# Modeling Dynamic Adaptation of Distributed Systems

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

Markus Endler
Departament of Computer Science
University of São Paulo, Brazil
`endler@ime.usp.br`

Fabio Kon  Roy H. Campbell  M. Dennis Mickunas
Department of Computer Science
University of Illinois at Urbana-Champaign
`{f-kon,rhc,mickunas}@cs.uiuc.edu`

*Report No.UIUCDCS-R-2000-2196, UILU-ENG-2000-1752*

December, 2000

*Phd student, Department of Computer Science - University of São Paulo, Brazil

# Modeling Dynamic Adaptation of Distributed Systems

**Abstract**

The computational infrastructure for information technology that is now being built is composed of a huge number of highly heterogeneous and mobile devices connected by different network technologies where a great number of resources and services will be available. This infrastructure imposes to the software architecture some complex requirements where adaptation is a key point. This work aims at addressing the problem of how to adapt computing systems at runtime by presenting a generic model for dynamic adaptation.

## 1 The Role of Adaptation in New Distributed Computing Environments

The computational infrastructure for information technology will be composed of a huge number of highly heterogeneous devices connected by different network technologies, where a great number of services and resources will be available to users. A key point of this infrastructure is mobility. The user will no longer be restricted to fixed points of access and, using different devices, will be able to access services and resources and also discover and interact with new ones as he(she) moves around. Mobile computing comprises systems as diverse as digital cellular telephony, road traffic information, and web browsing via wireless connections [KHR+00]. Another aspect to consider of those new environments are the now ongoing research on active spaces, such as the Gaia project [RC00]. An active space consists of physical spaces, such as offices, lecture and meeting rooms, homes, hospitals and cities that are augmented with computing devices integrated into the environment providing information to users of the space, helping them to perform more easily conventional activities and also providing new ways of interacting with the computational environment. In order to provide a seamless environment to users, the applications and systems must adapt themselves as mobility and other environment changes take place. The need for adaptation becomes clear when we analyze the following characteristics of these new environments.

- High bandwidth variability derived from the use of several different networks technologies (varying from a few Kbps to many Mbps in the case of wireless networking technologies [PS98]) and also derived from the mobility of users as they get far from base stations, leading to a weaker signal. In order to deal with this high variability, systems and applications must adapt, providing the user with a variable level of detail and quality of service.
- Heterogeneous networks: in different places, mobile devices may experience different QoS provided by the network. They must also be able to switch from one network interface to another and change the transport access protocol as they move across networks. For example, when switching from a cellular coverage in a city to satellite coverage in the country side [FZ94].
- Device heterogeneity with respect to both hardware and software properties such as screen size, color depth, processing power, primary and secondary memory availability, operating system, supported object component model, etc...
- Variable amount and diversity of services and resources available as the user moves from one place to another or as new entities arrive in the geographical area where the user is located.

Our work aims at addressing the problem of how to adapt computing systems at runtime by presenting a generic model for dynamic adaptation. Section 2 describes one of the application scenarios in which we intend to apply our model, as an example as how it can be used. Section 3 presents the proposed model, composed of several components, some of them influenced by recent research in this area, which is also described briefly. The adaptation model adopts the CORBA [OMG00] distributed object model as its underlying middleware in order to take advantage of its numerous facilities such as location transparency and the naming, trading, and security services. Section 4 describes, in more detail, the event evaluator, one of the components of this model. Finally, in Section 5, we describe future work and some conclusions we can already draw.

# 2 Application Scenario

We intend to demonstrate the adaptation model using the application defined in the SIDAM project [EdSeS+00]. SIDAM stands for Distributed Information Systems for Mobile Agents. The research aims at studying the main problems related to the design and implementation of decentralized information systems for mobile users. As a reference application it uses an on-line system for disseminating road traffic information but the model is sufficiently generic to be applied in a wide spectrum of applications. SIDAM adopts the *indirect model* for client-server interactions proposed in [BBIM93]. In this model, all communications to and from a mobile host *(Mh)* are intermediated by base stations, which act as representatives for all the mobile hosts within their cell and may also hold some of the *Mh* execution state. In the SIDAM model, there are two types of machines, the fixed hosts *(Fh)* and the mobile hosts *(Mh)*, where *Fhs* are connected to each other through a wired and reliable network. Among the static hosts, some serve as base stations (or *Mobile Support Stations - Mss*) for the mobile hosts and others may be data servers (e.g. a Traffic Information Server) that provide application specific services to both fixed and mobile hosts. The SIDAM model is illustrated in figure 1.

