Model-based Self-Adaptive Components: A Preliminary Approach

Pedro Rodrigues and Emil Lupu

Department of Computing, Imperial College London, UK

Abstract. Due to the increasing scale, complexity, dynamicity and heterogeneity of modern software systems, it is not feasible to solely rely upon human management to guarantee a good service level with such availability demand. Self-managing systems are needed as an effective approach to deal with those issues by exploiting adaptive techniques to adjust a system. On top of that, model-based adaptation improves reliability, hence enhancing trust in self-managing systems. However, a centralised approach can be too complex to manage thus compromising system dependability. This paper presents a preliminary decentralised approach on model-based self-adaptive components.

1 Introduction

Self-management can be decomposed in various functions, as identified by IBM [1] as the MAPE-K loop: Monitoring, Analysis, Planning and Execution, all underpinned by system knowledge. The Monitoring service supervises the system and notifies system changes. The Analysis service receives these notifications and analyses system consistency as well as optimality, and sends a request to the Planning service to change the system when the system is not behaving as expected. The Planning service decides which changes have to be made and passes them to the Execution service to apply them.

To the best of our knowledge, most proposals in self-adaptive system are based on centralised management and a centralised model of the system. Centralised control of the details of all components in a system implies great complexity since adaptation concerning different levels of a system is dealt at the centralised manager; does not allow components to be completely autonomous and limits the reusability of management specifications concerning one component in other systems. Moreover, adaptation actions mainly focus on structural modifications, such as replacing components, hence not dealing with behavioural modifications.

In this paper we present a preliminary approach for the design of self-managing system composed of autonomous components for pervasive environments. The assumptions of systems administrators regarding adaptation action effects may not be verified at runtime. However, little work has been conducted on online reasoning about adaptation repercussions in terms of its outcome and costs.

This paper is organised as follows. Section 2 presents some of the relevant proposals regarding the services of the MAPE loop. Section 3 discusses our

Andrew V. Jones (Ed.): Proceedings of the 1st Imperial College Student Workshop (ICCSW '11), pp. 73–79. September 29th–30th 2011, London UK.

preliminary proposal for the design of composite autonomous components based on model-based adaptation. Section 4 ends this paper with closing remarks.

2 Background

This section discusses some proposals for the main concepts regarding self-managing systems. Monitoring is the foundation service of these systems as it provides online knowledge on system state. Relying upon that knowledge, decision-making services verify the need for adaptation and decide the most suitable adaptation action. Component models provide means for the specification of component structure and behaviour, which may support the decision-making service. Finally, self-adaptation frameworks combine those concepts to add self-managing properties to software systems.

2.1 Monitoring

A self-adaptive monitoring service should control the detail of collected monitoring data from sensors in the following manner: supervising main component metrics while the component fulfils its requirements and switching to more detailed monitoring when the system deviates from its normal behaviour. In this regard, an automated method for selecting a subset of metrics to be collected in the context of correlation-based monitoring was proposed in [11], resulting in detecting on average 66% of faults in case of all metrics were being connecting, though collecting only 30% of them. Alternatively, in the approach presented in [12] when an anomaly is detected the monitoring level is progressively increased until a fault root is found or the monitoring level achieved its maximum. These techniques allow to reduce power consumption while not compromising fault diagnosis accuracy.

2.2 Decision-Making

The decision-making service is responsible for choosing the most suitable adaptation action to be executed in face of a context or state change. Policies have been a successful way of expressing automated management of distributed systems and changes in the system behaviour at runtime [14]. While Event-Condition-Actions (ECA) policies express reactive actions based on the current system state, Goal and Utility-function policies express desirable system states.

The Stitch language [5] proposes a modification to ECA policies based on two constructions: *tactics* and *strategies*. Tactics implement the Condition-Action part and introduce a construct to indicate the expected behaviour of the tactic actions. Strategies are defined as a tree of Condition-Tactics nodes which define a condition for the tactic to be applied, an estimation of the time it needs to adapt the system and a list of conditional branches that define the following steps in the tree.

