

The Making of SimEAC

David Chisnall
University of Wales Swansea
United Kingdom
Email: csdavec@swan.ac.uk

Min Chen
University of Wales Swansea
United Kingdom
Email: m.chen@swansea.ac.uk

Abstract—In this short paper, we give a brief overview a simulation environment, SimEAC, which was designed for testing algorithms with strong autonomic features. SimEAC enables a user to configure a system infrastructure to be simulated by specifying a collection of hardware attributes, and it provides a fine degree of control over operating system and task simulation, hence allowing a variety of autonomic algorithms to be prototyped on a virtual system infrastructure.

I. INTRODUCTION

The development of an autonomic system will inevitably involve the design, test, verification, and optimization of a collection of autonomic elements. As pointed out by [1], there is a significant engineering challenge in the development of such elements for large-scale systems, because:

- It is hard to anticipate the combined effects of many autonomic elements when they interact with the system, users, and one another.
- It is difficult to create a variety of possible scenarios in which autonomic elements are expected to function, and to recreate a particular condition for the purpose of testing and optimization.
- It is risky to install any autonomic element that has not been through an adequate engineering process.
- It is costly to provide a replica of the live system in order to engineer autonomic elements.

This lead to the necessity for a development environment for support the engineering of autonomic elements. The core of an autonomic element usually consists of one or a few algorithms that encode strategies, methods and operations for self-management in response to the dynamic environment where it functions. In this work, we focused on the provision of support for the development of such algorithms through a simulation environment, SimEAC (Simulated Environment for Autonomic Computing).

SimEAC enables a system designer to create a system infrastructure to be simulated by specifying a collection of hardware attributes, and it provides a fine degree of control over operating system and task simulation, hence allowing new autonomic algorithms to be prototyped, tested and optimized on a virtual system infrastructure. SimEAC supports modeling and simulating a variety of system architectures, including large scale networked systems, in a relatively abstract manner. It provides a user interface for configuring a virtual system infrastructure to be simulated, and for specifying the statistic and stochastic behavior of the system and running tasks,

including system failures and dynamic job loads. SimEAC allows an algorithm designer to conduct experimentation on different virtual architectures without the risks of bringing down a service system, and enables more scientific evaluation of autonomic algorithms in analyzing typical attributes of an autonomic system such as scalability and problem localization.

We have also conducted a few case studies to demonstrate the use of some key features of SimEAC. One case study involves a simulation of a self-organization algorithm for managing distributed computation in an ad hoc network. Another is a simulation of an agent-driven resource manager in a virtual cluster-based visualization environment.

II. MOTIVATION AND REQUIREMENTS

Autonomic computing is a relatively new research area, and for the last a few years, there have been proposals for new architectural designs (e.g., [2]) and programming environments (e.g.,[3]). However, many development tools required for building a medium or large autonomic system are not yet available. One of such tools would be for algorithm simulation, which plays a vital part in a development process and enables us to determine how well an autonomic algorithm would work without the risks and limits of testing it on a live system. Here, we use the term *algorithm* in a more general sense, encompassing a variety of software components that encode strategies, methods, event-driven operations, decision processes, optimization steps, and so on, but excluding components that are implementation-dependent, such as specific APIs.

In the context of autonomic computing, we consider the following set of requirements for SimEAC, which should be able to:

- **operate at an abstract level** allowing cost-effective coarse-grain specification and simulation of large-scale infrastructure.
- **support complex systems** allowing the evaluation of algorithms in simulated environments that can be large, ubiquitous, interconnected and/or heterogeneous.
- **support dynamic system behaviors** allowing the simulation of temporal events such as system failures and load changes, and hence enabling evaluation of event-driven autonomic algorithms.
- **be dynamically re-configurable** allowing the configuration and parameters of a virtual system be modified

during simulation, and enabling observation of the effects of various self-management strategies.

- **provide an intuitive API** allowing quick and easy algorithm prototyping.
- **have an extensible design** allowing easy addition of new simulation models to accommodate future directions of autonomic research.