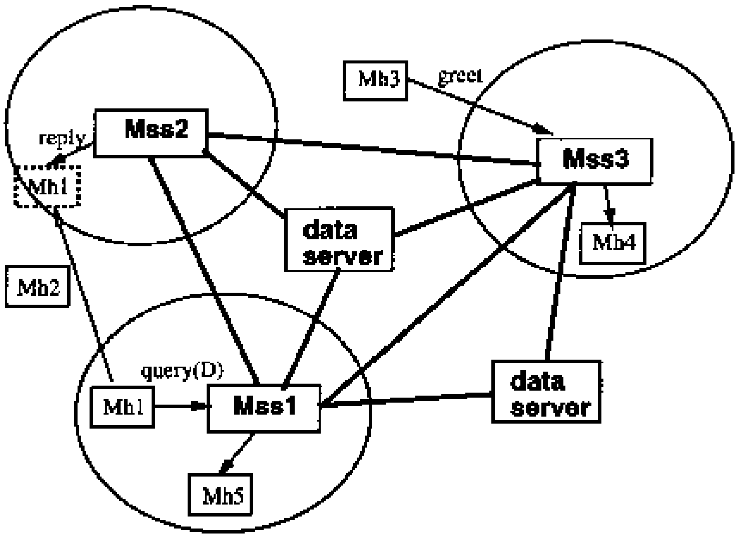


Figure 1: SIDAM Model with three Mobile Support Stations and five Mobile Hosts

## 2.1 The Road Traffic Information System

The SIDAM's reference application aims at developing a distributed information service for disseminating road traffic information that can be queried and updated by mobile users. These users may be either common citizens requiring information about the traffic situation in a specific part of a city or may be the Traffic Engineering Company (CET) staff (in a car or helicopter) feeding the system with data about the traffic. 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. some blocks) that are the atomic units holding traffic information. The micro-regions traffic data are stored in *Information Servers* (IS). Each information server can hold data from one or more micro-regions. Information servers reside in *Data Servers* (DS), which are fixed hosts.

The following operations are available to the user:

- query: the user provides the name of a street, street segment, or micro-region and receives back attributes such as mean velocity, congestion rate, and special events (e.g. accidents, public demonstrations, floods).
- update: the user provides the street name or segment, any of the attributes above, and its identification.
- subscribe: the user provides the street (segment) or micro-region name and a threshold on any of the attribute values, which determines when he or she should be notified about a significant change of traffic condition.
- broadcast: the user provides its identification, the identification of a group of users (configured *a priori*), and a message to be sent to the group. This operation is intended to be used primarily by the Traffic Engineering Company staff for communication with each other, in order to plan and execute coordinated actions.

Some dynamic adaptations useful for the SIDAM reference application are:

- **Split and Join of Information Severs**. Information servers can be highly dynamic entities. If an IS becomes overloaded, we can divide the micro-regions it stores into two or more sets originating new information servers. If two or more are under utilized, they can be collapsed into a single IS.
- **Replication of Information Servers**. It is possible to have a large number of requests issued by the group of mobile hosts located in a cell whose destination is an IS $A$. If $A$ is located far away from that cell's Mobile Support Station (Mss) in the wired network, the requests must be routed through a large number of hosts delaying the response. In this situation, we should replicate the IS to the data server nearest to that cell's Mss.
- **Data Prefetching**. If the bandwidth for communicating with a mobile host is sufficiently large, the system could deliver the traffic information about the micro-regions that are neighbors from the one requested. This can minimize the amount of requests issued and also improve the response time for future requests. If the bandwidth becomes scarce, the system would deliver just the traffic data for the micro-region requested.
- **Reallocation and Preallocation of Resources**. Special mobile hosts, such as the ones carried by ambulances and fire trucks, should have resources preallocated in their destination cell in order to guarantee their access to the traffic information. This may require the reallocation of the resources assigned to less important hosts.

