Demonstrating a Runtime Machine-centric Emergent Software Architecture Framework

Roberto Rodrigues Filho and Barry Porter
School of Computing and Communications
Lancaster University
Lancaster, UK
Email: {r.rodriguesfilho, b.f.porter}@lancaster.ac.uk

Abstract—**Current solutions to self-adaptive software architecture are very human-centric, depending on humans to define policies or update models that guide software adaptation at runtime. We argue that this approach is not sufficient to provide fast responses to continual changes that occur in the current operating environment. Our approach derives a continually *emergent* software architecture by: autonomously exploring all possible architectures that can be used to realise a given software system; monitoring that system in execution in terms of its performance and its operating environment; and identifying the optimal architecture for each set of operating environment conditions that are encountered. This demonstration illustrates our framework in two scenarios: first, we enable participants to act as the autonomous agent, exploring various possible architectures over time, and manually constructing the rules by which adaptation will be driven, thereby demonstrating the complexity of human-centric architectural adaptation; and second, we then show a fully autonomous system that performs the same tasks automatically, resulting in emergent software architectures that are highly responsive to changes in the operating environment. Both scenarios are visualised through a graphical user interface.**

I. Introduction

Current approaches to architectural self-adaptation rely heavily on human experts to define adaptation policies that guide runtime adaptation. This has two problems: firstly it depends on predictions made by experts at the design phase which establish a static connection between architectural configurations and expected operating conditions; and secondly the increasing complexity of modern systems, which contain a large number of system configurations, make this kind of static specification increasingly infeasible.

Recent work in architectural self-adaptation, such as [1], [2], [3] and [4] attempt to provide more dynamic self-adaptive techniques. Although these techniques help to reduce the need for making predictions, they still require significant human involvement in the process of defining models and updating those models to accommodate new scenarios.

Our approach, by contrast, is completely machine-centric, removing all burden on humans from the process of defining adaptation rules or models. In this way, we allow autonomous and continuous experimentation of architecture variations in the face of changes in the operating environment, in the system or in the underlying infrastructure. This human-independent process allows the system to autonomously gather information on different architectures under different conditions and enables the most suitable architecture to emerge with no predefined adaptation policies or models. Our framework is also generic across multiple applications, being oblivious to a specific application's features and goals, and requiring no application-specific domain knowledge.

This paper presents our runtime machine-centric framework for emergent software architectures. We briefly describe our implementation and present two interactive demonstration scenarios, using a web server as an application example. The framework is able to assemble functional architectural variations for this web server and autonomously learn which architecture best suits different patterns of client requests. Our demo features a highly visual way for participants to both act as the emergent architecture driver in real time, showing the volume of decisions that must be made in order to do this; and also to experience a fully autonomous version of this.

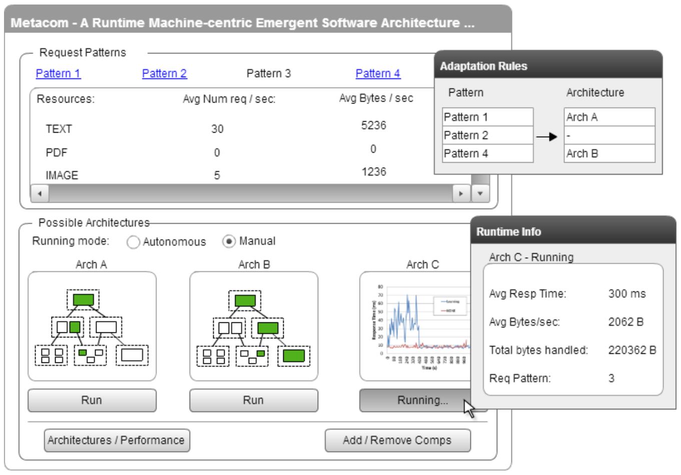
II. Demonstration

A. Framework

This demonstration features our machine-centric emergent software architecture framework. Our framework is divided into three main modules, called Assembly, Perception and Learning. The Assembly module is responsible for autonomously composing software architectures based on a pool of available components; the Perception module is responsible for collecting performance-related and operating condition-related data from running components; and the Learning module is responsible for processing the collected data and establishing correlations between an architecture's performance and the different identified operating environments. To do this the Learning module operates in two phases, exploration and exploitation. During exploration the Learning module experiments with all available architectures in order to collect data about their constituent components and operating conditions. During exploitation, the best-performing architecture is selected for the current operating environment.

B. Application

In order to demonstrate the framework in action we use a web server as an application example. The web server was created using the Dana programming language [5], which provides an advanced component-based runtime. Our web server can be realised from a large pool of available components,



Metacom - A Runtime Machine-centric Emergent Software Architecture ...

