs*), parameters and, as a consequence, more differential relations between indicators and parameters. The A-CAD was intentionally simplified for the purposes of this experiment (which is, after all, a lab demo and not a full-fledged case study with an industrial partner).

4.3 Refinement

The last step of System Identification is to refine the relations that were identified in the previous step of the process. According to [Souza *et al.*, 2011a], we should analyze indicators that have more than one relation associated to them and identify: (a) if any parameter has a greater effect over the indicator with respect to other parameters; and (b) what is the effect of combined parameters on the indicator. With respect to (a), to be able to compare relations that represent positive effect ($>$ operator) with the ones that represent negative effect ($<$ operator) we compare their absolute value, which results in comparing the effect of increasing parameters that have positive effect with the effect of decreasing parameters that have negative effect. Regarding (b), the default behavior is that homogeneous impact is additive, so unless stated otherwise, combining two or more parameter changes which contribute positively (negatively) to an indicator results in an even more positive (negative) effect.

Moreover, when comparing a numeric control variable with other parameters, one should specify the *unit of increment* of the variable in order to compare the changes of other parameters to a change of one unit of the numeric variable. For example, the unit for *NoSM* is *one staff member* — specified $U_{NoSM} = 1$ —, so when other parameters are compared with *NoSM*, they are comparing to "hiring or laying off one staff member". Enumerated control variables and variation points (which are themselves enumerated) have a default unit of increment of choosing the next value in their given order.

Table 4.3 shows the result of the refinement step for the A-CAD. Of the fourteen indicators, six had more than one parameter associated with them: *AR3*, *AR4*, *AR6*, *AR9*, *AR11* and *AR12*. All of them follow the default combination rules (homogeneous impact is additive) and no relation was added for *AR6* because *VP3* is only relevant if *VP2* is "increased" to pursue *Obtain map info manually* instead of assuming *Gazetteer working and up-to-date*.

Indicators *AR3* and *AR4* refer to quality constraint *Ambulances arrive in 8 min*, which is affected by parameters *VP1* and *LoA*. Analyzing these two parameters we have concluded that providing route assistance to drivers has a greater positive effect on this indicator than executing tasks *Determine best ambulances* and *Inform stations / ambulances* using more manual procedures. This is represented in relations 4.24 and 4.25.

Indicator *AR6* is affected by variation points *VP2* and *VP3*. Note, however, that *VP3*'s value is only relevant when *VP2*'s choice is *Obtain map info manually*. Relation 4.25 represents this fact explicitly, although this is not strictly necessary, given that it can be inferred by the goal refinements by the runtime

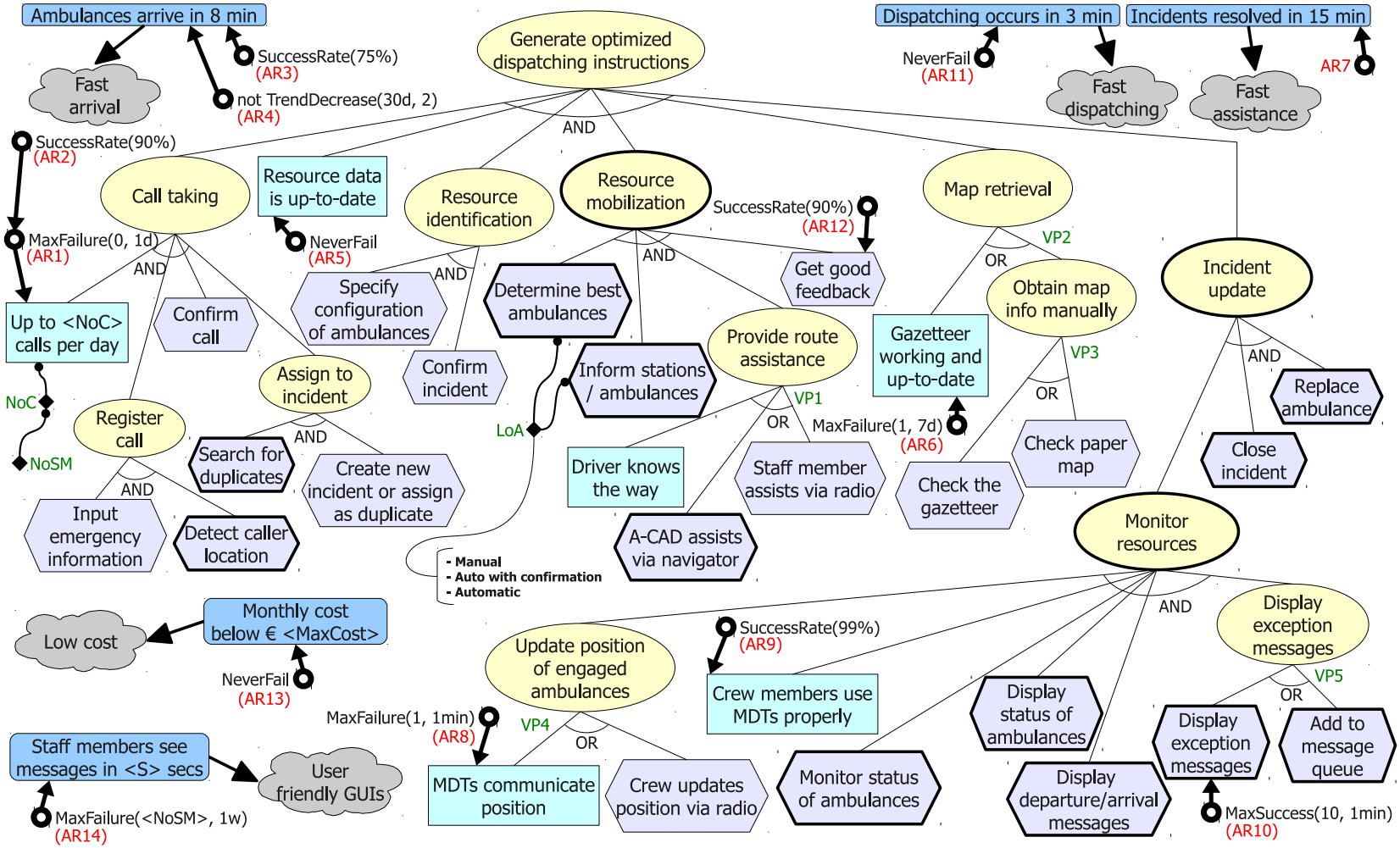

Figure 4.5: Goal model for the A-CAD system-to-be, after System Identification.

$$|\Delta(AR3/VP1)| > |\Delta(AR3/LoA)| \quad (4.24)$$
$$|\Delta(AR4/VP1)| > |\Delta(AR4/LoA)| \quad (4.25)$$
$$VP2 \neq \langle Obtain\ map\ info\ manually \rangle \rightarrow |\Delta(AR6/VP3)| = 0 \quad (4.26)$$
$$|\Delta(AR9/VP4)| > |\Delta(AR9/LoA)| \quad (4.27)$$
$$|\Delta(AR11/VP2)| > |\Delta(AR11/LoA)| > |\Delta(AR11/VP3)| > |\Delta(AR11/VP1)| > |\Delta(AR11/VP4)| \quad (4.28)$$
$$|\Delta(AR12/VP2)| \approx |\Delta(AR12/VP3)| \approx |\Delta(AR12/LoA)| \quad (4.29)$$

Table 4.3: Refinements for the differential relations of the A-CAD

framework. For the same reason, it does not make sense to compare *VP2* to *VP3*, because to change the latter one needs first to change the former.

