Dynamic Adaptation of Distributed Systems

Francisco José da Silva e Silva*
Federal University of Maranhão, Brazil
fssilva@ime.usp.br

Markus Endler
PUC-Rio, Brazil
endler@inf.puc-rio.br

Fabio Kon
University of São Paulo, Brazil
kon@ime.usp.br

Abstract

Adaptation techniques allow software to modify its own functions and configuration in response to changes in its execution environment. In order to incorporate an adaptable behavior, an application must have a way to monitor the execution environment and discover relevant variations that requires reconfiguration.

The developer of an adaptive software must take care of several issues such as the selection of what elements of the environment should be monitored and how to perform the monitoring, what adaptations of the software behavior should be performed and when they should take place.

Our work aims at addressing the problem of how to change computing systems at runtime in order to adapt to variations in the execution environment, such as resource availability and user mobility. We present a generic framework to support the construction of adaptive distributed applications. It provides a set of integrated tools for environment monitoring, detecting events and reconfiguring distributed applications.

1 Introduction

Adaptation techniques allow software to modify its own functions and configuration in response to changes in its execution environment. It has a number of potential benefits such as the ability to respond rapidly to security threats and the opportunity to optimize performance as changes in the execution environment takes place [2]. They are particularly useful in distributed and mobile computing because of the intrinsic characteristics that make their execution environment susceptible to great variations such as high bandwidth variability, variable error rates, eventual disconnections, hardware and software heterogeneity, variable resource availability and user mobility [11, 5, 12].

In order to incorporate an adaptive behavior, an application must have a way to monitor the execution environment and discover relevant variations that require reconfiguration. The developer of an adaptive software must take care of several issues such as the selection of what elements of the environment should be monitored and how to perform the monitoring, what adaptations of the software behavior should be performed and when they should take place.

*PhD student, Department of Computer Science, University of São Paulo, Brazil

2 Main Objective and Related Work

Our work aims at addressing the problem of how to change computing systems at runtime in order to adapt to variations in the execution environment, such as resource availability and user mobility. Our main interest is on adapting distributed information systems composed of several components spread over a network where data are also decentralized and users are mobile.

The problem of how to provide dynamic adaptation of software has been addressed recently by several research groups. Sudame and Badrinath [13] describe a mechanism based on ICMP messages [3] to propagate information about network environment parameters such as the cost of communication, latency, and bandwidth. Mobility specific parameters, such as signal strength, location, energy consumption and availability can also be addressed by the described mechanism. Once variations on network parameters are detected, actions that adjust parameters of the transport layer protocol (TCP or UDP) can be automatically triggered. There are a fixed predefined set of actions the system can take. Sudame and Badrinath [13] do not address the problems of dynamic reconfiguration of system components neither the monitoring of inter-component interactions. It focuses on adapting transport layer protocol parameters and not on generic system adaptations. Finally, the monitoring and adaptation are mainly local to a host and the work does not address the adaptation of a system composed of several components distributed over a network.

Fox et al. [6] address the problem of network bandwidth and client variability with respect to both hardware and software properties. Adaptation is achieved via data-type-specific lossy compression. Compression is performed by *distillers*, which preserve most of the semantic content of the data object. A proxy is placed between the client and the server. Its role is to retrieve content from Internet servers on the client's behalf, determine the high-level types of the various components (e.g., images, text runs), and determine which distillation engine must be employed to adapt the data to the current bandwidth availability and client type. Fox et al. focus their work on applications composed of a client and a server, where a proxy is placed between them. In our work, the application is composed of several components spread over a network. Fox's proxy adaptation is based on data-type-specific transformations; other adaptation mechanisms, such as component reconfiguration and relocation are not considered. We believe that component migration, replication and reconfiguration are essential to support user mobility and to cope with resource availability fluctuations.

Odyssey [9] addresses the problem of providing information access to mobile computers. Its approach is different from Fox's proxies but it uses the same principle of adjusting the data according to the environment status. It is implemented as a set of extensions to the NetBSD operating system. In Odyssey's view, the operating system is responsible for monitoring resource availability, notifying applications of relevant changes to those resources, and enforcing resource allocation decisions. Each application is responsible for deciding how to exploit available resources by selecting the fidelity level of the transmitted data. As with Fox et al., an Odyssey application is divided on a client and server components. The adaptation is based on choosing between different versions of the data (fidelity levels) in order to match the resource availability. It lacks other adaptation mechanisms such as component reconfiguration and relocation.

