Developing Adaptive Distributed Applications: a Framework Overview and Experimental Results

Francisco José da Silva e Silva[1], Markus Endler[2], and Fabio Kon[3]

[1] Federal University of Maranhão
Av dos Portugueses, s/n, Campus do Bacanga
65085-580 São Luís - MA, Brazil
fssilva@deinf.ufma.br

[2] PUC-Rio
Rua Marquês de São Vicente, 225, Gávea
22453-900 Rio de Janeiro - RJ, Brazil
endler@inf.puc-rio.br

[3] University of São Paulo
Rua do Matão, 1010, Cidade Universitária
05508-090 São Paulo - SP, Brazil
kon@ime.usp.br

Abstract. Building self-adaptive applications is a complex and challenging task. Developers must consider several issues in addition to the implementation of the application-specific functionalities. These issues include the selection of which environment elements must be monitored for detecting when adaptations should take place, how to perform the monitoring, which software adaptations should be carried out, and when should they take place.

Our work aims at reducing the complexity of building adaptive distributed applications. To achieve this, we developed an object-oriented framework that provides a set of integrated tools for monitoring resource usage and component interactions, detecting composite events, and reconfiguring distributed applications.

This paper describes the framework model and how it was used to incorporate adaptive mechanisms to a distributed information service for mobile users. It also presents several results obtained through the execution of a set of simulated scenarios. The results indicate that, besides detecting resource availability fluctuations, it is also important to monitor the interactions between application components, looking for patterns that indicate the necessity to perform dynamic adaptation. We drive conclusions about the applicability of using the framework for the design and implementation of adaptive distributed applications.

1 Introduction

Self-adaptive software is able to modify its own behavior at runtime to adapt to changes in its execution environment for optimizing its performance and availability in different environmental situations. [17, 7]. Adaptation is particularly useful for distributed, mobile, and ubiquitous systems because of the dynamic nature of these environments. This dynamism can be observed in form of variations in resource availability or error rates, in frequent disconnections, reconfigurations of hardware and software, and device mobility [20, 14].

Dynamic adaptation requires software to monitor its execution environment to detect when a significant change has happened and then to 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 functionalities. These include the selection of which resources elements should 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. All those issues make the development of adaptive software a complex and challenging task.

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, detecting complex events, and reconfiguring distributed applications dynamically.

This paper gives an overview of the framework model, and focuses on our experience on using the framework to add adaptive mechanisms to a prototype implementation of an information service for mobile users. It also describes several experimental results obtained through the execution of the adaptive distributed application on a set of simulated scenarios, which shall clarify the benefits of our approach. 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.

Specific details about the framework architecture are described elsewhere [6, 5]; in this paper we give a short description of the framework model and then focus on experimental results and on the evaluation of the framework usability.

2 Framework Model

Our framework model satisfies the following design requirements.

- The framework should be **generic** so as to be usable 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 in providing mechanisms to **coordinate reconfiguration actions** between distributed components.
- Framework components should be **flexible and extensible**, in order to accommodate specific applications needs.
- Framework components 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 [3], 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.

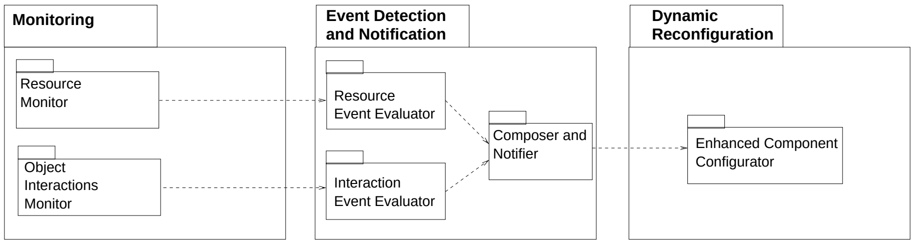

Fig. 1. Framework UML Package Diagram

The framework is composed of a collection of CORBA objects and services. Our implementation comprises 21,852 lines of Java code and uses JacORB [22] as the CORBA ORB. The framework is available for download as open-source software at `http://www.ime.usp.br/~sidam/software/AdaptationFramework`.