# 3 A Model for Dynamic Adaptation

Figure 2 shows a UML package diagram [PJ00, BRJ99] of the proposed model for dynamic adaptation of distributed systems. There are three main packages that compose the model. The first is the monitoring package, which is responsible for monitoring the entities that compose the system. The data collected by this package is stored in logs or directly sent to the event detection and notification service. The latter is responsible for analyzing that data and determining the occurrence of events in which there is some interest. These events are sent to the dynamic reconfiguration package, which is responsible for applying the required reconfiguration actions to adapt the system.

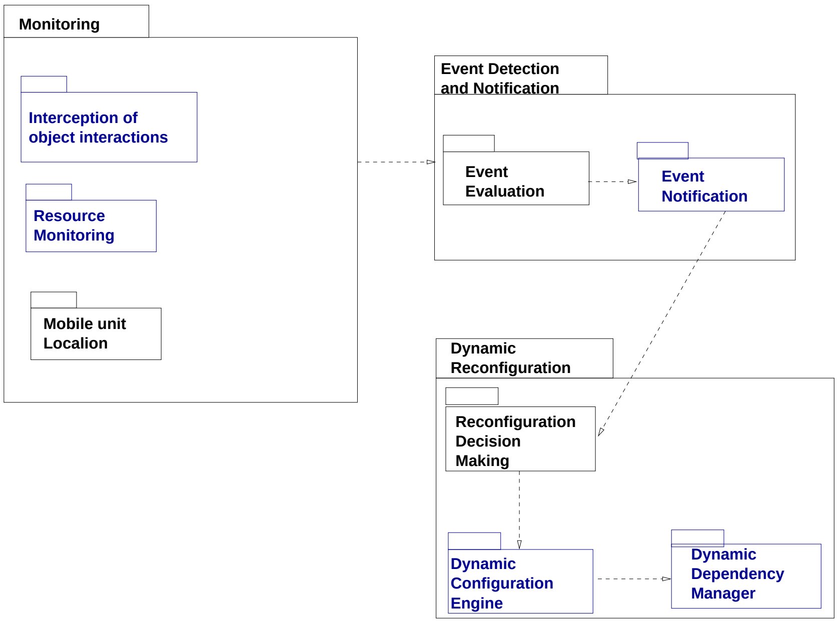


Figure 2: Model for dynamic adaptation

## 3.1 The Monitoring Package

The monitoring package is composed of three other packages. The first is responsible for the interception of the interactions between the objects that comprise the applications in the distributed system. Our intention is to follow the model described by [Mao99]. In this work, *Interceptors*, as defined in the CORBA standard [OMG00], are inserted into the ORB invocation path. Every time an object invokes a method of another object in the system, the message corresponding to this invocation is intercepted and later re-dispatched to the target object. Using interceptors, we can extract useful information from each method invocation and store it in a log for later analysis by the event evaluator package. Furthermore, each request can be time-stamped so that we know exactly when a message invocation request was received at the server side. We 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.

The interception of object interactions is particularly useful for monitoring the objects that compose a distributed application. However, it is also useful to have a package for monitoring distributed resources, such as the memory and CPU available at the hosts. For building this package we intend to use some ideas described in [SB97]. We call a *parameter* the aspect of a resource that we intend to monitor. For instance, we could be interested in monitoring the percentage of CPU usage in a host. For each parameter, we define a set of ranges in which it operates. We notify the event evaluator only when a parameter that was operating in one range passes to operate in another one. To implement such monitoring functionality we propose the use of a *resource monitoring object*. For each parameter, we define a specific monitoring object that must be instanciated in each host that we intend to monitor. The monitoring object checks the current range in which the parameter that it is responsible for monitoring is operating and sends a message to the event evaluator package with this information. It continuously checks the range in which the parameter is operating in a predefined frequency. It only sends another message to the event evaluator when the parameter again changes the range in which it is operating.

Yet another interesting aspect related to monitoring is the location of mobile devices. As described in the Introduction, we believe that location awareness will play a key role in future distributed computing environments. We assume that the mobile device is equipped with a hardware resource that can be monitored in order to obtain information about the device location. This could be the exact geographic position in the case of a mobile unit equipped with a GPS interface or a coarser location information such as the detection of the current cell in which the device is located. The information about a mobile unit location can be useful for a large range of applications. As an example, consider the retrieval or dissemination of location dependent data such as traffic information, theaters and restaurants located near the mobile unit, etc...

