A Framework for Building Adaptive Distributed Applications *

Francisco Jos da Silva e Silva†
Federal University of Maranhão
fssilva@ime.usp.br
Markus Endler
PUC-Rio
endler@inf.puc-rio.br
Fabio Kon
University of São Paulo
kon@ime.usp.br

Abstract

Building adaptive applications is a complex and challenging task. Developers must consider several issues in addition to the implementation of the application functional behavior. This work aims at reducing the complexity of building adaptive distributed applications by presenting a framework that provides a set of integrated tools for monitoring resource usage as well as application-specific interactions, detecting environmental changes, and reconfiguring distributed applications. This paper describes the framework model and its implementation. We also present experimental results that show that besides detecting resources availability fluctuations it is also important to monitor the interactions between application components, looking for patterns that indicate the necessity to perform dynamic adaptation.

1 Introduction

A self-adaptive software should be able to modify its own behavior at runtime to adapt to changes in its execution environment, in order to maintain the required level of performance or even enhance it. Adaptation is particularly useful for distributed, mobile, and ubiquitous systems because of the dynamic nature of these environments. This dynamism can be observed as large variations in resource availability and error rates, in frequent disconnections, hardware and software heterogeneity, and user mobility [Helal].

Dynamic adaptation requires software that can monitor its execution environment to detect when significant change has happened and then reconfigure itself to adapt to this change. The developer of an adaptive application must take into account several issues besides the implementation of the application's functional behavior. These issues include the selection of which environment elements must be monitored, how to perform the monitoring, how to detect relevant environmental changes, which software adaptations should be carried out and when should they take place.

Our work aims at addressing the complexity of building adaptive distributed applications and systems by presenting a framework that provides a set of integrated tools for monitoring the environment, detecting events, and reconfiguring distributed applications dynamically. The framework supports the construction of distributed adaptive applications, composed of several components spread over a network. As a proof of concept, we used the framework to add adaptive behavior to a prototype implementation of an information service for mobile users. With this experience, we collected evidence that, besides detecting variations in resource availability, it is also important to monitor the interactions among application components, searching for patterns that indicate the necessity to perform dynamic adaptation.

*supported by CNPq, CAPES and FAPESP.
†PhD student of Computer Science at USP.

2 Related Work

Fox et al. [Fox] address the problem of adapting client/server applications, such as Web browsing, to variations in network bandwidth and the properties of client devices. Adaptation is achieved via data-type-specific lossy compression. Fox et al. focus their work on applications composed of a client and a server, where a proxy is placed between them. We followed a more generic approach, considering the case of applications composed of several components spread over a network. Fox's proxy adaptation is based on data-type-specific transformations. We believe that other adaptation mechanisms, such as switching to different algorithms as the environment changes, and component reconfiguration, migration, and replication, are complementary changes which are able to deal with a wider range of adaptive applications. Our framework follows a more flexible approach, allowing a choice of the form of adaptation that best suit the application.

Moura et al. [Moura] present an infrastructure that simplifies the development of adaptive distributed applications through a set of mechanisms that allow applications to dynamically select the components that best suit their requirements, to verify whether the system is satisfying these requirements, and to eventually react to variations in the nonfunctional properties of the used services. They use the CORBA Trading Service to support dynamic component selection an have implemented an extensible monitoring facility that supports monitoring of dynamically defined requirements. Moura et al. does not address the problem of how to coordinate reconfiguration actions between application components, an important aspect of highly distributed adaptive applications.

The Quality Objects (QuO) [Vanegas] project offers a framework for creating applications that adapt to different Quality of Service (QoS) guarantees offered by the resources. QuO provides tools that allows the developer to specify the level of service that the application desires, the behavior alternatives and strategies for adapting to changing levels of service, and to measure and control system resources. QuO also lacks a mechanism to coordinate reconfiguration actions between distributed application components. QuO, as well as the other research projects described in this section, does not explore the analysis of the interactions between application components as a clue to determine when adaptation should be triggered.

3 Framework Architecture

Our framework model satisfies the following design requirements.

- The framework should be **generic** in order to be used for the development of a wide range of distributed applications.
- It should promote a clear **separation of concerns** between the application's functional code and the adaptation-specific code.
- It should support the development of highly distributed applications, which implies providing mechanisms to **coordinate reconfiguration actions** between distributed components.
- Framework components must be **flexible and extensible** in order to accommodate specific applications needs.
- Framework components to be shared between applications at runtime must **support concurrent execution** of adaptive applications.
- Framework components must provide a **management interface** to allow the user to configure and customize its execution parameters.

Figure 1 shows the framework UML package diagram, which is composed of three high-level packages. The `Monitoring` package is responsible for monitoring the application execution environment. The data collected by the components of this package are saved in persistent storage or sent directly to the `Event Detection and Notification` package. The latter is responsible for analyzing these data and determining the occurrence of relevant events which are of interest for adaptation decision-making. These events are then sent to the `Dynamic Reconfiguration` package, which is responsible for applying the required reconfiguration actions to adapt the system.