AwReq AR9, which refers to domain assumption *Crew members use MDTs properly*, is affected by *VP4* and also *LoA*. Again, the choice at the variation point has a greter positive effect than changing the value of the controlled variable, as pointed out by relation 4.27. Having the crew update position via radio is more effective in helping them use MDTs than decreasing the *Level of Automation*.

The effects on indicator *AR11* are ordered in relation 4.28, showing that assuming the gazetteer works instead of checking for maps manually is the most time-saving change possible (*VP2*), followed by using a higher level of automation (*LoA*), checking the gazetteer instead of using manual maps (*VP3*), assisting drivers with automatic navigators or not at all (*VP1*) and, finally, reducing the minimum search time by 10 seconds (*MST*).

Lastly, relation 4.29 shows that changes in the parameters that affect indicator *AR12* are equally effective. Obviously, to "increase" *VP3* the system would have to "increase" *VP2* first, and their combined effect is greater than the effect of isolated parameters, as previously stated.

4.4 Reconfiguration

Wang & Mylopoulos [Wang and Mylopoulos, 2009] provide a GORE-based definition of a system *configuration*: "a set of tasks from a goal model which, when executed successfully in some order, lead to the satisfaction of the root goal". We add to this definition the values assigned to each *control variable* elicited during System Identification[1]. *Reconfiguration*, then, is the act of replacing the current configuration of the system with a new one with the purpose of overcoming or preventing a system failure (repairing, healing), or achieving better performance with respect to non-functional requirements (optimization).

Several proposals in the literature provide methods and algorithms that are capable of finding a new system configuration, given that requirements are represented as goal models. To cite a few:

- Wang and Mylopoulos [2009] propose algorithms that suggest a new configuration without the component that has been diagnosed as responsible for the failure;
- Nakagawa *et al.* [2011] developed a compiler that generates architectural configurations by performing conflict analysis on KAOS goal models [van Lamsweerde, 2009];
- Fu *et al.* [2010] use reconfiguration to repair systems based on an elaborate statemachine diagram that represents the life-cycle of goal instances at runtime;
- Peng *et al.* [2010] assign preference rankings to softgoals (which can be dynamically changed at runtime) and determine the best configuration using a SAT solver;

[1] As discussed in [Souza *et al.*, 2011a], however, control variables are just abstractions over large or repetitive variation points. Therefore, if each control variable were to be expanded to a variation point, the original definition in [Wang and Mylopoulos, 2009] could be applied, unaltered.

- Khan *et al.* [2008] apply Case-Based Reasoning to the problem of determining the best configuration;
- In our research group, Dalpiaz *et al.* [2012] propose an algorithm that finds all valid variants to satisfy a goal and compares them based on their cost (to compensate tasks that failed or the ones that already started and will be canceled) and benefit (e.g., contribution to softgoals). In [Dalpiaz *et al.*, 2010], reconfiguration is discussed in terms of interaction among autonomous, heterogeneous agents based on commitments.

Note that different reconfiguration algorithms may require different information from the model. For instance, Wang and Mylopoulos' proposal requires a goal model and a diagnosis pointing to the failing component, whereas Dalpiaz *et al.*'s framework needs the compensation/reaction costs associated wity tasks and priorities associated with softgoals.

In our research we have proposed a framework to perform qualitative adaptation [Souza *et al.*, 2012b], i.e., find a new configuration for the system using the qualitative information provided by System Identification. Given that this framework supports adaptation algorithms with different levels of model precision, the choice of algorithm to use (and the extra precision required by it) for each of the A-CAD's possible failures should also be specified. This is covered in the next chapter.

Chapter 5

Qualitative Adaptation of the A-CAD

The Awareness Requirements modeled in chapter 3, coupled with the parameters and differential relations elicited in chapter 4 provide us enough information to reason over the goal model to find out, for each of the A-CAD *AwReqs*, which (if any) parameters can be changed (and to which direction, i.e., increase or decrease) in order to try and improve the outcome of the system with respect to its *AwReqs*. At runtime, such algorithm could be run when an *AwReq* failure is detected with the purpose of adapting the system to this failure.

However, the qualitative information modeled during System Identification can come in different levels of precision. For instance, in 4.3 differential relations of the same indicator were compared amongst themselves in order to establish an order with respect to the magnitude of the effect parameter changes have on indicators. For most indicators an order was given, but for *AR12*, the information elicited was that all parameters have roughly the same impact on the indicator (relation 4.29). The result is that an adaptation algorithm can be more precise when dealing with most indicator failures, but when *AR12* fails, the choice of parameter to change will have to be random or arbitrary.

Given that the availability of more precise information can vary from one system to another or even change in time for the same system, in [Souza *et al.*, 2012b] we propose a framework that performs qualitative adaptation using the information from the requirements models that is able to accommodate varying levels of precision by offering different adaptation algorithms. For each *AwReq*, then, modelers should specify the adaptation algorithm to use, making sure all the required information for that algorithm are present in the models.

In section 5.2 we specify the adaptation algorithms for failures of the A-CAD. Before that, however, section 5.1 extends the A-CAD model produced during System Identification (c.f. figure 4.5, page 34) with new elements that will help us illustrate some adaptation algorithms.

5.1 Extending the A-CAD Models

To better illustrate some adaptation algorithms in our experiments, we have included a few new elements in the goal model of the A-CAD, resulting in the model shown in figure 5.1 (page 38). The new elements are numeric control variable *MST* (*Minimum Search Time*), softgoal *Unambiguity*, quality constraint *No unnecessary extra ambulances dispatched* and *AwReq AR16*[1].