### 3.2 Event Detection and Notification Package

We are currently concentrating our efforts in defining a complete specification for the event evaluator package, described in more detail in the next section. The event evaluator package is responsible for analyzing the data stored in the logs created by the monitoring package. By analyzing those logs, the event evaluator can determine the occurrence of events that are relevant to the application. The event evaluator also receives messages from the resource monitoring objects informing that some parameters being monitored had changed their operation range. By collecting and combining these messages for evaluating user defined expressions the event evaluator can also determine the occurrence of events, as described in Section 4.

When the event evaluator 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. Particularly, it will deliver the event for the dynamic reconfiguration package.

### 3.3 Dynamic Reconfiguration Package

The dynamic reconfiguration package is composed of three sub-packages. The reconfiguration decision making sub package is responsible for receiving the event notification and determining which reconfiguration actions are required to adapt the system in the occurrence of that event. A group of reconfiguration actions are sent to the dynamic configuration engine sub package, which applies them with the help of the dynamic dependence manager. To implement the dynamic dependence manager, we intend to use the component configurator infrastructure described in [Kon00, KC00]. In this work, two types of dependencies are explored:

- *Prerequisites* necessary for loading an inert component into the runtime system. They are usually expressed as dependencies on persistent hardware and software components. A prerequisite list can contain three different kinds of information: the nature of the hardware resources the component needs, the capacity of the hardware resources it needs and the software services (such as other components it requires).

- *Dynamic dependencies* between loaded components in a running system. A *component configurator* manages the dependencies of each component. The component configurator is responsible for storing the runtime dependencies between a specific component and others system and application components. The dependencies of a component $C$ are managed by a component configurator $C^c$. Each configurator $C^c$ has a set of *hooks* to which other configurators can be attached. These are the configurators for the components on which $C$ depends; they are called *hooked components*. The components that depend on $C$ are called *clients*; $C^c$ also keeps a list of references to the clients' configurators.

## 4 Event Evaluator

The major requirements that guided us when defining the architecture for the Event Evaluator were:

- **Flexibility.** Each application has a set of events that is useful only in its own context. Therefore, the Event Evaluator must allow the user to extend the set of events that are being monitored and detected as well as modify the specifications of the evaluators responsible for event detection.

- **Dynamic configuration.** The extensions and changes in the code of the Event Evaluator described in the last item should not be hard coded into the Event Evaluator, otherwise we would need to recompile it every time a user develops a new application. In order to apply the changes and extensions, we should not even interrupt its execution or we would be affecting the event detection of other applications.

- **Management.** The user should be able to customize the Event Evaluator by dynamic adding and removing evaluator definitions, loading into memory as well as activating and deactivating their evaluation during the Event Evaluator execution. The user should also be able to inspect the internal configuration of the Event Evaluator (introspection).

To fulfill the above requirements efficiently, it is essential to define a way to support dynamic change of the Event Evaluator code. To do this, we intend to use the research described in [KRL+00]. This work describes *dynamicTAO*, a CORBA-compliant reflective ORB that supports dynamic configuration. Component implementations are shipped as dynamically loadable libraries, so they can be linked to the ORB process at runtime. A *Persistent Repository* stores category implementations in the local file system and a *Dynamic Service Configurator* supplies common operations for dynamically configuring components at runtime. The *Dynamic Configurator* interface specifies operations that allows the user to store or erase components in the local file system, load and remove components in the main memory as well as suspend and resume their execution. We intend to use these principles in the context of event detection.

## 4.1 Component Collaborations

Figure 3 shows a UML collaboration diagram [PJ00, BRJ99] of the log based event evaluation as proposed in the Event Evaluator architecture. The main components are:

1. *Event Evaluator*: defines the code that detects the occurrence of a specific type of event. This component reads the log stored by the monitoring package in order to detect the occurrence (or not) of a specific event. There can be multiple event evaluators loaded in memory as represented in the figure, one for each type of event that is being periodically evaluated. The code of the event evaluator as well as the frequency in which it must be executed are provided by the user, giving him (her) the flexibility described earlier.

2. *Configurator*: provides an interface through which the user can store or erase event evaluator definitions in a permanent storage, load or unload event evaluators, and suspend or resume their execution, which implies suspending or resuming the evaluation of certain events.

3. *Event Evaluator Storage*: responsible for the management of the permanent storage that keeps the event evaluator definitions.

4. *Scheduler*: responsible for starting the execution of the event evaluators loaded in memory in a schedule defined by the user.