2.1 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 object.

Monitoring is based on *Monitoring Parameters* [23] which correspond to specific aspects of a distributed resource to be monitored. For instance, for each host one can monitor the amount of main memory available, or the percentage of processor utilization. For each network link we can monitor the available bandwidth and latency.

Every Monitoring Parameter has a set of associated operation ranges, which are defined by the framework user. For example, one could use the following operation ranges for monitoring percentage of processor utilization: [0%, 10%), [10%, 25%), [25%, 50%), [50%, 75%), and [75%, 100%].

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. Furthermore, each invocation can be time-stamped and one can measure the delay between the method invocation and the receipt of the response.

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 method in/out arguments. Our Interceptor implementation can be extended in order to incorporate application-specific needs in this way.

Normally, it is not necessary to monitor the interactions corresponding to all methods of each application component interface. This would impose a heavy overhead on application execution. Thus, the framework allows the user to specify which application methods must be monitored and the overhead imposed to other methods is minimized.

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.

2.2 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` sub-package.

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 [13]. We also allow the replacement of event definitions without interrupting the event detection service.

Resource Event Evaluation. As described in Section 2.1, a Resource Monitoring Object notifies a component of the `Resource Event Evaluation` sub-package (called Resource Manager) whenever there is a change on the operation range of the parameter being monitored. The Resource Manager keeps the last value informed by all RMOs instantiated on the environment, obtaining a global view of the state of the distributed resources.

The evaluation of a resource event is based on a boolean expression provided by the user as part of the event definition. The expression is composed of one or more Monitoring Parameters, as illustrated in the following expression: `(cpu_use > 4 AND memory_available < 2)`. This expression evaluates to `true` when, in a host, the operation range for the percentage of processor utilization is greater than 4 and the operation range that describes the availability of main memory is bellow 2. In order to trigger event notification, the corresponding boolean expression must stay `true` during a user-specified time. This avoids the generation of an event when a temporary situation, such as a peak in CPU usage, occurs.

Interaction Event Evaluation As described in Section 2.1, the data collected by the interception of method invocations is kept in persistent storage. The `Interaction Event Evaluator` sub-package periodically searches these data, looking for patterns that indicate some abnormal or undesired situation, or simply a change in application behavior.

The detection of interaction events is performed by *Evaluators*. Each Evaluator is responsible for the detection of an specific type of interaction event. The framework provides an abstract class, called `Evaluator`, that implements the basic functionality of an Evaluator.

For each application-specific interaction event, the developer of an adaptive application must implement a subclass of `Evaluator`, overriding two methods:

`user_evaluation_code()`, that reads the persistent storage and searches for a pattern of interest; and `clean_database()`, responsible for removing analyzed data from the persistent storage. `user_evaluation_code()` must return an object of type `EventNotification` when the pattern is found.

A component of the framework, the Interaction Manager, is responsible for the execution of Evaluators, each of which is executed in an independent thread. The Interaction Manager executes the `user_evaluation_code()` method in a periodicity defined by the user as part of the interaction event definition. It also allows dynamic loading and replacement of Evaluators using Java reflection mechanisms [13].

Event Notification The framework uses the CORBA Event Service [19] to define an event channel used to notify the occurrence of resource and interaction events. This channel uses a push approach, where the Resource and Interaction Managers are registered as event producers and components of adaptive applications responsible for carrying out dynamic reconfiguration are registered as consumers.

2.3 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 [16, 15], which defines an architecture for reconfiguring distributed applications. In this architecture, each application component has a corresponding Component Configurator object. The Component 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. Examples of common events are the failure or migration of a component, internal reconfiguration, or replacement of the component implementation. The rationale is that those events affect all the dependent components. This communication mechanism is used in order to coordinate reconfiguration actions between the application components.

