Towards Requirements-Driven Autonomic Systems Design

Alexei Lapouchnian Sotirios Liaskos John Mylopoulos Yijun Yu
Department of Computer Science
University of Toronto
10 King's College Road, Toronto, ON, Canada, M5S 3G4
{alexei, liaskos, jm, yijun}@cs.toronto.edu

ABSTRACT

Autonomic computing systems reduce software maintenance costs and management complexity by taking on the responsibility for their configuration, optimization, healing, and protection. These tasks are accomplished by switching at runtime to a different system behaviour – the one that is more efficient, more secure, more stable, etc. – while still fulfilling the main purpose of the system. Thus, identifying and analyzing alternative ways of how the main objectives of the system can be achieved and designing a system that supports all of these alternative behaviours is a promising way to develop autonomic systems. This paper proposes the use of requirements goal models as a foundation for such software development process and sketches a possible architecture for autonomic systems that can be built using the this approach.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications – *methodologies*; D.2.2 [**Software Engineering**]: Design Tools and Techniques – *modules and interfaces, state diagrams*; D.2.10 [**Software Engineering**]: Design – *methodologies*; D.2.11 [**Software Engineering**]: Software Architectures – *patterns*; D.2.13 [**Software Engineering**]: Management – *software configuration management*; K.6.3 [**Management of Computing and Information Systems**]: Software Management – *software maintenance*.

General Terms

Management, Design.

Keywords

Goal-oriented requirements engineering, autonomic computing software customization, software variability, self-management.

1. INTRODUCTION

As management complexity and maintenance cost of software systems keep spiraling upward, Autonomic Computing (AC) [8] promises to move most of this complexity from humans to the software itself and to reduce software maintenance costs, thereby drastically reducing the dominant cost factor in the software lifecycle. This reduction is expected to come about because autonomic software can self-configure at runtime to match changing operating environments; it can self-optimize to tune its performance or other software qualities; it can self-heal instead of crashing when its operating environment turns out to be inconsistent with its built-in design assumptions; and it can self-protect itself from malicious attacks.

There are three basic ways to make a system autonomic. The first is to design it so that it supports a space of possible behaviours. These are realized through an isomorphic space of possible system configurations. To make such designs possible, we need concepts for characterizing large spaces of alternative behaviours/configurations. Goal models in requirements engineering and feature models in software product line design offer such concepts [1][5]. For example, the possible behaviours of an autonomic meeting scheduling system might be characterized by a goal model that indicates all possible ways of achieving the goal "Schedule Meeting."

The second way of building an autonomic system is to endow it with planning capabilities and social skills so that it can delegate tasks to external software components (agents) thereby augmenting its own capabilities [12]. Evolutionary approaches to autonomic systems [11], such as those found in biology, constitute a third way of building autonomic software. We only explore the first way in this paper.

Specifically, the purpose of this position paper is to argue that requirements goal models can be used as a foundation for designing software that supports a space of behaviours, all delivering the same function, and that is able to select at runtime the best behaviour based on the current context. We also sketch a possible autonomic systems architecture that can be derived from these goal models. We then outline how feedback mechanisms of different kinds can be used as a basis for self-configuring, self-tuning and self-repairing behaviour and how properly enriched goal models can serve as sources of knowledge for these activities. Self-protection mechanisms will de addressed elsewhere.

The rest of the paper is structured as follows. We introduce goal-oriented requirements engineering – the foundation of our approach – in Section 2, outline how design-level views can be created from goal models in Section 3, sketch a way goal models can be used for designing autonomic systems in Section 4. Section 5 concludes the paper.