The parameter *MST* represents the minimum amount of time (in seconds) staff members must dedicate to the task of searching for duplicates. This parameter is directly related to the new softgoal, *Unambiguity*: if staff members are forced to spend some time searching for duplicate calls, this will lower the probability

[1]Note that the newly added *AwReq* was called *AR16* and not *AR15* as it would have been expected. The reason is that *AR16* was defined in [Souza *et al.*, 2012b] (publication referred to by this chapter) whereas *AR15*, which will be presented later, was defined in [Souza *et al.*, 2012a] (subject of the next chapter), a publication that was written and submitted first. To keep the models of this report in sync with these publications, *AR16* was presented before *AR15* here.

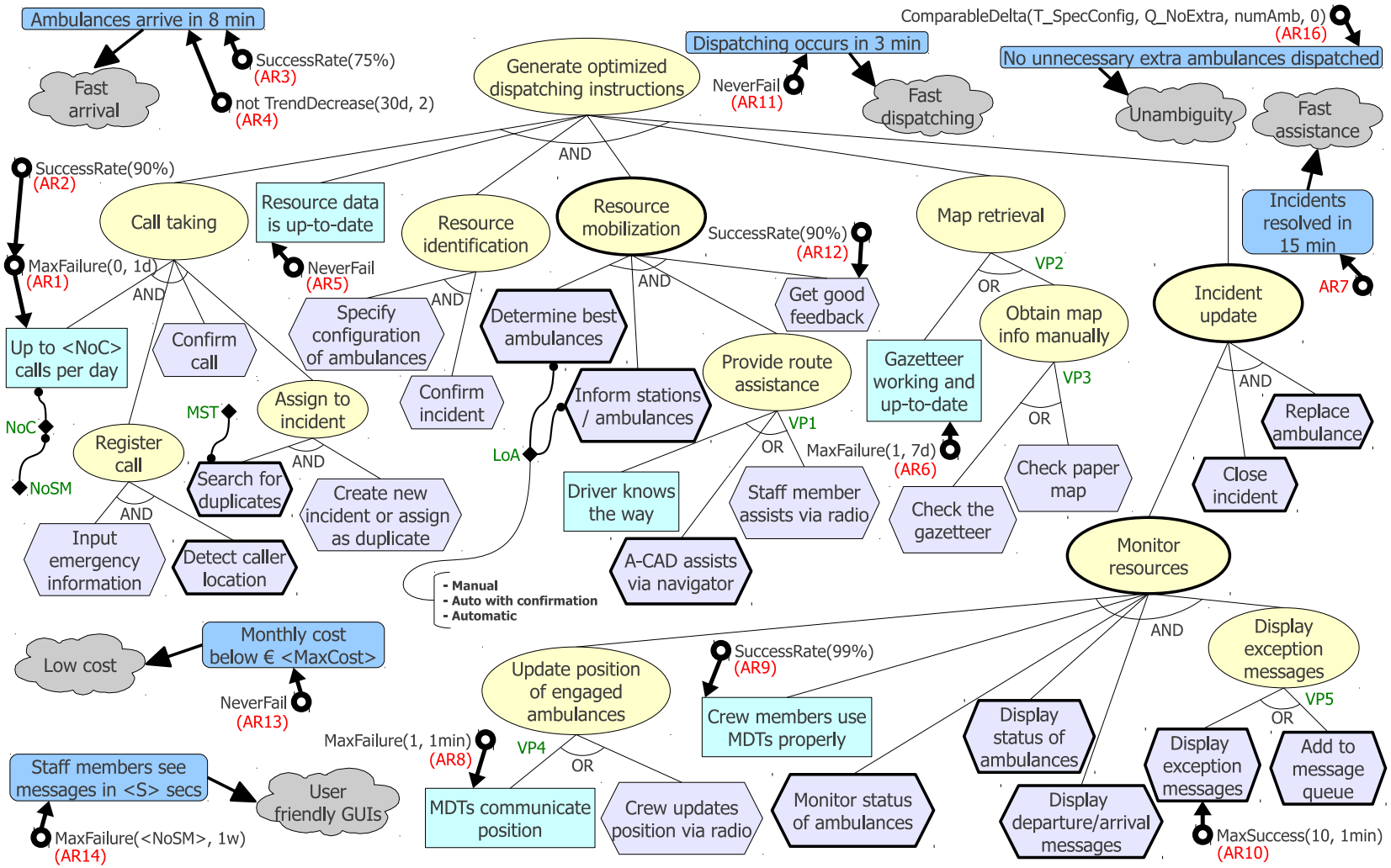

Figure 5.1: Goal model for the A-CAD system-to-be, extended to illustrate some adaptation algorithms.

$$\Delta\left(AR12/MST\right)[0,180] > 0 \tag{5.1}$$
$$\Delta\left(AR11/MST\right)[0,180] < 0 \tag{5.2}$$
$$\Delta\left(AR13/MST\right)[0,180] > 0 \tag{5.3}$$
$$\Delta\left(AR16/MST\right)[0,180] > 0 \tag{5.4}$$
$$\Delta\left(AR16/LoA\right) < 0 \tag{5.5}$$
$$\ldots > |\Delta\left(AR11/VP4\right)| > |\Delta\left(AR11/MST\right)| \tag{5.6}$$
$$|\Delta\left(AR12/VP2\right)| \approx |\Delta\left(AR12/VP3\right)| \approx |\Delta\left(AR12/LoA\right)| \approx |\Delta\left(AR12/MST\right)| \tag{5.7}$$
$$|\Delta\left(AR13/NoSM\right)| > |\Delta\left(AR13/MST\right)| \tag{5.8}$$
$$|\Delta\left(AR16/MST\right)| > |\Delta\left(AR16/LoA\right)| \tag{5.9}$$

Table 5.1: New differential relations and refinements, after the inclusion of *MST* and *AR16*.

of missing a duplicate and registering a call as a new incident, which would in turn result in duplicate (ambiguous) dispatch. On the other hand, the trade-off here is that higher values for *MST* may imply harming softgoals such as *Fast arrival* and *Fast dispatching*.