Alternatively, the application of genetic algorithms is proposed in [13] to determine the best configuration of a system in face of state or context change. However, it takes a considerable amount of time to compute the most fitting system configuration, as a considerable number of possible configurations is explored.

2.3 Component Models

The component model in Darwin [10] specifies component structural view in terms of required and provided services (ports) and component interactions through bindings between provided and required ports. A composite component defines bindings among internal components ports as well as binding the composite component ports to the ones of internal components. Although, a FRACTAL component [3] is also based on the principle of provided and required interfaces, each component is involved in a *membrane* that provides external control interfaces to introspect and reconfigure the component internal details, and a *content* that consists in a set of sub-components. The membrane control interfaces normally correspond to several controller and interceptor objects.

The above component models mainly focus on providing means for structural adaptation. Modes [9] extend the Darwin component model with a representation of the expected interaction behaviour between required and provided services. Each of the identifiable component states is defined as a behaviour type that is characterised by an interaction process, constrains and properties. The interaction process is represented by Finite State Processes that define a set of scenarios in which the component can operate. This representation can be used to construct a Labelled Transition System, which can then be passed to the LTSA toolset to detect the presence of deadlocks and other properties analysis. On the other hand, the MOCAS model [2] only focuses on behavioural adaptation. Each component sets a UML state machine at runtime to characterise and realise its behaviour. This state machine consists in a set of states which are connected through transitions, each one being designated by an input signal, a guard (a boolean expression) and effects. A state also includes invariants, that together with guards designate business properties.

2.4 Self-Adaptation Framework

The Rainbow framework [7] relies upon Acme ADL [8] as a generic architectural model to manage a given system. Each component can be annotated with functional and non-functional properties, expected interactions with other components and specific architectural constraints. Furthermore, the framework uses the Stitch language [5] to express adaptation policies, which are triggered by reasoning over the architectural model. However, the adaptation actions are directly applied to the underlying software system.

The GRAF framework [6] proposes using a runtime abstract model between the adaptable software and the adaptation manager, where the adaptation manager does not directly control the adaptable software. The Runtime Model Man-

ager evaluates the pre-conditions of the adaptation policy before applying adaptation actions on the runtime model. Thereafter, the conducted modifications are validated using the policy post-conditions as well as the model invariants. If they do not conform with such constraints the Runtime Model Manager rollbacks the alterations performed on the Runtime Model.

Both frameworks rely upon a centralised representation of the system as well as centralised management, which can result in significant management complexity.

2.5 Summary

Centralised representation and management of a software system can become too complex that may overwhelm the benefits of having a self-managing system, while restricting the design of autonomous components. Dealing with adaptation concerning different levels of the software system at a centralised management may compromise system dependability. On the other hand, minimising the set of metrics of online monitoring and analysis reduces the complexity of system management while minimising power consumption. Furthermore, work on online reasoning on behavioural and runtime changes is insubstantial, limiting the trust in self-managing systems as well as their autonomy.

3 Model-based Self-Adaptive Component

We propose that each component has self-managing capabilities in order to reduce the complexity of specifying the underlying mechanisms for autonomous system management. A system is structured as a hierarchy of component compositions, *i.e.* single components can be used to construct a composite one which in turn can be used in other composite relationship. The resulting composite component is responsible for the management of its sub-components, though their internal details are managed by themselves. Each component defines the level of management details a parent component is allowed to control. The hierarchical structure provides means to handle adaptation concerns at different levels. The underlying mechanisms and models of self-adaptive components will be presented in the next sub-sections.

