SOFTWARE – PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 2015; **45**:689–725
Published online 11 December 2013 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/spe.2245

Designing an adaptive computer-aided ambulance dispatch system with Zanshin: an experience report

Vítor E. Silva Souza[1,*,†] and John Mylopoulos[2]

[1]*Department of Informatics, Federal University of Espírito Santo (UFES), Vitória, Brazil*
[2]*Department of Information Engineering and Computer Science, University of Trento, Trento, Italy*

SUMMARY

We have been witnessing growing interest in systems that can adapt their behavior to deal with deviations between their performance and their requirements at run-time. Such adaptive systems usually need to support some form of a feedback loop that monitors the system's output for problems and carries out adaptation actions when necessary. Being an important feature, adaptivity needs to be considered in early stages of development. Therefore, adopting a requirements engineering perspective, we have proposed an approach and a framework (both called *Zanshin*) for the engineering of adaptive systems based on a feedback loop architecture. As part of our framework's evaluation, we have applied the Zanshin approach to the design of an adaptive computer-aided ambulance dispatch system, whose requirements were based on a well-known case study from the literature. In this paper, we report on the application of Zanshin for the design of an adaptive computer-aided ambulance dispatch system, presenting elements of the design, as well as the results from simulations of run-time scenarios. Copyright © 2013 John Wiley & Sons, Ltd.

Received 2 January 2013; Revised 29 October 2013; Accepted 31 October 2013

KEY WORDS: software requirements; adaptive systems; feedback loops; requirements awareness; software evolution; software reconfiguration; system identification; case studies

1. INTRODUCTION

We have been witnessing growing interest in software systems that can adapt to changes in their environment or their requirements in order to continue to fulfill their mandate. Indications for this growing interest can be found in recent workshops and conferences on topics such as adaptive, autonomic, and autonomous software systems (e.g., [1–4]).

In order to develop an adaptive system, one needs to account for some form of a feedback loop that introduces monitoring and adaptation functionalities in the overall system. In other words, even if implicit or hidden in the system's architecture, adaptive systems must have a feedback loop implemented within their components in order to evaluate the overall behavior of the system and act accordingly [5]. A consensus is building in the adaptive software engineering community that these loops should be first-class citizens in the design of adaptive systems [3, 6], rather than embedded and diffused in the code they are supposed to control.

Because of their importance, adaptivity concerns need to be considered in early stages of the software development process. Therefore, in our research, we adopt a requirements engineering (RE) perspective to the design of adaptive software systems and, given that feedback loops constitute a (architectural) prosthesis intended to support adaptation, we try to answer the following question: what are the requirements this prosthesis is intended to fulfill? Motivated by this question, we have proposed *Zanshin*, an approach for engineering requirements for adaptive systems founded on a

*Correspondence to: Vítor E. Silva Souza, Departamento de Informática (CT 7), Universidade Federal do Espírito Santo, Av. Fernando Ferrari, 514, Goiabeiras, Vitória, ES, 29075-910, Brazil.
†E-mail: vitorsouza@inf.ufes.br

feedback loop architecture. Different parts of the approach have been presented in the literature [7–11] and were compiled into the first author's PhD thesis [12].

In this paper, we focus on evaluating Zanshin. Hevner *et al.* [13] describe five categories of evaluation methods in Design Science: Observational, Analytical, Experimental, Testing, and Descriptive. Previous publications on Zanshin use descriptive methods to validate the different parts of the approach. Here, the goal is to report on the execution of experimental evaluation methods that provide further validation for our proposal as a whole. Such evaluation consisted of designing an adaptive computer-aided ambulance dispatch (A-CAD) system, whose requirements were based on the well-known London Ambulance Service Computer-Aided Despatch (LAS-CAD) failure report [14] and some of the publications that analyzed the case.

Moreover, our proposal includes a Java™-based framework that implements the generic features of a feedback loop that operationalizes adaptation. Therefore, after eliciting requirements for the A-CAD, we conducted simulations of real failure scenarios in order to evaluate the performance of Zanshin, that is, the degree to which it provided sensible adaptation responses to these scenarios.

The main purpose of this paper is to share with researchers and practitioners experiences and lessons learned on the use of Zanshin to design adaptive software systems. In addition, the paper presents details about the design of the A-CAD system and the Zanshin foundation on top of which it was built. This can hopefully facilitate other efforts to build adaptive systems.

The remainder of the paper is organized as follows. Section 2 introduces the Zanshin approach. Section 3 presents base requirements for a CAD system based on the LAS-CAD case study. Sections 4–6 summarize the results from applying the Zanshin approach to the CAD system in order to design the A-CAD. In Section 7, we introduce the Zanshin framework, explaining how external systems can take advantage of its feedback loop implementation. Section 8 shows how A-CAD requirements were coded and describes some of the simulations that were conducted. Section 9 compares related work. Section 10 discusses lessons learned from this experiment and its threats to validity. Finally, Section 11 concludes the paper.

2. THE ZANSHIN APPROACH

Zanshin is an approach to the design of adaptive systems that combines concepts of goal-oriented RE (GORE) [15] with principles of feedback control theory [16] and qualitative reasoning [17]. Figure 1 presents an overview of the approach.

State-of-the-art GORE techniques are used to analyze aspects of the problem that are not related to adaptation concerns. The analyst thus produces a goal model that represents the requirements of

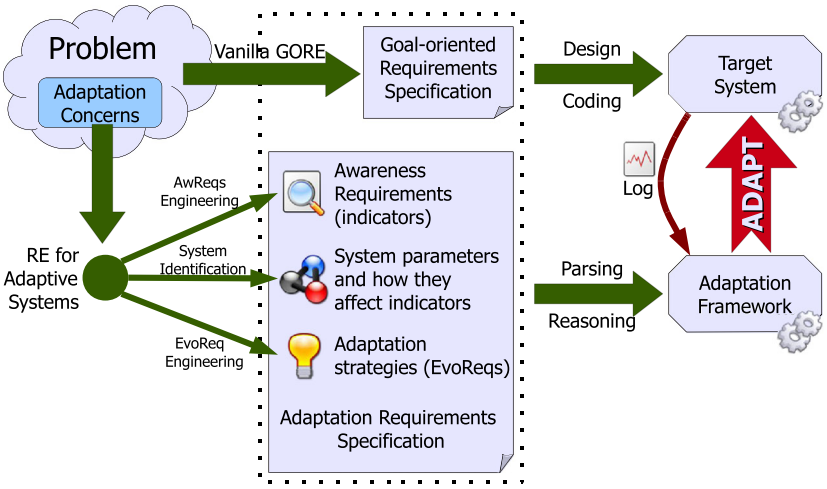

Figure 1. An overview of the Zanshin approach.

DOI: 10.1002/spe

Input: goal model $M = \{G, S, T, D, Q\}$
Output: augmented goal model $M' = \{G', S', T', D', Q', A, P, R, E\}$
Where:

- G is the set of goals;
- S is the set of softgoals;
- T is the set of tasks;
- D is the set of domain assumptions;
- Q is the set of quality constraints;
- A is the set of Awareness Requirements (*AwReqs*);
- P is the set of system parameters;
- R is the set of differential relations between parameters and *AwReqs*;
- E is the set of Evolution Requirements (*EvoReqs*).

Figure 2. Inputs and outputs of the systematic process proposed by Zanshin.

a system-to-be, using modeling primitives such as goals, softgoals, tasks, quality constraints (QCs), and domain assumptions [18]. This model embodies system requirements and circumscribes what the system should do and how it should behave.

A feedback loop activates corrective actions whenever *requirements divergences* are detected, that is, whenever the base system violates its requirements or some of its domain assumptions are found to not hold. Hence, Zanshin extends 'vanilla' goal models of RE by allowing the specification of what requirements/indicators are to be monitored and what corrective actions are to be performed when divergences are detected.

In short, Zanshin proposes a systematic process consisting of three major steps: *Awareness Requirements Engineering*, *System Identification*, and *Evolution Requirements Engineering*. The process takes as input a vanilla goal model M and produces an augmented goal model M', as shown in Figure 2. As with most software engineering processes, this one can be conducted in an iterative fashion. M' is then used by the Zanshin framework to provide adaptation capabilities to the base system, as later explained in Sections 7 and 8. It is important to note that the entire process is carried out manually by an analyst.

2.1. Awareness requirements engineering

Input: $M = \{G, S, T, D, Q\}$;
Output: $M' = \{G', S', T', D', Q', A\}$.

Awareness Requirements (AwReqs) are requirements that talk about the states of other requirements (such as their success or failure) at run-time. AwReqs can refer to the following: (i) tasks, to determine if the system has performed a specific set of actions successfully; (ii) QCs, to determine if the system abides by quality requirements; (iii) goals/softgoals, to determine if the system has satisfied a requirement; and (iv) domain assumptions, to determine if things that were assumed to be true in the environment indeed are during system operation.

For purposes of illustration, we use the Meeting Scheduler System, a well-known exemplar in the RE literature [19] and featured in previous Zanshin publications [7, 11, 12]. Examples of AwReqs are the following: (i) task *Have the system schedule (a meeting)* should never fail; (ii) QC *At least 90% of participants attend (meetings)* should have 75% success rate; (iii) considering one week periods, the success rate of goal *Collect timetables* should not decrease three times consecutively; (iv) QC *At least 90% of participants attend (meetings)* should have 75% success rate.

AwReqs can refer to single instances of a requirement (e.g., (i) and (iv) in the previous text) or the whole set of instances of a requirement in an aggregate way (e.g., (ii) and (iii) in the previous text). At run-time, the elements of a goal model represented as classes and instances of these classes are created every time the system starts pursuing a requirement (or verifying a domain assumption). The state of these instances (*succeeded*, *failed* etc.) is then monitored by the feedback controller. AwReqs can also refer to other AwReqs, for example, '*AwReq* AR1 should succeed 90% of the time', constituting meta-AwReqs or level 2 AwReqs. Further levels of awareness are also possible. All AwReqs are monitored in the same fashion.

DOI: 10.1002/spe

In summary, AwReqs represent situations the system should be aware of and possibly take action when they occur. These requirements determine the functionality of the monitoring component of the feedback loop. Thus, requirements (or domain assumptions) are not necessarily treated as invariants that must always be achieved (at best, a highly optimistic assumption). Instead, we accept the fact that the system may fail in achieving its requirements and specify the degree of failure that can be tolerated. As such, AwReqs are control requirements rather than function/non-functional ones. In Section 4, AwReqs are elicited for the A-CAD system.

2.2. System identification

Input: $M' = \{G', S', T', D', Q', A\}$;
Output: $M' = \{G', S', T', D', Q', A, P, R\}$.

A possible adaptation strategy in case of failures is to search the solution space to identify a new configuration (i.e., values for system parameters) that would move system indicators toward AwReqs fulfillment. As in control systems, the purpose of system identification is to identify the effect that each parameter change has on indicators for the controlled system. On the basis of qualitative reasoning [17], Zanshin proposes a language to represent parameters, indicators and their relations, plus a system identification process to elicit and represent this information in the goal model. In this process, Zanshin uses AwReqs to identify indicators.

For instance, suppose an AwReq referring to the success rates of goal *Collect timetables* fails at run-time. To reconfigure the system, the controller needs to determine which parameter can be modified in order to improve the chances of success of this goal. If there was a parameter *FhM* that indicates *From how Many* participants we should collect timetables (a percentage value), decreasing its value could be considered an adaptation action here. Likewise, switching how the goal is operationalized (from *Collect automatically* to *Email participants* or to *Call participants*) would also be considered as an alternative. Note that any parameter change may also have impact on other indicators, for example, QC *At least 90% of participants attend (meetings)*.

Two types of parameters can be elicited during this process: *variation points* consist of OR-refinements that are already present in a goal model and merely need to be labeled (e.g., the goal operationalization example in the previous text); *control variables* are abstractions over large/repetitive variation points, simplifying the OR-refinements that would have to be modeled in order to represent such variability (e.g., *FhM*). After indicators and parameters have been identified, the effects that changes on the latter have on the outcome of the former are analyzed and represented using differential relations such as shown below. Here, AR2 is an AwReq that refers to *Collect timetables*, whereas I_{AR2} represents the indicator that is constrained by AR2 (the success rate of the goal):

$$\Delta\,(I_{AR2}/FhM) < 0 \tag{1}$$

$$\Delta\,(I_{AR2}/VP1)\,\{email \rightarrow call, email \rightarrow system, call \rightarrow system\} > 0 \tag{2}$$

$$|\Delta\,(I_{AR2}/VP1)\,| > |\Delta\,(I_{AR2}/FhM)\,| \tag{3}$$

Relation (1) states that increasing the value of *FhM* has a negative effect on the success rate of AR2, whereas relation (2) means that performing the changes indicated between curly brackets can have a positive effect on the same AwReq. Of course, the analogous opposite relations are also inferred. Relation (3) tells us that the effect of *VP1* over AR2 is greater (in absolute values) than that of *FhM*, thus helping the controller decide which parameter to use in certain situations.

Finally, the analyst specifies what kind of reconfiguration algorithm to use for each AwReq failure. The choice depends on the amount of information elicited during system identification. For example, having relation (3) ordering parameters by the magnitude of their effect, one can choose to use an algorithm that picks the parameter with highest (or lowest) effect on a failing indicator. Zanshin provides an extensible reconfiguration framework that can cope with varying levels of precision about indicators and parameters.

DOI: 10.1002/spe

For more details, refer to [7] for the language and system identification process or [8] for the reconfiguration framework. In Section 5, the system identification process is applied to the A-CAD.

2.3. Evolution requirements engineering

Input: $M' = \{G', S', T', D', Q', A, P, R\}$;
Output: $M' = \{G', S', T', D', Q', A, P, R, E\}$.

With reconfiguration, Zanshin searches for the best system configuration to adapt to for a given failure. However, it is often the case that stakeholders or domain experts know exactly how they want to respond to undesirable situations, stating requirements such as 'If we detect so many problems in satisfying requirement R, replace it with a less strict version, $R-$' or 'Starting January 1st, 2013, replace requirement S with S' to comply with new legislation that has been recently approved'. Because these prescribe desired evolutions for other requirements, we refer to them as *evolution requirements* or simply EvoReqs.

EvoReqs are specified using the languages for AwReqs augmented with a set of primitive operations that change other requirements. Operations can be combined using patterns to compose *adaptation strategies*, such as 'Retry', 'Delegate', 'Relax' and so on. For instance, the Meeting Scheduler's AwReq 'task *Characterize meeting* should never fail' can be augmented to 'if *Characterize meeting* fails, retry the task in 5 seconds'.

At run-time, an event-condition-action (ECA)-based process uses the information expressed by EvoReqs in order to direct the system on how to adapt. The framework selects which adaptation strategy to apply and checks if it has worked. Further details on EvoReqs (their specification and operationalization) can be found in [11].

EvoReqs are used to specify the requirements for the adaptation component of the feedback loop, integrating with the reconfiguration framework [8] to represent requirements such as 'if this fails, reconfigure' (in this sense, 'reconfigure' becomes one of the available adaptation strategies). A complete specification of adaptation strategies for the A-CAD system is provided in Section 6.

3. THE COMPUTER-AIDED AMBULANCE DISPATCH SYSTEM

The failure of the 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 [14], papers on the subject were published in different venues, for example, [20–23].

Being a real case study with much available information—because of its failure and subsequent inquiry—makes the LAS-CAD a good choice for the evaluation of new research proposals. In fact, the focus of the discussions at the 8th International Workshop on Software Specification and Design 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 [23]. Other examples of this use can be seen in theses and dissertations, for example, [24, 25].