5. *Event Composer and Notifier*: responsible for notifying the occurrence of events to the objects that register their interest on them. We use the term *composer* because we intend to let the user define an event that depends on other events. For instance, the user could define an event $C$ that is trigged when an event $A$ and an event $B$ occurs.

Figure 4 shows the collaboration diagram between the objects that deal with the evaluation of events based in the monitoring of system resources. Here, the main components are:

1. *Resource Monitoring Object*. Part of the Resource Monitoring package, it is responsible for monitoring a certain parameter (such as percentage of CPU usage) in a host. When it detects that the operating range for this parameter has changed, it sends a notification message to the Resource Monitoring Manager. We intend to develop some monitoring objects to monitor parameters such as: percentage of CPU usage, physical memory, virtual memory, and disk space. The user can also build his (her) own monitoring objects for the parameters he (she) wants to monitor in the context of his (her) application. As an example, we intend to develop a monitoring object to monitor the number of threads created by a Traffic Information Server, used in the SIDAM application.

2. *Resource Monitoring Manager*. The component with which the user interacts to:

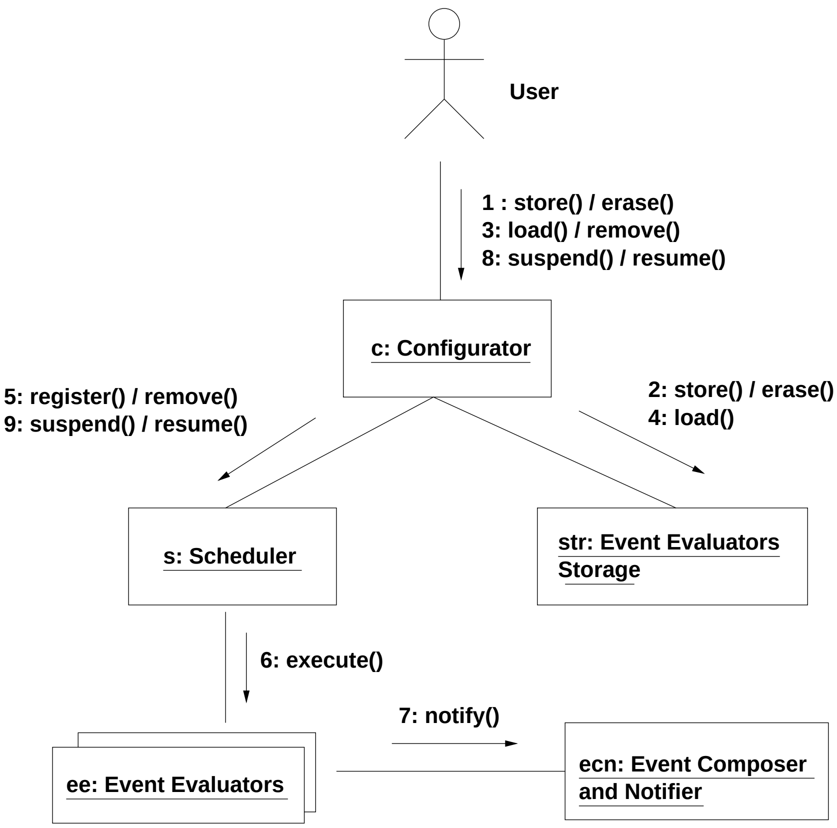


Figure 3: Event Evaluator: log based event evaluation

- Define the events to be detected. The user will do this by specifying a boolean expression and a duration. The boolean expression is composed of parameters. Each parameter has a target range. For example, we could have the overload of a Traffic Information Server defined as when the percentage of CPU usage gets higher than 80% and the memory availability drops bellow 20%. The duration refers to the time in which the expression must hold to trigger the generation of the event. With this we can avoid the generation of an event when we have a temporary situation such as a peak in CPU usage.
- Remove the definition of an event.
- Suspend or resume the evaluation of an event.

The resource monitoring manager stores the definitions of events provided by the users. When it receives a message from a resource monitoring object informing that one parameter has changed its operating range, it evaluates all the boolean expressions that have this parameter. If one expression becomes true it starts a timer that is responsible for sending an alarm message when the time defined by the user for this expression expires, indicating the occurrence of the event. If, between the time of evaluating an expression as true and receiving an alarm message, the manager receives another message from a resource monitoring object that turns the expression to false then the timer will be turned off and the event will not be triggered.