2. GOAL-ORIENTED REQUIREMENTS ENGINEERING

A major breakthrough of the past decade in (Software) Requirements Engineering is the development of a framework for capturing and analyzing stakeholder intentions to generate functional and non-functional (hereafter quality) requirements [1][9][14]. In essence, this work has extended upstream the software development process by adding a new phase (*early* requirements analysis) that is also supported by engineering concepts, tools and techniques, like its downstream cousins. The fundamental concepts used to drive the new form of analysis are those of *goal* and *actor*. For example, a stakeholder goal for a library information system may be "Fulfill Every Book Request". This goal may be decomposed in different ways. One might consist of ensuring book availability by limiting the borrowing period and also notifying users who requested a book that the book is available. This decomposition may lead (through intermediate steps) to functional requirements such as "Remind Borrower" and "Notify User". A different decomposition of the initial goal, however, may involve buying a book whenever a request can't be fulfilled[1]. The point is: there are in general many ways to fulfill a stakeholder goal. Analyzing the space of alternatives makes the process of generating functional and quality requirements more systematic in the sense that the designer is exploring an *explicitly represented* space of alternatives. It also makes it more rational in that the designer can point to an explicit evaluation of these alternatives in terms of stakeholder criteria to justify his choice. An authoritative account of Goal-Oriented Requirements Engineering can be found in [13].

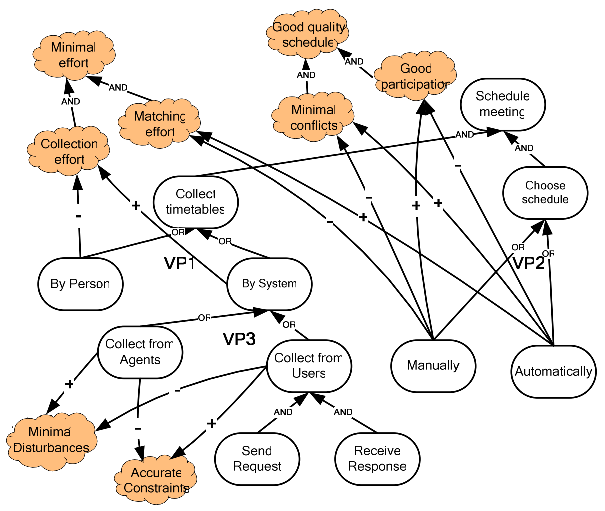

Figure 1. A goal model showing interdependencies among goals and qualities.

At the very heart of this new phase of Software Engineering are goal models that represent stakeholder intentions and their refinements using formally defined relationships. Functional stakeholder goals are modeled in terms of hard goals (or simply goals, when there is no ambiguity). For example, "Schedule Meeting" and "Fulfill Every Book Request" are functional goals that are either fulfilled (satisfied) or not fulfilled (denied). Other stakeholder goals are qualitative and are hard to define formally. For instance, "Have Productive Meeting" and "Have Satisfied Library Users" are qualitative goals and they are modeled in terms of *softgoals*. A softgoal by its very nature doesn't have a clear-cut criterion for its fulfillment, and may be fully or partially satisfied or denied.

[1] Admittedly not a very practical one!

Goals and/or softgoals may be related through AND/OR relationships that have the obvious semantics. In addition, goals/softgoals can be related to softgoals through help (+), hurt (–), make (++), or break (--) relationships. This simple language is sufficient for modeling and analyzing goals during early requirements, covering both functional and quality requirements. Note that in this framework, quality requirements are treated as first-class citizens.

To illustrate what goal models are, and what they can do for the design of autonomic software, let's suppose that the task is to design a system that supports the scheduling of meetings (Figure 1). Clearly, several stakeholders here (managers, engineers, admin staff, etc.) share the goal "Schedule Meeting", which can be AND-decomposed into "Collect Timetables" and "Choose Schedules". Each of the subgoals has two alternative solutions: it can either be done "By Person" ("Manually") or "By System" ("Automatically"). A system can collect a timetable "From Agents" for each potential meeting participant (e.g., from his secretary) or directly from participants ("From Users"); the latter goal is further AND-decomposed into "Send Request" and "Receive Response" (regarding timetables).