In particular, the LAS-CAD failure report [14] states 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.

The high criticality of many of the components of the LAS-CAD make it a good case for adaptive systems, because self-adaptation to failures—which invariably occurs 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 LAS, a set of standards (called ORCON) indicate what percentage of ambulances should arrive on the scene within 3 min, 10 min, and so on. There is no way to simply put that table into the system and guarantee that the standards will be followed [23]. Instead, adaptation actions can be taken whenever the system does not satisfy such requirements.

DOI: 10.1002/spe

Note, however, that it is not our intention to prove that the LAS would not have failed if it had been built as an adaptive system. Many of the studies conducted over the failure indicate that the procurement process and the development process were flawed, producing a bad quality system in general. 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 that would, in theory, be designed properly and have good quality in general.

In the remainder of this section, we describe the domain of ambulance dispatching on the basis of the information obtained on the LAS-CAD (§ 3.1); then, we narrow the scope of this domain for our CAD system (§ 3.2), producing a list of requirements in the form of 'SHALL statements' (§ 3.3); finally, we represent the 'vanilla' requirements for the CAD using a goal model (§ 3.4) and discuss the issue of variability (§ 3.5). Adaptation requirements for the A-CAD are discussed later, in sections 4 through 6.

3.1. The London Ambulance Service Computer-Aided Despatch domain

The following list defines primitive terms for the domain of ambulance dispatching, mostly adapted from [23]. Italicized words refer to other terms that are also in the list (for cross-reference):

- Emergency service: a service provided by the public authorities of a specific *region* (e.g., the LAS for the city of London) to the citizens of that region that consists of the dispatching of ambulances and their *crews* and *equipments* to *incidents*. The purpose of the CAD is to automate elements 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.);
- Gazetteer: a geographical directory, providing information about places in the *serviced region* together with a map;
- Call: a telephone call to the *emergency service* (e.g., 999 in the UK, 911 in the USA, and 190 in Brazil), 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, and helicopter). Vehicles are identified by their license plate numbers (or similar), are 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, and stretcher). Equipments of different types are assigned to ambulances and are uniquely identified by an ID code;

DOI: 10.1002/spe

- Crew member: human resources that work in *ambulances* and provide aid to citizens in case of *incidents* (e.g., drivers, paramedics, and firemen). 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 firemen might also be needed at the *location*; and
- Staff: people that work in the *emergency service*'s dispatching function, that is, employees of the *emergency service* that are not part of a *crew* (e.g., telephone operators, resource allocators, and dispatchers).

3.2. The adaptive computer-aided ambulance dispatch scope

A real CAD system is very large and complex. In our experiments, we focused on the core functions of a CAD software, assuming that there are other systems that produce events related to ambulances and that are monitored by the core CAD software. An analysis of the dependencies between the CAD software and these other systems is shown in [26], which was used as the basis for the definition of the scope of our CAD system. Figure 3 shows the states an ambulance can assume during its life-cycle and the events that trigger the transitions. These events, described in the succeeding text, were adapted from [23]:

- Creation (label omitted in the diagram): ambulance has been registered within the system;
- Commissioning: 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., for repairs and refueling). 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 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 s; and
- 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.

Events of commissioning 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

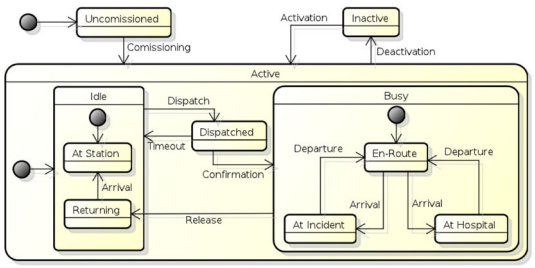

Figure 3. State machine diagram for ambulances.

DOI: 10.1002/spe

a lunch break for 1 h 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 sub-categories each.[‡] 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 responsibility 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 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 a non-emergency service is provided by the public authorities, a separate system should manage these situations and deactivate ambulances whenever they get dispatched to non-emergencies; and
- 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 the nature of incidents in each sector of the serviced region in the past.

3.3. Stakeholder requirements

Given the scope of our problem, we extracted a list of stakeholder requirements for the vanilla CAD system, again analyzing the publications that described the LAS-CAD. Requirements are represented using 'SHALL statements', divided in three categories: *incident response*, *resource monitoring*, and *exception messages*. Each requirement received a unique identifier for cross-referencing purposes.

Although Zanshin is based on GORE, we decided to use SHALL statements at this point for two main reasons: (i) given that requirements were elicited from documents and not real stakeholders, it would be very hard to guess the rationale behind them and (ii) SHALL statements provide a baseline for experiments using non-GORE approaches such as, for instance, RELAX [27].

Incident Response

REQ-1. The system shall allow staff to register calls they receive from citizens.
REQ-2. The system shall, whenever possible, detect the location of the caller and associate it with the call registry (public phone locations, triangulated cell phones, etc.).
REQ-3. The system shall allow staff to dismiss calls as non-emergencies.
REQ-4. The system shall assist staff in identifying, through the information from the call, if it refers to an open incident in the system.
REQ-5. The system shall allow staff to assign calls to open incidents as duplicates or create new incidents for calls.
REQ-6. The system shall allow staff to indicate the number of ambulances needed and their respective configurations (e.g., ambulance with paramedics, and fire truck and firemen);

[‡]See 'Categorised', posted at the blog 'Random Acts of Reality' (filed at the Internet Archive on May 20, 2011): http://web.archive.org/web/20110520163639/http://randomreality.blogware.com/blog/_archives/2004/2/18/21077.html.

DOI: 10.1002/spe

REQ-7. The system shall allow staff to confirm the information related to new incidents, clearing them for dispatch by the system.
REQ-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.
REQ-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, otherwise.
REQ-10. The system shall close incidents when all related resources are released (see REQ-13).
REQ-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. REQ-9 should follow accordingly.
REQ-12. The system shall perform in such a way that at least 75% of the ambulances arrive within 8 min to the location of the incident once dispatching instructions have been sent (see REQ-9). This constraint is based on the LAS-CAD standard for category A calls.[§]

Resource Monitoring

REQ-13. The system shall monitor for ambulance-related events (cf. Section 3.2) and keep the status of each ambulance up-to-date, including ambulance configuration.
REQ-14. The system shall show accurate and up-to-date information about ongoing incidents, including status, configuration, and position of engaged ambulances.
REQ-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

REQ-16. The system shall generate exception messages if the dispatching process does not conclude within 3 min. The process is considered concluded after the number of ambulances and their configurations have been assigned (see REQ-6), the system has dispatched ambulances that fit the configuration (see REQ-8 and REQ-9), and all ambulances have confirmed the dispatch (see REQ-13).
REQ-17. The system shall generate exception messages if ambulances engaged to incidents are not released within 15 min of their confirmation (see REQ-13)—in other words, incidents should be resolved within 15 min of dispatch.
REQ-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 REQ-13).

3.4. Goal-based specification of the computer-aided ambulance dispatch software

In this section, we present base system requirements for the CAD software using a goal model. Goal models can be used at different levels of abstraction. For example, in *early requirements* analysis, one could use goals to represent the objectives of stakeholders, a few alternative solutions to satisfy these objectives, and how each solution contributes to the satisfaction of relevant quality criteria (also known as softgoals). Here, however, we use goal models to represent a particular software-based solution. In other words, Zanshin starts from a *late requirements* analysis perspective and expects a goal-based specification of the solution.

Our goal-based specifications use a core RE ontology [18] to define a goal model that starts with a top goal (and optionally one or more cross-cutting softgoals), which is then further refined using ***refinement relations***. Refinements can be of type *AND* or *OR*, with obvious Boolean semantics, being used to represent *decomposition*, when connecting elements of the same type (e.g., a goal decomposed into sub-goals), or *operationalization*, when connecting elements of different types (e.g., a goal can be operationalized by a task or a domain assumption).

[§]See 'ORCON!', posted at the blog 'Random Acts of Reality' (filed at the Internet Archive on May 20, 2011): http://web.archive.org/web/20110520165433/http://randomreality.blogware.com/blog/_archives/2004/3/15/21076.html.

DOI: 10.1002/spe

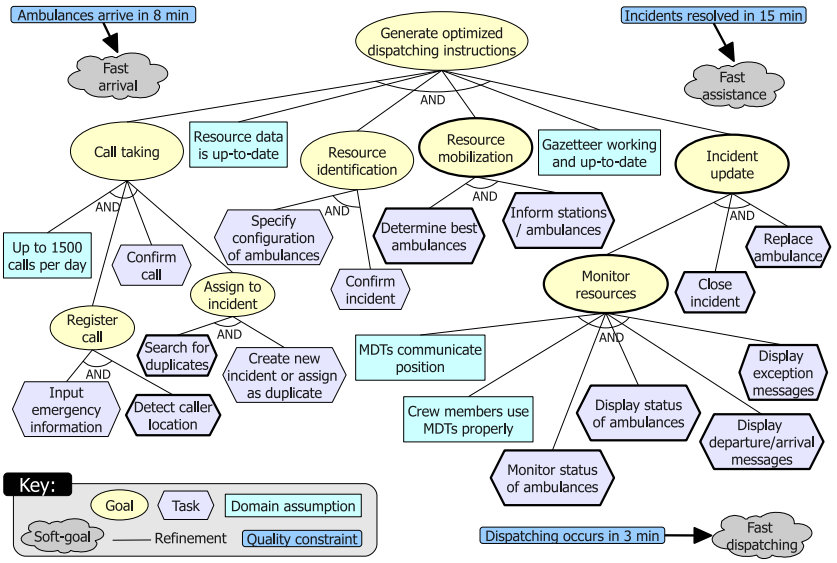

Figure 4. Goal-based specification for the computer-aided ambulance dispatch system-to-be, showing its 'vanilla' requirements.

On the basis of the stakeholder requirements elicited in Section 3.3 and early requirements $i^\star$ [28] models produced by Jane You for her masters thesis [26], we produced a goal model for the CAD system, shown in Figure 4. In the following paragraphs, we describe this model, associating its elements with the stakeholder requirements of Section 3.3 (IDs are shown in square brackets, e.g., [REQ-1]). Note that each requirement is associated with a task in the model and vice versa, showing that there are no tasks without purpose.

Call taking, an activity performed mostly by staff members, consists of responding to 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 [REQ-3], and *assigning* them to an incident. During registration, the system should try to *detect the caller's location* [REQ-2] to help staff expedite the activity of *inputting emergency information* [REQ-1]. Analogously, the system should *search for duplicates* [REQ-4] to help staff decide if they should *create a new incident or assign as duplicate* to an existing one [REQ-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 of *specifying the configuration of ambulances* [REQ-6]—that is, 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* [REQ-7] for dispatch by the system. *Resource mobilization* is then conducted by the system itself, *determining the best ambulance* [REQ-8] from those available and on the basis of the provided configuration and, finally, *informing stations / ambulances* about dispatch instructions [REQ-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 of goals into *achieve* and *maintain* goals have been proposed in [29–31]. This means that the CAD system should attempt to satisfy this goal sub-tree periodically, every t units of time (t to be specified later in the software development process).

To satisfy incident update, the CAD system should *monitor resources*, *close incidents* [REQ-10] when the ambulances are released, and *replace ambulances* [REQ-11] that break down (or have any other disabling event) during service. *Monitoring resources* consists of *monitoring the status of ambulances* [REQ-13]—including all events described in Section 3.2—and displaying *the status of ambulances* [REQ-14], *departure/arrival messages* [REQ-15], and eventual *exception messages* [REQ-16, REQ-17, and REQ18].

DOI: 10.1002/spe

For resource monitoring to work, the CAD system depends on a couple of assumptions being true. First, mobile data terminals (MDTs) should communicate position of busy (engaged) ambulances at regular intervals of time (at every 13 s, as specified in Section 3.2). Second, it is assumed that crew members use MDTs properly to notify about events in the ambulance state-chart (also, see Section 3.2) that cannot be triggered automatically by the ambulance's position, namely, commissioning, 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 4 also shows three softgoals and their respective QCs that refer to time-related requirements that have been elicited from the different LAS-CAD publications:

- 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 min [REQ-16].
- Once ambulances have acknowledged dispatching instructions, they should arrive at the incident's location in up to 8 min [REQ-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 min [REQ-17].

3.5. Variability in goal-based specifications

Goal models with high variability are those that offer alternative ways of satisfying its goals. This redundancy is fundamental to the task of reconfiguration in adaptive systems as it allows the selection of different alternatives to satisfy system goals at run-time.

The topic of requirements variability has been quite explored by research literature. According to [32], variability in the design of software-intensive systems can come from many different sources, such as goal refinements (decomposition/operationalization), different countermeasures for risks, different resolutions for conflicts, and different actors to which goals/tasks can be assigned. These situations lead to design decisions, which in turn lead to different system proposals and different software architectures. Different approaches for variability in goal models can be seen, for example, in [33–37].

Note, however, that we have modeled the system requirements so far with no variability whatsoever: there are no OR-refinements in Figure 4. This has been done on purpose to keep the model simpler at this stage. In the next sections, we show the result of applying Zanshin (cf. Section 2) to the base GORE model presented in this section, adding variability to the requirements during the application of our approach.

4. AWARENESS REQUIREMENTS ENGINEERING FOR THE A-CAD

In this section, we apply the first step of the Zanshin approach to the CAD system with the objective of designing the A-CAD.

4.1. Possible computer-aided ambulance dispatch failures

The input for this activity is the goal-based specification of Figure 4. We now proceed to identify requirements and assumptions that are critical to the success of the system in order to attach to them certain adaptation actions in a later step that would be taken whenever the system does not satisfy such requirements.

Again, using available publications on the LAS-CAD case as source, we analyzed what are some possible situations to which a CAD software might have to adapt in order to specify AwReqs to some of these situations as part of our experiment. The following list contains some CAD-related failures that were considered possible causes for the LAS-CAD demise:

- misusage: lack of cooperation from staff and crew, ranging from willful misusage to direct sabotage of the system; staff/crew members unfamiliar with the system or improperly trained

DOI: 10.1002/spe

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, and so on;
- 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), and so on;
- unreliable software: errors or incorrect information produced by any of the softwares associated with the CAD system;
- 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. This 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. This can cause the system to generate dispatching instructions that are not optimal, causing other problems such as sending crews to unfamiliar territory;
- mobile data terminal problems: MDTs that lock up, are not readable, or malfunction because of poor installation or maintenance can cause transmission problems, misusage, or stale information;
- slow response speed: ambulances take too long to arrive because of 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); and
- flood of exception messages: exception messages should be generated in a few specific situations (c.f. Section 3.3). Other errors, such as transmission problems, misusage, and MDT problems, could cause a flood of exceptions that hinders the work of the staff.

4.2. Awareness requirements elicitation

We have identified 12 AwReqs for the A-CAD, covering most of the aforementioned problems. It is important to note, however, that the list of AwReqs 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 A-CAD that would be used in practice in a big city such as London would most certainly require a lot more effort and elicit many other AwReqs in the process.

Figure 5 shows the A-CAD goal model with added AwReqs. A new task—*Get feedback*, under goal *Resource mobilization*—was also added to cope with the *unfamiliar territory* problem. Table I summarizes the elicited AwReqs with a short description, the CAD problem from which they originated, and the pattern that represents them (for more on AwReq patterns, refer to [9]). In what follows, we justify the elicitation of each AwReq, explaining its rationale.