The Component Configurator is the place where programmers insert the code to deal with these configuration-related events. This provides a clear separation of concerns between the application functional code and the code that supports application reconfiguration.

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 [12].

3 Experimental Results

Our framework was developed within the scope of the SIDAM research project [10], which aims at studying problems related to the design and implementation of decentralized information systems for mobile users. The SIDAM reference application is a distributed information service disseminating information about traffic conditions in a large metropolitan area. Generation, storage, and query of traffic information is fully 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.

We used the SIDAM application as a use case in order to validate the facilities provided by our framework. This section gives a brief overview of the SIDAM application and describes the experimental results we obtained.

3.1 The SIDAM Application

The SIDAM reference application is a distributed information service disseminating road traffic information for a metropolis such as São Paulo or New York. In order to keep the size of the information base and the corresponding queries manageable, the SIDAM application logically divides the city into micro-regions (a few blocks), which are the atomic units to which a piece of traffic information can be associated. Traffic data of a micro-region is stored at *Information Servers*, each of which can hold data from one or more micro-regions. Information servers reside at fixed hosts that are connected to each other through a reliable wired overlay network.

Traffic information can be queried and updated by mobile users, which may be either common citizens requiring information about the traffic situation in a specific part of a city or staff from the Traffic Engineering Department feeding the system with data about the observed traffic conditions. The mobile network is infra-structured, and hence all communications to and from a mobile host are intermediated by some base stations [1].

3.2 Simulated Scenarios

We have evaluated the benefit of using our framework through the execution of the adaptation-instrumented SIDAM application in several simulated scenarios in order to validate the gain of performance when interactions among application components are also monitored. For each simulated scenario we varied the amount of requests originated from each city region as well as the frequency by which this pattern changed.

A hypothetic city was divided into four regions: North, South, West and East. The simulations were performed in an Ethernet local area network with 10 Mbps, having the communication costs (e. g. latencies) between computers belonging to different regions simulated by adding communication delays as described on Table 1. The SIDAM application components as well as the framework components were instantiated in several nodes, as would happen in the real scenario. Each node of the network consisted of a Pentium II with 400 MHz, 128 MB of RAM, running Linux 2.4.18.

	North	**South**	**West**	**East**
North	-	20	8	10
South	20	-	10	8
West	8	10	-	15
East	10	8	15	-

Table 1. Communication cost between city regions in milliseconds

We used the framework to intercept calls 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 directed to each Information Server grouped according to the city region where the mobile clients were when the request was made. The system then uses this knowledge to choose a better distribution of the Information Servers across the network of fixed hosts. The main form of adaptation done by the SIDAM application is just a relocation of Information Servers to a host which is closer to most of its current clients, aiming at reducing the overall access latency. This is advantageous in a network suffering from significant variations of the access pattern, such it is the case in a network with mobile users.

In order to simplify the analysis of the collected data, we used just one Information Server in our tests, but our implementation does not restrict the number of Information Servers that can be instantiated.

First Scenario The first simulated scenario is composed of two phases. In the first one, the majority of accesses (50%) is performed by clients located in the South region, while the North, East, and West regions are responsible for 10%, 20%, and 20% of the queries, respectively. After performing a fixed number of `queryInfo` calls, we switched to the second phase, changing the access pattern to one in which clients located in a different region (e.g. West) become responsible for 70% of the accesses. All the other regions accounted for only 10% of the queries. As expected, the adaptive SIDAM application reacts to such a change by migrating the Information Server to a host “close to” the West region, which in fact, reduces the mean response time of the queries, because the communication overhead is reduced.