Quality attributes are represented as softgoals (cloudy shapes in the figure). For our example, four top-level desired qualities are "Minimal (scheduling) Effort", "Good Quality Schedule", "Minimal Disturbance" and "Accurate (timetable) Constraints". These can be decomposed into sub-softgoals. For example, "Minimal Effort" can be fulfilled by minimizing "Collection Effort" and "(human) Matching Effort". Similarly, "Good Quality Schedule" is fulfilled by having "Minimal Conflicts" and "Good Participation". Clearly, collecting timetables manually is a tedious task. Thus, it hurts the softgoal "(minimize) Collection Effort". As shown in Figure 1, such partial contributions are explicitly expressed in the goal model. In order to not clutter the figure, we don't show all partial contributions. For instance, when timetables are collected by a person, they tend to be more accurate. Thus, there should be a positive contribution from the "By Person" goal to the "Minimal Conflicts" softgoal.

In all, the goal model of Figure 1 shows six alternative ways for fulfilling the goal "Schedule Meeting". It is easy to verify that generally the number of alternatives represented by a goal model depends exponentially on the number of OR decompositions (labeled as variation points "VP1" through "VP3" in Figure 1) present in the goal model (assuming a "normalized" goal model where AND and OR decompositions are interleaved). As such, goal models make it possible to capture during requirements analysis – in stakeholder-oriented terms – all the different ways of fulfilling top-level goals. Now, if one were designing an autonomic software system, it would make sense to ensure that the system is designed to accommodate all ways of fulfilling top-level goals (i.e., delivering the desired functionality), rather than just some.

Another feature of goal models is that alternatives can be ranked with respect to the qualities modeled in the figure. Assigning to the system the responsibility for collecting timetables and generating a schedule is in general less time-consuming (for people), but results more often in sub-optimal schedules, since the system doesn't take into account personal/political/social considerations. So, the model of Figure 1 represents a space of alternative behaviours that can lead to the fulfillment of top-level goals, and also captures how these alternatives stack up with respect to desired stakeholder qualities.

A more sophisticated Goals-Skills-Preferences approach for ranking alternatives is presented in [4]. It was proposed in the context of "personal software" (e.g., an email system) that needs to be fine-tuned for each particular user. The approach takes into consideration the user *preferences* (desired quality attributes) as well as the user's physical and mental *skills* to find the best alternative for achieving the user's goals. We envision a generic version of this approach where *capabilities* will be used to prune the space of alternatives for achieving goals, while *preferences* will be used to rank them.

3. FROM GOAL MODELS TO HIGH-VARIABILITY SOFTWARE DESIGN

We use goal models to represent variability in the way high-level stakeholder objectives can be met by the system-to-be together with its environment. Thus, goal models capture variability in the problem domain. However, properly augmented goal models can be used to create models that represent variability in the solution domain. We use textual *annotations* to add the necessary details to goal models. For example, the sequence annotation ("**;**") can be added to the appropriate AND goal decomposition to indicate that the subgoals are to be achieved in sequence from left to right. Sequence annotations are useful to model data dependencies or precedence constraints among subgoals. For instance, it is easy to see that the goal "Collect Timetables" must be achieved before achieving the goal "Choose Schedule" (see Figure 1). The absence of any dependency among subgoals in an AND decomposition can be indicated by a concurrency ("||") annotation. It is important to note that the above-mentioned annotations capture properties of the problem domain. However, annotations that apply to OR decompositions are usually more solution-oriented and indicate how (e.g., in parallel to save time or in sequence to conserve resources) the alternatives are to be executed.

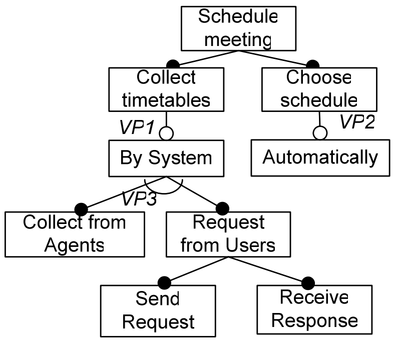

Figure 2. A feature model generated from the goal model of Figure 1.