Flood of calls: The CAD's initial solution assumed that up to 1500 calls are received per day. If many more calls 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, in turn, says that AwReq AR1 should succeed 90% of the time within any one month period. This meta-AwReq raises awareness of the possibility that the average number of calls per day is rising and the system should reconfigure to support a greater 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

DOI: 10.1002/spe

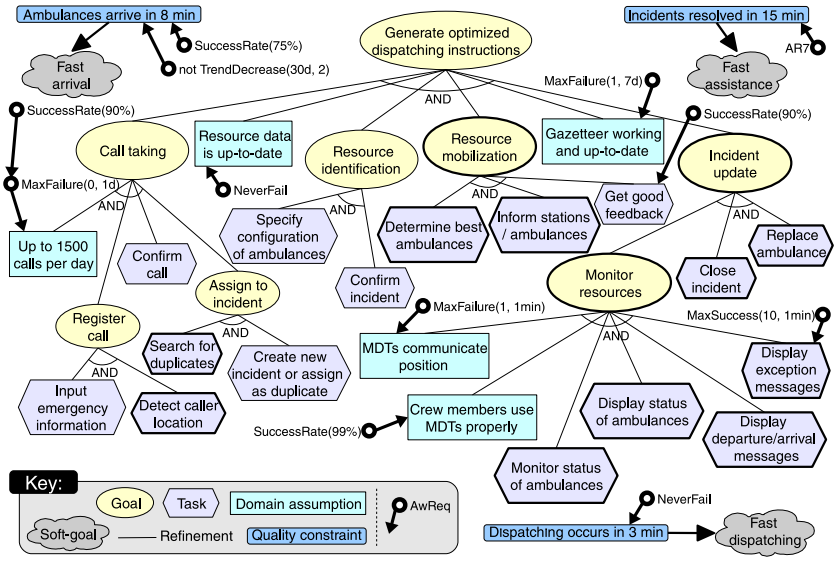

Figure 5. Goal model for the adaptive computer-aided ambulance dispatch system-to-be, with the elicited awareness requirements (AwReqs).

Table I. Summary of the awareness requirements elicited for the adaptive computer-aided ambulance dispatch system.

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%)

AwReq: awareness requirement; QC: quality constraint; DA: domain assumption.

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 surpass 1500. If AR1 were not aggregate, AR2's percentage would be applied to the number of calls, not the number of days. ORCON standard: This the standard the LAS is supposed to comply with. This standard motivated functional requirement REQ-12, which says that 75% of the ambulances should arrive within

DOI: 10.1002/spe

Listing 1: Some of the *AwReqs* of the A-CAD, specified in OCL_{TM}.

```
package acad

-- AwReq AR1: domain assumption 'Up to 1500 calls received per day' should always be true.
context D_MaxCalls
  inv AR1: never(self.oclInState(Failed))

-- AwReq AR2: AwReq 'AR1' should succeed 95% of the time considering month periods.
context AR1
  def: all : Set = AR1.allInstances()
  def: month : Set = all->select(x | new Date().difference(x.time, DAYS) <= 30)
  def: monthSuccess : Set = month->select(x | x.oclInState(Succeeded))
  inv AR2: always(monthSuccess->size() / month->size() >= 0.95)

-- AwReq AR3: quality constraint 'Ambulances arrive in 8 min' should have 75% success rate.
context Q_AmbArriv
  def: all : Set = Q_AmbArriv.allInstances()
  def: success : Set = all->select(x | x.oclInState(Succeeded))
  inv AR3: always(success->size() / all->size() >= 0.75)

-- AwReq AR4: the success rate of quality constraint 'Ambulances arrive in 8 min' should
     not decrease 2 months in a row.
context Q_AmbArriv
  def: all : Set = Q_AmbArriv.allInstances()
  def: m1 : Set = all->select(x | new Date().difference(x.time, MONTHS) == 1)
  def: m2 : Set = all->select(x | new Date().difference(x.time, MONTHS) == 2)
  def: m3 : Set = all->select(x | new Date().difference(x.time, MONTHS) == 3)
  def: success1 : Set = m1->select(x | x.oclInState(Succeeded))
  def: success2 : Set = m2->select(x | x.oclInState(Succeeded))
  def: success3 : Set = m3->select(x | x.oclInState(Succeeded))
  def: rate1 : Double = success1->size() / m1->size()
  def: rate2 : Double = success2->size() / m2->size()
  def: rate3 : Double = success3->size() / m3->size()
  inv AR4: never((rate1 < rate2) and (rate2 < rate3))

-- AwReq AR6: domain assumption 'Gazetteer working and up-to-date' should not be false
     more than once per week.
context D_GazetUpd
  def: all : Set = D_GazetUpd.allInstances()
  def: week : Set = all->select(x | new Date().difference(x.time, DAYS) <= 7)
  def: weekFail : Set = week->select(x | x.oclInState(Failed))
  inv AR6: always(weekFail.size() <= 1)

-- 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.
context T_MonitorStatus
  def: related : Set = T-InformAmbs.allInstances()->select(x | x.argument("incident") =
       self.argument("incident"))
  inv AR7: eventually(self.argument("status") = "released") and never(related->exists(x |
       x.time.difference(self.time, MINUTES) > 12))

-- AwReq AR10: task 'Display exception messages' should successfully execute no more than
     10 times per minute.
context T_Except
  def: all : Set = T_Except.allInstances()
  def: minute : Set = all->select(x | new Date().difference(x.time, SECONDS) <= 60)
  def: minuteSuccess : Set = minute->select(x | x.oclInState(Succeeded))
  inv AR10: always(minuteSuccess.size() <= 10)
endpackage
```

8 min to the location of the incident. That is precisely what AwReq AR3 imposes over the QC *Ambulances arrive in 8 min*. Furthermore, AwReq AR4 alerts staff about a decreasing trend in the success rate of the QC, which could allow one to fix the causes of this problem before it goes lower than 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 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 QC *Dispatching occurs in 3 min* and AwReq AR11, which indicates the QC 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: 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

DOI: 10.1002/spe

12 min. Counting the 3 min of dispatching, that gives a total of 15 min for incident response, as prescribed by QC *Incidents resolved in 15 min*.
Transmission problems: The CAD goal model of Figure 5 includes the domain assumption *MDTs communicate position*, because current position of each ambulance is essential to a proper ambulance dispatch. AwReq AR8 establishes that this can fail at most once per minute.
Misusage: 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.

4.3. Verification of domain assumptions

Nearly half of the AwReqs in Table I impose constraints on domain assumptions being true, which denotes the importance of adapting to changes in the environment in which the A-CAD operates. The following list specifies how each domain assumption should be checked:

- Up to 1500 calls received per day: there are many ways of keeping the count of how many calls the emergency service receives. For instance, calls could be logged into a database that is queried periodically;
- Resource data is up-to-date: deemed false if any crew or staff member reports inconsistencies between the information shown by the system and the reality;
- Gazetteer working and up-to-date: checked in the same fashion as the previous assumption, plus it is 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 every 13 s; and
- 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, refueling, and so on).

4.4. Awareness requirement specification

Finally, to prevent ambiguity from reading the AwReqs' descriptions in natural language, each AwReq has been specified in OCL_{TM} [38] (see [9] for more details). Listing 1 shows the specification of some of the AwReqs from Table I. The ones that were omitted are very similar to specifications already included in the listing. These AwReqs provide monitoring requirements for the A-CAD.

5. SYSTEM IDENTIFICATION FOR THE ADAPTIVE COMPUTER-AIDED AMBULANCE DISPATCH

Using the result of AwReqs engineering as input (i.e., Figure 5, Table I, and Listing 1), we have conducted system identification for the A-CAD. During this process, different alternatives to goal satisfaction are added to the A-CAD specification in the form of *variation points* and *control variables* [7]. We start system identification with the elicitation of parameters (§ 5.1), followed by relation identification (§ 5.2) and refinement (§ 5.3). Lastly, some final additions are made, showing that the process can be iterative (§ 5.4).

DOI: 10.1002/spe

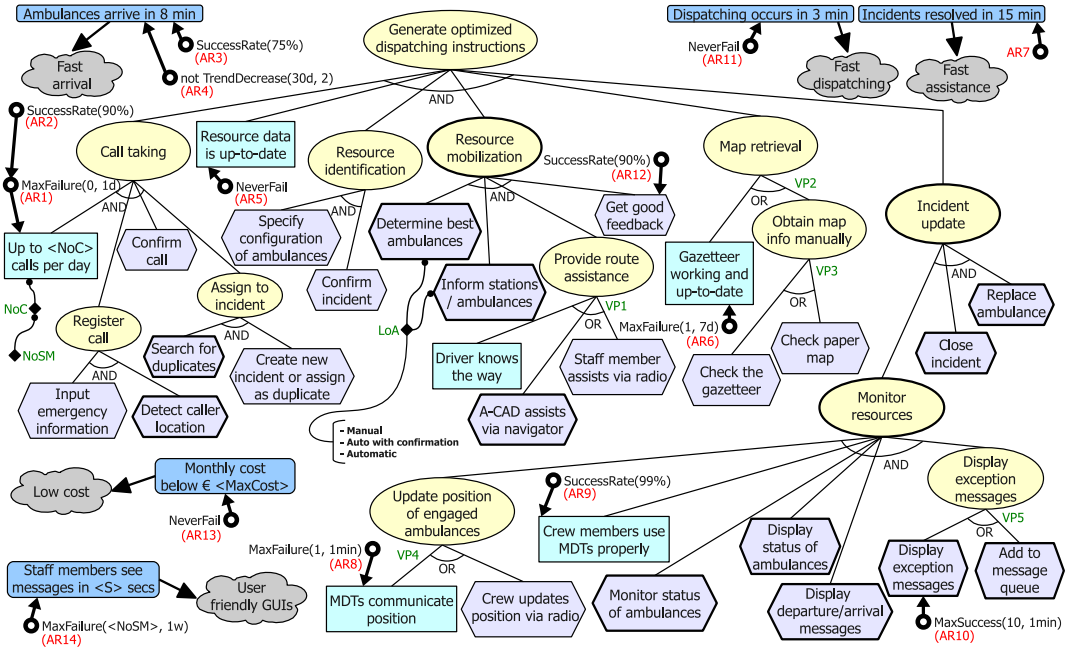

Figure 6. Goal model for the adaptive computer-aided ambulance dispatch system-to-be, with identified control variables and variation points.

5.1. *Parameter identification*

Considering AwReqs as indicators and trying to come up with possible variability scenarios that could help in case they fail, the goal model was augmented with eight new parameters: control variables *NoC*, *NoSM*, and *LoA* and variation points *VP1* through *VP5*. New goals and tasks were added to the goal model as well. Figure 6 shows the resulting goal model from system identification, including all identified parameters. This new model also shows the name of the AwReqs next to their graphical representation for an easier reference in the explanations that follow.

Rationale: Control variables *NoC*—maximum *Number of Calls* that can be handled daily—and *NoSM*—*Number of Staff Members* working on the present day—are associated with domain assumption *Up to 1500 calls per day*, which has 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.

Parameter *NoC* is a special kind of parameter that cannot be set directly. Instead, it is declared as a function of another parameter, namely, *NoSM*. The maximum number of calls the service can take in a day is then calculated on the basis of the number of staff members working on that specific day. Changing the number of staff members on duty can affect positively or negatively the success rate of the assumption, thus affecting AwReqs AR1 and AR2.

Control variable *LoA*—*Level of Automation* of the dispatch procedure—is an enumerated parameter that is associated with tasks *Specify configuration of ambulances* and *Inform stations / ambulances* and can assume one of three values: *manual* (dispatch will be done completely manually by staff members, communicating with ambulances and stations via radio), *semi-automatic* (dispatch orders are suggested by the A-CAD but are sent only if a staff member confirms that it is indeed the best choice), or *automatic* (the A-CAD autonomously generates dispatch orders and sends them to ambulances/stations).

Changing the value of this parameter can affect 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*

DOI: 10.1002/spe

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* was elicited to provide alternatives for improving 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: (i) assuming that the *Driver knows the way* and, thus, doing nothing; (ii) having the *A-CAD assist via navigator*; or (iii) having the *Staff member assist via radio*. Again, there is a trade-off between how personalized this assistance is and how much time it takes from staff members. AR7 could also be affected by changes in *LoA*: 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.

Variation points *VP2* and *VP3* have been elicited along with a new sub-tree of the main goal of the system in order to include an alternative to the gazetteer for map provision, therefore affecting 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 the alternative to using the gazetteer automatically. When 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*. Again, 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.

Finally, there are 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 of a manual fallback for when MDTs are not working properly, thus affecting 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 AR9.

Parameter *VP5* provides a simple solution to the flood of exception messages monitored by 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*. ■

5.2. Relation identification

We now specify the effect that changes in parameters have on indicators (AwReqs) using differential relations. Table II shows the initial set of indicator/parameter relations for the A-CAD. For the relations that refer to enumerated control variable *LoA* to make sense, it is required that a total order of the parameter's enumerated values be provided. This order shall be as follows: *manual* $\prec$

Table II. Initial set of differential relations of the adaptive computer-aided ambulance dispatch system.

$\Delta(I_{AR1}/NoSM)[0,maxSM] > 0$	(4)	$\Delta(I_{AR11}/VP2) < 0$	(15)
$\Delta(I_{AR2}/NoSM)[0,maxSM] > 0$	(5)	$\Delta(I_{AR12}/VP2) > 0$	(16)
$\Delta(I_{AR3}/LoA) < 0$	(6)	$\Delta(I_{AR6}/VP3) > 0$	(17)
$\Delta(I_{AR4}/LoA) < 0$	(7)	$\Delta(I_{AR11}/VP3) < 0$	(18)
$\Delta(I_{AR9}/LoA) < 0$	(8)	$\Delta(I_{AR12}/VP3) > 0$	(19)
$\Delta(I_{AR11}/LoA) > 0$	(9)	$\Delta(I_{AR8}/VP4) > 0$	(20)
$\Delta(I_{AR12}/LoA) < 0$	(10)	$\Delta(I_{AR9}/VP4) > 0$	(21)
$\Delta(I_{AR3}/VP1) > 0$	(11)	$\Delta(I_{AR11}/VP4) < 0$	(22)
$\Delta(I_{AR4}/VP1) > 0$	(12)	$\Delta(I_{AR10}/VP5) > 0$	(23)
$\Delta(I_{AR7}/VP1) > 0 \Delta(I_{AR11}/VP1) < 0$	(13)	$\Delta(I_{AR13}/NoSM) < 0$	(24)
$\Delta(I_{AR6}/VP2) > 0$	(14)	$\Delta(I_{AR14}/VP5) < 0$	(25)

DOI: 10.1002/spe

semi-automatic $\prec$ *automatic*. Variation points assume their default order, that is, ascending from left to right according to their position in the model. For more details, please refer to [7].

The rationale for most relations were presented in the previous step because they motivated the very elicitation of the parameters. However, at this step of the process, each parameter was again compared with each AwReq 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 I_{AR11} (QC *Dispatching occurs in 3 min* should never fail). A higher level of automation (*LoA*) improves it, compared with relation (9), whereas choosing to do tasks manually with the involvement of a staff member (*LoA* and *VP1* through *VP4*) has a negative effect on it, compared with relations (9), (13), (15), (18), and (22).
- Variation points *VP2* and *VP3* also have an effect on indicator I_{AR12} (task *Get good feedback* should succeed 90% of the time), compared with relations (16) and (19). The rationale is that obtaining map information manually may help in the choice of the best ambulance to dispatch.
- Parameter *VP1*, which prescribes the kind of route assistance to give ambulance drivers, also affects indicators I_{AR3} and I_{AR4} (both refer to QC *Ambulances arrive in 8 min*), compared with relations (11) and (12). Providing route assistance may help satisfy ORCON standards.