3. *Timers*: responsible for sending an alarm message to the manager indicating that the time defined by the user for a specific boolean expression has expired.

4. *Event Composer and Notifier*: same object described in the log based event evaluation. Allows the user to define composite events that depend on log based events and on resource monitoring events.

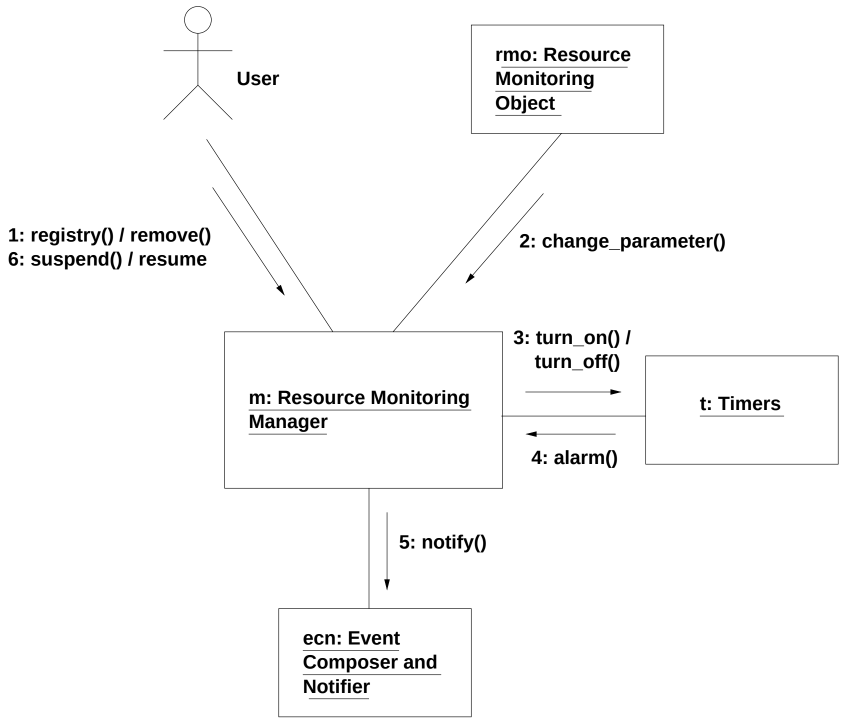


Figure 4: Event Evaluator: resource measurement event evaluation

# 5 Related Work

Sudame and Badrinath [SB97] describe a mechanism based on ICMP messages to propagate information about network environment parameters such as the cost of communication, latency, and bandwidth. Mobility not only introduces large variations in these parameters, but it also introduces some new ones such as signal strength, location, and energy consumption and availability that can also be addressed by the mechanism described in the paper. Each parameter is associated with a set of levels (values). Each level is either a low watermark or a high watermark. A level with a high watermark is violated when the parameter value crosses it from below. A level with low watermark is violated when the parameter crosses it from above. The reporting mechanism uses a new ICMP message, *ICMP STATUS*, to propagate information about the environment every time a watermark is violated. The message carries the name of the device for which the event has occurred, the name of the parameter that caused the event and a list of watermarks that have been violated. The mechanism provides a handler for this new ICMP message. The handler invokes a function at the socket layer. This function takes actions for various transport layer protocols and also notifies all interested applications. Actions are functions provided by the transport layer to adjust these protocol specific parameters. There are a fixed predefined set of actions that can be taken. An API is provided to allow applications to register interest in receiving notifications about watermarks violations and also to specify which actions should be taken in that case. The later are stored in *action tables*. The socket layer maintains one action table per socket that requires adaptation. Each entry in the action table has the following template: `parameter(p),watermark(w)=>ACTION(a),arg`. When parameter $p$ violates watermark $w$, perform action $a$ where $a$ is an index corresponding to some action provided by the transport layer and *arg* is passed to the action as an argument. We borrow from this work the notions of parameters and operational levels for the monitoring of system resources. However, this work does not addresses the problems of dynamic reconfiguration of system components as well as the monitoring of application components interactions. It also focus in adapting transport layer protocols parameters and not in general system adaptation. Finally, the monitoring and adaptation are mainly local to a host and the work does not addresses the adaptation of a distributed system composed of several components spreaded in a network.