3.1 Runtime Model

Similar to the GRAF framework [6] we propose that a component comprises a Runtime Model which incorporates a structural and a behavioural view. The structural view consists in a ADL specification of its provided and required services, annotated with some properties, *e.g.* requirements and capabilities. For instance, a given service provides an average response time of 500ms; a service client requires a service provider with 10Mbps of available bandwidth for a file transfer service, etc. The behavioural view represents the behavioural model of its provided services and internal details. Moreover, the set of provided services

can be dynamically evolved in order to accommodate new functionality or replace existing one in face of a change in system requirements.

Furthermore, each component provides a list of relevant metrics to be monitored along with the dependencies among them. Such information can be exploited by the aforementioned techniques to reduce monitoring complexity when the component is behaving properly. In addition, based on the values of those metrics, utility functions can be specified to evaluate the system. The domain of each metric can be discretised in order to reduce the complexity of specifying utility functions, *e.g.* response time $\in [0, 100]ms \rightarrow$ `low`, $]100, 500]ms \rightarrow$ `medium`, $]500, \infty[\rightarrow$ `high`.

Finally, the Runtime Model includes a set of functional and non-functional requirements, structural and behavioural invariants and goals, that guide the decision-making service when choosing a suitable adaptation action.

3.2 Adaptive Monitoring

For each of the relevant metrics specified in the component's description a configuration determining the type of monitoring, interval or period based, and the correspondent parameters is specified. For period-based monitoring, the metric's value is periodically propagated based on a specified interval. When using interval-based monitoring, the metric value is propagated when it falls outside a specified numeric interval. Such parameters as well as the set of currently monitored metrics are dynamically adapted using reactive policies based on the monitored values and the dependencies among metrics to reduce power consumption. For example, when the number of clients exceeds a given threshold, increase the monitoring rate of system response time; stop monitoring available bandwidth when the system response time is below 100ms. Moreover, the monitoring component is also responsible for updating the aforementioned annotations on the structural model.

3.3 Decision-Making

Each component applies its adaptation actions relying upon the current view of its behavioural and structural models. Therefore, the monitoring component updates those two models instead of directly propagating metrics values to the decision-making service. Based on the Stitch language [5], we suggest adaptation actions to be specified using ECA policies, each one including an expected outcome of its adaptation actions in terms of structural and behavioural modifications as well as metric variations and an estimation of the cost of applying those actions.

By reasoning over the runtime behavioural model, the decision-making service verifies which ECA policies need to be activated. If two or more policies are triggered, the decision-making service applies an utility function to the expected outcome and the cost estimation; the one with the highest utility value is selected. However, the adaptation actions are firstly executed in the runtime

behaviour model in order for the decision-making service to verify if their execution violates components goals, invariants or requirements. If the simulation of the adaptation actions does not lead the runtime model to an inconsistent state, the modifications are applied in the component; otherwise the runtime model is rollbacked to the previous state before simulating the ECA policy execution. Moreover, after the execution of an ECA policy the expected outcome and cost estimation are updated using a suitable statistic method based on the metrics values captured by the monitoring service. Alternatively, statistical models can be used to predict the outcome of a given adaptation action, using collected values to improve prediction [4].

When a goal, an invariant, a functional or non-functional requirement is invalidated and there is not an ECA policy to fix the identified problem, a plan is generated using the estimation, evaluation or prediction of the outcome of actions in ECA policies. The Runtime Model is used to validate the generated plan, *i.e.* if the plan does not violate the aforesaid components requirements, goals and invariants. The planning algorithm can be parameterised to generate a plan as quick as possible or an optimal one using utility functions. Additionally, utility-functions can also be used to periodically improve system configuration parameters or structural configuration, *i.e.* sub-components replacement on a composite relationship. Since adaptation can imply a significant cost in terms of system availability, optimisation can be conducted only when the system utility is below a given threshold.

4 Final Remarks

Self-managing systems should be designed as a hierarchy of self-adaptive composite components to ease their specification and to provide means for management at different levels while allowing to reuse components and their managing features in similar systems. Online reasoning on structural and behavioural adaptation repercussions can improve the reliability of self-managing systems, as well as increase the confidence of systems administrators towards such systems.