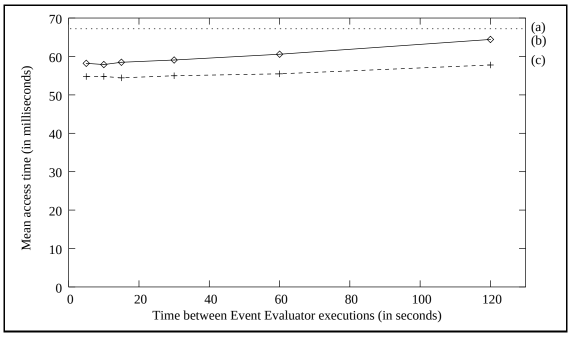

Fig. 2. First Scenario

We performed several experiments with this scenario, changing the execution frequency of the Evaluator responsible for detecting changes of the access pattern. Figure 2 shows the results we obtained. The x axis indicates the frequency of the Evaluator execution while the y axis represents the mean response time spent by the Information Server.

At Figure 2, the solid line (b) shows the results obtained with 600 queries, while the dotted line (c) shows the results when we executed 1,200 queries. The dotted horizontal line (a) indicates the mean response time obtained when executing the non-adaptive version of the SIDAM application.

The results show that the best adaptation is obtained with a frequent event evaluation.

The graph on Figure 3 shows the access time for each of the 600 queries performed when executing the non-adaptive version of the SIDAM application. For the majority of queries, the access time remained between 50 and 100 milliseconds, most of them around 75 milliseconds.

Figure 4 illustrates a similar graph, obtained with the execution of the adaptive SIDAM application. As seen, the adaptive version of the SIDAM application reduced the access time of most of the queries to a value around 50 milliseconds.

These results indicate clearly that, in this scenario, adaptation enhances application performance significantly.

Second Scenario The second scenario was chosen to tests a situation where the amount of accesses performed from each region is approximately the same. This scenario is also composed of two phases. In the first one, each city region generates the same amount of requests. Then, the access pattern changes to a situation where the clients located at one region (South) become responsible for 40% of the requests, while each of the other regions generate only 20% of the

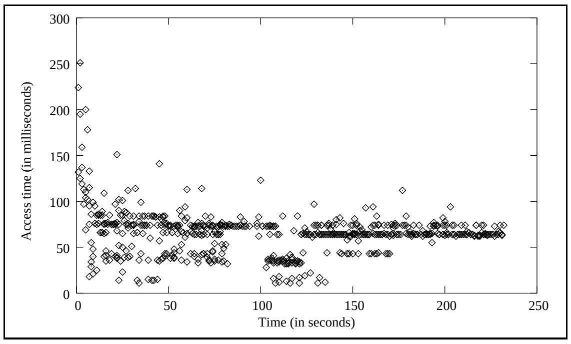

Fig. 3. First Scenario: Non-Adaptive SIDAM Application

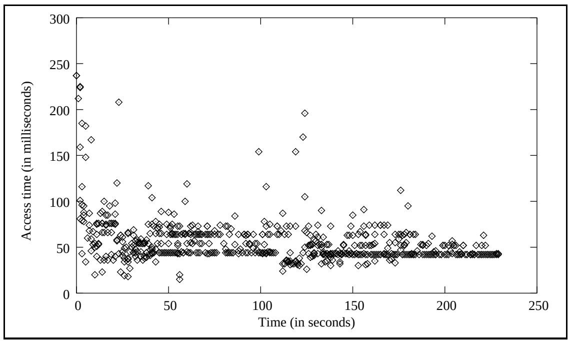

Fig. 4. First Scenario: Adaptive SIDAM Application

total amount of requests. During this second phase, the adaptive application migrates the Information Server to the south, but only a little gain is obtained, as shown by Figure 5.

Third Scenario The third scenario illustrates a situation where the access pattern changes with high frequency. Each access pattern is kept only for a short time, approximately 1 minute and consist of a fixed number of requests originated from a single region. We have then switched back and forth between two such patterns (origin: South and North).

