An Architecture for Gradual Transition Towards Self-Managed Software Systems*

Edin Arnautovic
Institute of Computer Technology
Vienna University of Technology, Vienna, Austria
arnautovic@ict.tuwien.ac.at

ABSTRACT

While management of today's software systems is usually performed by humans using some user interface (UI), *autonomic systems* would be self-managed.

This doctoral research addresses the research problem of gradual transition towards self-managed software systems and proposes and investigates a particular architecture for its solution.

In particular, we propose unified communication between a system to be managed and its (human or autonomic) manager. Such communication is specified using our high-level discourse metamodel based on insights from theories of human communication. This should make such communication easier to design and understand by humans.

Categories and Subject Descriptors

D.2 [**D.2 Software Engineering**]: Software Architectures; K.6 [**Management of Computing and Information Systems**]: System Management

General Terms

Design, Management

Keywords

self-managing systems, autonomic computing

1. INTRODUCTION

Large and complex software systems need to be managed with respect to, e.g., trouble shooting, dynamic reconfiguration and parametrization. Managing such systems requires dedicated and especially trained personnel and is, therefore, costly. For example, companies now spend between 33% and 50% of their total cost of ownership recovering from or preparing against failures and 80% is spent on operations, maintenance and minor enhancements [19]. In addition, a special user interface for management tasks has to be provided, either built into the software to be managed or plugged onto it. In order to address these issues, there is relatively new research on *autonomic systems* [8] that would be *self-managed*. The basic idea is to have a so-called autonomic manager, which is itself built in software — as a separate component — and supposed to manage the software system in question [8]. Unfortunately, this ambitious approach is difficult to implement at once and to put into wide-spread industrial use, especially for legacy systems.

The doctoral research presented in this paper addresses the research problem of gradual transition towards self-managed software systems and proposes and investigates a particular architecture for its solution. In our approach, "human-managed" and "autonomic-managed" live together, where more and more related tasks will be moved from the human administrator to the autonomic manager. For such an approach, it is desirable to have the communication between the managed software system and its (human or software) manager on the same semantic level. In particular, this architecture utilizes high-level interaction specifications based on discourse models. Such discourse models are based on insights from theories of human communication and will allow for a unified approach to high-level communication between the managed software and both its human and its autonomic manager. As a consequence, this communication will be better understandable as well as easier to design by humans. Additionally, the user interfaces can even be generated automatically from such well-defined interaction specifications.

In the remainder of this paper we give a brief survey of related work and background on human communication theories used. Then, we sketch our transition architecture and our discourse metamodel. Finally, we conclude with an outlook on future work.

2. BACKGROUND

2.1 Self-Managed Software Systems

Management of software systems includes processes and tasks required to control, measure, optimize and configure software in a computing system. It is crucial to satisfy quality requirements such as performance, availability and security. Nowadays, software systems are managed by humans with possibly some more or less sophisticated tools.

*This doctoral research is supervised by Professor Hermann Kaindl (kaindl@ict.tuwien.ac.at) and is partially carried out in the OntoUCP project (No. 809254/9312) funded by the Austrian FIT-IT Program and Siemens AG Österreich.

.

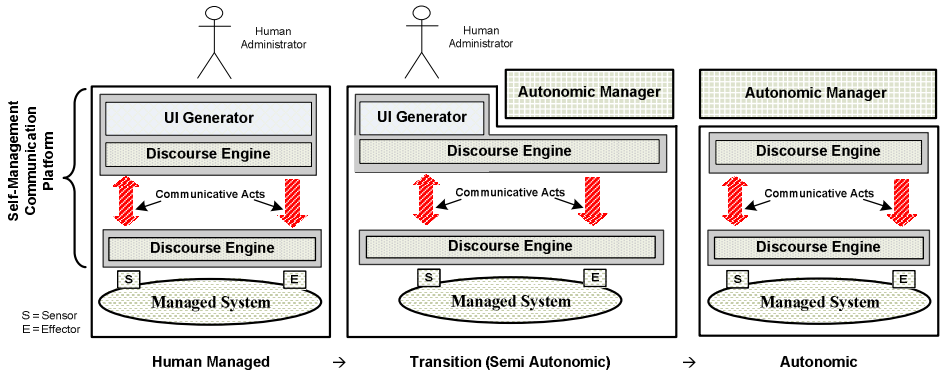

Figure 1: Transition Architecture.

