FERMILAB-CONF-07-103-TD

A framework for constructing adaptive and reconfigurable systems

Pierre-Etienne Poirot, Jerzy Nogiec, and Shangping Ren

***Abstract*—This paper presents a software approach to augmenting existing real-time systems with self-adaptation capabilities. In this approach, based on the control loop paradigm commonly used in industrial control, self-adaptation is decomposed into observing system events, inferring necessary changes based on a system's functional model, and activating appropriate adaptation procedures. The solution adopts an architectural decomposition that emphasizes independence and separation of concerns. It encapsulates observation, modeling and correction into separate modules to allow for easier customization of the adaptive behavior and flexibility in selecting implementation technologies.**

***Index Terms*—control, adaptation.**

I. Introduction

Evolution in software systems has created the need for real-time systems to be self-adaptive and autonomous. Due to technological advances (ubiquitous networks, powerful microcontrollers), today's industrial systems are often immersed in complex and changing environments, which makes their construction, evolution and maintenance increasingly more challenging [1]. Notably, as human intervention in those environments can be costly and time-consuming, those systems are increasingly expected to be made self-adaptive and to handle internal errors, resource variability or changing user needs [4].

Additionally, as software control benefits from Moore's law in performance and sophistication, current systems are becoming prohibitively complex, to the point that their complexity is becoming a major problem [1]. As many control activities are increasingly depending on computers, it can not be expected that most users or organizations will have the means to deal with that complexity [2]. As a result, there is a strong incentive for these systems to become autonomous and self-adaptive in order to manage themselves efficiently and with minimal human intervention.

In this paper, we present a solution to extend existing systems with self-adaptation capabilities while maintaining the new features separated from the original functionality. An externalized adaptation scheme based on the observe-reason-modify paradigm is applied, which decomposes the adaptation process into three orthogonal subtasks: observation, modeling and correction. The approach itself is general and could be applied to a broad class of observable systems.

Manuscript received May 10, 2007; This work was supported by the U.S. Department of Energy.

P. E. Poirot and J. Nogiec are with the Technical Division, Fermi National Accelerator Laboratory, Batavia, IL, 60510.

P. E. Poirot and S. Ren are also with the Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616.

The rest of this paper is organized as follows: section II details related work with focus on architectural self-adaptation. Section III details our approach for automating reconfiguration and creating systems that can adapt to their runtime conditions. Section IV briefly outlines our prototype implementation. Finally, section V summarizes and concludes the paper.

II. Related work: Architecture-based self-adaptation

Most industrial systems are not designed to be self-adaptive, but rather to be extremely stable in order to operate reliably in a production environment. By design, in those systems, users may hardly customize the system or tailor its behavior at runtime. External self-adaptive approaches have been studied recently and have shown to be a practical solution for providing self-adaptation to these systems [4], [8], [7]. These approaches adopt traditional control theory that has been used and proved to be an effective solution [9] in hardware design and industrial automation. They place general adaptation mechanisms in separate modules that can be created, modified, extended and reused across various applications. Nevertheless, while the concept of a control feedback loop is simple, i.e., sensing (observing), calculating (reasoning) and acting (adapting), what to observe, how to reason and when to act are difficult to define and the decisions regarding the what, how and when play an important role in the success of external, feedback loop - based adaptation schemes.

The seminal Rainbow by Garlan and al. [4], [5] illustrates the challenges of architecture-based self-adaptation, such as latency and suitable decomposition. It provides generic self-adaptation through gauges (monitoring), a model manager, a constraint evaluator, an adaptation engine (reasoning), an adaptation executor and effectors (adapting). To use it, it requires an extensive knowledge of the system properties, constraint rules, adaptation strategies, and adaptation operators, split among multiple components. Other works in this area include [6], [7], [8], [9].

III. Extending existing systems with self-adaptation capability

Dissimilarly from the approaches mentioned above, our solution aims to streamline the adaptation process by creating a small, modular adaptation loop encapsulating adaptation behaviors into separate modules that can be created, modified and reused across multiple systems. Our solution emphasizes decomposition into three levels (observation, modeling and correction) that support independent specifications, while keeping the adaptation process external to the controlled system (Figure 1).

A. Control-loop architecture

As a solution, we employ an external feedback loop designed to externalize the adaptation logic out of the controlled system to allow for easier conception, maintenance and modification of the adaptive behaviors. In this approach, the system is monitored and compared to a system model that determines if or how the system should be adapted, in response to either internal behavioral changes or external stimuli. It differs from the ad hoc and static approaches that may often be used in practice, which are typically system-specific and restricted to a class of similar applications. Figure 1 illustrates our adaptive architecture.