During our tests we realized the importance of adjusting the parameter which determines the frequency in which the Evaluator analyses the data collected by the monitoring package. 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, as shown in figure 6. 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

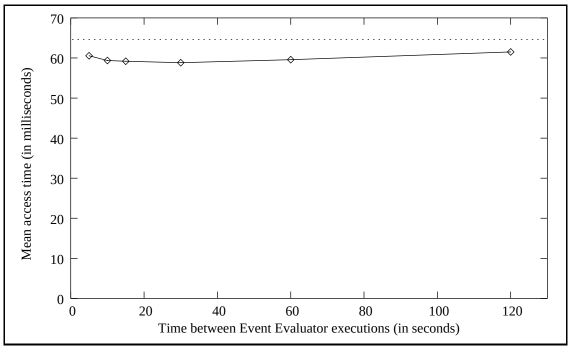

Fig. 5. Second Scenario

parameter value is too high (comparing to the rate in which the environment changes), the response times become closer to the non-adaptive ones. This is shown in Figure 6, where the access times of the adaptive application were only less then the non-adaptive application when there was a high frequency of event evaluations.

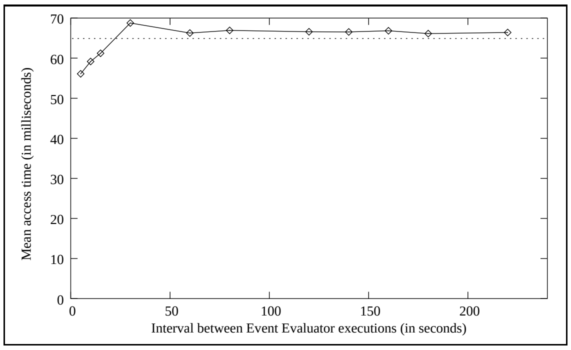

Fig. 6. Third Scenario

Forth Scenario This scenario demonstrates a situation where the adaptive mechanisms promote a very significant gain of performance to the application. In a first phase, the majority of accesses (90%) is performed by clients located in one region (North). After 600 accesses, the access pattern is changed to one in which clients located in a different region (e.g. South) become responsible for 90% of the accesses. In this second step we also perform 600 access. For this scenario, we defined different communication costs for clients issuing requests from each of the regions, illustrated on Table 2.

	North	South	West	East
North	-	100	100	100
South	100	-	100	100
West	100	100	-	100
East	100	100	100	-

Table 2. Communication cost between city regions in milliseconds

Figure 7 shows the results we obtained. The use of the non-adaptive version of the application resulted in a mean response time 132,92% greater than the adaptive one when the interval between the evaluation of the access pattern was set to 5 seconds.

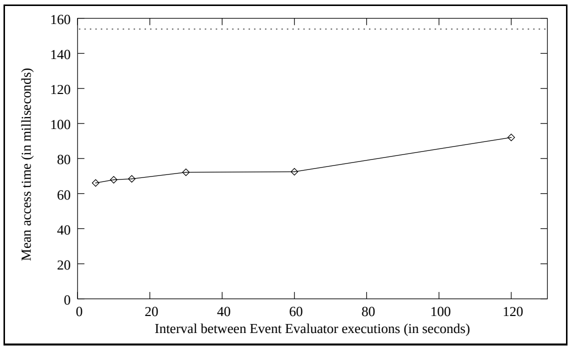

Fig. 7. Forth Scenario

3.3 Experience on Using the Framework

The experiments with the four scenarios aimed at identifying the situation in which dynamic adaptation triggered by detection of unbalanced access patterns can bring performance gains. The main focus of our experiments were not the design and evaluation of monitoring and adaptation strategies tailored for a specific application such as the SIDAM, but rather the validation of the framework usability for building adaptive distributed applications.

In fact, the experience of using our framework for incorporating adaptive behavior into a non-trivial application such as the SIDAM information service turned out to be a very positive one, since we could validate the requirements established for its architecture, described in Section 2. The major benefits observed were:

- The framework simplified the requirement and architectural analysis by defining a clear separation between the aspects related to environment mon-

itoring, detection of environmental changes, and dynamic application reconfiguration;
- System design was also simplified, since the framework "suggests" the components in charge of performing each of the above tasks, defining the collaboration model between them trough well defined interfaces. In that manner, the framework achieves the goal of enabling *design reuse*;
- The framework enabled intense *code reuse* through the reuse of its components, minimizing the effort on programming;
- The framework showed to be very flexible, letting the developer customize the aspects related to monitoring, event detection, and reconfiguration considering specific application needs;
- The framework promoted a clear separation of concerns between the application functional code and the code related to dynamic reconfiguration;
- The framework successfully provided the adequate mechanisms necessary to reconfigure an application composed of several components spread throughout a distributed system.