Request Patterns

Pattern 1 Pattern 2 Pattern 3 Pattern 4

Resources:	Avg Num req / sec:	Avg Bytes / sec
TEXT	30	5236
PDF	0	0
IMAGE	5	1236

Adaptation Rules

Pattern	Architecture
Pattern 1	Arch A
Pattern 2	-
Pattern 4	Arch B

Possible Architectures

Running mode: Autonomous [✓ Manual]

Runtime Info

Arch C - Running

Avg Resp Time:	300 ms
Avg Bytes/sec:	2062 B
Total bytes handled:	220362 B
Req Pattern:	3

Fig. 1. A prototype of the graphical user interface which allows the exploration of our runtime machine-centric emergent software architecture framework using a fully functioning web server as an application example.

each of which implement small independent behaviours. These components can be composed in different ways (selecting some components from the pool), where each possible composition performs differently under different operating environment conditions – for example compositions that include a cache component work best in request patterns with high repetition, whereas compositions without a cache component work better in request patterns with high variation.

In total our web server has 42 different possible architectures, which can be autonomously assembled by our emergent architecture framework. The framework experiments with each architecture and gathers real-time information from its constituent components, which is then used to learn which architectures best suit each set of observed operating conditions.

C. Scenarios

The demonstration will use two computers, one acting as a client and the other acting as a server. On the client computer, participants will be able to select different client request patterns from a user interface. Each request pattern creates different performance characteristics at the server. On the server computer, participants will see a web-based graphical user interface to interact with our framework, shown in Fig. 1.

The interface is divided into two sections. The first section, named "Request Patterns", is located in the upper part of the window and presents information on the current operating environment – in this case we show variations in client request patterns. The second section, named "Possible Architectures", presents a list of available web server architectures, and allows participants to select two different modes of "Autonomous" and "Manual". The bottom part of the interface also contains two buttons, one which switches between architecture and performance views, and the other which allows participants to upload new components to form new architecture variations.

The demonstration then presents two scenarios as follows:

Manual exploration: This scenario allows participants to manually perform all of the decision making that our framework provides, giving an insight into the level of complexity involved in this. When operating in this mode, the task of exploring the available architectures becomes the participant's responsibility. The participant can select from the list of available web server architectures and, for the currently selected architecture, will see a real-time graph of performance being drawn. This will show the average response time and other detailed information in a dialogue window. To fully explore the design space, the participant must try every possible architecture to gain an understanding of the performance characteristics of each one. The participant is then able to make an informed decision about which architecture best suits the current deployment environment. When the client request pattern changes (at the client computer), a new request pattern will be detected by our framework (shown at the top of the screen) and all currently gathered information will be considered invalid for that pattern. The participant will then need to explore the available architectures again for this new pattern, to build up enough knowledge to make a choice (potentially consulting the stored information from the previous request pattern when deciding which architectures to explore first). Overall, this scenario allows participants to actively experience the processes taken by the framework.

Autonomous exploration: In this scenario, the framework operates in fully autonomous mode, enabling the most suitable architectures to emerge as client request patterns change. This part of the demo uses the same graphical interface but shows the buttons being 'pressed' automatically as our framework explores the design space. Here the participants can observe the selections as they occur, along with monitored data that arrives, and adaptation rules that are learned over time for each identified pattern. Finally, the participants are able to upload new components to observe the framework learn about new architecture variations at runtime and watch the system creating new adaptation rules as a result.

Acknowledgment

Roberto Rodrigues Filho would like to thank his sponsor, CAPES, Brazil, for the scholarship grant BEX 13292/13-7. This work was partially supported by the UK's EPSRC under research grant number EP/M029603/1.

References

[1] A. Elkhodary, N. Esfahani, and S. Malek, "Fusion: A framework for engineering self-tuning self-adaptive software systems," in *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE '10. New York, NY, USA: ACM, 2010, pp. 7–16.

[2] D. Menasce, H. Gomaa, S. Malek, and J. Sousa, "SASSY: A Framework for Self-Architecting Service-Oriented Systems," *Software, IEEE*, vol. 28, no. 6, pp. 78–85, Nov 2011.

[3] A. Filieri, H. Hoffmann, and M. Maggio, "Automated design of self-adaptive software with control-theoretical formal guarantees," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 299–310.

[4] B. Chen, X. Peng, Y. Yu, B. Nuseibeh, and W. Zhao, "Self-adaptation through incremental generative model transformations at runtime," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 676–687.

[5] B. Porter, "Runtime modularity in complex structures: A component model for fine grained runtime adaptation," in *Component-Based Software Engineering*. ACM, June 2014, pp. 26–32.