Fox et al. [FGBA96, FGCB98] address the problem of network bandwidth and client variability with respect to both hardware and software properties by proposing an adaptation model based in the following design principles:

- Adapt to client variation via data type specific lossy compression. The compression is performed by *distillers*, which preserve most of the semantic content of the data object.
- Perform adaptation on the fly.
- Move complexity away from both clients and servers by performing adaptation at a shared infraestructural proxy.

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 destination engine must be employed in order to adapt the data to the current bandwidth availability and client type. This work focus in the Internet data access application. Two parameters are used to trigger the adaptation. The first one is network bandwidth and the other one is the client hardware and software architecture. The system dynamically chooses the distiller to be used depending on the value of these parameters. We could map this adaptation mechanism using our model in the following way: for each connection between a client and the server, start a resource monitoring object. Every time the bandwidth available changes significantly, the monitoring object will send a message to the event evaluator. Within the message, the resource monitoring object passes the current range in which the connection bandwidth is operating and also the hardware/software architecture of the client. The user should also define an event based on the network bandwidth parameter. This will allow the event evaluator to detect the event and send a notification to the reconfiguration decision making. Depending on the values passed within the notification message (bandwidth available and hardware/software architecture), the reconfiguration decision making asks the dynamic configuration engine to dynamically load the proper distiller. The notification of the event is also delivered to the application server that would now know witch distiller to use when a request arrive from the client.

Odyssey [NSN+97, NS99] also address the problem of providing information access to mobile computers. It does this in a different way of Active Proxies but it uses the same principle of adjusting the data according to the environment situation. Odyssey is a set of extensions to the NetBSD operating system. It defines adaptation as trading data quality for resource consumption. To quantify this notion of quality, Odyssey defines a type specific property called *fidelity*. For any data item there is a most complete, current, detailed version of that item, called the *reference copy*. Ideally the reference copy is always presented to the mobile user. However, when resources become scarce, the item is degraded in some way. Fidelity is defined as the degree to which a present item matches the reference copy. 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 best to exploit available resources. Odyssey objects are available to applications as part of the file name space. The *interceptor* provides a VFS client [Kle96] that forwards file system requests on Odyssey objects to the *viceroy*. The viceroy's most important task is monitoring the availability of resources and managing their use. Since fidelity is a type-specific notion, each Odyssey object must carry with it a notion of type. The *wardens* are a set of type aware code components, one warden per type. Wardens are responsible for providing a menu of fidelities from which applications can pick. Wardens also provide semantically rich access methods to applications, and can take advantage of type knowledge to optimize resource usage, consistency, and so on. Odyssey adds two calls to the standard system API. The first, *resource request*, is used to inform Odyssey of the changes to resource availability of interest to the calling application. For example, a video player might wish to know if the bandwidth ever drops bellow a certain level. The second, *type specific operation*, is used by an application to change the fidelity at which data is accessed. In the example above, the video application might reduce the frame rate when bandwidth does drop significantly. When the *viceroy* detects that a resource stays outside of one or more of the requested levels, each affected application is notified via an *upcall*. The application responds by changing the fidelity of the data it is accessing.

Chang and Karamcheti [CK00] describe an adaptation framework that eliminates the need for adaptation decisions to be explicitly programmed into the application by relying on two components: a *tunability interface* and a *virtual execution environment*. The tunability interface provides language support to express the 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). The tunability interface relies upon unverified source-level annotations, which are processed by a simple macro preprocessor. Different execution paths are achieved by expanding the tunable module constructs (which set up the alternate execution paths) into conditional statements involving the associated guard expressions. The preprocessor also generates *monitoring* and *steering* agents that monitor resource availability and control application execution respectively. The framework also provides a way for creating virtual execution environments that run on top of a static distributed system and can be configured to accurately emulate a variety of resource availability scenarios. A driver script repeatedly executes the application in these 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. This work address the dynamic configuration using a different approach then the one used in our adaptation model [KRL+00]. All the different execution paths (algorithms) used are coded into a single execution process as well as the monitoring and steering agents. In our approach they are dynamically loaded only when needed and can also be removed when they are not in use. The monitoring is also separated from the application code. This work also focus in a single application adaptation and not in adapting a distributed system composed of several components spreaded in a network.

# 6 Future Work and Conclusions

The adaptation model described here aims at providing the user with a flexible and powerful tool to simplify the development of self-aware and adaptable distributed applications. In order to achieve this, we propose a model composed of three main components:

1. Monitoring: provides the user with the tools necessary for monitoring the interactions between the objects that compose the distributed application, the resources available in the system, and the location of mobile devices.