Table 5.1 shows the new differential relations and new and changed refinements added to the A-CAD model after the addition of the new elements. The following list describes the new/modified relations:

- Increasing the *Minimum Search Time* will affect negatively the success of quality constraint *Dispatching occurs in 3 min* (*AR11*) for an obvious reason: the time spent searching for duplicates could be spent with other tasks related to dispatching and incident resolution in order to finish them faster (relation 5.2);
- On the other hand, increasing *MST* affects positively *AR16* — the more time spent searching for duplicates, the less chance of an ambiguous dispatch (relation 5.4) —, *AR12* — duplicate dispatches will most likely get bad feedback from crew members who will be sent to assist an incident unnecessarily (relation 5.1) — and *AR13* — duplicate dispatches represent waste of resources, and therefore money (relation 5.3);
- The *Level of Automation* also affects `AR16` (i.e., quality constraint *No unnecessary extra ambulances dispatched*): on a more manual setting staff members can check amongst themselves if the dispatch they are currently doing is ambiguous and cancel one of the dispatches before ambulances are mobilized (relation 5.5);
- Regarding *AwReqs AR11* and *AR13*, parameter *MST* is the one with the lowest effect (relations 5.6 and 5.8). On the other hand, when dealing with *Unambiguity* (i.e., *AwReq AR16*), *MST* is better than *LoA* (relation 5.9). For *AR12*, all parameters have roughly the same effect, including the new parameter *MST* (relation 5.7).

5.2 Algorithm Specification for the A-CAD

Given the new elements and relations introduced in the previous section, we can now proceed to specify the adaptation algorithm that will be used for each *AwReq* failure in the A-CAD. Table 5.2 summarizes the choices of algorithms, whose rationale is explained in the following paragraphs.

AwReqs AR1 and *AR2* monitor if the domain assumption *Up to* $\langle NoC \rangle$ *calls per day* is true and the only way to improve the success rate of this assumption is by increasing the number of staff members (*NoSM*), which in turn automatically increases the number of calls (*NoC*) the service can take per day. Given that only one parameter is related to these *AwReqs* the default procedure (represented by $\varnothing$) will be used to deal

Table 5.2: Adaptation algorithms selected for each *AwReq* failure in the A-CAD.

AwReq	**Algorithm**	**Attributes**
AR1	∅	MT_{NoSM} = *5 days*
AR2	∅	MT_{NoSM} = *5 days*
AR3	{*Ordered Effect Parameter Choice*}	*order = descending*
AR4	{*Ordered Effect Parameter Choice*}	*order = ascending*
AR6	∅	$N = 2$
AR7	∅	
AR8	∅	*Immediate Resolution*
AR9	{*Ordered Effect Parameter Choice*}	*order = descending*
AR10	∅	*Immediate Resolution*
AR11	{*Oscillation Value Calculation*, *Oscillation Resolution Check*}	
	{*Ordered Effect Parameter Choice*}	*order = descending*
AR12	∅	
AR13	{*Ordered Effect Parameter Choice*}	*order = ascending* *repeat policy = max 2 times*
AR14	∅	*Immediate Resolution*
AR16	{*Ordered Effect Parameter Choice*}	*order = ascending*

with their failures. However, it is important to note that the *maturation time* of parameter *NoSM* is five days, meaning it takes that amount of time to see the results of hiring new staff (hiring and training takes time). The adaptation algorithm will wait for this amount of time before considering new failures of *AR1* or *AR2*.

AR3 and *AR4* also refer both to the same element, namely, the quality constraint *Ambulances arrive in 8 min*. To increase its success rate, the framework can choose between variation point *VP1* or control variable *LoA*, the former having a higher effect than the latter. Since *AR3* sets the threshold for the success rate of the quality constraint, it is set to use *descending order*, choosing to change first the element with greater effect. On the other hand, *AR4* just indicates a trend of decline, but the current rate could still be well over the threshold and the choice here is to use the parameters with lowest effect first, i.e., *ascending order*.

AwReq AR6 imposes a maximum failure constraint on the domain assumption *Gazetteer working and up-to-date* and the related parameters are variation points *VP2* and *VP3* and, as specified in section 4.3, *VP2* has to be changed first, otherwise changing *VP3* has no effect. However, we have specified the *number of parameters* to choose to be $N = 2$ and, thus, both parameters will be changed at the same time. This will make the A-CAD switch always from assuming proper functioning of the gazetteer to using paper maps.

It is also the case for *AR7*, *AR8*, *AR10* and *AR14* that there is just one parameter that has an effect on the indicator. For this reason, they will all use the default algorithm. With the exception of *AR7*, however, these *AwReqs* have been marked as *immediate resolution*, which means that the adaptation algorithm will consider the problem solved immediately after making the parameter change. This makes sense for *AR8* and *AR10* because changes on their associated parameters, respectively *VP4* and *VP5*, switch the system to a branch that does not contain the elements to which the *AwReq* refers. Changing *VP5* back to *Display exception messages* also makes *AR14* irrelevant because messages would be shown immediately to staff members. The default algorithm is also the choice for *AR12*, because all of the parameters that can affect it have roughly the same effect, so one of them will be chosen randomly.

For *AR9*, *AR13* and *AR16* the ordered parameter choice was also selected, being used in an *ascending order* for the latter two *AwReqs*. Furthermore, for *AR13* the *repeat policy* was set to *max 2 times* so we try to reduce costs by avoiding ambiguous dispatches (increase *MST*) a few times first before firing staff members (reducing *NoSM*).

Finally, *AR11* indicates that *Dispatching occurs in 3 min* should never fail. In case it does, however, two different algorithms were chosen. The first one is the *Oscillation Algorithm*, which applies only to *MST*, as it is the only numeric variable associated with *AR11*. If this algorithm is not applicable (e.g., *MST* is not incrementable), use descending order and change other related parameters.

Note that Table 5.2 does not include *AwReq AR5*. The reason is that there are no parameters related to this indicator and, thus, no reconfiguration algorithm can be used to adapt in this case. In situations such as these (but which is also possible when there are related parameters), requirements engineers can use *Evolution Requirements* to associate specific changes in the goal model to some system failures. The application of this kind of requirement on the A-CAD is described in the following chapter.

Chapter 6

Evolution Requirements for the A-CAD

In the last chapter, we have associated adaptation algorithms to most *AwReqs* of the A-CAD, in order to adapt the system to failures of these *AwReqs* at runtime by analyzing the information added to the goal model during System Identification. In some cases, however, reconfiguration might not be available or even the preferred solution by the stakeholders.

For instance, we have mentioned before the case of *AR5*, which does not have related parameters and, thus, cannot be associated to a reconfiguration algorithm. Furthermore, other *AwReqs* such as *AR1* or *AR2* for instance, could benefit from a more specific adaptation action, one that would manipulate the goal model itself in a pre-determinate way instead of searching for the best parameter to change.

For this reason, we have proposed an approach that allows requirements engineers to model *Evolution Requirements* (or *EvoReqs*), which specify changes to other requirements when certain conditions apply [Souza *et al.*, 2012a]. Such changes have an effect on the target system (i.e., the A-CAD) at runtime, effectively instructing it on how to adapt.