Another activity of this step is the identification of landmark values for numeric control variables, which establish intervals in which the identified relations can be applied [7]. The only applicable numeric parameter is *NoSM* and all of its relations, namely, (4) and (5), are valid in the interval $[0, maxSM]$, *maxSM* representing the maximum number of staff members the ambulance service infrastructure can hold. *NoC* is also numeric, but it is not applicable as it cannot be directly modified (it is a function of *NoSM*). Hence, no differential relation or landmark value were identified for it.

Given the relations in Table II 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 run-time to change their values whenever there is a system failure. This change, however, may require some kind of trade-off analysis at run-time. 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 I_{AR11}.

5.2.1. Trade-off analysis. A careful analysis of the relations described previously indicates there are some indicators missing in our model of the A-CAD: I_{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, one can set everything to manual and increase the number of staff members to the maximum. Also, switching *VP5* to *Add to message queue* solves the flood of messages problem, so why not use it exclusively?

The answer to these questions relies on some implicit quality indicators, that is, 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 existence of the following stakeholder requirements (which for the sake of convenience have already been depicted in Figure 6):

- We should aim for *low cost* (softgoal). In particular, stakeholders would like *Monthly cost below* €⟨*MaxCost*⟩ (QC), where *MaxCost* is a 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* ⟨*S*⟩ *secs*, where *S* is a variable representing the maximum amount 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.

DOI: 10.1002/spe

Table III. Refinements for the differential relations of the adaptive computer-aided ambulance dispatch system.

$$|\Delta(I_{AR3}/VP1)| > |\Delta(I_{AR3}/LoA)| \quad (26)$$

$$|\Delta(I_{AR4}/VP1)| > |\Delta(I_{AR4}/LoA)| \quad (27)$$

$$VP2 \neq \langle \textit{Obtain map info manually} \rangle \rightarrow |\Delta(I_{AR6}/VP3)| = 0 \quad (28)$$

$$|\Delta(I_{AR9}/VP4)| > |\Delta(I_{AR9}/LoA)| \quad (29)$$

$$|\Delta(I_{AR11}/VP2)| > |\Delta(I_{AR11}/LoA)| > |\Delta(I_{AR11}/VP3)| > |\Delta(I_{AR11}/VP1)| > |\Delta(I_{AR11}/VP4)| \quad (30)$$

$$|\Delta(I_{AR12}/VP2)| \approx |\Delta(I_{AR12}/VP3)| \approx |\Delta(I_{AR12}/LoA)| \quad (31)$$

Table II already shows differential relations regarding these two new indicators: relation (24) relates *NoSM* to I_{AR13}, whereas relation (25) relates I_{AR14} to *VP5*, thus completing this step.

5.3. Relation refinement

By comparing parameters associated with the same indicator, we have identified six new relations, shown in Table III. When reading these comparisons, consider that the unit for *NoSM* is *one staff member*—specified $U_{NoSM} = 1$—so when other parameters are compared with *NoSM*, they are comparing with '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.

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

Finally, it is important to note once again 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 QCs 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.

5.4. Final additions to the adaptive computer-aided ambulance dispatch model

To better illustrate some adaptation strategies, we have included a few new elements in the goal model of the A-CAD, resulting in the model shown in Figure 7. The new elements are

- numeric control variable *Minimum Search Time* (*MST*), representing the minimum amount of time (in seconds) staff members must dedicate to the task of searching for duplicates;
- softgoal *Unambiguity*, operationalized by QC *No unnecessary extra ambulances dispatched*;
- AwReq AR15, which specifies that the goal *Register call* should never fail (`NeverFail (G_RegCall)`); and
- AwReq AR16, imposing a comparable delta constraint that verifies that, in fact, the number of ambulances at the scene is the same as the number of ambulances in the configuration of the dispatch (`ComparableDelta(T_SpecConfig, Q_NoExtra, numAmb, 0)`);

DOI: 10.1002/spe

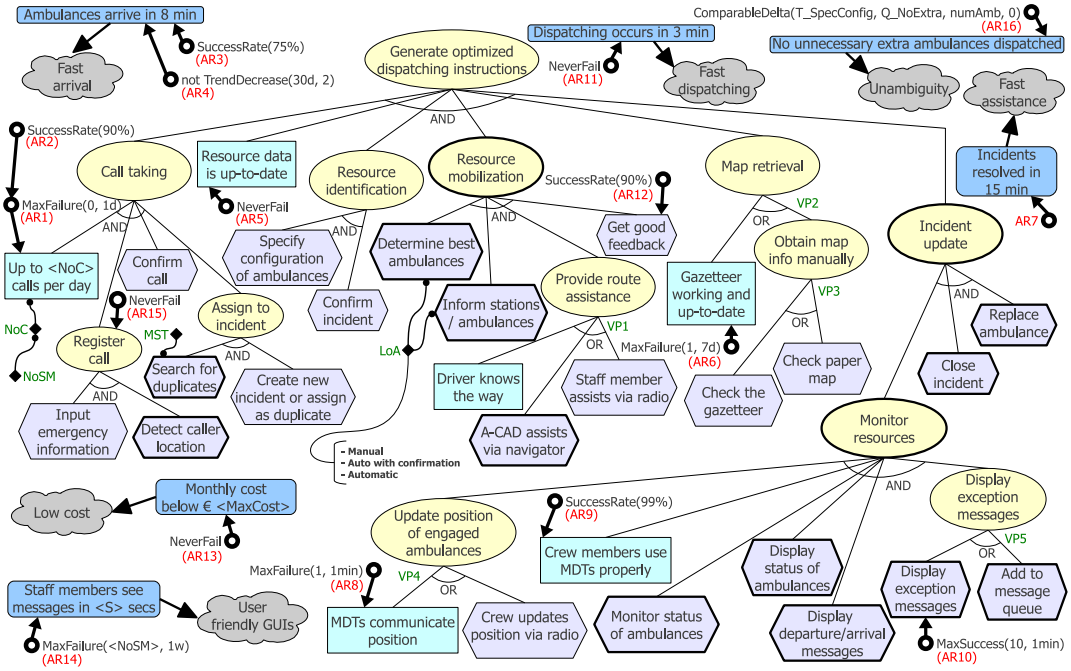

Figure 7. Final goal model for the adaptive computer-aided ambulance dispatch system-to-be.

Table IV. New differential relations and refinements after the final additions to the specification.

$$\Delta\left(I_{AR12}/MST\right)[0,180] > 0 \quad (32)$$
$$\Delta\left(I_{AR11}/MST\right)[0,180] < 0 \quad (33)$$
$$\Delta\left(I_{AR13}/MST\right)[0,180] > 0 \quad (34)$$
$$\Delta\left(I_{AR16}/MST\right)[0,180] > 0 \quad (35)$$
$$\Delta\left(I_{AR16}/LoA\right) < 0 \quad (36)$$
$$\ldots > |\Delta\left(I_{AR11}/VP4\right)| > |\Delta\left(I_{AR11}/MST\right)| \quad (37)$$
$$|\Delta\left(I_{AR12}/VP2\right)| \approx |\Delta\left(I_{AR12}/VP3\right)| \approx |\Delta\left(I_{AR12}/LoA\right)| \approx |\Delta\left(I_{AR12}/MST\right)| \quad (38)$$
$$|\Delta\left(I_{AR13}/NoSM\right)| > |\Delta\left(I_{AR13}/MST\right)| \quad (39)$$
$$|\Delta\left(I_{AR16}/MST\right)| > |\Delta\left(I_{AR16}/LoA\right)| \quad (40)$$

Rationale: *MST* 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 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*. AR15 represents the fact that the goal *Register call* 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. ■

Table IV 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 *MST* will affect negatively the success of QC *Dispatching occurs in 3 min* (I_{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 (33).
- On the other hand, increasing *MST* affects positively I_{AR16}—the more time spent searching for duplicates, the less chance of an ambiguous dispatch (35)—I_{AR12}—duplicate dispatches

DOI: 10.1002/spe

will most likely get bad feedback from crew members who will be sent to assist an incident unnecessarily (32)—and I_{AR13}—duplicate dispatches represent waste of resources and therefore money (34).

- The *Level of Automation* also affects I_{AR16} (i.e., QC *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 (36).
- Regarding indicators I_{AR11} and I_{AR13}, parameter *MST* is the one with the lowest effect (37) and (39). On the other hand, when dealing with *Unambiguity* (i.e., indicator I_{AR16}), *MST* is better than *LoA* (40). For I_{AR12}, all parameters have roughly the same effect, including the new parameter *MST* (38).

6. EVOLUTION REQUIREMENTS ENGINEERING FOR THE ADAPTIVE COMPUTER-AIDED AMBULANCE DISPATCH

The last activity of the Zanshin approach provides the last piece of the adaptation requirements of the A-CAD: EvoReqs represent a series of precise changes to be performed in the goal model [11] or the execution of a reconfiguration algorithm that uses the information collected during system identification [8]. When associated with AwReqs, we refer to them as adaptation strategies.

The complete specification of the adaptation strategies for the A-CAD is provided in Table V, associating to each AwReq one or more strategies that are attempted in the specified order and respecting each strategy's applicability condition, shown in the right-most column of the table. Only the first applicable strategy is executed, ignoring any other possibly applicable strategy down the list. The following paragraphs explain the table's contents, mentioning reconfiguration algorithms and their properties, which are defined in more detail in [8] and, therefore, not repeated here.

Rationale: 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. Because hiring and firing staff members should not be done automatically by a software system, the first strategy is to warn the ambulance service managers.

Then, reconfiguration is provided as a second strategy to try, in case the managers do not respond. Given that only one parameter is related to these AwReqs, the default procedure (represented by $\varnothing$) will be used to deal with their failures. The default procedure is the simplest reconfiguration algorithm provided by Zanshin, in which a parameter is chosen randomly from the list of parameters that affect the failed indicator (details in [8]). However, it is important to note that the *maturation time* of parameter *NoSM* is 5 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 QC *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. Because AR3 sets the threshold for the success rate of the QC, 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, that is, *ascending order*.

Moreover, when the negative trend of the QC is still high (over 90%), before executing the reconfiguration strategy for AR4, the AwReq is first relaxed to consider periods of 60 days instead of 30. Concurrently, however, AR3 is changed to fail at 80% instead of 75%, strengthening the constraint on the success rate of the requirement.

AR5 does not have any parameters associated with it, and thus, reconfiguration is not applicable. Because it refers to failures of the domain assumption *Resource data is up-to-date*, we delegate the solution to the staff member whose session of use triggered the AwReq, waiting for her to check the system responsible for registration of resources and fix the problem manually.

AwReq AR6 imposes a maximum failure constraint on the domain assumption *Gazetteer working and up-to-date* with related parameters *VP2* and *VP3*. *VP2* has to be changed first; otherwise, changing

DOI: 10.1002/spe

Table V. Final specification of adaptation strategies for the adaptive computer-aided ambulance dispatch experiment.

AwReq	Adaptation strategies	Applicability conditions
AR1	1. *Warning("AS Management")* 2. *Reconfigure(∅)*	1. Once per adaptation session. 2. Always.
AR2	1. *Warning("AS Management")* 2. *Reconfigure(∅)*	1. Once per adaptation session. 2. Always.
AR3	1. *Reconfigure({Ordered Effect Parameter Choice} [order = descending])*	1. Always.
AR4	1. *RelaxReplace(AR4, AR4_60Days) + StrengthenReplace(AR3, AR3_80Pct)* 2. *Reconfigure({Ordered Effect Parameter Choice} [order = ascending])*	1. Only if *AR3*'s success rate is above 90%. 2. Always.
AR5	1. *Delegate("Staff Member")*	1. Always.
AR6	1. *Reconfigure(∅ [n = 2])*	1. Always.
AR7	1. *Reconfigure(∅)*	1. Always.
AR8	1. *RelaxReplace(D_MDTPos_20Secs)* 2. *RelaxReplace(AR8, AR8_45Secs)* 3. *RelaxReplace(AR8_45Secs, AR8_30Secs)* 4. *Retry(60000)* 5. *Reconfigure(∅ [Immediate Resolution])*	1. Once per adaptation session. 2. Once per adaptation session. 3. Once per adaptation session. 4. Up to three times per session. 5. Always.
AR9	1. *Reconfigure({Ordered Effect Parameter Choice} [order = descending])*	1. Always.
AR10	1. *Reconfigure(∅ [Immediate Resolution])*	1. Always.
AR11	1. *Reconfigure({Oscillation Value Calculation, Oscillation Resolution Check})* 2. *Reconfigure({Ordered Effect Parameter Choice} [order = descending])*	1. Always. 2. Always.
AR12	1. *Reconfigure(∅)*	1. Always.
AR13	1. *Reconfigure({Ordered Effect Parameter Choice}* [order = ascending, repeat policy = max 2 times])	1. Always.
AR14	1. *Reconfigure(∅ [Immediate Resolution])*	1. Always.
AR15	1. *Retry(5000)* 2. *RelaxDisableChild(T_DetectCaller)*	1. Once per adaptation session. 2. Once per adaptation session.
AR16	1. *Reconfigure({Ordered Effect Parameter Choice} [order = ascending])*	1. Always.

AwReq: awareness requirement.

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.

The reconfiguration strategy is also applied to AR7, AR8, AR10, and AR14. Because there is just one parameter that has an effect on each of these indicators, 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,

DOI: 10.1002/spe

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.

AR8, however, also has other adaptation strategies associated to it. The original specification for the transmission of ambulance positions is very strict; therefore we apply three relax strategies (one on the domain assumption itself and two on the AwReq) and a retry strategy to make sure it is not a temporal glitch before reconfiguring. Each of these strategies can be applied at most once per adaptation session, except for retrying, which can be done up to three times.

For AR9, AR13, and AR16, the ordered parameter choice was also selected, being used in an ascending order for the latter two AwReqs. 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*).

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.

Finally, AR15, like AR5, is also not affected by reconfiguration and is associated with two adaptation strategies: retrying goal *Register call* after 5 s (in case the failure is due to a temporary error in the input form) and relaxing the goal by disabling task *Detect caller location* (in case caller detection is not working), as it is not essential to ambulance dispatch (the staff member can ask the caller for her location). Each of these strategies can only be tried once, so if both of them fail, the controller will resort to the *Abort* strategy, the default fallback for all AwReqs, in which the system is instructed to give up adapting and fail as gracefully as possible. ■

The goal model of Figure 7, the differential relations in tables II, III, and IV and, finally, the adaptation strategies specified in Table V provide a complete requirements specification for the A-CAD. From these specs, one could go on to design the architecture and implement the system with the prescribed requirements for adaptation. Zanshin offers a framework that already implements the generic functionality of the feedback loop that provides the system with adaptation capabilities, relieving developers from the coding effort. We present it next.

7. THE ZANSHIN FRAMEWORK

The Zanshin framework works as depicted in Figure 8. Its source code is available at the public version control repository https://github.com/sefms-disi-unitn/Zanshin to any developer who wishes to use and/or modify it.

First, the *target system* (the base system to which the framework will add adaptation capabilities, e.g., the A-CAD) is instrumented in order to provide a log that indicates when instances of requirements have changed state (i.e., have succeeded, failed, and been canceled).