2. Event evaluation and notification: provides the tools for evaluating the data collected in the monitoring stage in order to detect events and a notification tool to notify the objects interested in them.

3. Dynamic Reconfiguration: provides tools that encapsulate mechanisms through which the user can define the actions needed to reconfigure the system faced with the occurrence of an event and apply these actions. Dynamic dependences of the components are also managed in order to provide a safe reconfiguration.

As this document describes, many of the components in this model are based in recent research but others will have to be designed and developed in order to have a complete operational framework. We started by defining the architecture of the event evaluation package. The following are some future work we intend to do:

- define how to support the specification of composite events.
- define the architecture of the Composer and Notification package.
- implement some basic resource monitoring objects.
- implement the Event Evaluator and Notification packages.
- Tie the monitoring and the evaluation packages by implementing some events that we have already defined in the context of the SIDAM project.

In a subsequent stage of the research, we plan to investigate the reconfiguration decision making package.

## References


[BBIM93] 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.

[BRJ99] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*. Addison Wesley, 1999.

[CK00] 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.

[EdSeS+00] 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.

[FGBA96] Armando Fox, Steven D. Gribble, Eric A. Brewer, and Elan Amir. Adapting to network and client variability via on-demand dynamic distillation. *ACM SIGPLAN Notices*, 31(9):160–170, September 1996. Co-published as SIGOPS Operating Systems Review **30**(5), December 1996, and as SIGARCH Computer Architecture News, **24**(special issue), October 1996.

[FGCB98] 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.

[FZ94] G. Forman and J. Zahorjan. The challenges of mobile computing. *IEEE Computer*, apr 1994.

[KC00] Fabio Kon and Roy H. Campbell. Dependence management in component-based distributed systems. *IEEE Concurrency*, 8(1):26–36, January/March 2000.

[KHR+00] Fabio Kon, Christopher Hess, Manuel Román, Roy H. Campbell, and M. Dennis Mickunas. A flexible, interoperable framework for active spaces. In *OOPSLA'2000 Workshop on Pervasive Computing*, Minneapolis, oct 2000.

[Kle96] S. R. Kleiman. An architecture for multiple file systems types in sun unix. In *Summer Usenix Conference Proceedings*, Atlanta, GA, 1996.

[Kon00] Fabio Kon. *Automatic Configuration of Component-Based Distributed Systems*. PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, May 2000.

[KRL+00] Fabio Kon, Manuel Román, Ping Liu, Jina Mao, Tomonori Yamane, Luiz Claudio Magalhães, and Roy H. Campbell. Monitoring, Security, and Dynamic Configuration with the dynamicTAO Reflective ORB. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'2000)*, number 1795 in LNCS, pages 121–143, New York, April 2000. Springer-Verlag.

[Mao99] Jina Mao. Monitoring and Analyzing Method Invocations in the 2K Operating System. Master's thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, May 1999.

[NS99] B. Noble and M. Satyanarayanan. Experience with adaptive, mobile applications in odyssey, 1999.

[NSN+97] Brian D. Noble, M. Satyanarayanan, Dushyanth Narayanan, James Eric Tilton, Jason Flinn, and Kevin R. Walker. Agile application-aware adaptation for mobility. In *Sixteen ACM Symposium on Operating Systems Principles*, pages 276–287, Saint Malo, France, October 1997.

[OMG00] OMG - Object Management Group. *The Common Object Request Broker: Architecture and Specification*, oct 2000. version 2.4.

[PJ00] Meilir Page-Jones. *Fundamentals of Object-Oriented Design in UML*. Addison Wesley, 2000.

[PS98] Evaggelia Pitoura and George Samaras. *Data Management for Mobile Computing*. Kluwer Academic Publisher, 1998.

[RC00] Manuel Román and Roy H. Campbell. Gaia: Enabling Active Spaces. In *Proceedings of the 9th ACM SIGOPS European Workshop*, Kolding, Denmark, September 2000.

[SB97] P. Sudame and B. Badrinath. On providing support for protocol adaptation in mobile wireless networks. Technical report, Department of Computer Science, Rutgers University, jun 1997. http://www.cs.rutgers.edu/pub/technical-reports/dcstr -333.ps.Z.