In [15], we described how one can generate three design views from goal models, specifically *feature models*, *statecharts*, and *component-connector models*. These views can serve as a starting point in developing a design for a software system that can deliver the desired functionality in multiple ways. To generate each of these views, we enrich goal models with the information that is required by the desired view, but cannot be represented by the basic goal decompositions.

For constructing a feature model (which represents the configuration variability of a software product line family) from a goal model we need to consider the subset of goals whose attainment will be the responsibility of the system-to-be. Each of these goals will turn into a distinct feature of the system. Then, goal model patterns corresponding to various feature types are identified and resultant features are created. The variation points of Figure 1 are preserved in the feature model through optional (VP1 and VP2) and alternative (VP3) features.

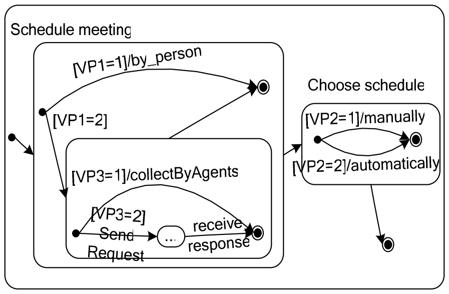

Figure 3. A fragment of the statechart generated from the goal model in Figure 1.

To generate a statechart, which models the behavioural variability of the system-to-be, for each goal the software system is responsible for a state that represents the system achieving that goal is introduced. We use super-/substates for organizing the states into a hierarchy that is isomorphic to the goal hierarchy from the source goal model. The generation of statecharts is based on a set of patterns that take into account goal decompositions and the temporal annotations that were used to enrich the original goal models. Here, the behaviour of the system depends on the selected process alternative (note that the conditions on state transitions refer to variation points of the goal model).

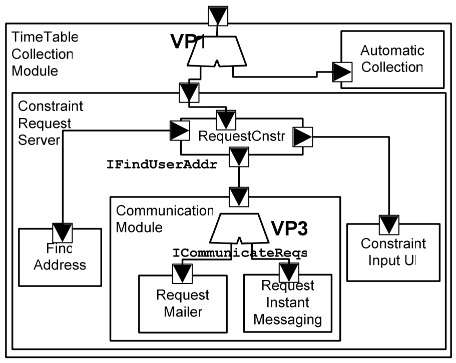

Figure 4. A fragment of the component-connector model.

Finally, the generation of a component-connector model, which is used to model structural variability of a software system, is based on the intuition that the achievement of each goal that has been delegated to the system should be a responsibility of at least one of its components. Then, again through the use of patterns, the structure of the goals in the goal model as well as their input and output information (the necessary enrichment) is used for defining the relationships among components. Further, through the use of special types of components called switches, alternative subgoals are mapped into alternative bindings between components.

Thus, having the initial goal model representing the requirements for the system-to-be and the appropriate process-level enrichments, it is possible to generate initial design views that preserve the variability in the way the system-to-be can meet its objectives.

Overall, this approach is systematic and requirements-driven. It allows for the gradual increase of the level of detail of the goal models through the use of annotations. This process turns requirements goal models into solution domain models that can be either utilized as high-level design specifications or used to generate other design-level models of the system. In this approach, requirements traceability is supported through the tight mapping between notations.

4. TOWARDS AUTONOMIC COMPUTING SYSTEMS

4.1 From Goal Models to AC Systems

The building blocks of autonomic computing are architectural components called Autonomic Elements (AEs). An autonomic element is responsible for providing resources, delivering services, etc. Its behaviour and its relationships with other AEs are "driven by goals that its designer embedded in it" [6]. An AE typically consists of an autonomic manager and a set of managed elements, such as resources, components, etc. The manager must be able to monitor and control the managed elements.