In order to continue to grow in terms of complexity and still being manageable and operating at reasonable costs, software systems should function with least possible human intervention. Humans should ideally only define high-level business policies and the system should adapt to any changes and influences during runtime. Following this, software systems should operate analogously to the human autonomic nervous system, controlling most important body functions without concision intervention from the brain. Researchers at IBM came up with this idea in 2001, addressing increasing complexity and total cost of ownership [5, 8]. However, some approaches for self-adaptive software existed already (e.g., [16]). Since then, several approaches have been presented for adding the autonomic capabilities to software systems [4, 10], however very few of them consider the transition towards self-managed systems.

Parekh et al. [17] address the addition of autonomic capabilities into legacy systems and thus the transition towards self-managed systems in some sense. The interaction between legacy system to be managed and the autonomic controllers is based on publish/subscribe event notification in their approach and is still on a low abstraction level.

In [20] a programming model for effectors is proposed in order to abstract from their technical details. This abstraction would probably contribute to easier transition. We use insights from human communication theories in order to make such an abstraction understandable to humans.

Unfortunately, the previously proposed protocols for communication between managed elements and autonomic managers are still on a low semantic level (e.g., standard Web services [6]).

Generally, a lack exists in the generic approaches which would give the foundations for the field [9]. Our work intends to contribute filling this gap. We are not aware of any complete, defined approach for the transition from human- to self-managed software systems.

2.2 Communication Theories

Communication with and within software systems can be specified in many ways, where traditionally a major distinction is made whether it is with a human user or within software. We strive for a unified and high-level approach to communication based on the following work:

Communicative acts Philosophers observed that human speech is also used to do something with intention — to act. Early and seminal work on speech acts was done by Searle [18]. In this essay Searle claims that "speaking a language is performing speech acts, act such as making statements, giving commands, asking questions and so on". Such speech acts are basic units of language communication. Since speech act theory provides a formal and clean view of communication, computer scientists have found speech acts very useful for describing communication also apart from speech or natural language (e.g., inter-agent communication in FIPA Agent Communication Language (ACL)[1] and information systems [14]). To emphasize their general applicability, the notion communicative act is used in this context.

Rhetorical Structure Theory Rhetorical Structure Theory (RST) [12] is a linguistic theory focusing on the function of text, widely applied to the automated generation of natural language. It describes internal relationships among text portions and associated constraints and effects. The relationships in a text are organized in a tree structure, where the rhetorical relations are associated with non-leaf nodes, and text portions with leaf nodes. In our work we make use of RST for linking communicative acts.

We have already worked on discourse models derived from human-human communication for modeling human computer interaction [2]. The work prior to this research showed that it is possible to generate user interfaces automatically from specifications based on communicative acts [3]. Our preliminary thoughts on using such discourse models for unified communication within systems-of-systems (some systems including humans) are promising as well [7].

3. TRANSITION ARCHITECTURE

Figure 1 illustrates a sketch of our new transition architecture. It is supposed to facilitate the transition to self-managed systems by supporting unified communication with human administrators and an autonomic manager. During transition towards self-managed operation of the software system, human administrators would be gradually replaced by the software component autonomic manager.

For software systems being manageable, they have to provide two types of interfaces: *Sensors* and *Effectors* [6] (**S** and

[1]Foundation for Intelligent Physical Agents, Communicative Act Library Specification, www.fipa.org

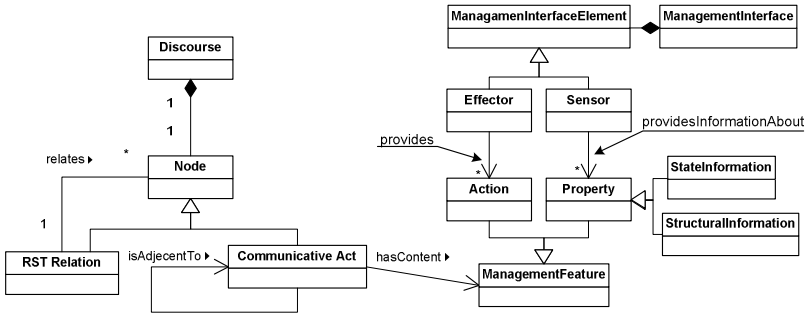

Figure 2: Self-Management Discourse Metamodel.