Chang and Karamcheti [1] describe an adaptation framework that eliminates the need for adaptation decisions to be explicitly programmed into the application. The framework is based on two components: a *tunability interface* and a *virtual execution environment*. The tunability interface provides language support to express availability of multiple execution paths for the application, and the means by which application progress can be monitored and influenced (by switching to a different path). A driver script repeatedly executes the application in virtual environments, obtaining application behavior profiles. The execution system incorporates these profiles into adaptation decisions at run time to choose the appropriate execution path that the application must take in order to satisfy user-defined preferences, such as transmission time. All the different execution paths (algorithms) are coded into a single, monolithic execution process together with the monitoring and steering agents. This approach is not well suited to our needs since we must consider resource availability and user request load in multiple nodes in order to determine the best adaptation action to be triggered. Application reconfiguration is supported (by choosing between different execution paths) but migration and replication of components are not considered.

3 Our Approach

Considering the limitations of the adaptation mechanisms described above, we focused our work in the development of a generic framework to support the construction of adaptive distributed applications. We began by designing an object oriented model to serve as the basis for the implementation of such framework. The framework is intended to be flexible enough to be used in a wide range of adaptive distributed applications.

The model is composed of three main packages. The first is responsible for monitoring the entities present in the system. The data collected by the *monitoring package* is stored in logs or sent directly to the *event detection and notification package.* The latter is responsible for analyzing that data and determining the occurrence of events in which there is interest. These events are sent to the *dynamic reconfiguration package*, which is responsible for applying the required reconfiguration actions to adapt the system.

3.1 Monitoring Package

The monitoring objective is to collect sufficient data to keep track of the environmental state in which the application is running. The analysis of these data will determine when reconfiguration should take place and what are the reconfiguration actions that should be applied to adapt the application to the changes in environmental state.

One important aspect of the execution environment to monitor is the status of distributed resources, such as memory, CPU, disk, and network connectivity. To implement such monitoring functionality we use *resource monitoring objects*, which are responsible for monitoring system specific parameters, such as processor utilization, in the various hosts of the distributed system. The resource monitoring objects notify the event detection and notification package whenever the value of such parameters changes significantly.

In addition to resource monitoring it is also important to monitor the *interactions* between the objects that compose the distributed application. We intend to use *Interceptors*, as defined in the CORBA standard [10] Interceptors are inserted into the object invocation path. Each time a client invokes a method of an object, the message corresponding to this invocation is intercepted and later re-dispatched to the target object. Using interceptors, the system can extract useful information from each method invocation, storing it in a log for analysis by the event detection and notification package. Furthermore, each request can be time-stamped so that one can know when a message invocation request was received at the server side. The system can also measure the time between the reception of the message at the server and the dispatching of the response to the client, which corresponds to the duration of the method execution at the server.

3.2 Event Detection and Notification Package

The event evaluation package is responsible for analyzing the data stored in the logs created by the monitoring package and determine the occurrence of events that are relevant to the application. The event evaluation also receives notifications from the resource monitoring objects about changes in resource utilization. By collecting and combining these messages and evaluating user-defined expressions, the event evaluation can determine the occurrence of key events in the distributed system that will trigger the adaptation process. When the event evaluation detects the occurrence of an event, it sends a message to the event notification package, which is responsible for delivering the event to the objects that had registered interest in it.

3.3 Distributed Dynamic Reconfiguration Package

In order to provide the necessary mechanism to trigger the adaptation processes, we are extending the functionality of the *Component Configurator* [8, 7] that defines an architecture for reconfiguring distributed applications. For each application component, we instantiate a component configurator object. The configurator keeps track of the dynamic dependencies between the component and others system and application components. By maintaining an explicit representation of those dependencies we can guarantee runtime consistency. For example, if one component fails, the components that depends on it can be notified in order to promote a graceful exit. Component Configurators cooperate in order to trigger the necessary actions to dynamically reconfigure the application.

We extend the original component configurator architecture with an environment-aware thread that registers itself with the event detection and notification package. This thread receives the notification of events that indicates relevant environmental changes. Depending on the type of the event, it executes the appropriate algorithm that defines the actions that should be taken in order to adapt the application to the new environment state.

Using this approach we achieve two primary project requirements:

- A clear separation of concerns between the application functionality and the adaptation processes. All the code necessary to make the application aware of the execution environment as well as the code that defines the reconfiguration actions that should be taken in order to respond to a significant environment variation are encapsulated inside the component configurator;
- Safe reconfiguration. By maintaining an explicit representation of the prerequisites and dynamic dependencies in the distributed system, the component configuration acquires the necessary knowledge to reconfigure the system in a safe and consistent way.