We start from the requirements model for the A-CAD after the new elements added in the previous chapter (shown in figure 5.1, page 38). However, for the sake of demonstrating a wider variety of *EvoReqs*, we have once again added one more *AwReq* to this model: *AR15* — the goal *Register call* should never fail (`NeverFail(G_RegCall)`). This requirement represents the fact that this goal is critical to the dispatch process, for the very simple reason that the A-CAD cannot process an incident that has not been registered into the system and, thus, the entire process will have to be conducted manually if this goal is not satisfied. The complete goal model with this last *AwReq* added is shown in figure 6.1 (page 43). This is the final model for the A-CAD, as no more elements will be included in the remaining chapters of this report.

In the following sections, we present the *EvoReq* patterns that form the adaptation strategies used in the specification of the A-CAD (§6.1) and then complete the A-CAD's adaptation requirement specification by associating these strategies to *AwReq* failures (§6.2).

6.1 Adaptation Strategies (Patterns)

As described in [Souza *et al.*, 2012a], *EvoReqs* are modeled as a sequence of adaptation instructions that are sent to the target system in order to coordinate the adaptation procedure according to a specific strategy. Many of them follow recognizable patterns, therefore a strategy can be defined by a name, a list of arguments (with optional default values) and an algorithm in Java™-style pseudocode that uses the adaptation instructions. Strategies are then associated with *AwReqs* and are enacted in case they fail at runtime. For this reason, we can always refer to the failing *AwReq* in the strategy's pseudocode using the keyword `awreq`, whereas other parameters have to be explicitly declared as arguments. Some of these arguments (including

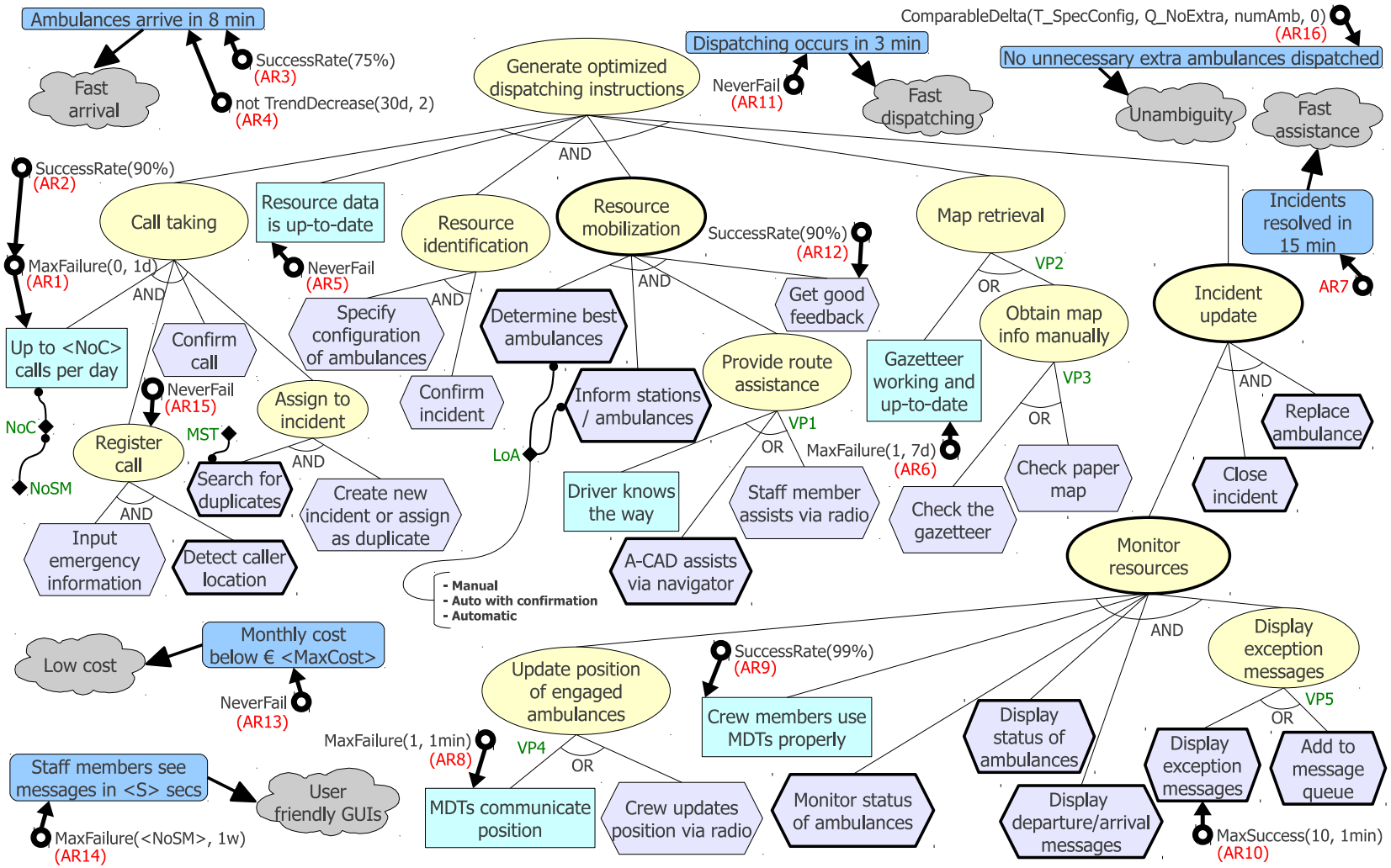

Figure 6.1: Final goal model for the A-CAD system-to-be, with new *AwReq AR15* added.

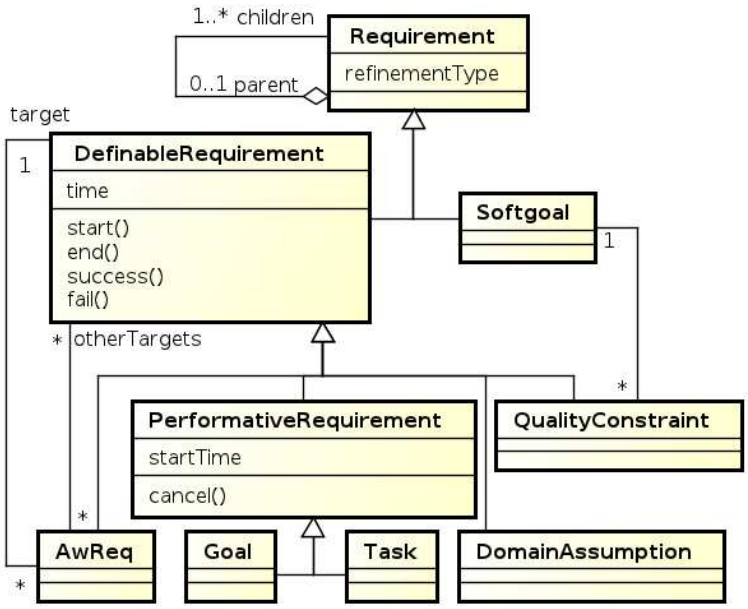

Figure 6.2: Class model for requirements in GORE, adapted from [Souza *et al.*, 2011b].