E in Figure 1). Sensors provide information about current system properties, either on request or on their own initiative. (That is why the related arrow in Figure 1 is bidirectional.) Effectors offer functionality to invoke some action in the system, e.g., for changing some property. (Since these actions are invoked from the outside only, the related arrow is unidirectional.) For example, an application server could provide information about processor and memory load at its sensors and offer functionality for changing parameter values at its effectors.

Today's software systems typically provide such access on the level of object-oriented interfaces or Web services. This involves some form of *message passing*, metaphorically defined on a slightly higher level than pure procedure calls. Still, such interfaces would be on a lower level than our proposed discourse models presented below.

Therefore, it will be necessary to "wrap" the lower-level interfaces in a Discourse Engine as indicated in Figure 1. For the autonomic manager as the communication partner, the same kind of "wrapper" has to be provided by another instance of the Discourse Engine. The Discourse Engines control basically the flow of discourse. For the human administrator as the communication partner, a UI Generator will be provided for automated generation of a user interface. For both communication partners, however, the same communicative acts would flow back and forth in the course of enacting the discourse. For which communication partner they are translated — human or software, respectively — and how, is to be handled by the Communication Platform.

This architecture covers the whole spectrum of management possibilities. Without an autonomic manager first, it provides for high-level communication with the human administrator only, through a generated user interface. During the transition, the Communication Platform handles the management partly done by the human administrator and partly by the autonomic manager. Over time, more and more management tasks will be taken over by the autonomic manager. Finally, the vision is to have a fully self-managed system, where the high-level communication between the autonomic manager and the managed system is better understandable and easier to specify by humans.

4. DISCOURSE METAMODEL

Our self-management discourse metamodel defines what the models of an interaction design specification should look like in our approach and is based on insights from several theories of human communication. Discourse models according to this metamodel can specify the communication within software systems, more precisely between the managed software system and its autonomic manager.

Our new approach for interaction modeling in terms of discourses can be sketched as follows. In essence, it has communicative acts as its "atoms", from which "molecular" structures can be composed in two ways:

- According to Conversation Analysis [11] there are frequently occurring pairs of communicative acts — **adjacency pairs**. E.g., a question should have a related response, and a request should have an accept or a reject. So, adjacency pairs of communicative acts create dialogue structure and are modeled in our metamodel by the *isAdjacentTo* relation.
- **RST relations** provide structure relating communicative acts (or adjacency pairs) and further structures made up of RST relations.

The metamodel shown in Figure 2 as UML class diagram consists of two main parts. The left part contains communicative acts, RST relations, and their hierarchical structure; the right part consists of classes involved in the description of management interfaces.

The *Communicative Act* class carries the intention of the interaction like asking a *Question* about Total CPU Load. The communicative acts can be further classified into *Assertions*, *Directives* and *Commissives*. *Assertions* convey information without requiring receivers to act beside changing their beliefs (e.g., Informing and Answer). *Directives* (e.g., Question, Request, Accept) and *Commissives* (e.g., Offer) require an action by the receiver or sender and the advancement of the discourse by further communicative acts. This classification is not shown in Figure 2 to avoid cluttering the diagram.

RST relations relate communicative acts and further structures made up of RST relations. The top left part of Figure 2 makes up the hierarchical structure of RST, by generalizing RST relations and communicative acts into nodes which are related by RST relations and thus form a tree structure. There exist several types of rhetorical relations such as *Result, Condition, Background, etc.* For example, a *Result* relation represents that the sequence of actions is a consequence of the "situation" resulting from the preceding interactions. A *Condition* represents that some conditions have to be fulfilled by preceding interactions in order to continue with the discourse. These particular relations are not shown in Figure 2 to avoid cluttering the diagram. Part of our research is to investigate the applicability of particular relations to the self-management domain.

The right part of Figure 2 represents the content of discourse for managing a software system. There are two kinds of information to be exchanged between a software system and its (autonomic or human) manager, namely the information about current system properties and the actions requested by the autonomic manager which are to be executed by the Managed System. We define two types of properties: *StateInformation* and *StructuralInformation*. *StateInformation* represent the system as seen from outside using its parameters (black box). *StructuralInformation* can carry the information about runtime architecture and structure of the system. However, our metamodel does not define the format of the content. So, common standards such as Common Base Event format [15] could be used.