An autonomic element manages itself to deliver its service in the best possible way. In order to achieve this, its autonomic manager must be armed with tools for monitoring its managed elements and the environment, for analyzing the collected data to determine whether the AE is performing as expected, for planning a new course of action if a problem is detected, and for executing these plans by, for example, tuning the parameters of its managed elements. Most importantly, these activities require the knowledge about the goal of the autonomic element, the configurations and capabilities of its managed elements, the environment of the AE, etc.

Kephart and Chess suggest that overall system self-management results from the internal self-management of its individual autonomic elements [6]. Moreover, in their view, autonomic elements are full-fledged intelligent agents that, when assigned individual goals, will use complex social interactions to communicate, negotiate, form alliances, etc. and ultimately deliver the objective of an autonomic system. However, deriving a set of goals and policies that, if embedded into individual autonomic elements, will guarantee certain global system properties is nontrivial.

We believe that goal models can be useful in the design of autonomic computing systems in several ways. First, goal models provide a means to represent many ways in which the objectives of the system can be met and analyze/rank these alternatives with respect to stakeholder quality concerns. This allows for exploration and analysis of alternative system behaviours at design time, which can lead to more predictable and trusted autonomic systems. It also means that if the alternatives that are initially delivered with the system perform well, there is no need for complex social interactions among autonomic elements. Of course, not all alternatives can be identified at design time. In an open and dynamic environment, new and better alternatives may present themselves and some of the identified and implemented alternatives may become impractical. Thus, in certain situations, new alternatives will have to be discovered and implemented by the system at runtime. However, the process of discovery, analysis, and implementation of new alternatives at runtime is complex and error-prone. By exploring the space of alternative process specifications at design time, we are minimizing the need for this difficult task.

Second, goal models can provide the traceability mechanism from AC system designs to stakeholder requirements. When a change in stakeholder requirements is detected at runtime (e.g., by using the approach in [2]), goal models can be used to reevaluate the system behaviour alternatives with respect to the new requirements and to determine if system reconfiguration is needed. For instance, if a change in stakeholder requirements affected a particular goal in the model, it is easy to see how this goal is decomposed and which components/autonomic elements implementing the goal are in turn affected. By analyzing the goal model, it is also easy to identify how a failure to achieve some particular goal affects the overall goal of the system.

Third, goal models provide a unifying intentional view of the system by relating goals assigned to individual autonomic elements to high-level system objectives and quality concerns. These high-level objectives or quality concerns serve as the common knowledge shared among the autonomic computing elements to achieve the global system optimization. This way, the system can avoid the pitfalls of missing the global optimal configuration due to only relying on local optimizations.

4.2 A Hierarchical Autonomic Architecture

Enriching goal models to generate design-level views is one of the possible ways to create autonomic systems. Here, the goal model enrichments will include, among other things, the data to be collected to allow for determining whether and to what degree the goals are attained, etc. Additionally, goal models may include goals that are to be achieved by the environment of the system-to-be (e.g., its users, legacy systems, etc.). In requirements engineering, these goals can be viewed as the system's expectations of its environment. To support self-management, the system must monitor for the achievement of these goals in order to detect if the expectations are still valid and to be able to change its behaviour if they are not.

We now outline a possible architecture for autonomic software systems that can be derived from high-variability requirements goal models. We envision a hierarchy of autonomic elements that is structurally similar to the goal hierarchy of the corresponding goal model. In the most straightforward case, each goal in the goal model is associated with an autonomic element whose purpose is the achievement of that goal. The managed elements of the leaf-level autonomic elements (which correspond to leaf-level goals) are the actual components, resources, etc. Leaf-level AEs can tune and optimize these resources to deliver their objective in the best way. On the other hand, higher-level autonomic elements are not directly associated with physical components, but are used to orchestrate the lower-level elements. The root autonomic element represents the whole software system. Thus, an AE corresponds to any subtree of the goal model. This approach has an advantage that the global high-variability design space is partitioned into autonomic elements with lower variability, thereby facilitating management and administration tasks.