AwReq themselves) are instances of the classes depicted in figure 6.2, proposed in [Souza *et al.*, 2012a], which is a modified version of the earlier *AwReqs* GORE model published in [Souza *et al.*, 2011b].

This section defines all patterns used in the elicitation and modeling of the *EvoReqs* of the A-CAD. We start with the *Retry* strategy, defined in listing 6.1. This strategy obtains a reference to the requirements instance `r` that should be tried again, creates a new instance `r'` from the same class of `r` and copies the data from the execution section of `r` to `r'` (if argument `copy` is `true`, which is its default value). Then, it terminates all components associated with `r`, rolls back any partial changes that would leave the system in an inconsistent state, waits the amount of time specified in argument `time` and finally initiates the components related to the new requirement instance `r'`.

Listing 6.1: Definition of the *Retry* adaptation strategy.

```
1  /* Try to satisfy the requirement again after a given period of time (in ms). */
2  Retry(copy : boolean = true; time : long) {
3    r = awreq.target;
4    R = r.class;
5    r' = new-instance(R);
6    if (copy) copy-data(r, r');
7    terminate(r);
8    if (R = PerformativeRequirement) rollback(r);
9    wait(time);
10   initiate(r');
11 }
```

A very simple strategy is to tell an actor of the system about the failure so he, she or it (in the case of non-human actors) becomes aware of the problem and can possibly do something about it. There are two slightly different strategies based on this notion: *Warning*, defined in listing 6.2, consists on just sending the notification and ignoring the failure, whereas *Delegation*, defined in listing 6.3, has an extra instruction that tells the target system to wait for the issue to be fixed before proceeding. The former is interesting for the cases in which it is too difficult or even impossible to check if the problem has been fixed or when the target system cannot suspend the execution session until the problem is fixed.

Listing 6.2: Definition of the *Warning* adaptation strategy.

```
1  /* Send a warning to an actor, which can be human or not. */
2  Warning(a : Actor) {
3    send-warning(a, awreq);
4  }
```

Listing 6.3: Definition of the *Delegate* adaptation strategy.

```
1  /* Delegate the solution of a failure to an actor, which can be human or not. */
2  Delegate(a : Actor) {
3    send-warning(a, awreq);
4    wait-for-fix(awreq);
5  }
```

A more sophisticated strategy consists on relaxing a requirement, which means making it easier to be satisfied. This strategy has two versions. *Relax by Disabling Child* assumes the requirement is AND-refined and disables one of its children, which means there is one less child requirement to satisfy in order to satisfy the parent. On the other hand, *Relax by Replacement* assumes there exists another version of the requirement that is less strict and is currently disabled and replaces the original requirement with the relaxed version of it. These strategies can be applied to both the class and the instance levels and they have corresponding *Strengthen by Enabling Child* and *Strengthen by Replacement* counterparts.

Listings 6.4 and 6.5 show, respectively, the definitions of strategies *Relax by Disabling Child* and *Strengthen by Enabling Child*, which are very similar. When the strategies are applied to the class level, they correspond directly to evolution operations `disable` / `enable`. When applied to the instance level, the strategy temporarily suspends the target requirement, locates the instance of the child that has to be disabled (enabled), performs a `terminate` (`initiate`) rolling back any partial changes (only when disabling), indefinitely suspends (resumes) the child requirement and resumes the target requirement so its satisfaction can be reassessed.

Listing 6.4: Definition of the *Relax by Disabling Child* adaptation strategy.

```
1  /* Relaxes a requirement by disabling one of its enabled children (assuming AND-
       refinement). */
2  RelaxDisableChild(r : Requirement = awreq.target; level : Level = INSTANCE; child
       : Requirement) {
3    if ((level == CLASS) || (level == BOTH)) {
4      disable(child.class);
5    }
6
7    if ((level == INSTANCE) || (level == BOTH)) {
8      suspend(r);
9      terminate(child);
10     if (child.class = PerformativeRequirement) rollback(child);
11     suspend(child);
12     resume(r);
13   }
14 }
```

Listing 6.5: Definition of the *Strengthen by Enabling Child* adaptation strategy.

```
1  /* Strengthens a requirement by enabling one of its disabled children (assuming
       AND-refinement). */
2  StrengthenEnableChild(r : Requirement = awreq.target; level : Level = INSTANCE;
       child : Requirement) {
3    if ((level == CLASS) || (level == BOTH)) {
4      enable(child.class);
5    }
6
7    if ((level == INSTANCE) || (level == BOTH)) {
8      suspend(r);
9      resume(child);
10     initiate(child);
11     resume(r);
```

```
  }
}
```

Note that unlike the *Retry* strategy, the *Relax/Strengthen* strategies can be applied to a different requirement than the one referred to by the failing *AwReq* (i.e., `awreq.target`). Therefore, the requirement `r` to be relaxed/strengthened is passed as the first argument of the pattern. However, for the most common case of relaxing/strengthening the target requirement, `awreq.target` is the default value for this argument.

Listing 6.6 defines the *Relax by Replacement* strategy (*Strengthen by Replacement* is exactly the same, but with name `StrengthenReplace`, and therefore it is now shown). When applied at the class level, the target requirement is disabled and the specified new one is enabled for future executions. When used at the instance level, the replacement is similar to a *Retry*, copying data from one requirement to another if appropriate, terminating, indefinitely suspending and rolling back the target requirement instance and initiating the new requirement instance.

Listing 6.6: Definition of the *Relax by Replacement* adaptation strategy.

```
/* Relaxes a requirement by replacing it with a less strict version. */
RelaxReplace(r : Requirement = awreq.target; copy : boolean = true; level : Level
    = INSTANCE; r' : Requirement) {
  R = r.class;
  R' = r'.class;
  if ((level == CLASS) || (level == BOTH)) {
    disable(R);
    enable(R');
  }

  if ((level == INSTANCE) || (level == BOTH)) {
    if (R = PerformativeRequirement) && (R' = PerformativeRequirement) && (copy)
        copy-data(r, r');
    terminate(r);
    if (R = PerformativeRequirement) rollback(r);
    suspend(r);
    initiate(r');
  }
}
```

Finally, listing 6.7 shows the simplest strategy of all, *Abort*, which corresponds directly to the homonimous evolution operation.