5. CONCLUSION AND FUTURE WORK

In essence, this research proposes and investigates a new architecture for transition towards self-managed software systems where a high-level discourse metamodel will be used for unified communication within this architecture. The expected results of this research will facilitate and hopefully speed up the transition from human- to self-managed software systems. Since this transition is challenging and should happen gradually, foundations and tools — as to be developed in the proposed project — are crucial.

Currently we are working on prototypic implementation of our approach on the top of the Java application server (JBoss[2]). We will evaluate autonomic capabilities as well as a transition process according to experiences from the research community (e.g., [1]). Future research will include further development of our architecture and its evaluation as well as validation of our discourse metamodel against self-management scenarios. Another part of the research will investigate a transition process using our approach according to the IBM autonomic maturity levels criteria [13].

6. REFERENCES

[1] A. B. Brown and C. Redlin. Measuring the effectiveness of self-healing autonomic systems. In *ICAC '05: Proceedings of the Second International Conference on Automatic Computing*, pages 328–329. IEEE Computer Society, 2005.

[2] J. Falb, H. Kaindl, H. Horacek, C. Bogdan, R. Popp, and E. Arnautovic. A discourse model for interaction design based on theories of human communication. In *CHI '06 Extended Abstracts on Human Factors in Computing Systems*, pages 754–759. ACM Press, 2006.

[3] J. Falb, R. Popp, T. Röck, H. Jelinek, E. Arnautovic, and H. Kaindl. Using communicative acts in high-level specifications of user interfaces for their automated synthesis. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE'05)*, pages 429–430. ACM Press, 2005. Tool demo paper.

[4] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*, 37(10):46–54, 2004.

[5] P. Horn. Autonomic Computing: IBM's Perspective on the State of Information Technology. Technical report, IBM Corporation, 2001.

[2]www.jboss.org

[6] IBM Corporation. *An architectural blueprint for autonomic computing*, third edition, June 2005. White Paper.

[7] H. Kaindl, J. Falb, E. Arnautovic, and R. Popp. High-level Communication in Systems-of-Systems. In *Proceedings of the Fourth Annual Conference on Systems Engineering Research*, Los Angeles, CA, April 2006. INCOSE.

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

[9] P. Lin, A. MacArthur, and J. Leaney. Defining autonomic computing: A software engineering perspective. In *ASWEC '05: Proceedings of the 2005 Australian conference on Software Engineering*, pages 88–97. IEEE Computer Society, 2005.

[10] H. Liu and M. Parashar. Accord: A Programming Framework for Autonomic Applications. *Systems, Man and Cybernetics, Part C, IEEE Transactions on*, 36(3):341–352, May 2006.

[11] P. Luff, N. Gilbert, and D. Frohlich. *Computers and Conversation*. Academic Press, 1990.

[12] W. C. Mann and S. Thompson. Rhetorical structure theory: Toward a functional theory of text organization. In *Text*, pages 243–281, 1988.

[13] R. Murch. *Autonomic Computing*. Prentice Hall, 2004.

[14] M. Nowostawski, D. Carter, S. Cranefield, and M. Purvis. Communicative acts and interaction protocols in a distributed information system. In *AAMAS '03: Proceedings of the second international joint conference on autonomous agents and multiagent systems*, pages 1082–1083. ACM Press, 2003.

[15] D. Ogle, H. Kreger, J. Cornpropst, E. Labadie, M. Chessell, B. Horn, J. Gerken, J. Schoech, and M. Wamboldt. Canonical situation data format: The common base event v1.0.1, 2004.

[16] P. Oreizy, M. M. Gorlick, R. N. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D. S. Rosenblum, and A. L. Wolf. An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems*, 14(3):54–62, 1999.

[17] J. Parekh, G. Kaiser, P. Gross, and G. Valetto. Retrofitting autonomic capabilities onto legacy systems. *Cluster Computing*, 9(2):141–159, 2006.

[18] J. R. Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, Cambridge, England, 1969.

[19] R. Sterritt and M. Hinchey. Why computer-based systems should be autonomic. In *ECBS '05: Proceedings of the 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'05)*, pages 406–412. IEEE Computer Society, 2005.

[20] G. Valetto, G. Kaiser, and D. Phung. A uniform programming abstraction for effecting autonomic adaptations onto software systems. In *ICAC '05: Proceedings of the Second International Conference on Automatic Computing*, pages 286–297. IEEE Computer Society, 2005.