A fragment of a properly enriched goal model will serve as the core of each AE's knowledge. For example, Figure 5 presents an AE, whose objective is to achieve the goal G. It has a fragment of the goal model showing the decomposition of this goal. Here, the goal G is AND-decomposed into G1 and G2, which means that the goal model identified only one way to achieve G. The managed elements of the AE in Figure 5 are themselves autonomic elements that achieve the goals G1 and G2. They have different fragments of the goal model assigned to them. For example, the AE achieving the goal G2 knows that to attain that goal it must either achieve G3 or G4. These goals are in turn handled by lower-level AEs (not shown).

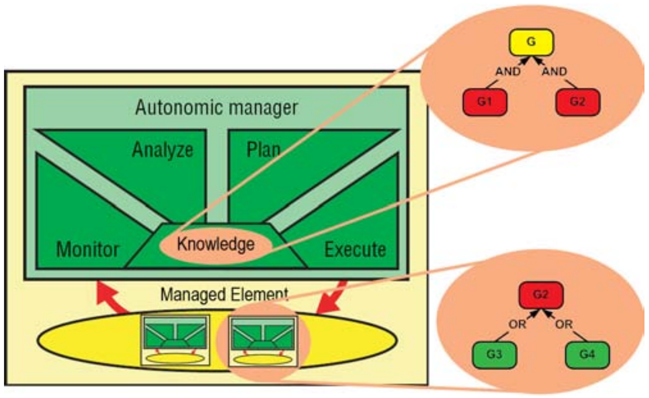

Figure 5. A hierarchical composition of AEs.

Because of the hierarchy of AEs, it is possible to propagate high-level concerns from the root AE down to the leaf-level elements, thus making sure that the system achieves its objectives and addresses the quality concerns of its stakeholders. In particular, the root autonomic element will receive high-level policies from stakeholders, act on these policies by identifying which alternative for achieving its goal fits the policy best (if such alternatives are identified in the corresponding goal model), produce policies for the AEs that it directly manages, and pass those policies down to these autonomic elements, which will in turn process them and formulate new policies for their children AEs. This process terminates when leaf-level AEs are reached. Note that each autonomic element retains the freedom to achieve its goal in the best way it can provided that it satisfies the policy set by its parent AE. This process can be viewed as an example of self-reconfiguration based on the changes in user requirements.

Thus, goal models can be used to configure complex systems given high-level quality objectives. It was shown in [7] that it is also possible to create goal models based on configuration options (such as Preferences, Options, etc. dialogs) provided by existing software systems. These goal models then can represent the quality concerns behind the software configuration choices for a system. Therefore, they can be used by the autonomic elements wrapping non-autonomic components/systems to make sure that these components' behaviours conform to the quality preferences of system stakeholders.

In order to see whether and how well an AE achieves its goal, it needs to monitor its managed element, collect the data, analyze it based on its knowledge, plan changes to its behaviour if necessary, and execute the plan. The planned change may include switching from one alternative behaviour to another, generating a new policy for its managed elements, etc. At the same time, each element sends the data about its current state to its parent autonomic element. So, if an AE is unable to deliver its service, the parent will immediately know that and will hopefully be able to switch to another behaviour not involving the failed AE, thus stopping the failure propagation and self-healing the system. If no alternatives exist in the system, a new AE can be found to replace the failed element. Because of the hierarchical structure of the AEs, this replacement can be performed quite easily.

A self-optimizing behaviour may involve an AE noticing that one of its two managed AEs that both deliver the same service is responding slower and slower and deciding to fine-tune its controlled parameters or proactively switching to the other AE if analysis shows that the improvement will compensate for the possible switching cost. Goal analysis supplemented with quantitative reasoning will allow for self-optimizing behaviour.