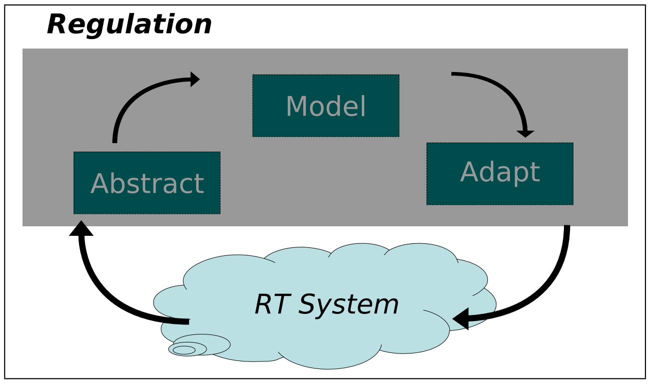

Fig. 1. Self-adaptation scheme

B. Decomposition

In our adaptation solution, each section of the framework is decomposed into independent components (event generators, inference engine, model evaluator and actuators) that encapsulate their design and implementation and work independently from the rest of the system. Called *modules*, these components communicate through standard interfaces that employ various technologies such as direct calls, JNI calls or interprocess communications (see section IV). Modules are designed, implemented and refined without regard to other parts of the framework and rely on standardized interactions, which makes them independent from the actual system and communications logic.

As shown in Figure 2, the approach employs a minimum of dimensions involved in adaptive behaviors: system observation, system modeling and system correction. These modules are separated, so that the adaptation loop can be substituted easily. In other words, different strategies or methodologies for observation, modeling and executing adaptation can be integrated based on the application needs. For instance, we use a knowledge-based system for observation, an event-driven Finite State Machine (FSM) for modeling the system adaptation and a multithreaded server to independently perform the system's automatic adaptation. Specific real-time applications can use different adaptation models (such as Specification and Description Language (SDL) models) for representing the running system, while having identical monitoring and adaptation modules. Such separation and independence not only enhance software reusability, but also, more importantly, provide additional dimensions of flexibility in choosing adaptation schemes and building self-adaptive applications.

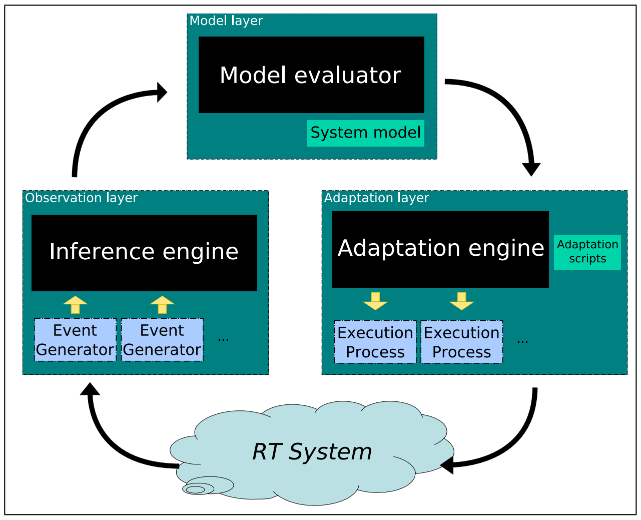

Fig. 2. Layers of self-adaptation

C. Observation layer

The observation layer consists of an interface to the controlled system, which allows for monitoring event traffic and introspecting component states, and an inference module, which *abstracts system-level events into model-domain events*, and hence abstracts the state of the controlled system. In our prototype implementation (see the next section), we use a knowledge-based system to analyze a stream of events and generate new information about the system. The purpose of the inference engine is to transform a stream of low-level monitoring events into higher-level state change events. For example, increasing message queues can be interpreted as network performance degradation and generate the network degradation event. The remainder of the adaptation loop works exclusively on the system model events, such as our example network performance degradation event, and is screened from the numerous low-level events coming from the controlled system. How this information is determined can be altered without disrupting the rest of the adaptation system.

D. Model layer

Following the information produced by the inference module, the model module *maps the system's activity onto a model of the system* to choose how or whether the application should adapt or reconfigure. The model engine adds an intermediate level between observation (such as a network degradation) and applied reaction (such as a higher compression rate). It evaluates the specified model to trigger the adaptation strategies.

In the current version, the adaptation model of the system is abstracted through an event-driven FSM that defines the

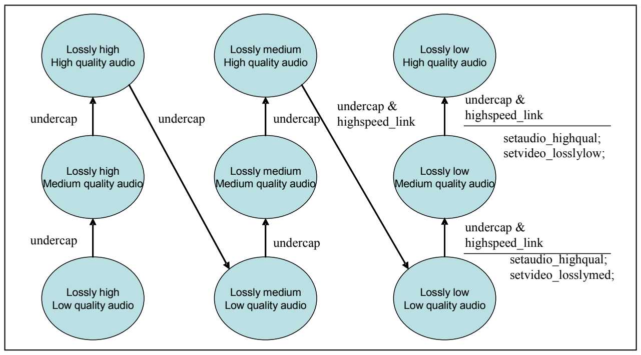

Fig. 3. Partial example of a model controlling a multimedia application. Adaptive behaviors and evolution of the software can be altered by changing the graph (`undercap` and `highspeedlink` are inferences from low-level events).

desired adaptive behavior of the system. This allows for sophisticated adaptive behaviors that are easily alterable and comprehensible. The states used in the FSM may differ from the functional model of the system, as various states of the system may not be considered by the adaptation. In our current design, the FSM produces *service requests* when it transitions between states in response to the events generated by the inference module (see figure 2).