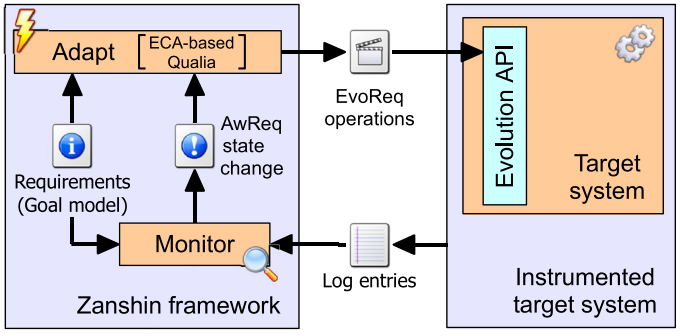

Figure 8. Overview of the Zanshin framework in action.

DOI: 10.1002/spe

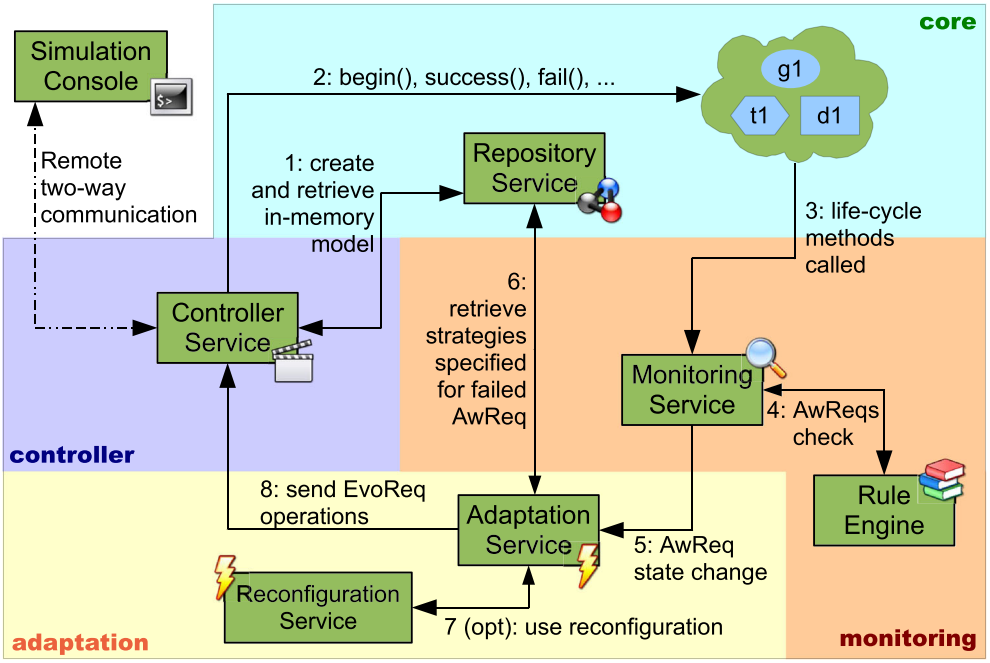

Figure 9. The architecture of the Zanshin framework.

Given this log and the requirements specification, such as the one presented for the A-CAD earlier, the *Monitor* component concludes if and when certain AwReqs have themselves changed state (which includes not only AwReq failures but also AwReqs being satisfied). These state changes then trigger an *Adapt* component that decides which requirement evolution operations the target system should execute. This component can be divided in two main parts:

- an ECA-based adaptation component [11] that chooses an adaptation strategy based on the list of strategies associated with the AwReq failure and their respective applicability conditions and
- a qualitative reconfiguration component [8], which is activated by the ECA-based process when reconfiguration is selected as the appropriate strategy and executes the reconfiguration algorithm that has been specified in the adaptation strategy.

In either case, the output of the *Adapt* component is a list of EvoReq operations [11] that are sent to the target system. The latter then carries on application and domain-specific actions on the basis of the instructions given by the EvoReq operations. In Figure 8, this is represented by the *Evolution API* component, which should be implemented by the target system.

7.1. Zanshin's architecture

The Zanshin framework was implemented with four main OSGi[¶] bundles (components): **`controller`**, **`core`**, **`monitoring`**, and **`adaptation`** (plus a few auxiliary bundles). Each component implements different services that together provide the feedback loop functionality for target systems. Figure 9 shows the internal architecture of the framework.

A plain Java project called **`zanshin-simulations`**, also available in the repository, offers a simple text-based interface for running simulations using the Zanshin framework (some of which are described in Section 8). By using RMI,[‖] it provides the controller with a model of the target system

[¶]The Open Services Gateway initiative framework is a module system and service platform for the Java programming language. It allows components to be implemented as bundles that can be remotely installed, started, stopped, updated, and uninstalled without requiring the component container itself to be restarted. See http://www.osgi.org/.
[‖]Java Remote Method Invocation is a technology that enables programmers to create distributed Java applications. See http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html.

DOI: 10.1002/spe

and logging information about its run-time execution (requirements and log entries in Figure 8). Once Zanshin receives this information, it executes the following process:

1. The *Controller Service* sends the model to the *Repository Service* so it can create a reification of the goal model in memory. For every user session, the controller obtains a separate copy of this in-memory model from the repository.
2. Once the controller receives logging entries indicating that the user started pursuing a requirement (`begin`), successfully completed a requirement (`success`), could not fulfill a requirement successfully (`fail`), and so on, it obtains the instance of the class that represents the given requirement and calls the appropriate life-cycle method in this instance.
3. Upon receiving life-cycle method calls, the objects that represent the target system's requirements notify the Monitoring Service about them. Moreover, basic propagation of these life-cycle events up the refinement hierarchy is also performed by the model itself (e.g., if a goal is OR-refined into N tasks, once any of these tasks receive a `success()` method, the same method will also be called in the parent goal).
4. The Monitoring Service relies on a *Rule Engine* to check if any AwReqs have been satisfied or failed. The Rule Engine performs this task on the basis of an OCL_{TM} specification of the AwReqs similar to the one presented in Listing 1. The monitoring of AwReqs using OCL_{TM} was implemented in [9], but integrating it to the Zanshin framework is currently in our to-do list. In the mean time, the framework works only with `NeverFail`-type of AwReqs.
5. If any AwReq state change is detected, it is sent to the *Adaptation Service*, who will manage separate adaptation sessions for each AwReq [11]. AwReq failures would trigger the creation of new sessions or the retrieval of an existing one, whereas AwReq successes might (depending on the AwReq's resolution condition) close existing sessions.
6. In case of AwReq failures, the Adaptation Service retrieves from the requirements model at the repository the appropriate adaptation strategies, checking for their applicability and executing the first strategy that is currently applicable (i.e., it uses the ECA-based process).

   Strategies are executed within adaptation sessions (explained previously). If a given strategy fails to solve the problem, the ECA algorithm will come back to the same adaptation session to try the same or another strategy, depending on their applicability conditions (e.g., if a strategy can be executed at most once per session, it will not be applied again if it fails).
7. If the selected adaptation strategy is to `Reconfigure`, the *Reconfiguration Service* will be called and asked to produce a new system configuration to respond to the failure.
8. The EvoReq operations relative to the selected adaptation strategy are submitted to the Controller Service, which finally sends them back to the target system (in this case, the *Simulation Console*), completing the feedback loop.

In the framework's current implementation, requirements models are specified using Eclipse Modeling Framework (EMF)[**] meta-models: the `core` component provides the basic GORE classes and the classes involved in the ECA-based process [11]. These meta-models should then be extended for each target system to provide classes representing its specific requirements. Once the system's meta-model is available, a model can be created to represent the system's goal model in a machine-readable format.

In the next section, we describe some of the simulations implemented in the `zanshin-simulations` project that are based on the A-CAD model described in this paper. Simulations with other systems, including a Meeting Scheduler [7, 11], an ATM machine [39], and the news website *ZNN.com* [40], are also provided. For readers interested in obtaining Zanshin, running the existing simulations or even creating their own, please refer to the project's *wiki* at https://github.com/sefms-disi-unitn/Zanshin/wiki.

[**]EMF is a modeling framework and code generation facility for the Eclipse platform. From a model described in an XML-based language, EMF can produce a set of Java classes representing the model, as well as adapters and editors. See http://www.eclipse.org/modeling/emf/.

DOI: 10.1002/spe

8. SIMULATIONS OF THE ADAPTIVE COMPUTER-AIDED AMBULANCE DISPATCH SYSTEM

We have adopted experimental methods from Design Science [13] in order to validate the Zanshin framework, simulating run-time failures of A-CAD on the basis of its requirements specification and evaluating the response of the framework. The objective of these experiments was to provide indications that the Zanshin framework offers a systematic process for the design of adaptive systems.

This section describes in detail two simulations that were conducted using A-CAD models. Section 10 includes a discussion on the lessons learned from these experiments. Simulations extend the base EMF meta-models and provide a model that represents target system requirements. Listing 2 shows the EMF encoding of the A-CAD goal model.

The entire goal tree of Figure 7 is represented in the aforementioned EMF model (lines 3–55), along with the A-CAD's softgoals (lines 58–62), QCs (lines 65–69), and AwReqs (lines 72–99). Lines 104 and 108 show, respectively, one of the parameters and differential relations of the A-CAD, used in simulation 2.

Although it may look complex and cumbersome to write (especially when XMI links are needed, e.g., `//@rootGoal/@children.6`), the Eclipse IDE allows developers to generate graphical editors for these kinds of models on the basis of their EMF meta-models, which greatly facilitates their creation. A detailed explanation of this procedure can be found in the project's wiki page titled 'How to create your own Zanshin simulation'.

8.1. Simulation 1: adaptation through evolution

The first simulation involves a failure of AwReq AR15, which refers to *Register call* as its target using the XMI link `//@rootGoal/children.0/children.1` (Listing 2, line 91), that is, starting at the root goal, navigate to the child with index 0 (`G_CallTaking`), then in that element navigate to the child of index 1 (`G_RegCall`). The numbers in the comments next to some elements of the listing show the index of the children of the root goal, facilitating their location.

In line 92, AR15 is specified to have a simple resolution condition—that is, if the AwReq evaluation succeeds, the problem is solved—and the two associated adaptation strategies specified in Table V: **`Retry(5000)`** (lines 93–95) and **`RelaxDisableChild(T_DetectLoc)`** (lines 96–98). Both strategies are applicable at most once during an adaptation session.

After sending the requirements models to Zanshin so it can create in-memory models, this simulation consists of two parts in a single user session: first, log information about the failure of task *Input emergency information* is sent, and the simulation waits for instructions from the framework; then, we simulate the success of this task followed by a failure of task *Detect caller location*, waiting once again for instructions.

The monitoring infrastructure detects that AR15 has changed its state (once for each part), and Zanshin conducts the ECA-based coordination process [11], producing a log similar to the one shown in Listing 3. On the other side of the remote connection, the Simulation Console also produces a log, shown in Listing 4.

Zanshin's log (Listing 3) shows the framework receiving notification of life-cycle methods being called and the failure of task *Input emergency information* being propagated to goal *Register call*, which triggers an AwReq (lines 6–9). It then creates a new adaptation session `S1` for it (line 10) and searches for a suitable adaptation strategy to be applied, executing the **`Retry(5000)`** strategy (lines 12–14). On the simulation side (Listing 4), we see how Zanshin responds with the EvoReq operations related to the *Retry* strategy [11] and the simulation acknowledges them (lines 4–8). In a real system, a controller on the target system's side would interpret each command and take the appropriate application-specific action.

The second part is analogous, with Zanshin being notified of the failure of task *Detect caller location*, which also propagates to goal *Register call* and, consequently, AwReq AR15 (Listing 3, lines 29–32). The adaptation framework then retrieves the same adaptation session `S1`

DOI: 10.1002/spe

as before, realizing that it has not yet been solved (lines 33–34), and proceeds to searching for a suitable adaptation strategy, but **`Retry(5000)`** cannot be used again in the same session because of its applicability condition (line 35). The framework ends up selecting **`RelaxDisableChild(T_DetectCaller)`** and executing it (lines 36–38), which again is recognized by the target system controller (Listing 4, lines 11–15).

Listing 2: EMF specification of A-CAD requirements.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <acad:AcadGoalModel xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http:
       //www.w3.org/2001/XMLSchema-instance" xmlns:ecore="http://www.eclipse.org/emf/2002/
       Ecore" xmlns:acad="http://acad/1.0" xmlns:it.unitn.disi.zanshin.model="http://zanshin.
       disi.unitn.it/1.0/eca">
3    <rootGoal xsi:type="acad:G_GenDispatch">
4      <children xsi:type="acad:G_CallTaking">                          <!-- 0 -->
5        <children xsi:type="acad:D_MaxCalls"/>
6        <children xsi:type="acad:G_RegCall">
7          <children xsi:type="acad:T_InputInfo"/>
8          <children xsi:type="acad:T_DetectLoc"/>
9        </children>
10       <children xsi:type="acad:T_ConfirmCall"/>
11       <children xsi:type="acad:G_AssignIncident">
12         <children xsi:type="acad:T_SearchDuplic"/>
13         <children xsi:type="acad:T_CreateOrAssign"/>
14       </children>
15     </children>
16     <children xsi:type="acad:D_DataUpd"/>                            <!-- 1 -->
17     <children xsi:type="acad:G_ResourceId">                          <!-- 2 -->
18       <children xsi:type="acad:T_SpecConfig"/>
19       <children xsi:type="acad:T_ConfIncident"/>
20     </children>
21     <children xsi:type="acad:G_ResourceMob">                         <!-- 3 -->
22       <children xsi:type="acad:T_DetBestAmb"/>
23       <children xsi:type="acad:T_InformStat"/>
24       <children xsi:type="acad:G_RouteAssist" refinementType="or">
25         <children xsi:type="acad:D_DriverKnows"/>
26         <children xsi:type="acad:T_AcadAssists"/>
27         <children xsi:type="acad:T_StaffAssists"/>
28       </children>
29       <children xsi:type="acad:T_Feedback"/>
30     </children>
31     <children xsi:type="acad:G_ObtainMap" refinementType="or">       <!-- 4 -->
32       <children xsi:type="acad:D_GazetUpd"/>
33       <children xsi:type="acad:G_ManualMap" refinementType="or">
34         <children xsi:type="acad:T_CheckGazet"/>
35         <children xsi:type="acad:T_CheckPaper"/>
36       </children>
37     </children>
38     <children xsi:type="acad:G_IncidentUpd">                         <!-- 5 -->
39       <children xsi:type="acad:G_MonitorRes">
40         <children xsi:type="acad:G_UpdPosition" refinementType="or">
41           <children xsi:type="acad:D_MDTPos"/>
42           <children xsi:type="acad:T_RadioPos"/>
43         </children>
44         <children xsi:type="acad:D_MDTUse"/>
45         <children xsi:type="acad:T_MonitorStatus"/>
46         <children xsi:type="acad:T_DispStatus"/>
47         <children xsi:type="acad:T_DispDepArriv"/>
48         <children xsi:type="acad:G_DispExcept" refinementType="or">
49           <children xsi:type="acad:T_Except"/>
50           <children xsi:type="acad:T_ExceptQueue"/>
51         </children>
52       </children>
53       <children xsi:type="acad:T_CloseIncident"/>
54       <children xsi:type="acad:T_ReplAmb"/>
55     </children>
56
57     <!-- Softgoals. -->
58     <children xsi:type="acad:S_FastArriv"/>                          <!-- 6 -->
59     <children xsi:type="acad:S_FastDispatch"/>                       <!-- 7 -->
60     <children xsi:type="acad:S_FastAssist"/>                         <!-- 8 -->
61     <children xsi:type="acad:S_LowCost"/>                            <!-- 9 -->
62     <children xsi:type="acad:S_UserFriendly"/>                       <!-- 10 -->
63
64     <!-- Quality Constraints. -->
65     <children xsi:type="acad:Q_AmbArriv" softgoal="//@rootGoal/@children.6"/>    <!-- 11
           -->
66     <children xsi:type="acad:Q_Dispatch" softgoal="//@rootGoal/@children.7"/>    <!-- 12
           -->
67     <children xsi:type="acad:Q_IncidResolv" softgoal="//@rootGoal/@children.8"/> <!-- 13
           -->
68     <children xsi:type="acad:Q_MaxCost" softgoal="//@rootGoal/@children.9"/>     <!-- 14
           -->
69     <children xsi:type="acad:Q_MaxTimeMsg" softgoal="//@rootGoal/@children.10"/> <!-- 15
           -->
70
71     <!-- AwReqs. -->
72     <children xsi:type="acad:AR1" target="//@rootGoal/@children.0/@children.0"/> <!-- 16
           -->
73     <children xsi:type="acad:AR2"/>                                  <!-- 17 -->
74     <children xsi:type="acad:AR3"/>                                  <!-- 18 -->
75     <children xsi:type="acad:AR4"/>                                  <!-- 19 -->
```


DOI: 10.1002/spe