Self-managing systems developed using the proposed approach will frequently need to determine how the current alternative (or to predict how some potentially useful alternative) satisfies its functional and non-functional requirements. Similarly, given some changes in user preferences, the system will need to find the best alternative that supports these changes. Top-down [10] and bottom-up [3] goal reasoning approaches can be employed by an autonomic system to support the above activities. The former approach helps in finding the alternatives that satisfy the desired valuations of root goals, while the latter one can be used to determine whether and to what extent a given alternative meets the requirements of the system.

4.3 Goal Model-Based Autonomic Behaviour

This section sketches how feedback mechanisms can be used for self-management of the meeting scheduling system modeled in Figure 1. Suppose that we have a system that actually does accommodate the six alternatives captured in that model. Moreover, suppose that the system is operating according to the full automation alternative, i.e., the system is responsible for collecting timetables and generating a schedule.

4.3.1 Self-Configuration and Reconfiguration

One form of feedback that can be provided to the system is the information on whether each meeting was actually held as scheduled. The system could be keeping track of postponed/cancelled meetings, along with their initiator and the participants. If the percentage of such meetings is above some threshold, the system could try to find the cause for this problem. The cause could be inaccurate timetables that omit many of the constraints of participants and lead to conflicts when a schedule is generated. Or, it could be that participants have very volatile schedules that keep them away from home base and prevent them from attending previously scheduled meetings. Depending on the cause, the system may switch to a different mode of delivering the "Schedule Meeting" functionality, perhaps by having a person collect timetables (so that he can confirm that they are fully updated). Alternatively, the system may switch to a mode where a person generates meeting schedules because a person can think strategically when it is a good time for each meeting.

Such diagnosis and reconfiguration can be difficult to perform fully automatically, without human input. Poorly motivated system reconfiguration can lead not only to suboptimal performance, but also to the loss of trust toward the autonomic system. Thus, a person may be in the loop to provide information (e.g., the actual cause of high failure rates) or to select/confirm a reconfiguration. Such human feedback can be used by the system for learning to improve its diagnosis. However, for some types of software expecting a meaningful user input may not be feasible – where do you find a person who understands your operating system enough to determine the causes of failures and to suggest/approve reconfigurations? For application software, however, we expect that having a person in the loop will be normal. Note that the existence of a goal model is crucial if one wants to have a person involved during reconfiguration because then the person is looking at alternatives expressed in stakeholder-oriented terms, rather than system-oriented terms.

4.3.2 Self-Optimization

Feedback could also be designed to give information on how well is functionality is being delivered with respect to stakeholder qualities. For example, participants may evaluate meetings with respect to their effectiveness (softgoal "Good Quality Schedule" in Figure 1) by considering factors such as attendance, degree of participation, and productivity. On the basis of such feedback, the system may choose to switch to a mode where schedules are generated by a person.

[4] describes a framework where stakeholder preferences can be taken into account in selecting a behaviour for a software system under design. However, much research remains to be done to refine our tools for modeling, analyzing and evaluating quality attributes.

4.3.3 Self-Repair

Suppose now that the system is unable to schedule a meeting because it is missing a timetable. Note that this is an application-level failure (rather than a system- or middleware-level one). Switching behaviour could solve the problem: a person could fetch the missing timetable through personal contact.

This scenario raises, again, the problem of diagnosis. The timetable may be missing because a participant is ignoring system requests, or because he is away and inaccessible. In the first case, requesting the timetable personally could help. In the second case, on the other hand, switching to an alternative behaviour will not help and a more appropriate response may be to revise the list of participants and repeat the scheduling process; or, to proceed with the scheduling on the basis of available timetables.