3.4 Framework Validation

In order to validate the framework we are using a prototype derived from the SIDAM project [4] as a case study. SIDAM stands for Distributed Information Systems for Mobile Agents. This research aims at studying the main problems related to the design and implementation of decentralized information systems for mobile users. The SIDAM reference application is a distributed information service disseminating road traffic information for a large metropolis.

The SIDAM reference application is a interesting example of a highly dynamic and distributed application since the generation, storage and query of information are decentralized and clients are mobile. In this case, adaptation becomes essential for achieving high service availability, dynamic load balancing, and short response times for clients.

In order to keep the size of the information base and the corresponding queries manageable, the SIDAM application divides the city into micro-regions (e.g., a few blocks) that are the atomic units holding traffic information. The micro-regions traffic data are stored at *Information Servers* (IS). Each information server can hold data from one or more micro-regions. Information servers reside at *Data Servers*, which are fixed hosts.

In each data server, we instantiate resource monitoring objects in order to monitor the amount of CPU used and main memory available. We also defined an event, called DataServerOverload. It is trigged when (1) the CPU utilization in one data server is above 80% and (2) the amount of main memory available is bellow a given value and that condition holds for at least 15 seconds. For each SIDAM information server, we instantiate a component configurator. When it receives a notification that one data server is overloaded, an algorithm is executed. This algorithm determines one information server from the overloaded host to be migrated to another machine. This action is then realized with the help of the component configurators.

We are currently implementing adaptations of the SIDAM prototype triggered by the analysis of the interactions between the objects that compose the application. One example is the analysis of what micro-regions are more frequently accessed. Using this information we can determine the best distribution of micro-regions across the information servers.

4 Conclusions

Incorporating adaptive behavior in a distributed application is a complex task. The framework presented here aims at simplifying this task through the use of object oriented technology. It provides a set of integrated tools for environment monitoring, detecting events and reconfiguring distributed applications.

References

[1] Fangzhe Chang and Vijay Karamcheti. Automatic configuration and run-time adaptation of distributed applications. In IEEE Computer Society, editor, *Ninth IEEE International Symposium on High Performance Distributed Computing*, Pittsburg, Pennsylvania, aug 2000.

[2] W.-K. Chen, M. Hiltunen, and R. Schlichting. Constructing adaptive software in distributed systems. In *Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS-01)*, pages 635–643, Los Alamitos, CA, April 16–19 2001. IEEE Computer Society.

[3] Douglas E. *Internetworking with TCP/IP: Principles, Protocols, and Architectures*, volume 1. Prentice Hall, 4th edition, 2000. Chapter 9. Internet Protocol: Error and Control Messages (ICMP).

[4] M. Endler, D.M. da Silva, F. Silva e Silva, R.A. da Rocha, and M.A. de Moura. Project SIDAM: Overview and Preliminary Results. In *Anais do 2o. Workshop de Comunicação sem Fio (WCSF), Belo Horizonte*, May 2000.

[5] G. Forman and J. Zahorjan. The challenges of mobile computing. *IEEE Computer*, abr 1994.

[6] A. Fox, S. Gribble, Y. Chawathe, and E. Brewer. Adapting to network and client variation using active proxies: Lessons and perspectives. *special issue of IEEE Personal Communications on Adaptation*, 1998.

[7] Fabio Kon. *Automatic Configuration of Component-Based Distributed Systems*. PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, May 2000.

[8] Fabio Kon and Roy H. Campbell. Dependence management in component-based distributed systems. *IEEE Concurrency*, 8(1), January/March 2000.

[9] B. D. Noble and M. Satyanarayanan. Experience with adaptive mobile applications in odyssey. *Mobile Networks and Applications*, 4, 1999.

[10] OMG - Object Management Group. *The Common Object Request Broker: Architecture and Specification*, dec 2001. version 2.6.

[11] Evaggelia Pitoura and George Samaras. *Data Management for Mobile Computing*. Kluwer Academic Publisher, 1998.

[12] M. Satyanarayanan. Fundamental challenges in mobile computing. In *In Fifteenth ACM Symposium on Principles of Distributed Computing*, Philadelphia, PA, May 1996.

[13] P. Sudame and B. Badrinath. On providing support for protocol adaptation in mobile wireless networks. Technical report, Department of Computer Science, Rutgers Universit, jun 1997. http://www.cs.rutgers.edu/pub/technical-reports/dcstr -333.ps.Z.