An agent-oriented approach to support change propagation in software evolution*

Khanh Hoa Dam
School of Computer Science and Information Technology
RMIT University
Melbourne, Australia
kdam@cs.rmit.edu.au

ABSTRACT

Software maintenance and evolution is arguably a lengthy and expensive phase in the life cycle of a software system. A critical issue at this phase is change propagation: given a set of primary changes that have been made to software, what additional secondary changes are needed? Although many approaches have been proposed, automated change propagation is still a significant technical challenge in software maintenance and evolution. This paper presents a Ph.D. research in the final stages of developing and evaluating a novel, agent-based, framework to support semi-automated change propagation in evolving software systems.

1. INTRODUCTION

Software maintenance and evolution is the modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment. It is an important part of the software development activity, which can account for a large percentage of the cost of software development [11, page 449]. Although software maintenance and evolution is a highly important area for both research and industry, there has been very little work that we are aware of on software maintenance in agent-oriented software engineering (AOSE). The main purpose of this PhD research is to fill that gap, as well as apply agent technology to the problem of software evolution in a broader context [2, 3, 4].

When software is modified, typically some primary changes are made and then additional, secondary, changes are made as a result. Determining and making these secondary changes is termed *change propagation* [8] and is widely considered to be complicated, labour-intensive and expensive, especially in complex software systems. Although several existing techniques [1] address this problem to a certain extent, these approaches are still labour-intensive solutions, and managing inconsistencies in software models remains a challenging problem [10].

The major goal of our research is developing a framework that provides more effective *automated* support for change propagation in design models. In our view, a tool cannot fully automate change propagation because a tool cannot make decisions involving trade-offs and design styles, where human intervention is required. However, our framework should provide an assistant tool that helps the designer by presenting feasible change propagation options.

The change propagation framework that we proposed [4] is based on a simple hypothesis: given a model (i.e. a design) which has been subjected to primary changes, the system finds inconsistencies in the model (with respect to given constraints), and then invokes repair options to fix these consistency violations, resulting in changes being propagated. However, differing from existing approaches, we use existing agent techniques to represent and implement several major parts of our framework. More specifically, the change propagation engine proposed uses a Belief-Desire-Intention (BDI) platform to perform change propagation. Furthermore, repair options are represented as BDI plans whilst fixing a constraint is considered as a goal/event. The use of BDI-style, event-triggered, plans matches well with the cascading nature of change propagation where a change can cause other changes to be made. In addition, there are usually many ways of fixing a given inconsistency, and this is naturally captured using multiple plans that respond to a given event. Although we do not use the full capabilities of BDI agents, these two properties of change propagation make the use of BDI plans beneficial and, we believe, well motivated.

The remainder of this paper is organised as follows. In the next section we briefly describes our change propagation framework and its components. We then conclude by discussing the current status and future directions of our work (section 3).

2. CHANGE PROPAGATION FRAMEWORK

Much of the work that has been done in change propagation has been addressing the issue at the code level [1]. Recently, however, as the importance of models in the software development process has been better recognised, there is an emerging need to deal with changes at the model level. As a result, the main focus of our work is to deal with propagating changes through design models. Our agent-oriented approach provides a generic change propagation framework (see figure 1) which can be applied to different software engineering methodologies, and in fact we have applied it to both UML and Prometheus [7]. Due to space limitation, we briefly describe the main process and the major components of the framework. Further details of the framework can be found in [2, 3, 4].

The key data items that the framework requires are a meta-model, a collection of well-formedness constraints, an application design model, and a collection of repair plans. The **meta-model** specifies, in the usual manner, what entities exist in a design model, and their relationships. The meta-model is captured in UML, and is exported to XMI format for use by our implementation. The **constraints**

*The author is very grateful for the supervision given on this dissertation research by Associate Professor Michael Winikoff and Professor Lin Padgham. This work is being supported by the Australian Research Council under grant LP0453486, in collaboration with Agent Oriented Software.

Cite as: Extended Thesis Abstract for Doctoral Mentoring Program, K.H.Dam Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008), Padgham, Parkes, Müller and Parsons (eds.), May, 12-16., 2008, Estoril, Portugal, pp. 1736-1737.

specify conditions that a well-formed design should satisfy. We use the Object Constraint Language [6] to specify constraints. OCL is part of the UML standards which is used to specify invariants, pre-conditions, post-conditions and other kinds of constraints imposed on elements in UML models. The application design **model** is a existing design, for example a Prometheus design in the case of using Prometheus methodology [7].

At design time the repair plan generator takes the constraints and the meta-model as inputs, and returns a parameterized set of event-triggered repair plan types that are able to repair violations of the constraint [2]. Repair plans are represented as BDI-style event-triggered plans (we use a syntax based on AgentSpeak(L) [9]). Our translation schema guarantees completeness and correctness, i.e. there are no repair plans to fix a violation of a constraint other than those produced by the generator; and any of the repair plans produced by the generator can fix a violation. The set of repair plan types is created ahead of time and forms the library of plans that the change propagation engine uses to fix constraint violations.