5. CONCLUSION

The essential characteristic of autonomic computing systems is their ability to change their behaviour automatically in case of failures, changing environment conditions, etc. In this paper, we outline an approach for designing autonomic computing systems based on goal models that represent *all* the ways high-level functional and non-functional stakeholder goals can be attained. These goal models can be used as a foundation for building software that supports a space of behaviours for achieving its goals and is able to analyze these alternatives (with respect to important quality criteria), its own state, and its environment to determine which behaviour is the most appropriate at any given moment. For such systems, goal models provide an intentional view unifying all the system components and demonstrating how they must work together to achieve the overall system objective. Goal models also support requirements traceability thus allowing for the easy identification of parts of the system affected by changing requirements. When properly enriched with relevant design-level information, goal models can provide the core architectural, behavioural, etc. knowledge for supporting self-management. Of course, an appropriate monitoring framework as well as, perhaps, learning mechanisms need to be introduced to enable self-management.

6. REFERENCES

[1] A. Dardenne, A. van Lamsweerde and S. Fickas. Goal-Directed Requirements Acquisitions, *Science of Computer Programming*, 20:3-50, 1993.

[2] M. S. Feather, S. Fickas, A. Van Lamsweerde, and C. Ponsard. Reconciling system requirements and runtime behavior. In Proc. *9th International Workshop on Software Specification and Design*, p. 50. IEEE Computer Society, 1998.

[3] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, R. Sebastiani. Reasoning with Goal Models. In Proc. *21st International Conference on Conceptual Modeling (ER2002)*, Tampere, Finland.

[4] B. Hui, S. Liaskos, and J. Mylopoulos. Requirements Analysis for Customizable Software: Goals-Skills-Preferences Framework. In Proc. *11th IEEE International Requirements Engineering Conference (RE'03)*, Monterrey, CA, pp. 117–126, September 2003.

[5] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-Oriented Domain Analysis (FODA) feasibility study (CMU/SEI-90-TR-21, ADA235785). Technical Report, SEI/CMU, 1990.

[6] J. Kephart and D. Chess. The vision of autonomic computing, *Computer*, 36(1):41–50, 2003.

[7] S. Liaskos, A. Lapouchnian, Y. Wang, Y. Yu, and S. Easterbrook. Configuring common personal software: a requirements-driven approach. Technical Report CSRG-512, University of Toronto, 2005. Available at: ftp://ftp.cs.toronto.edu/csrg-technical-reports/512/.

[8] R. Murch. Autonomic Computing. Prentice Hall, 2004.

[9] J. Mylopoulos, L. Chung, and B. Nixon. Representing and using non-functional requirements: a process-oriented approach, *IEEE Transactions on Software Engineering*, 18(6):483–497, 1992.

[10] R. Sebastiani, P. Giorgini, J. Mylopoulos. Simple and Minimum-Cost Satisfiability for Goal Models. In Proc. *CAiSE 2004*, Riga, Latvia.

[11] W. Spears, K. De Jong, T. Baeck, D. Fogel, H. Garis. An Overview of Evolutionary Computing. In Proc. *European Conference on Machine Learning*, 1993.

[12] K. Sycara, M. Klusch, S. Widoff, and J. Lu. Dynamic Service Matchmaking among Agents in open Information Environments, *ACM SIGMOD Record*, Special Issue on Semantic Interoperability in Global Information Systems, A. Ouksel, A. Sheth (Eds.), 28(1):47–53, 1999.

[13] A. van Lamsweerde. Requirements Engineering in the Year 00: A Research Perspective. Proc. *ICSE'00*, Limerick, Ireland, June, 2000.

[14] E. Yu. Modeling Organizations for Information Systems Requirements Engineering. In Proc. *1st IEEE International Symposium on Requirements Engineering*, San Diego, CA, 1993, pp. 34-41.

[15] Y. Yu, J. Mylopoulos, A. Lapouchnian, S. Liaskos, and J.C.S.P. Leite. From stakeholder goals to high-variability software designs. Technical Report CSRG-509, University of Toronto, 2005. Available at: ftp://ftp.cs.toronto.edu/csrg-technical-reports/509/.