Listing 6.7: Definition of the *Abort* adaptation strategy.

```
/* Abort. */
Abort() {
  abort(awreq);
}
```

As we can see from the commands used, the above strategies make changes to the requirements model, either at the instance or at the class level, which is quite different a solution than the reconfiguration algorithms presented in the previous chapter. However, as presented in [Souza *et al.*, 2012a], two *EvoReq* commands were included in order to allow one to specify the use of reconfiguration as an adaptation strategy. Listing 6.8, then, shows how these commands can be used in order to activate a reconfiguration algorithm.

Listing 6.8: Definition of the *Reconfiguration* adaptation strategy.

```
/* Uses a reconfiguration algorithm in order to adapt. */
Reconfigure(algo : FindConfigAlgorithm, ar : AwReq, level : Level = INSTANCE) {
  C' = find-config(algo, ar)
  apply-config(C', level)
}
```

Basically, the `find-config()` command activates the specified adaptation algorithm, which returns a new configuration for the system. Then, `apply-config()` instructs the target system to apply the new configuration at the specified abstraction level. The next section uses this and the other strategies defined above to specify how the A-CAD should adapt to its *AwReqs*' failures.

6.2 *EvoReqs* for the A-CAD

Having defined the strategies, a complete specification of the adaptation rules for the A-CAD can be provided. Since strategies are executed by an Event-Condition-Action framework (described in [Souza *et al.*, 2012a]), strategies need to be ordered and will be tried in the given order. Table 6.1 (page 48) presents the list of strategies associated to each *AwReq* failure, including the qualitative adaptation algorithms listed back in table 5.2, integrated through the *Reconfigure* strategy.

As mentioned in the title page, this is a work in progress. The next steps in writing this report are:

- Determine the level of abstraction (class, instance or both) for the *Reconfigure* strategy in each of its uses in table 6.1;
- Associate applicability conditions to each strategy;
- Associate resolution conditions to each *AwReq*;
- Describe the implementation in chapter 7 and conclude the report (chapter 8).

Table 6.1: *EvoReqs* elicited for the A-CAD experiment.

AwReq	***AwReq* Pattern**	**List of *EvoReqs***
AR1	`NeverFail(T_InputInfo)`	1. *Warning("AS Management")* 2. *Reconfigure(∅)*
AR2	`SuccessRate(AR1, 90%)`	1. *Warning("AS Management")* 2. *Reconfigure(∅)*
AR3	`SuccessRate(Q_AmbArriv, 75%)`	1. *Reconfigure({Ordered Effect Parameter Choice} [order = descending])*
AR4	`not TrendDecrease(Q_AmbArriv, 30d, 2)`	1. *RelaxReplace(AR4, AR4_60Days) + StrengthenReplace(AR3, AR3_80Pct)* 2. *Reconfigure({Ordered Effect Parameter Choice} [order = ascending])*
AR5	`NeverFail(D_DataUpd)`	1. *Delegate("Staff Member")*
AR6	`MaxFailure(D_GazetUpd, 1, 7d)`	1. *Reconfigure(∅ [n = 2])*
AR7		1. *Reconfigure(∅)*
AR8	`MaxFailure(D_MDTPos, 1, 1min)`	1. *RelaxReplace(D_MDTPos_20Secs)* 2. *RelaxReplace(AR8, AR8_45Secs)* 3. *RelaxReplace(AR8_45Secs, AR8_30Secs)* 4. *Retry(60000)* 5. *Reconfigure(∅ [Immediate Resolution])*
AR9	`SuccessRate(D_MDTPos, 1, 1min)`	1. *Reconfigure({Ordered Effect Parameter Choice} [order = descending])*
AR10	`MaxSuccess(T_Except, 10, 1min)`	1. *Reconfigure(∅ [Immediate Resolution])*
AR11	`NeverFail(Q_Dispatch)`	1. *Reconfigure({Oscillation Value Calculation, Oscillation Resolution Check})* 2. *Reconfigure({Ordered Effect Parameter Choice} [order = descending])*
AR12	`SuccessRate(T_Feedback, 90%)`	1. *Reconfigure(∅)*
AR13	`NeverFail(Q_MaxCost)`	1. *Reconfigure({Ordered Effect Parameter Choice} [order = ascending, repeat policy = max 2 times])*
AR14	`MaxFailure(Q_MsgTime, <NoSM>, 1w)`	1. *Reconfigure(∅ [Immediate Resolution])*
AR15	`NeverFail(G_RegCall)`	1. *Retry(5000)* 2. *RelaxDisableChild(T_DetectCaller)*
AR16	`ComparableDelta(T_SpecConfig, Q_NoExtra, numAmb, 0)`	1. *Reconfigure({Ordered Effect Parameter Choice} [order = ascending])*

Chapter 7

Framework Implementation

As stated in the title page, this report is a work in progress. This section is yet to be written. It will talk about the Zanshin framework (http://github.com/vitorsouza/Zanshin/), which was implemented in order to validate the ideas of our research.

Chapter 8

Conclusions

As stated in the title page, this report is a work in progress. This section is yet to be written.

Appendix A

A-CAD Final Requirements Model

As stated in the title page, this report is a work in progress. This section is yet to be written.

Bibliography

Paul Beynon-Davies. Information systems 'failure': the case of the London Ambulance Service's Computer Aided Despatch project. *European Journal of Information Systems*, 4(3):171–184, 1995.

Karin K. Breitman, Julio C. S. P. Leite, and Anthony Finkelstein. The world's a stage: a survey on requirements engineering using a real-life case study. *Journal of the Brazilian Computer Society*, 6(1), 1999.

Lawrence Chung, Brian A. Nixon, Eric Yu, and John Mylopoulos. *Non-Functional Requirements in Software Engineering*. Springer, October 1999.

Fabiano Dalpiaz, Amit K. Chopra, Paolo Giorgini, and John Mylopoulos. Adaptation in Open Systems: Giving Interaction Its Rightful Place. In Jeffrey Parsons, Motoshi Saeki, Peretz Shoval, Carson Woo, and Yair Wand, editors, *Conceptual Modeling – ER 2010*, volume 6412 of *Lecture Notes in Computer Science*, pages 31–45. Springer, 2010.