III. AN ABSTRACT MODEL

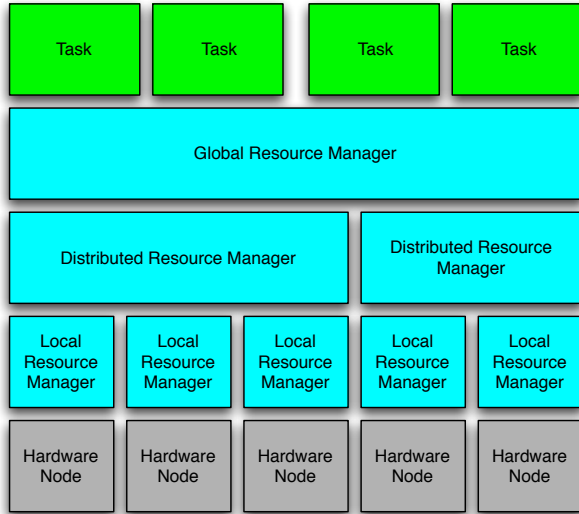


Fig. 1. SimEAC architectural overview.

SimEAC is constructed in three layers as shown in Figure 1. The first layer (i.e., the lowest) describes the capabilities of the hardware, including connections between individual hardware nodes. The second layer contains resource managers. These allocate access to hardware resources to tasks in the third layer. The resource managers can be arranged hierarchically, and used to model operating systems and autonomic middleware.

IV. APPLICATION PROGRAMMING INTERFACES

The architectural layout of a simulated system is defined in XML. XML is particularly well suited to this task since it naturally represents hierarchical structures. These XML specifications can either be created manually or via the user interface of SimEAC.

Resource managers and tasks are implemented in Objective-C. Each one is written as a class conforming to a formal protocol. To aid users in specifying resource managers and tasks with or without programming, a number of built-in classes have been provided, which perform some commonly required functionality. Adding a new task to the system requires nothing more than writing and compiling the class — it can then be added to an XML description file either manually or via the GUI of SimEAC (Figure 2).

The simulation engine runs using discreet time intervals, known internally as ticks. The size of each tick is controlled by the user — shorter ticks produce a more accurate simulation

at the expense of requiring more processor run time for the simulator.

Most interaction between components in the system takes place through a message passing system. Accesses to storage devices and communication between tasks, for example, are both accomplished via message passing.

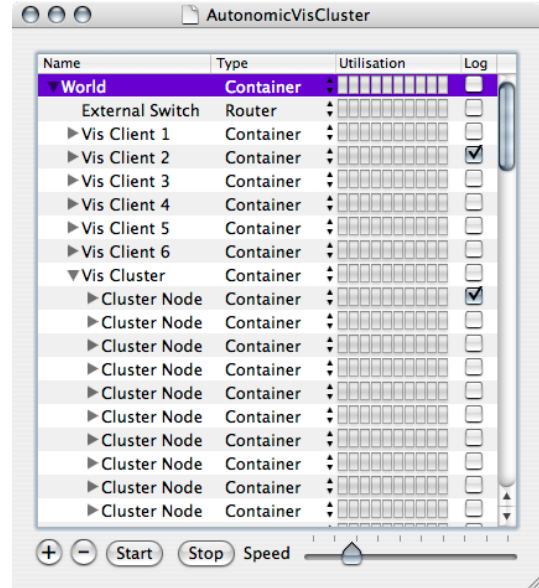


Fig. 2. SimEAC user interface.

V. SUMMARY AND CONCLUSIONS

SimEAC is a versatile platform to simulating autonomic algorithms. It can be used to evaluate and optimize the designs of the core of autonomic elements without the expense (in terms of both finance and time) and risks of testing such elements on a live system in service.

SimEAC is capable of simulating widely different system architectures and tasks within the same framework. With the aid of its abstract specification of hardware, its XML schema for building a virtual infrastructure, its GUI for customizing built-in resource managers and tasks, and its API for defining more complex ones, the user can carry out a simulation task with very little effort, especially in comparison with the complexity involved in constructing a testing environment, perhaps anything other than a straightforward PC. We are in the process of making SimEAC available to the autonomic research community.

REFERENCES

- [1] J. O. Kephart and D. M. Chess, “The vision of autonomic computing,” *IEEE Computer*, pp. 41–50, 2003.
- [2] S. R. White, J. E. Hanson, I. Whalley, D. M. Chess, and J. O. Kephart, “An architectural approach to autonomic computing,” in *Proc. 1st Autonomic Computing*, 2004, pp. 2–9.
- [3] H. Liu, M. Parashar, and S. Hariri, “A component-based programming model for autonomic applications,” in *Proc. 1st Autonomic Computing*, 2004, pp. 10–17.