The framework is composed of a collection of CORBA objects and services. Our implementation comprises 21,852 lines of Java code and uses JacORB as the CORBA ORB. The framework is available for download as

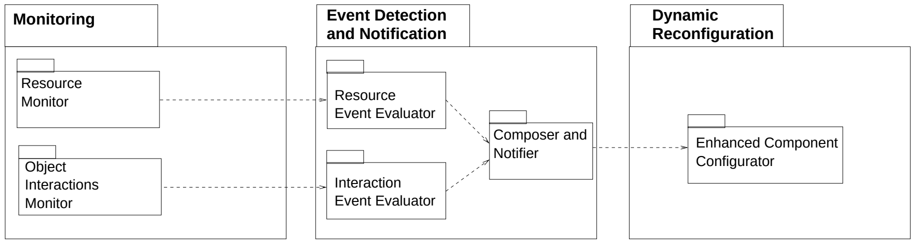

Figure 1: Framework UML Package Diagram

open-source software at `http://www.ime.usp.br/~sidam/software/AdaptationFramework`.

Monitoring Package: The main goal of monitoring is to collect sufficient data to keep track of the state of the execution environment (e.g. the network nodes and links) in which the application is running. Analysis of these data will determine when reconfiguration is required and which are the reconfiguration actions that should be applied to adapt the application to the recent changes in the environment.

Monitoring Distributed Resources: The `Resource Monitor` sub-package monitors the status of distributed resources that normally corresponds to physical resources such as memory, CPU, disk, and network links, but it can also monitor software parameters, such as the number of open threads in a server. To implement such monitoring functionality, we use *Resource Monitoring Objects (RMO)*. Each RMO monitors a specific Monitoring Parameter and notifies the `Event Detection` package whenever there is a change of the operation range of the parameter being monitored.

An example of an application that uses monitoring of distributed resources to adapt its behavior to improve its performance, could be a Web server that converts color images to black-and-white if the available bandwidth to the client drops bellow a given operation range. If the bandwidth drops even more, the client could simply omit the figures.

Monitoring Interactions Among Application Components: The `Object Interactions Monitor` sub-package uses software components called *Interceptors*, which are inserted into the object invocation path, in order to monitor the interactions between the components that comprise the distributed application. Using interceptors, the framework can extract useful information from each method invocation and record it (in persistent storage) for later analysis by the `Event Detection and Notification` package. Interceptors can store different kinds of information from each method invocation, such as the name of the called method, the request timestamp, the IP of the machines that were executing the client and server objects, and the invocation completion time. Besides this generic information, in several cases, it is also useful to store application-specific data, such as the method arguments. Our Interceptor implementation can be extended in order to incorporate application-specific needs in this way.

As an example where interaction monitoring is necessary, consider an distributed information service for mobile users. Interceptors could store the physical location of the clients when performing requests and the system could use this information to determine a better distribution of data in order to minimize response time to clients.

Event Detection and Notification Package: The `Event Detection and Notification` packages are used to detect relevant changes on the environment state, based on the analysis of the data collected by the `Monitoring` package. Our framework follows an event-driven approach to notify applications about relevant environmental changes. The framework provides support for two types of events: *Resource Events*, which are triggered when there is a significant change on the state of the distributed resources being monitored; and *Interaction Events*, which are triggered depending on the patterns discovered by the analysis of the data collected by the `Object Interactions Monitor` subpackage.

Since each adaptive application has interest on a particular set of events and this package works as a stand-alone service in the distributed system, a major requirement for this package is the ability to extend and modify dynamically the set of events that can be detected. This is done by allowing the user to provide new event definitions that are dynamically loaded into the system by using reflective Java mechanisms. We also allow the replacement of event definitions without interrupting the event detection service.

Dynamic Reconfiguration Package: The main goal of the Dynamic Reconfiguration package is to determine which adaptive action(s) should be performed when a significant change on the environmental state occurs and to support the application of those actions in a safe and consistent way.

In order to provide the required mechanism to trigger the reconfiguration, we extended the functionality of the Component Configurator [Kon], which defines an architecture for reconfiguring distributed applications. Each application component has a corresponding Component Configurator object. The Configurator keeps track of the dynamic dependencies between the component and other system or application components. By maintaining an explicit representation of those dependencies, it is possible to guarantee runtime consistency. Component Configurators are also responsible for disseminating events across inter-dependent components. This communication mechanism is used in order to coordinate reconfiguration actions between the application components.