Despite the fact that the framework eases the construction of adaptive distributed applications, it does not address some other issues related to the development of adaptive applications, such as:

- What is the goal when incorporating adaptation: performance gains, fault tolerance or optimizing resource consumption, etc ?
- Under which circumstances does adaptation lead to significant gains considering its costs?
- Which environment variables must be monitored ?

These questions can only be answered after considering the specific adaptive application to be constructed. The framework role is to provide the basic structure of the mechanisms and their interactions that are necessary to implement the functionality defined in response to the above questions, facilitating the analysis and design of the solution.

4 Related Work

Fox et al. [11] 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. Compression is performed by *distillers* that preserve most of the semantic content of the data object. A proxy, placed between the client and the server, retrieves content from Internet servers on the client's behalf, determines the high-level types of the various components (e.g., images, text runs), and determines which distillation engine should be employed to adapt the data to the current bandwidth availability and client device type.

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.

Odyssey [18] addresses the problem of providing information access to mobile computers. Their approach is different from Fox's proxies, but uses the same principle of adjusting the data stream according to the environment status.Odyssey is implemented as a set of extensions to the NetBSD operating system, and 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 in client and server components, and the adaptation mechanism allows only to choose between different versions of the data (fidelity levels) so as to be compatible with the resource availability.

Chang and Karamcheti [4] describe an adaptation framework which eliminates the need for adaptation specific 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 the application execution 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 uses this profile information for 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 execution paths (algorithms) are coded into a single, monolithic executable, together with monitoring and steering agents. We believe that this approach is not well suited for highly distributed applications since, in this case, one must consider resource availability and application specific load in multiple nodes in order to determine the most suitable adaptation action.

The next two systems, developed at PUC-Rio and BBN respectively, are the ones that share the most similarities with our work.

Moura et al. [8] 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 [19] 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) [2, 21] 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.

5 Conclusions

Incorporating adaptive behavior into 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.

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. To demonstrate the use of the framework, we implemented a concrete case study: a distributed service to disseminate traffic information. We have also collected several experimental results by executing a set of simulated scenarios. The results indicate the benefits of using the data collected by the monitoring of the interactions performed by the application components as a way to discover when adaptation must be triggered.

The framework tools that provide the mechanisms for performing resource and interaction monitoring and for detecting environmental changes are extensible, allowing the application developer to accommodate specific application needs.

Other important aspect of our approach is the mechanism to coordinate reconfiguration actions between distributed components, simplifying the development of distributed adaptive applications. The mechanism is based in an extension of the Component Configurator architecture and provides a clear separation of concerns between the application functional code and the adaptation code.

As future work, we intend to provide the support for network monitoring by integrating the framework to the ReMoS [9] system. We also intend for apply the framework to the development of a wider range of distributed adaptive applications and use this experience to build extended framework components, tailored to specific application domains.

6 Acknowledgments

The authors would like to thank FAPESP (State of São Paulo Research Foundation), sponsor of the SIDAM project. Francisco José da Silva e Silva would like to thank his sponsors, CAPES (Brazilian Federal Agency for Post-Graduate Education), Federal University of Maranhão and FAPEMA (State of Maranhão Research Foundation).