```
    <children xsi:type="acad:AR5"/>                                    <!-- 20 -->
    <children xsi:type="acad:AR6"/>                                    <!-- 21 -->
    <children xsi:type="acad:AR7"/>                                    <!-- 22 -->
    <children xsi:type="acad:AR8"/>                                    <!-- 23 -->
    <children xsi:type="acad:AR9"/>                                    <!-- 24 -->
    <children xsi:type="acad:AR10"/>                                   <!-- 25 -->
    <children xsi:type="acad:AR11" target="//@rootGoal/@children.12" incrementCoefficient=
        "2"> <!-- 26 -->
      <condition xsi:type="it.unitn.disi.zanshin.model:ReconfigurationResolutionCondition"
          />
      <strategies xsi:type="it.unitn.disi.zanshin.model:ReconfigurationStrategy"
          algorithmId="qualia">
        <condition xsi:type="it.unitn.disi.zanshin.
            model:ReconfigurationApplicabilityCondition"/>
      </strategies>
    </children>
    <children xsi:type="acad:AR12"/>                                   <!-- 27 -->
    <children xsi:type="acad:AR13"/>                                   <!-- 28 -->
    <children xsi:type="acad:AR14"/>                                   <!-- 29 -->
    <children xsi:type="acad:AR15" target="//@rootGoal/@children.0/@children.1"> <!-- 30
        -->
      <condition xsi:type="it.unitn.disi.zanshin.model:SimpleResolutionCondition"/>
      <strategies xsi:type="it.unitn.disi.zanshin.model:RetryStrategy" time="5000">
        <condition xsi:type="it.unitn.disi.zanshin.
            model:MaxExecutionsPerSessionApplicabilityCondition" maxExecutions="1"/>
      </strategies>
      <strategies xsi:type="it.unitn.disi.zanshin.model:RelaxDisableChildStrategy" child="
          //@rootGoal/@children.0/@children.1/@children.1">
        <condition xsi:type="it.unitn.disi.zanshin.
            model:MaxExecutionsPerSessionApplicabilityCondition" maxExecutions="1"/>
      </strategies>
    </children>
  </rootGoal>

  <!-- System parameters. -->
  <configuration>
    <parameters xsi:type="acad:CV_MST" type="ncv" unit="10" value="60" metric="integer"/>
  </configuration>

  <!-- Indicator / parameter differential relations. -->
  <relations indicator="26" parameter="//@configuration/@parameters.0" lowerBound="0"
      upperBound="180" operator="ft" />
</acad:AcadGoalModel>
```

Listing 3: *Zanshin* execution log for the *AR15* simulation.

```
Successfully created a new user session for target system acad: 1
Processing method call: start / T_InputInfo
Processing method call: start / G_RegCall
Processing method call: start / G_CallTaking
Processing method call: start / G_GenDispatch
Processing method call: fail / T_InputInfo
Processing method call: end / T_InputInfo
Processing method call: fail / G_RegCall
Processing state change: AR15 (ref. G_RegCall) -> failed
(Session: S1) Created new session for AR15
(Session: S1) The problem has not yet been solved...
(Session: S1) Strategy RetryStrategy is applicable.
(Session: S1) Selected adaptation strategy: RetryStrategy
(Session: S1) Applying strategy RetryStrategy(true; 5000)...
(Session: S1) The problem has not yet been solved...
Processing method call: end / G_RegCall
Processing method call: fail / G_CallTaking
Processing method call: end / G_CallTaking
Processing method call: fail / G_GenDispatch
Processing method call: end / G_GenDispatch

Processing method call: start / T_InputInfo
Processing method call: start / G_RegCall
Processing method call: start / G_CallTaking
Processing method call: start / G_GenDispatch
Processing method call: success / T_InputInfo
Processing method call: end / T_InputInfo
Processing method call: start / T_DetectLoc
Processing method call: fail / T_DetectLoc
Processing method call: end / T_DetectLoc
Processing method call: fail / G_RegCall
Processing state change: AR15 (ref. G_RegCall) -> failed
(Session: S1) Retrieved existing session for AR15, 1 event in the timeline
(Session: S1) The problem has not yet been solved...
(Session: S1) Strategy RetryStrategy is not applicable because it has been applied at
    least 1 time(s) this session.
(Session: S1) Strategy RelaxDisableChildStrategy is applicable.
(Session: S1) Selected adaptation strategy: RelaxDisableChildStrategy
(Session: S1) Applying strategy RelaxDisableChildStrategy(G_RegCall; instance-level;
    T_DetectLoc)...
(Session: S1) The problem has not yet been solved...
Processing method call: end / G_RegCall
Processing method call: fail / G_CallTaking
Processing method call: end / G_CallTaking
Processing method call: fail / G_GenDispatch
Processing method call: end / G_GenDispatch

Processing method call: success / G_RegCall
Processing state change: AR15 (ref. G_RegCall) -> succeeded
(Session: S1) Retrieved existing session for AR15, 2 events in the timeline
(Session: S1) Problem solved. Adaptation session will be terminated.
```


DOI: 10.1002/spe

Listing 4: Simulation Console execution log for the *AR15* simulation.

```
Created a new user session with id: 1
Staff member tries to Input Emergency Information but it fails!
Waiting for a response from Zanshin...
Received EvoReq operation: copy-data(G_RegCall, G_RegCall) [session: 1]
Received EvoReq operation: terminate(G_RegCall) [session: 1]
Received EvoReq operation: rollback(G_RegCall) [session: 1]
Received EvoReq operation: wait(5,000) [session: 1]
Received EvoReq operation: initiate(G_RegCall) [session: 1]

In the 2nd try, Input Info suceeeds, but Detect Caller Location fails!
Received EvoReq operation: suspend(G_RegCall) [session: 1]
Received EvoReq operation: terminate(T_DetectLoc) [session: 1]
Received EvoReq operation: rollback(T_DetectLoc) [session: 1]
Received EvoReq operation: suspend(T_DetectLoc) [session: 1]
Received EvoReq operation: resume(G_RegCall) [session: 1]
OK. Ending user session...
```

Listing 5: *Zanshin* execution log for the *AR11* simulation.

```
Successfully created a new user session for target system acad: 2
Processing method call: start / Q_Dispatch
Processing method call: start / G_GenDispatch
Processing method call: fail / Q_Dispatch
Processing state change: AR11 (ref. Q_Dispatch) -> failed
(Session: S2) Created new session for AR11
(Session: S2) The problem has not yet been solved...
(Session: S2) Selected adaptation strategy: ReconfigurationStrategy
(Session: S2) Applying strategy ReconfigurationStrategy(qualia; class-level)...
Parameters chosen: [CV_MST]
Values to inc/decrement in the chosen parameters: [20.00000]
Produced new configuration with 1 changed parameter(s)
(Session: S2) Indicator AR11 has been evaluated to false
(Session: S2) Evaluating resolution: false
(Session: S2) The problem has not yet been solved...
Processing method call: end / Q_Dispatch
Processing method call: fail / G_GenDispatch
Processing method call: end / G_GenDispatch

Successfully created a new user session for target system acad: 3
Processing method call: start / Q_Dispatch
Processing method call: start / G_GenDispatch
Processing method call: fail / Q_Dispatch
Processing state change: AR11 (ref. Q_Dispatch) -> failed
(Session: S2) Retrieved existing session for AR11, 1 event in the timeline
(Session: S2) Indicator AR11 has been evaluated to false
(Session: S2) Evaluating resolution: false
(Session: S2) The problem has not yet been solved...
(Session: S2) Selected adaptation strategy: ReconfigurationStrategy
(Session: S2) Applying strategy ReconfigurationStrategy(qualia; class-level)...
Parameters chosen: [CV_MST]
Values to inc/decrement in the chosen parameters: [20.00000]
Produced new configuration with 1 changed parameter(s)
(Session: S2) Indicator AR11 has been evaluated to false
(Session: S2) Evaluating resolution: false
(Session: S2) The problem has not yet been solved...
Processing method call: end / Q_Dispatch
Processing method call: fail / G_GenDispatch
Processing method call: end / G_GenDispatch

Successfully created a new user session for target system acad: 4
Processing method call: start / Q_Dispatch
Processing method call: start / G_GenDispatch
Processing method call: success / Q_Dispatch
Processing state change: AR11 (ref. Q_Dispatch) -> succeeded
(Session: S2) Retrieved existing session for AR11, 2 events in the timeline
(Session: S2) Indicator AR11 has been evaluated to true
(Session: S2) Evaluating resolution: true
(Session: S2) Problem solved. Adaptation session will be terminated.
```

Finally, given that a child of goal *Register call* has been disabled, its satisfiability is recalculated and the goal succeeds because task *Input emergency information*, now its only child, succeeded. The same adaptation session `S1` is again retrieved and marked as closed (Listing 3, lines 46–49). From this point on, further failures of AR15 from the same user will create a new adaptation session.

As this simulation demonstrates, the framework is able to execute the specified adaptation strategies, sending EvoReq operations to the target system, which should then adapt according to the instructions.

8.2. Simulation 2: adaptation through reconfiguration

The second simulation involves the failure of AwReq AR11, which as specified in its `target` attribute (line 82 of Listing 2), refers to QC *Dispatching occurs in 3 min* (look for the `<!--12 -->` comment to locate the 12th child of the root goal, line 66).

DOI: 10.1002/spe

Listing 6: Simulation Console execution log for the *AR11* simulation.