We extended the original component configurator architecture with an environment-aware thread that registers itself with the notification channel provided by the `Event Detection and Notification` package. This thread receives the notification of events that indicate relevant environmental changes. Depending on the type of the event, it executes the appropriate actions required to adapt the application to the new environment state, using the Component Configurators to coordinate reconfiguration actions between the application components. The framework organizes the code that treats each environment event as a set of strategies, using the Strategy Pattern [Gamma].

4 Experimental Results

In order to validate the framework we used as a case study a prototype of the SIDAM project [Endler]. The application is a distributed information service disseminating information about traffic conditions in a large metropolis. In order to keep the size of the information base and the corresponding queries manageable, the SIDAM application divides the city into micro-regions (a few blocks), which are the atomic units holding traffic information. Traffic data of a micro-region is stored at *Information Servers*. Each information server can hold data from one or more micro-regions. Information servers reside at fixed hosts that are connected with each other through a wired and reliable network and information can be queried by mobile users. All communications to and from a wireless mobile host are intermediated by base stations.

Simulated Scenarios We have evaluated our framework in a set of simulated scenarios. For each simulated scenario we varied the amount of requests originated by each city region as well as the frequency on which this pattern of access changes. We have implemented the interception of calls made to the `queryInfo()` method, provided by the Information Servers. This method returns the traffic situation in a specific city region. By analyzing the monitored data, we could determine the amount of requests performed for each Information Server grouped by the city region where the mobile clients were when the request was made. This knowledge is used to choose a better distribution of the Information Servers across the network of fixed hosts. The main adaptation is the relocation of Information Servers to a host which is nearer to most of its clients, aiming at reducing the access latency.

The experimental results indicated that the adaptation mechanisms incorporated into the application caused a significant enhancement of the performance gains. For some simulated scenarios, the use of the non-adaptive version of the application resulted in a mean response time 132,92% greater than the adaptive one. During our tests we also noticed the importance of adjusting a framework parameter that determines the frequency in which the code that analyses the data collected by the monitoring package is executed. If the parameter value is close to the rate in which the environment changes, the adaptive application can show worse performance than the non-adaptive implementation. This happens because shortly after the application reconfigures itself to adapt to a certain pattern, this pattern is changed to a new one, and hence the application becomes more busy with reconfiguration than with serving client requests. On the other side, if the parameter value is high (comparing to the rate in which the environment changes), the response times become closer to the non-adaptive ones.

The main goal focus of our experiments were not the design and evaluation of some specific monitoring / adaptation strategies, but instead the validation of the usability of our framework. In fact, the experience of using our framework for incorporating adaptive behavior to the SIDAM application showed to be a very positive one, since we could validate the requirements established for its architecture. The framework successfully simplified the development process by providing design guidelines for incorporating adaptive mechanisms and promoting an extensible reuse of the framework components, minimizing the effort on programming.

5 Conclusions

Incorporating adaptive behavior in a distributed application is a complex task. We address this complexity through the use of object oriented technology by providing a framework that simplifies the development of adaptive distributed applications. The framework can be seen as a development tool for building reflective distributed applications, where the objective of reflection is to promote reconfigurations on application's components in response to significant variations on the execution environment.

A key point of our work is the support for detecting environmental changes based not only on resource availability but also on changes in the interaction pattern among application components.

As a future work, we intend to provide the support for network monitoring by integrating the framework to the ReMoS [Dinda] system. We also intend to apply the framework to the development of a wider range of distributed adaptive applications and use this experience to build extended framework components, customized for application specific domains.

References

[Vanegas] R. Vanegas and J. A. Zinky and J. P. Loyall and D. A. Karr and R. E. Schantz and D. E. Bakken, *QuO's Runtime Support for Quality of Service in Distributed Objects*, Middleware (1988).

[Helal] A. Helal and B. Haskell and J. L. Carter and R. Brice and D. Woelk and M. Rusinkiewicz, *Any Time, Anywhere Computing*, Kluwer Academic Publishers (1999).

[Fox] A. Fox and S. Gribble and 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).

[Moura] A. L. Moura and C. Ururahy and R. Cerqueira and N. Rodriguez, *Dynamic Support for Distributed Auto-Adaptive Applications*, Proceedings of AOPDCS - Workshop on Aspect Oriented Programming for Distributed Computing Systems (2002).

[Kon] F. Kon and R. H. Campbell, *Dependence Management in Component-Based Distributed Systems*, IEEE Concurrency, 8-1 (2000).

[Gamma] E. Gamma and R. Helm and J. Vlissides and R. Johnson *Design Patterns: Elements of Reusable Object Oriented Software*, Addison-Wesley (1994).

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

[Dinda] P. Dinda and T. Gross and R. Karrer and B. Lowekamp and N. Miller and P. Steenkiste and D. Sutherland, *The Architecture of the Remos System*, 10th IEEE Symposium on High-Performance Distributed Computing (HPDC'01) (2001).