Work in Progress: Availability-Aware Self-Configuration in Autonomic Systems

David M. Chess, Vibhore Kumar, Alla Segal, and Ian Whalley

IBM Thomas J. Watson Research Center, P.O. Box 704,
Yorktown Heights, NY 10598, USA
{chess,vibhore,segal,inw}@us.ibm.com

Abstract. The Unity project is a prototype autonomic system demonstrating and validating a number of ideas about self-managing computing systems. We are currently working to enhance the self-configuring and self-optimizing aspects of the system by incorporating the notion of component availability into the system's policies, and into its models of itself.

1 Introduction

The vision of autonomic computing is of a world where complex distributed systems manage themselves to a far greater extent than they do today [1]. The Unity project [2] is a prototype autonomic system, designed to develop and validate various ideas about how this self-management can be achieved in practice. The aspects of self-management that we explore in Unity include self-configuration and self-optimization; the system initially configures parts of itself with a minimal amount of explicit human input, and during operation it reallocates and reconfigures certain of its resources to optimize its behavior according to human-specified policies.

In our current research, we are investigating ways to enhance the self-configuration and self-optimization aspects of Unity by incorporating the concept of availability into the system's policies and models of itself. The notions of availability and related concepts that we employ here are essentially those of [3]; availability is the condition of readiness for correct service.

(Space does not allow a significant list of previous work relevant to this project; the reader is invited to consult references such as [4].)

1.1 Availability in the Present System

In the current Unity system, availability is implicitly supported in one way: one component of the system (the policy repository) is implemented as a self-healing cluster of synchronized services; other components of the system ensure that if one member of the cluster goes down, another service instance is brought up and enters the cluster to take its place.

A. Sahai and F. Wu (Eds.): DSOM 2004, LNCS 3278, pp. 257-258, 2004.
© IFIP International Federation for Information Processing 2004

2 Stages of Availability Awareness

The first and simplest enhancement we plan to make will involve moving the constant representing the number of clustered copies of the repository to create, from its current location in a configuration file into a simple policy in the system's policy infrastructure. While this still does not involve the system in any availability-related calculations, it will at least leverage the commonality and deployment abilities of the policy subsystem.

In the second stage, we will replace the constant number with a desired availability target, representing a required percentage of uptime. We will then do a straightforward calculation based on MTBF and MTTR estimates for the repository component, to determine the initial size of the repository cluster. (Estimating the proper MTBF and MTTR figures from logs and other data will be addressed in a related project.)

In the third stage, we will replace the static required availability target with a service-level utility function as defined in [5], representing the business value of various levels of repository availability. Together with a similar representation of the cost impact of running additional repositories (including the impact on the rest of the system of devoting resources to those repositories), this will allow the system to calculate the optimum number of repositories for the given utility and cost functions.

The third stage requires obtaining the utility and cost functions from outside the system. In the fourth and final stage of this design, we will enrich the system's model of itself sufficiently to derive those functions from the model (and from the higher-level utility functions describing the business value of the system's overall behavior). The system (or more accurately, the solution manager component which determines and maintains the composition of the system) will use its model of the system's behaviors to estimate the value of repository availability (in terms of the impact of having the repository unavailable on the overall value produced) and the cost of running multiple repositories, and use these functions to do the calculations as in the third stage.

We solicit communication from others investigating similar problems or related technologies in this area.

References

1. Kephart, J., Chess, D.: "The Vision of Autonomic Computing", IEEE Computer 36(1): 41-50, 2003.
2. Chess, D., Segal, A., Whalley, I., White, S.: "Unity: Experiences with a Prototype Autonomic Computing System", International Conference on Autonomic Computing (ICAC-04), 2004.
3. Avizienis, A., Laprie, J-C., Randell, B.: "Fundamental Concepts of Dependability," Research Report N01145, LAAS-CNRS, April 2001.
4. Marcus, E., Stern, H.: "Blueprints for High Availability: Designing Resilient Distributed Systems", John Wiley & Sons, 1st Edition, January 31, 2000.
5. Walsh, W., Tesauro, G., Kephart, J., Das, R.: "Utility Functions in Autonomic Systems," International Conference on Autonomic Computing (ICAC-04), 2004.