E. Correction layer

The correction layer applies the service requests determined by the model module. The execution of a service request is hence decoupled from its selection. The correction module is designed to execute service requests while maintaining the system's consistency (such as flushing a subsystem before reordering the components in a dataflow subsystem [10]). Inspired by [4], [5], service requests are currently implemented as scripts operating on the target system. These scripts implement the activation procedures used to perform functional or structural adaptation.

The activation layer is designed to hide the specificity of the adaptation scripts (e.g. steps for changing compression algorithms) from the model of the system. This significantly simplifies the adaptive model of the system, as the details of these requests (e.g. switch to a higher compression rate) can be redefined later on without any changes to the adaptive model.

IV. Implementation

Currently, a prototype implementation of the proposed framework in Java is available. Relying on a design by interfaces, the current framework is centered around the concept of *events*, which are discrete tuples of information transiting between modules, and abstract *channels* that transfer events between modules. The framework includes an inference module based on Jess, a sophisticated knowledge engine developed at Sandia Lab [11], an event-driven FSM module to represent the system as a state machine, and a multithreaded adaptation module, to service and assign worker threads to execute adaptation scripts. In addition to the presented features, the framework also implements necessary utility classes (specialized channels, display GUI, etc.).

V. Conclusion

In this article, we presented a framework and methodology for adding an external control and adaptation capabilities to existing systems. The solution emphasizes separation of concerns and modularity of the adaptation process. Similarly to other efforts, its purpose is to design general mechanisms to develop systems that would adapt to their operating conditions and operate autonomously, and therefore fulfill the need for self-adaptive software expressed by the industry.

Our solution relies on previous work in external, control-based and model-based self-adaptation, but is distinguished by its decomposition of the adaptation process. The framework is decomposed into the observation, model and correction layers that are virtually independent from each other. Hence, it supports modularization and allows for the adaptation process to be customized for specific needs or application domains. The observation, model and correction modules can be substituted to fulfill a particular need. The framework relies on the first layer, the observation module, to infer information about the controlled system and allow adaptation decisions to be based on high level information rather than many low-level events. Then, the model module maps these observations onto an actual model of the system, which in turn permits situational adaptation. Finally, the correction module independently executes appropriate adaptation tasks and abstracts away the concerns necessary to realize the adaptation.

References

[1] A. G. Ganek and T. A. Corbi, "The dawning of the autonomic computing era" *IBM Systems Journal*, vol. Volume 42, Number 1, pp. 33–38, 2003.
[2] Microsoft, "Microsoft dynamic systems initiative: The drive to self-managing dynamic systems" 2005, http://www.microsoft.com/windowsserversystem/dsi/.
[3] J. O. Kephart and D.M. Chess. The vision of autonomic computing. *Computer*, Volume 36, Number 1:41–50, 2003.
[4] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-based self-adaptation with reusable infrastructure" *IEEE Computer*, vol. Volume 37, Issue 10, pp. 46–54, October 2004.
[5] D. Garlan and B. Schmerl, "Model-based adaptation for self-healing systems" in *WOSS '02: Proceedings of the first workshop on Self-healing systems*, New York, NY, USA, 2002, pp. 27–32.
[6] G. Karsai, Á. Lédeczi, J. Sztipanovits, G. Péceli, G. Simon, and T. Kovácsházy, "An approach to self-adaptive software based on supervisory control" in *IWSAS*, 2001, pp. 24–38.
[7] G. Kaiser, J. Parekh, P. Gross, and G. Valetto, "Kinesthetics eXtreme: An external infrastructure for monitoring distributed legacy systems" in *Autonomic Computing Workshop Fifth Annual International Workshop on Active Middleware Services (AMS'03)*, 2003.
[8] Y. Qun, Y. Xian-Chun, and X. Man-Wu, "A framework for dynamic software architecture-based self-healing" *SIGSOFT Softw. Eng. Notes*, vol. 30, no. 4, pp. 1–4, 2005.
[9] Y. Diao, J. Hellerstein, S. Parekh, R. Griffith, G. Kaiser, and D. Phung, "Self-managing systems: A control theory foundation" in *12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS '05)*, 2005.
[10] D. J. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik, "Aurora: a new model and architecture for data stream management" *The VLDB Journal*, vol. 12, no. 2, pp. 120–139, 2003.
[11] S. N. L. Ernest Friedman-Hill, "Jess" http://herzberg.ca.sandia.gov/jess/.