Fabiano Dalpiaz, Paolo Giorgini, and John Mylopoulos. Adaptive socio-technical systems: a requirements-based approach. *Requirements Engineering*, pages 1–24, 2012.

Mehdi Dastani, M. Birna van Riemsdijk, and John-Jules C. Meyer. Goal Types in Agent Programming. In *Proc. of the 5^{th} International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1285–1287. ACM, 2006.

M.S. Feather, Stephen Fickas, Axel van Lamsweerde, and Christophe Ponsard. Reconciling system requirements and runtime behavior. In *Proc. of the 9^{th} International Workshop on Software Specification and Design*, pages 50–59. IEEE, 1998.

Anthony Finkelstein and John Dowell. A Comedy of Errors: the London Ambulance Service case study. In *Proc. of the 8^{th} International Workshop on Software Specification and Design*, pages 2–4. IEEE, 1996.

Lingxiao Fu, Xin Peng, Yijun Yu, and Wenyun Zhao. Stateful Requirements Monitoring for Self-Repairing of Software Systems. Technical report, # FDSE-TR201101 (available online: http://www.se.fudan.sh.cn/paper/techreport/1.pdf), Fudan University, China, 2010.

Ivan Jureta, John Mylopoulos, and Stephane Faulkner. Revisiting the Core Ontology and Problem in Requirements Engineering. In *Proc. of the 16^{th} IEEE International Requirements Engineering Conference*, pages 71–80. IEEE, 2008.

Malik J. Khan, Mian M. Awais, and Shafay Shamail. Enabling Self-Configuration in Autonomic Systems using Case-Based Reasoning with Improved Efficiency. In *Proc. of the 4^{th} International Conference on Autonomic and Autonomous Systems*, pages 112–117. IEEE, 2008.

Jeff Kramer and Alexander L. Wolf. Succeedings of the 8^{th} International Workshop on Software Specification and Design. *ACM SIGSOFT Software Engineering Notes*, 21(5):21–35, 1996.

Alexei Lapouchnian, Yijun Yu, and John Mylopoulos. Requirements-Driven Design and Configuration Management of Business Processes. In Gustavo Alonso, Peter Dadam, and Michael Rosemann, editors, *Business Process Management*, volume 4714 of *Lecture Notes in Computer Science*, pages 246–261. Springer, 2007.

Emmanuel Letier. *Reasoning about Agents in Goal-Oriented Requirements Engineering*. Phd thesis, Université Catholique de Louvain, Belgium, 2001.

Mirko Morandini, Loris Penserini, and Anna Perini. Operational Semantics of Goal Models in Adaptive Agents. In *Proc. of the 8^{th} International Conference on Autonomous Agents and Multiagent Systems*, pages 129–136. ACM, 2009.

Hiroyuki Nakagawa, Akihiko Ohsuga, and Shinichi Honiden. gocc: A Configuration Compiler for Self-adaptive Systems Using Goal-oriented Requirements Description. In *Proc. of the 6^{th} International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 40–49. ACM, 2011.

Xin Peng, Bihuan Chen, Yijun Yu, and Wenyun Zhao. *Self-Tuning of Software Systems through Goal-based Feedback Loop Control*. IEEE, 2010.

Reynolds. Categorised, posted at the blog "Random Acts of Reality", http://randomreality.blogware.com/blog/_archives/2004/2/18/21077.html (last access: July 21^{st}, 2011).

Reynolds. ORCON!, posted at the blog "Random Acts of Reality", http://randomreality.blogware.com/blog/_archives/2004/3/15/21076.html (last access: July 21^{st}, 2011).

Vítor E. S. Souza and John Mylopoulos. From Awareness Requirements to Adaptive Systems: a Control-Theoretic Approach. In *Proc. of the 2^{nd} International Workshop on Requirements@Run.Time*, pages 9–15. IEEE, 2011.

Vítor E. S. Souza, Alexei Lapouchnian, and John Mylopoulos. Awareness Requirements for Adaptive Systems. Technical report, # DISI-10-049, University of Trento (available at http://eprints.biblio.unitn.it/archive/00001893/), 2010.

Vítor E. S. Souza, Alexei Lapouchnian, and John Mylopoulos. System Identification for Adaptive Software Systems: a Requirements Engineering Perspective. In Manfred Jeusfeld, Lois Delcambre, and Tok-Wang Ling, editors, *Conceptual Modeling – ER 2011*, volume 6998 of *Lecture Notes in Computer Science*, pages 346–361. Springer, 2011.

Vítor E. S. Souza, Alexei Lapouchnian, William N. Robinson, and John Mylopoulos. Awareness Requirements for Adaptive Systems. In *Proc. of the 6^{th} International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 60–69. ACM, 2011.

Vítor E. S. Souza, Alexei Lapouchnian, William N. Robinson, and John Mylopoulos. Awareness Requirements for Adaptive Systems. Technical report, # DISI-11-352, University of Trento, 2011.

Vítor E. S. Souza, Alexei Lapouchnian, and John Mylopoulos. (Requirement) Evolution Requirements for Adaptive Systems. In *Proc. of the 7^{th} International Symposium on Software Engineering for Adaptive and Self-Managing Systems (to appear)*, 2012.

Vítor E. S. Souza, Alexei Lapouchnian, and John Mylopoulos. Requirements-driven Qualitative Adaptation. In *submitted for publication (under review)*, 2012.

Vítor E. S. Souza. Awareness Requirements for Self-Adaptive Socio-technical Systems. Technical report, (Qualifying Paper), DISI, University of Trento, 2010.

Report of the inquiry into the London Ambulance Service. South West Thames Regional Health Authority, 1993.

Axel van Lamsweerde. *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley, 1 edition, 2009.

Yiqiao Wang and John Mylopoulos. Self-Repair through Reconfiguration: A Requirements Engineering Approach. In *Proc. of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, pages 257–268. IEEE, 2009.

Roel Wieringa. Design Science Research Methodology: Principles and Practice, workshop presented at the 32^{nd} International Conference on Software Engineering (handout slides available at http://wwwhome.cs.utwente.nl/~roelw/DesignScienceMethodology-handout.pdf), 2010.

Zheng You. Experiences with applying the i* framework to a real-life system. Technical report, Requirements Engineering (CSC2106) Course Project, University of Toronto, Canada, 2001.

Zheng You. *Using Meta-Model-Driven Views to Address Scalability in i* Models*. Master thesis, University of Toronto, Canada, 2004.

Eric S. K. Yu, Paolo Giorgini, Neil Maiden, and John Mylopoulos. *Social Modeling for Requirements Engineering*. MIT Press, 1^{st} edition, 2011.