In this paper we have presented a preliminary design to approach the aforementioned drawbacks of current self-managing systems. We intend to elaborate on each one of the presented services and combine them to design systems of self-managing components for pervasive scenarios. The main goal of deploying self-managing systems is to decrease managing costs by reducing human intervention. One one hand, such goal cannot be achieved if the underlying mechanisms of self-managing systems do not have the level of reliability so that system administrators can truly trust them to perform their job. On the other hand, self-managing systems do not completely replace the role of system administrators, *i.e.* leading to completely autonomous management, as the actions performed by self-managing systems are governed by high-level goals specified by system administrators. Consequently, system administrators still have the control of software systems, only delegating system management to the self-managing frameworks.

Acknowledgements

This work was supported by Fundação para a Ciência e Tecnologia under the grant SFRH/BD/73967/2010.

References

1. An Architectural Blueprint for Autonomic Computing. June 2006.
2. C. Ballagny, N. Hameurlain, and F. Barbier. Mocas: A state-based component model for self-adaptation. In *Proceedings of the 2009 Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, SASO '09, pages 206–215, Washington, DC, USA, 2009. IEEE Computer Society.
3. E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J.-B. Stefani. The fractal component model and its support in java: Experiences with auto-adaptive and reconfigurable systems. *Softw. Pract. Exper.*, 36:1257–1284, September 2006.
4. O. Celiku, D. Garlan, and B. Schmerl. Augmenting architectural modeling to cope with uncertainty. In *Proceedings of the International Workshop on Living with Uncertainties (IWLU'07), co-located with the 22nd International Conference on Automated Software Engineering (ASE'07)*, 5 November 2007.
5. S.-W. Cheng, D. Garlan, and B. Schmerl. Stitch: A language for architecture-based self-adaptation, 2011. Submitted for Publication.
6. M. Derakhshanmanesh, M. Amoui, G. O'Grady, J. Ebert, and L. Tahvildari. Graf: graph-based runtime adaptation framework. In *Proceeding of the 6th international symposium on Software engineering for adaptive and self-managing systems*, SEAMS '11, pages 128–137, New York, NY, USA, 2011. ACM.
7. D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*, 37:46–54, October 2004.
8. D. Garlan, R. T. Monroe, and D. Wile. Foundations of component-based systems. chapter Acme: architectural description of component-based systems, pages 47–67. Cambridge University Press, New York, NY, USA, 2000.
9. D. Hirsch, J. Kramer, J. Magee, and S. Uchitel. Modes for software architectures. In *of Lecture Notes in Computer Science*, pages 113–126. Springer, 2006.
10. J. Magee, N. Dulay, S. Eisenbach, and J. Kramer. Specifying distributed software architectures. In *Proceedings of the 5th European Software Engineering Conference*, pages 137–153, London, UK, 1995. Springer-Verlag.
11. M. A. Munawar, M. Jiang, T. Reidemeister, and P. A. S. Ward. Filtering system metrics for minimal correlation-based self-monitoring. In *Proceedings of the 2009 Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, SASO '09, pages 233–242, Washington, DC, USA, 2009. IEEE Computer Society.
12. M. A. Munawar and P. A. S. Ward. Leveraging many simple statistical models to adaptively monitor software systems. *Int. J. High Perform. Comput. Netw.*, 7:29–39, February 2011.
13. A. J. Ramirez, B. H. Cheng, P. K. McKinley, and B. E. Beckmann. Automatically generating adaptive logic to balance non-functional tradeoffs during reconfiguration. In *Proceeding of the 7th international conference on Autonomic computing*, ICAC '10, pages 225–234, New York, NY, USA, 2010. ACM.
14. M. Sloman. Policy driven management for distributed systems. *Journal of Network and Systems Management*, 2:333–360, 1994. 10.1007/BF02283186.