References

1. B. R. Badrinath, A. Bakre, T. Imielinski, and R. Marantz. Handling mobile clients: A case for indirect interaction. In *1993. In Proceedings of the Fourth Workshop on Workstation Operating Systems*, Napa, California, October 1993.
2. BBN Technologies. *QuO ToolKit User's Guide, release 3.0.10*, April 2002. http://quo.bbn.com/.
3. Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*. Addison Wesley, 1999.
4. 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*, pages 11–20, Pittsburg, Pennsylvania, August 2000.
5. Francisco José da Silva e Silva. *Adaptacão Dinâmica de Sistemas Distribuídos*. PhD thesis, Universidade de São Paulo, February 2003.
6. Francisco José da Silva e Silva, Markus Endler, and Fabio Kon. A framework for building adaptive distributed applications. In *2nd Workshop on Reflective and Adaptive Distributed Middleware. International Middleware Conference*, pages 110–114, Rio de Janeiro, Brazil, june 2003. ACM, IFIP, USENIX, Pontifícia Universidade Católica do Rio de Janeiro.
7. DARPA. Self adaptative software, 1998. BAA 98-12 Proposer Information Pamphlet.
8. Ana Lúcia de Moura, Cristina Ururahy, Renato Cerqueira, and Noemi Rodriguez. Dynamic support for distributed auto-adaptive applications. In *Proceedings of AOPDCS - Workshop on Aspect Oriented Programming for Distributed Computing Systems (held in conjunction with IEEE ICDCS 2002)*, pages 451–456, Vienna, Austria, July 2002.
9. Peter Dinda, Thomas Gross, Roger Karrer, Bruce Lowekamp, Nancy Miller, Peter Steenkiste, and Dean Sutherland. The architecture of the Remos system. In *10th IEEE Symposium on High-Performance Distributed Computing (HPDC'10)*, San Francisco, August 2001. IEEE.
10. 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.
11. 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*, 5(4), August 1998.
12. Erich Gamma, Richard Helm, John Vlissides, and Ralph Johnson. *Design Patterns: Elements of Reusable Object Oriented Software*. Addison-Wesley, 1994.
13. Dale Green. *The Reflection API*. Sun Microsystems, available at http://java.sun.com/docs/books/tutorial/reflect, 2002.
14. Abdelsalam Helal, Bert Haskell, Jeffery L. Carter, Richard Brice, Darrell Woelk, and Marek Rusinkiewicz. *Any Time, Anywhere Computing*. Kluwer Academic Publishers, 1999.
15. Fabio Kon. *Automatic Configuration of Component-Based Distributed Systems*. PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, May 2000.
16. Fabio Kon and Roy H. Campbell. Dependence management in component-based distributed systems. *IEEE Concurrency*, 8(1):26–36, January/March 2000.

17. Alex C. Meng. On evaluating self-adaptive software. In Paul Robertson, Howard E. Shrobe, and Robert Laddaga, editors, *IWSAS 2000: International Workshop on Self-Adaptive Software*, volume 1936 of *Lecture Notes in Computer Science*, pages 65–74, Oxford, England, April 2000. Springer.
18. B. D. Noble and M. Satyanarayanan. Experience with adaptive mobile applications in odyssey. *Mobile Networks and Applications*, 4(4):245–254, 1999. Kluwer.
19. OMG - Object Management Group. *The Common Object Request Broker: Architecture and Specification*, November 2002. version 3.0.1.
20. Evaggelia Pitoura and George Samaras. *Data Management for Mobile Computing*. Kluwer Academic Publisher, 1998.
21. Vanegas R, Zinky JA, Loyall JP, Karr DA, Schantz RE, and Bakken DE. Quo's runtime support for quality of service in distributed objects. In *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'98)*, The Lake District, England, September 1998.
22. Software Engineering and Systems Software Group at Freie Universitt Berlin and Xtradyne Technologies AG. *JacORB Programming Guide*, August 2002. http://jacorb.inf.fu-berlin.de.
23. P. Sudame and B. Badrinath. On providing support for protocol adaptation in mobile wireless networks. Technical report, Department of Computer Science, Rutgers Universit, June 1997. http://www.cs.rutgers.edu/pub/technical-reports/dcstr -333.ps.Z.