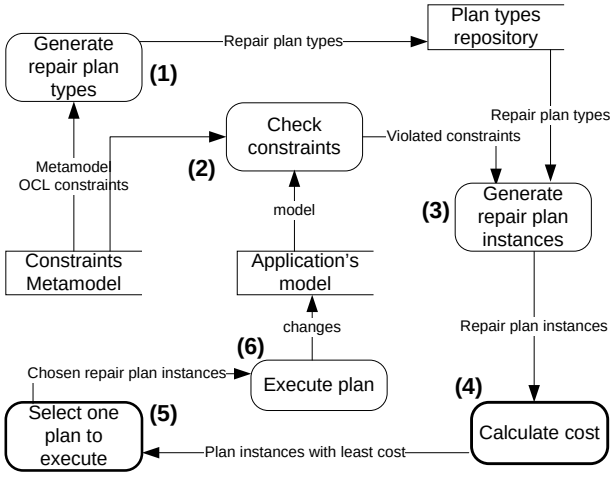

Figure 1: Change Propagation Framework

At runtime we check whether the constraints hold in the design model. We use the repair plans to generate plan instances (i.e. repair options) for the violated constraints, i.e. inconsistencies. Typically a given inconsistency will have a number of repair plans that could be used to restore consistency. In order to deal with the issue of how to select amongst these repair plans, we have proposed and implemented a cost calculation mechanisms for repair plans [3]. We select a repair option, possibly by picking the single cheapest, if it exists, or by asking the user. The selected repair option is executed, and it updates the application design model.

3. CURRENT STATUS AND FUTURE WORK

We have fully developed a theoretical foundation for our framework as well as refined it with details. The repair plan generator, an important component of the framework which represents a new method for automatically generating repair plans from OCL constraints, has been developed. Our repair plan generator is able to take common OCL constraints as inputs, and perform a translation that has been proven to be sound and complete [2].

In addition, we have successfully resolved another important issue as part of this framework involving the selection between different applicable (repair) plan instances to fix a given constraint violation [3]. We addressed this problem by defining a suitable notion of repair plan cost that takes into account the cascading nature of change propagation by incorporating both conflict and synergies between plans. We were also able to prove that, with regard to this notion of cost in this particular domain, in order to repair a set of violated constraints we can consider a single constraint at a time, in an arbitrary order, with no loss of generality. This has helped us develop and implement an algorithm, based on the notion of cost, that finds cheapest options and proposes them to the user.

We have recently implemented the Change Propagation Assistant (CPA) tool that demonstrates how our approach works in practice. The tool is integrated with the Prometheus Design Tool (PDT)[1], a modeling tool that supports the Prometheus methodology for building agent-based systems. We then performed empirical evaluations in both artificial settings and a real situation (the design of a weather alerting system [5]) to assess the efficiency and effectiveness of our change propagation framework generally and the CPA tool in particular. The results have shown that our approach is effective given that a reasonable amount of primary changes are provided and the cost algorithms are practical for small to medium realistic applications [3].

The evaluation outcomes leads us to some specific areas for future work. We intend to investigate the interaction between constraints in order to limit the number of plans to be explored in our algorithm and consequently improve its performance and scalability. In addition, we consider dealing with the issue that in some cases there may be a large number of repair options returned by the CPA tool, which makes it hard for the user to select which one to choose.

4. REFERENCES

[1] R. Arnold and S. Bohner. *Software Change Impact Analysis*. IEEE Computer Society Press, 1996.

[2] K. H. Dam and M. Winikoff. Generation of repair plans for change propagation. In M. Luck and L. Padgham, editors, *Agent Oriented Software Engineering (AOSE)*, pages 30–44, Honolulu, Hawaii, May 2007.

[3] K. H. Dam and M. Winikoff. Cost-based BDI plan selection for change propagation. In *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Estoril, Portugal, May 2008. (To appear).

[4] K. H. Dam, M. Winikoff, and L. Padgham. An agent-oriented approach to change propagation in software evolution. In *Proceedings of the Australian Software Engineering Conference (ASWEC)*, pages 309–318. IEEE Computer Society, 2006.

[5] I. Mathieson, S. Dance, L. Padgham, M. Gorman, and M. Winikoff. An open meteorological alerting system: Issues and solutions. In V. Estivill-Castro, editor, *Proceedings of the 27th Australasian Computer Science Conference*, pages 351–358, Dunedin, New Zealand, 2004.

[6] Object Management Group. Object Constraint Language (OCL) 2.0 Specification, 2006.

[7] L. Padgham and M. Winikoff. *Developing intelligent agent systems : a practical guide*. John Wiley & Sons, Chichester, 2004. ISBN 0-470-86120-7.

[8] V. Rajlich. Changing the paradigm of software engineering. *Commun. ACM*, 49(8):67–70, 2006.

[9] A. S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In *MAAMAW '96: Proceedings of the 7th European workshop on Modelling autonomous agents in a multi-agent world : agents breaking away*, pages 42–55. Springer-Verlag, 1996.

[10] G. Spanoudakis and A. Zisman. Inconsistency management in software engineering: Survey and open research issues. In K. S. Chang, editor, *Handbook of Software Engineering and Knowledge Engineering*, pages 24–29. World Scientific, 2001.

[11] H. V. Vliet. *Software engineering: principles and practice*. John Wiley & Sons, Inc., 2nd edition, 2001. ISBN 0471975087.

[1] http://www.cs.rmit.edu.au/agents/pdt