```
1  Created a new user session with id: 2
2  Current incident took more than 3 minutes do dispatch!
3  Received EvoReq operation: apply-config({CV_MST=40.00000}) [from now on]
4
5  Created a new user session with id: 3
6  Not enough, dispatch still took more than 3 minutes in another incident!
7  Received EvoReq operation: apply-config({CV_MST=20.00000}) [from now on]
8
9  Created a new user session with id: 4
10 OK, for a third incident dispatching now took less than 3 minutes!
```

Instead of EvoReqs, as the previously presented simulation, AR11 has defined the reconfiguration strategy as response to its failures (line 84, *Qualia* is the name of Zanshin's reconfiguration algorithm [8]). The reconfiguration strategy has to be associated with special resolution and applicability conditions (respectively, lines 83 and 85) in order to properly execute the algorithm. Moreover, AR11 also defines its increment coefficient $K_{AR11} = 2$.

In lines 103–105, the initial system configuration specifies the existing parameters and their values. Line 104 defines numeric control variable (`ncv`) *MST*, with unit of increment $U_{MST} = 10$, initial value `60`, and `integer` metric, which tells the framework how to perform increments. Finally, the specification includes a differential relation between AwReq AR11 and *MST*, with lower bound set to `0`, upper bound set to `180`, and `ft` (fewer than) as operator, that is, $\Delta(AR11/MST)[0, 180] < 0$ (line 108). When ran, the simulation produces a log similar to the one shown in Listings 5 (for Zanshin) and 6 (for the Simulation Console).

It can be seen from Listing 5 that whenever Zanshin is made aware of a failure in AR11 (lines 5 and 24), it executes the strategy associated to this indicator in the specification (lines 9 and 30), delegating the adaptation to the reconfiguration algorithm (Qualia). The latter, in turn, chooses randomly the parameter *MST* (lines 10 and 31), decreasing it by $V = K_{AR11} \times U_{MST} = 20$ two consecutive times (lines 11 and 32), until the problem is deemed solved (lines 46–49).

On the simulation side (Listing 6), we can see that the target system receives the new values for the *MST* control variable (40 in line 3, then 20 in line 7). As this simulation demonstrates, Zanshin is able to determine a new configuration for the target system using the information encoded in the requirements specification, instructing the system on how to reconfigure itself in order to adapt.

8.3. Simulation 3: scalability tests

In previous publications [8, 9, 11], we have reported on scalability tests on the monitoring and adaptation components of Zanshin and the reconfiguration algorithms implemented by Qualia. Results from these tests indicate that the performance of these frameworks, which are still prototypes and have not been optimized for use in production, are satisfactory, scaling linearly with respect to increasing number of elements in the requirements specification.

The focus of the experiments reported in the preceding text, however, is to show that the Zanshin framework produces sensible responses to failures at run-time on the basis of the augmented requirements specification produced by the steps described earlier. For this reason, readers interested in performance tests with these frameworks can refer to the aforementioned publications or download the framework's source code and try the tests for themselves.

9. RELATED WORK

Many approaches have been proposed in the literature for the design of adaptive systems. Surveys of the literature on the topic (e.g., [78–82]) cite several of them. In Table VI, we cite those that are related to Zanshin, classified into relevant categories. A short description of most of these approaches can be found in [12]. In what follows, we compare Zanshin with the approaches in each category.

Architecture-based approaches: Such approaches concentrate on adaptation mechanisms founded on architectural models. The architectural models used often consist of components and connectors [81]. The difference between these approaches and Zanshin is their focus: the former propose

DOI: 10.1002/spe

Table VI. A categorization of works related to Zanshin.

Category	Related work
Architecture-based approaches	Oreizy *et al.* [41]; Laddaga and Robertson [42]; Kramer and Magee [43, 44]; the Rainbow Framework [45, 46]; Sousa *et al.* [47]; and the SASSY framework [48].
Requirements-based approaches	LoREM [49]; Zhang and Cheng [50]; Tropos4AS [31]; CARE [51]; Ma *et al.* [52]; RELAX [53]; LoREM+RELAX [54]; FLAGS [55]; and Dalpiaz *et al.* [56].
Control-theoretic approaches	Schmitz *et al.* [57]; Hebig *et al.* [58]; and Filieri *et al.* [59].
Run-time monitoring of requirements	Requirements monitoring [60–62]; requirements reflection [63]; and requirements-aware Systems [64].
Reconfiguration approaches	DCR [65]; Brake *et al.* [66]; Khan *et al.* [67]; Tropos4AS [31]; Wang and Mylopoulos [68]; Peng *et al.* [69]; Nakagawa *et al.* [70]; Fu *et al.* [71]; GAAM [72]; Dalpiaz *et al.* [56]; and Ali *et al.* [84].
Design-time approaches	Letier and van Lamsweerde [73]; DDP [74]; Menzies and Richardson [75]; Heaven and Letier [76]; and Elahi and Yu [77].

adaptation mechanisms at the architectural level, whereas the latter relies on requirements models to define the feedback loop that operationalizes adaptation. In this sense, requirements and architecture-based approaches are complementary and could be used in combination.

Requirements-based approaches: Similar to Zanshin, they focus on requirements. The common trait in these proposals is the design of adaptation mechanisms based on requirements models. One of the novelties of Zanshin rests on its control-theoretic perspective, making the requirements for the feedback loop that operationalizes adaptation at run-time first-class citizens in the requirements model. Prominent requirements-based approaches are listed in Table VI. Only some of these approaches offer a run-time infrastructure that uses requirements models to support adaptation for the target/base system. However, they either use heavy-handed modeling formalisms [50, 55], which can turn out to be difficult in some practical settings, or focus on specific architectures such as agent-oriented [31, 56] or service-oriented architectures [51, 55].

Some of the adaptation strategies chosen for the A-CAD may bear a resemblance to RELAX [53] and FLAGS [55]. However, our control-theoretic approach to adaptive systems constitutes a fundamental difference between these approaches and Zanshin. In their approach, by default, requirements are treated as invariants that must always be achieved, except for non-critical goals that can be relaxed. In our approach, on the other hand, we accept the fact that a system may fail in achieving any of its requirements. We then suggest that critical requirements are supplemented by AwReqs that ultimately lead to the introduction of feedback loop functionality into the system to execute adaptation actions (EvoReqs) when their failure is detected. Nonetheless, strategies such as *RelaxReplace* and *RelaxDisableChild* were definitely inspired by these related works.

Control-theoretic approaches: Other non-requirements-based approaches have applied control theory to the design of adaptive systems. However, not supporting qualitative information in their models makes them very difficult to apply in situations in which stakeholders and requirements engineers do not have precise information about the relation between system parameters and indicators. Zanshin uses qualitative reasoning, allowing adaptation requirements to be modeled at different levels of precision.

Run-time monitoring of requirements: From the seminal work of Fickas and Feather [60] to the roadmap provided by Robinson [62], the topic of requirements monitoring has been addressed by many proposals in the literature. Along with proposals on requirements reflection [63] and requirements-aware systems [64], these works are centered on the run-time representation of requirements.

This is also a key aspect of Zanshin, which needs a run-time representation of the goal model in order to evaluate AwReqs satisfaction during system execution and provide adaptation instructions in case of AwReq failure. We based our approach in Robinson's requirements monitoring framework [61], as it provided the Event Engineering and Analysis Toolkit, a ready-to-use software for specifying, instrumenting, and collecting data to enable goal-satisfaction and data-mining analysis for run-time requirements monitoring [83].

DOI: 10.1002/spe

Reconfiguration approaches: Many approaches propose reconfiguration algorithms for adaptive systems, each of which focusing on a different aspect of the system to determine the new system configuration: social relationships and commitments [56, 84]; preferences in softgoals [71]; diagnosis of failures [68]; and so on. Our approach differs from the existing body of work by combining a control-theoretic view of the problem of adaptation with qualitative reasoning techniques on top of goal-oriented models. Moreover, Zanshin can be integrated with any of these existing reconfiguration algorithms, allowing the requirements engineer to choose from our own reconfiguration approach (Qualia) or any other integrated mechanism.

Design-time approaches: Although not explicitly designed for run-time system adaptation, approaches that propose design-time trade-off analysis [73, 76, 77] or risk management [74, 75] could be adapted to be used at run-time in order to decide the best system configuration. The former analyzes alternatives to choose the best one for a given problem, whereas the latter is concerned with modeling things that can go wrong with a software system; both of which are activities that adaptive systems have to perform at run-time.

10. DISCUSSION

In this section, we present a more subjective view on the lessons learned from the experience of applying Zanshin to the A-CAD (§ 10.1) and discuss threats to the validity of our experiments (§ 10.2), which also lays down the work ahead of us in the context of Zanshin.

10.1. Lessons learned

The experiment reported herein was conducted in the context of the first author's PhD thesis work. Unfortunately, proper measures of effort and statistics about the use of the approach (e.g., how many times new patterns were needed and how frequently were the different kinds of EvoReqs used) were not taken. At this time, the approach was used in an artificial (although based on a real) system by the author of the approach himself. Such measurements, taken in experiments with practitioners and real-world problems, are still in our to-do list. Nonetheless, we can discuss many lessons learned from using Zanshin to design and simulate adaptation in the A-CAD.

A prerequisite for software adaptation is that the system is aware of its own requirements at run-time in order to support monitoring activities. After all, the system will not be able to determine that its performance is deficient unless it has a model of requirements (even if implicit, represented in architectural models or even hard-coded in its implementation) against which to compare performance. Our experience suggests that the use of goal models (or an alternative requirements model) facilitates a more transparent and rigorous approach to monitoring in adaptive systems.

We also learned that monitoring requirements are determined by answering the question 'How much failure can be tolerated with respect to each system requirements?' Moreover, the design of adaptation mechanisms demands identification of a space of alternative controls that can change system behavior. Without a clear delineation of such a space, the design of the adaptive component of the system-to-be is incomplete and/or ad hoc. Therefore, the question 'In case failures go beyond what you tolerate, what would you like to happen?' should also be asked, associated with the requirements for adaptation.

This last question can then spawn a series of follow-up questions depending on what is to happen in response to a failure. If stakeholders would like the system to reconfigure itself, then models should contain enough parameters, relations with system indicators, and the reconfiguration algorithm to use, and that should be elicited. Instead, if they know exactly what to do, the components of the ECA algorithm that works with EvoReqs should be there, elicited as well. By making these elements first-class citizens in the model, Zanshin naturally guides the elicitation of feedback-loop requirements.

Another interesting lesson on adaptive software systems is that, unlike many of their control system cousins, they can't rely on quantitative techniques to drive adaptation because of the lack of

DOI: 10.1002/spe

quantitative models that relate system control parameters to often social and/or intentional indicators. For the A-CAD, it would have been impossible for us (and we believe for practitioners as well) to provide precise equations such as those commonly used in control systems.

An important aspect of moving from requirements models to run-time adaptation is the relationship of the system's models and code with its architecture. Our approach could greatly benefit if integrated with an architecture-based approach, separating adaptation concerns between those related to requirements (in the problem space, specified by stakeholders) and architecture (in the solution space, specified by developers). We have initiated work in this direction by comparing our approach with an architecture-based one using the same base system [40]. We have also recently submitted a proposal on generating architectural behavioral models (namely, UML state-charts) from requirements [85].

Finally, we confirmed our initial intuition that eliciting adaptation requirements for a complex systems such as the A-CAD and providing a framework that helps in operationalizing them at run-time are very complex and error-prone tasks. Currently, Zanshin provides limited tool support to analysts and developers, as will be discussed next, and there is a lot to be accomplished before it can be applied in industrial case studies.

In summary, the design of adaptive software systems calls for new concepts, tools, and techniques that go well beyond conventional software engineering. Zanshin represents a contribution toward this direction.

10.2. Threats to validity and future work

Our conclusions in this study are based on many assumptions, representing threats to validity. Among the many current limitations of our approach and framework, we highlight the following:

High implementation burden: Zanshin is generic and can be applied to many kinds of systems that operationalize adaptation through a feedback loop. The down side of this is that developers are responsible for implementing all the application-specific logic, including logging (for monitoring) and the effect of EvoReq operations (for adaptation). Approaches that are specific for some kinds of architectures can harness what the architecture has to offer (e.g., service-oriented approaches can use existing tools for service lookup, composition, and orchestration). We have recently started moving our approach toward architectural models: a first step of this agenda was presented in [40].

Lack of consistency and correctness guarantees: Our approach does not provide any process or technique to help analysts guarantee the consistency and correctness of the requirements models, leaving this responsibility at their hands. The use of ECA rules in EvoReqs constitutes a significant limitation, as large rule sets are hard to evolve, given that it becomes increasingly difficult to understand what a change entails. Moreover, attention needs to be paid to the case where conflicting rules fire at the same time.

More experiments needed: Further experiments are needed to demonstrate the applicability of Zanshin in real and practical scenarios. Surveys with practitioners can evaluate the approach and modeling language, demonstrating that developers other than the authors can be trained to successfully use Zanshin. The recently published experience papers using the *ZNN.com* exemplar [40] and the ATM machine software [39] are first steps in this agenda.

No assistance on code–goal model traceability: Although Zanshin concentrates on requirements, some assistance toward mapping goal model elements (tasks) to implemented components could help developers read goal-based specifications when providing logging information about the target system. Mapping elements from goal models to components in the system's implementation in software is still an open research problem. Currently, this mapping is also under the responsibility of developers;

Framework still a prototype: Although the necessary features to execute the described simulations are present, many functionalities are still to be implemented in the Zanshin framework: there is no integration with domain models (e.g., an AwReq that refers to a class of the application domain), only the most basic reconfiguration algorithms have been implemented, there is no support for legacy systems and third-party components, and so on.

DOI: 10.1002/spe

All of these limitations provide directions for future work in the context of this research, some of which have already been started in our research group.

11. CONCLUSIONS

In this experience report, we describe an experimental evaluation of the Zanshin approach for the design of adaptive systems. The evaluation consisted of modeling the adaptation requirements for an A-CAD system based on a well-known case study from the literature (the LAS-CAD system) and simulating run-time failures of this system to validate that our prototype framework responds accordingly.

The results from this experiment offer an initial evaluation for Zanshin, suggesting that it can systematize the design of adaptive systems. We have also shared the lessons learned from this experiment hoping that, along with all the documentation produced in the process, they provide value to other researchers and practitioners in the area of software engineering, especially so for those working with adaptive systems.

ACKNOWLEDGEMENTS

This work has been partially supported by the ERC advanced grant 267856 'Lucretius: Foundations for Software Evolution' (http://www.lucretius.eu) unfolding during the period of April 2011–March 2016. The authors would also like to acknowledge the contributions of Alexei Lapouchnian and William N. Robinson (Georgia State University, USA) to some of the examples contained herein.

REFERENCES

1. Baresi L, Müller HA (eds). In *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE: Piscataway, NJ, USA, 2012.
2. Brueckner S, Geihs K (eds). In *Proceedings of the 5th IEEE International Conference on Self-Adaptive and Self-Organizing Systems*. IEEE: Piscataway, NJ, USA, 2011.
3. Cheng BHC, de Lemos R, Giese H, Inverardi P, Magee J, Andersson J, Becker B, Bencomo N, Brun Y, Cukic B, Di Marzo Serugendo G, Dustdar S, Finkelstein A, Gacek C, Geihs K, Grassi V, Karsai G, Kienle H, Kramer J, Litoiu M, Malek S, Mirandola R, Müller HA, Park S, Shaw M, Tichy M, Tivoli M, Weyns D, Whittle J. Software engineering for self-adaptive systems: a research roadmap. In *Software Engineering for Self-adaptive Systems*, Vol. 5525, Cheng BHC, de Lemos R, Giese H, Inverardi P, Magee J (eds), Lecture Notes in Computer Science. Springer: Berlin, Germany, 2009; 1–26.
4. Schmeck H, Rosenstiel W, Abdelzaher T, Hellerstein JL (eds). In *Proceedings of the 8th ACM International Conference on Autonomic Computing*. ACM: New York, NY, USA, 2011.
5. Brun Y, Di Marzo Serugendo G, Gacek C, Giese H, Kienle H, Litoiu M, Müller HA, Pezzè M, Shaw M. Engineering self-adaptive systems through feedback loops. In *Software Engineering for Self-adaptive Systems*, Vol. 5525, Cheng BHC, de Lemos R, Giese H, Inverardi P, Magee J (eds), Lecture Notes in Computer Science. Springer: Berlin, Germany, 2009; 48–70.
6. Andersson J, de Lemos R, Malek S, Weyns D. Modeling dimensions of self-adaptive software systems. In *Software Engineering for Self-Adaptive Systems*, Vol. 5525, Cheng BHC, de Lemos R, Giese H, Inverardi P, Magee J (eds), Lecture Notes in Computer Science. Springer: Berlin, Germany, 2009; 27–47.
7. Souza VES, Lapouchnian A, Mylopoulos J. System identification for adaptive software systems: a requirements engineering perspective. In *Conceptual Modeling – ER 2011*, Vol. 6998, Jeusfeld M, Delcambre L, Ling T-W (eds), Lecture Notes in Computer Science. Springer: Berlin, Germany, 2011; 346–361.
8. Souza VES, Lapouchnian A, Mylopoulos J. Requirements-driven qualitative adaptation. In *On the Move to Meaningful Internet Systems: OTM 2012*, Vol. 7565, Meersman R, Panetto H, Dillon T, Rinderle-Ma S, Dadam P, Zhou X, Pearson S, Ferscha A, Bergamaschi S, Cruz IF (eds), Lecture Notes in Computer Science. Springer: Berlin, Germany, 2012; 342–361.
9. Souza VES, Lapouchnian A, Robinson WN, Mylopoulos J. Awareness requirements. In *Software Engineering for Self-Adaptive Systems II*, Vol. 7475, Lemos R, Giese H, Müller HA, Shaw M (eds), Lecture Notes in Computer Science. Springer: Berlin, Germany, 2013; 133–161.
10. Souza VES, Mylopoulos J. From awareness requirements to adaptive systems: a control-theoretic approach. In *Proceedings of the 2nd International Workshop on Requirements@Run.Time, Trento,* TN, Italy. IEEE: Piscataway, NJ, USA, 2011; 9–15.
11. Souza VES, Lapouchnian A, Angelopoulos K, Mylopoulos J. Requirements-driven software evolution. *Computer Science - Research and Development* 2013; **28**(4):311–329.

DOI: 10.1002/spe

12. Souza VES. Requirements-based software system adaptation. *PhD Thesis*, University of Trento, Italy, 2012.
13. Hevner AR, March ST, Park J, Ram S. Design science in information systems research. *MIS Quarterly* 2004; **28**(1):75–105.
14. Finkelstein A. Report of the inquiry into the London Ambulance Service (electronic version). *Technical Report*, South West Thames Regional Health Authority, 1993.
15. van Lamsweerde A. Goal-oriented requirements engineering: a guided tour. In *Proceedings of the 5th IEEE International Symposium on Requirements Engineering,* Toronto, ON, Canada. IEEE: Piscataway, NJ, USA, 2001; 249–262.
16. Hellerstein JL, Diao Y, Parekh S, Tilbury DM. *Feedback Control of Computing Systems*, 1st edn. Wiley: Hoboken, NJ, USA, 2004.
17. Forbus KD. Qualitative reasoning. In *Computer Science Handbook*, 2nd edn., chap. 62,. Chapman and Hall/ CRC: Boca Raton, FL, USA, 2004; 62-1–62-19.
18. Jureta I, Mylopoulos J, Faulkner S. Revisiting the core ontology and problem in requirements engineering. In *Proceedings of the 16th IEEE International Requirements Engineering Conference,* Barcelona, Spain. IEEE: Piscataway, NJ, USA, 2008; 71–80.
19. van Lamsweerde A, Darimont R, Massonet P. Goal-directed elaboration of requirements for a meeting scheduler: problems and lessons learnt. In *Proceedings of the 2nd IEEE International Symposium on Requirements Engineering,* York, UK. IEEE: Piscataway, NJ, USA, 1995; 194–203.
20. Beynon-Davies P. Information systems 'failure': the case of the London Ambulance Service's Computer Aided Despatch project. *European Journal of Information Systems* 1995; **4**(3):171–184.
21. Breitman KK, Leite JCSP, Finkelstein A. The world's a stage: a survey on requirements engineering using a real-life case study. *Journal of the Brazilian Computer Society* 1999; **6**(1).
22. Finkelstein A, Dowell J. A comedy of errors: the London ambulance service case study. In *Proceedings of the 8th International Workshop on Software Specification and Design,* Schloss Velen, Germany. IEEE: Piscataway, NJ, USA, 1996; 2–4.
23. Kramer J, Wolf AL. Succeedings of the 8th international workshop on software specification and design. *ACM SIGSOFT Software Engineering Notes* 1996; **21**(5):21–35.
24. Letier E. Reasoning about agents in goal-oriented requirements engineering. *PhD Thesis*, Université Catholique de Louvain, Belgium, 2001.
25. You Z. Using meta-model-driven views to address scalability in i* models. *Masters Thesis*, University of Toronto, Canada, 2004.
26. You Z. Experiences with applying the i* framework to a real-life system. *Technical Report*, Requirements Engineering (CSC2106) Course Project, University of Toronto, Canada, 2001.
27. Whittle J, Sawyer P, Bencomo N, Cheng BHC, Bruel JM. RELAX: incorporating uncertainty into the specification of self-adaptive systems. In *Proceedings of the 17th IEEE International Requirements Engineering Conference,* Atlanta, GA, USA. IEEE: Piscataway, NJ, USA, 2009; 79–88.
28. Yu ESK, Giorgini P, Maiden N, Mylopoulos J. *Social Modeling for Requirements Engineering*, 1st edn. MIT Press: Cambridge, MA, USA, 2011.
29. Dardenne A, van Lamsweerde A, Fickas S. Goal-directed requirements acquisition. *Science of Computer Programming* 1993; **20**(1-2):3–50.
30. Dastani M, van Riemsdijk MB, Meyer JJC. Goal types in agent programming. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems,* Hakodate, Japan. ACM: New York, NY, USA, 2006; 1285–1287.
31. Morandini M, Penserini L, Perini A. Operational semantics of goal models in adaptive agents. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems,* Budapest, Hungary. ACM: New York, NY, USA, 2009; 129–136.
32. van Lamsweerde A. Reasoning about alternative requirements options. In *Conceptual modeling: foundations and applications*, Borgida AT, Chaudhri VK, Giorgini P, Yu ESK (eds), chap. 20. Springer: Berlin, Germany, 2009; 380–397.
33. Ali R, Dalpiaz F, Giorgini P. A goal-based framework for contextual requirements modeling and analysis. *Requirements Engineering* 2010; **15**(4):439–458.
34. Gonzalez-Baixauli B, Sampaio do Prado Leite J, Mylopoulos J. Visual variability analysis for goal models. In *Proceedings of the 12th IEEE International Requirements Engineering Conference,* Kyoto, Japan. IEEE: Piscataway, NJ, USA, 2004; 183–192.
35. Lapouchnian A, Mylopoulos J. Modeling domain variability in requirements engineering with contexts. In *Conceptual Modeling - ER 2009*, Vol. 5829, Laender A, Castano S, Dayal U, Casati F, de Oliveira J (eds), Lecture Notes in Computer Science. Springer: Berlin, Germany, 2009; 115–130.
36. Liaskos S, Lapouchnian A, Yu Y, Yu ESK, Mylopoulos J. On goal-based variability acquisition and analysis. In *Proceedings of the 14th IEEE International Requirements Engineering Conference,* Minneapolis/St. Paul, MN, USA. IEEE: Piscataway, NJ, USA, 2006; 79–88.
37. Semmak F, Gnaho C, Laleau R. Extended Kaos to support variability for goal oriented requirements Reuse. In *Proceedings of the International Workshop on Model Driven Information Systems Engineering: Enterprise, User and System Models,* Montpelier, France, Ebersold S, Front A, Lopistéguy P, Nurcan S (eds). CEUR: Aachen, Germany, 2008; 22–33.

DOI: 10.1002/spe

38. Robinson WN. Extended OCL for goal monitoring. *Electronic Communications of the EASST* 2008; **9**:1–12.
39. Tallabaci G, Souza VES. Engineering adaptation with Zanshin: an experience report. In *Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems,* San Francisco, CA, USA. IEEE: Piscataway, NJ, USA, 2013; 93–102.
40. Angelopoulos K, Souza VES, Pimentel Ja. Requirements and architectural approaches to adaptive software systems: a comparative study. In *Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems,* San Francisco, CA, USA. IEEE: Piscataway, NJ, USA, 2013; 23–32.
41. Oreizy P, Gorlick MM, Taylor RN, Heimhigner D, Johnson G, Medvidovic N, Quilici A, Rosenblum DS, Wolf AL. An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems* 1999; **14**(3):54–62.
42. Laddaga R, Robertson P. Self adaptive software: a position paper. In *Proceedings of the 2004 International Workshop on Self-* Properties in Complex Information Systems*, Bertinoro, FC, Italy, 2004; 1–4.
43. Kramer J, Magee J. Self-managed systems: an architectural challenge. In *Future of Software Engineering (FOSE '07)*. IEEE: Piscataway, NJ, USA, 2007; 259–268.
44. Sykes D, Heaven W, Magee J, Kramer J. Exploiting non-functional preferences in architectural adaptation for self-managed systems. In *Proceedings of the 2010 ACM Symposium on Applied Computing,* Sierre, Switzerland. ACM: New York, NY, USA, 2010; 431–438.
45. Cheng SW, Garlan D, Schmerl B. Evaluating the effectiveness of the rainbow self-adaptive system. In *Proceedings of the ICSE 2009 Workshop on Software Engineering for Adaptive and Self-managing Systems,* Vancouver, BC, Canada. IEEE: Piscataway, NJ, USA, 2009; 132–141.
46. Garlan D, Cheng SW, Huang AC, Schmerl B, Steenkiste P. Rainbow: architecture-based self-adaptation with reusable infrastructure. *Computer* 2004; **37**(10):46–54.
47. Sousa JaP, Balan RK, Poladian V, Garlan D, Satyanarayanan M. A software infrastructure for user–guided quality–of–service tradeoffs. In *Software and Data Technologies*, Vol. 47, Cordeiro J, Shishkov B, Ranchordas A, Helfert M (eds), Communications in Computer and Information Science. Springer: Berlin, Germany, 2009; 48–61.
48. Menasce DA, Gomaa H, Malek S, Sousa JaP. SASSY: a framework for self-architecting service-oriented systems. *IEEE Software* 2011; **28**(6):78–85.
49. Goldsby HJ, Sawyer P, Bencomo N, Cheng BHC, Hughes D. Goal-based modeling of dynamically adaptive system requirements. In *Proceedings of the 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems,* Belfast, Northern Ireland, UK. IEEE: Piscataway, NJ, USA, 2008; 36–45.
50. Zhang J, Cheng BHC. Model-based development of dynamically adaptive software. In *Proceedings of the 28th International Conference on Software Engineering,* Shanghai, China. ACM: New York, NY, USA, 2006; 371–380.
51. Qureshi NA, Perini A. Engineering adaptive requirements. In *Proceedings of the 2009 ICSE Workshop on Software Engineering for Adaptive and Self-managing Systems,* Vancouver, BC, Canada. IEEE: Piscataway, NJ, USA, 2009; 126–131.
52. Ma W, Liu L, Xie H, Zhang H, Yin J. Preference model driven services selection. In *Advanced Information Systems Engineering*, Vol. 5565, van Eck P, Gordijn J, Wieringa R (eds), Lecture Notes in Computer Science. Springer: Berlin, Germany, 2009; 216–230.
53. Whittle J, Sawyer P, Bencomo N, Cheng BHC, Bruel JM. RELAX: a language to address uncertainty in self-adaptive systems requirement. *Requirements Engineering* 2010; **15**(2):177–196.
54. Cheng BHC, Sawyer P, Bencomo N, Whittle J. A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty. In *Model Driven Engineering Languages and Systems*, Vol. 5795, Schürr A, Selic B (eds), Lecture Notes in Computer Science. Springer: Berlin, Germany, 2009; 468–483.
55. Baresi L, Pasquale L, Spoletini P. Fuzzy goals for requirements-driven adaptation. In *Proceedings of the 18th IEEE International Requirements Engineering Conference,* Sydney, Australia. IEEE: Piscataway, NJ, USA, 2010; 125–134.
56. Dalpiaz F, Giorgini P, Mylopoulos J. Adaptive socio-technical systems: a requirements-based approach. *Requirements Engineering* 2012;1–24.
57. Schmitz D, Nissen HW, Jarke M, Rose T, Drews P, Hesseler FJ, Reke M. Requirements engineering for control systems development in small and medium-sized enterprises. In *Proceedings of the 16th IEEE International Requirements Engineering Conference,* Barcelona, Spain. IEEE: Piscataway, NJ, USA, 2008; 229–234.
58. Hebig R, Giese H, Becker B. Making control loops explicit when architecting self-adaptive systems. In *Proceedings of the 2nd International Workshop on Self-organizing Architectures,* Washington, DC, USA. ACM: New York, NY, USA, 2010; 21–28.
59. Filieri A, Ghezzi C, Leva A, Maggio M. Reliability-driven dynamic binding via feedback control. In *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems,* Zürich, Switzerland. IEEE: Piscataway, NJ, USA, 2012; 43–52.
60. Fickas S, Feather MS. Requirements monitoring in dynamic environments. In *Proceedings of the 2nd IEEE International Symposium on Requirements Engineering,* York, UK. IEEE: Piscataway, NJ, USA, 1995; 140–147.
61. Robinson WN. A requirements monitoring framework for enterprise systems. *Requirements Engineering* 2006; **11**(1):17–41.
62. Robinson WN. A roadmap for comprehensive requirements monitoring. *Computer* 2010; **43**(5):64–72.
63. Bencomo N, Whittle J, Sawyer P, Finkelstein A, Letier E. Requirements reflection: requirements as runtime entities. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE '10),* Cape Town, South Africa, Vol. 2. ACM: New York, NY, USA, 2010; 199–202.

DOI: 10.1002/spe

64. Sawyer P, Bencomo N, Whittle J, Letier E, Finkelstein A. Requirements-aware systems: a research agenda for RE for self-adaptive systems. In *Proceedings of the 18th IEEE International Requirements Engineering Conference,* Sydney, Australia. IEEE: Piscataway, NJ, USA, 2010; 95–103.
65. Hawthorne MJ, Perry DE. Exploiting architectural prescriptions for self-managing, self-adaptive systems: a position paper. In *Proceedings of the 1st ACM Sigsoft Workshop on Self-managed Systems,* Newport Beach, CA, USA. ACM: New York, NY, USA, 2004; 75–79.
66. Brake N, Cordy JR, Dancy E, Litoiu M, Popescu V. Automating discovery of software tuning parameters. In *Proceedings of the 2008 International Workshop on Software Engineering for Adaptive and Self-Managing Systems*. ACM: New York, NY, USA, 2008; 65–72.
67. Khan MJ, Awais MM, Shamail S. Enabling self-configuration in autonomic systems using case-based reasoning with improved efficiency. In *Proceedings of the 4th International Conference on Autonomic and Autonomous Systems,* Gosier, Guadeloupe, France. IEEE: Piscataway, NJ, USA, 2008; 112–117.
68. Wang Y, Mylopoulos J. Self-repair through reconfiguration: a requirements engineering approach. In *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering,* Auckland, New Zealand. IEEE: Piscataway, NJ, USA, 2009; 257–268.
69. Peng X, Chen B, Yu Y, Zhao W. Self-tuning of software systems through goal-based feedback loop control. In *Proceedings of the 18th IEEE International Requirements Engineering Conference,* Sydney, Australia. IEEE: Piscataway, NJ, USA, 2010; 104–107.
70. Nakagawa H, Ohsuga A, Honiden S. GOCC: a configuration compiler for self-adaptive systems using goal-oriented requirements description. In *Proceedings of the 6th International Symposium on Software Engineering for Adaptive an Self-Managing Systems,* Honolulu, HI, USA. ACM: New York, NY, USA, 2011; 40–49.
71. Fu L, Peng X, Yu Y, Mylopoulos J, Zhao W. Stateful requirements monitoring for self-repairing socio-technical systems. In *Proceedings of the 20th IEEE International Requirements Engineering Conference,* Chicago, IL, USA. IEEE: Piscataway, NJ, USA, 2012; 121–130.
72. Salehie M, Tahvildari L. Towards a goal-driven approach to action selection in self-adaptive software. *Software: Practice and Experience* 2012; **42**(2):211–233.
73. Letier E, van Lamsweerde A. Reasoning about partial goal satisfaction for requirements and design engineering. In *Proceedings of the 12th ACM SIGSOFT International Symposium on Foundations of Software Engineering,* Newport Beach, CA, USA, Vol. 29. ACM: New York, NY, USA, 2004; 53–62.
74. Cornford SL, Feather MS, Hicks KA. DDP: a tool for life-cycle risk management. *IEEE Aerospace and Electronic Systems Magazine* 2006; **21**(6):13–22.
75. Menzies T, Richardson J. Qualitative modeling for requirements engineering. In *Proceedings of the 30th Annual IEEE/NASA Software Engineering Workshop,* Columbia, MD, USA. IEEE: Piscataway, NJ, USA, 2006; 11–20.
76. Heaven W, Letier E. Simulating and optimising design decisions in quantitative goal models. In *Proceedings of the 19th IEEE International Requirements Engineering Conference,* Trento, TN, Italy. IEEE: Piscataway, NJ, USA, 2011; 79–88.
77. Elahi G, Yu ESK. Requirements trade-offs analysis in the absence of quantitative measures: a heuristic method. In *Proceedings of the 2011 ACM Symposium on Applied Computing,* Taiching, Taiwan. ACM: New York, NY, USA, 2011; 651–658.
78. Cheng BHC, de Lemos R, Giese H, Inverardi P, Magee J (eds). *Software Engineering for Self-Adaptive Systems*, Lecture Notes in Computer Science, Vol. 5525. Springer: Berlin, Germany, 2009.
79. de Lemos R, Giese H, Müller HA, Shaw M (eds). *Software Engineering for Self-Adaptive Systems II*, Lecture Notes in Computer Science, Vol. 7475. Springer: Berlin, Germany, 2013.
80. Di Nitto E, Ghezzi C, Metzger A, Papazoglou M, Pohl K. A journey to highly dynamic, self-adaptive service-based applications. *Automated Software Engineering* 2008; **15**(3):313–341.
81. Huebscher MC, McCann JA. A survey of autonomic computing–degrees, models, and applications. *ACM Computing Surveys* 2008; **40**(3):1–28.
82. Salehie M, Tahvildari L. Self-adaptive software: landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems* 2009; **4**(2):1–42.
83. EEAT, Available at: http://eeat.cis.gsu.edu:8080/xwiki/bin/view/EEAT/WebHome [last accessed 25 October 2013] 2013.
84. Ali R, Solis C, Omoronyia I, Salehie M, Nuseibeh B. Social adaptation - when software gives users a voice. In *Proceedings of the 7th International Conference on Evaluation of Novel Approaches to Software Engineering,* Wroclaw, Poland, Filipe J, Maciaszek LA (eds). SciTePress: Setubal, Portugal, 2012; 75–84.
85. Pimentel J, Castro JFB, Mylopoulos J, Angelopoulos K, Souza VES. From requirements to statecharts via design refinement. In *Proceedings of the 2014 ACM Symposium on Applied Computing*. ACM, 2014. to appear.

DOI: 10.1002/spe