CASA – A Contract-based Adaptive Software Architecture Framework*

Arun Mukhija Martin Glinz

Institut für Informatik
University of Zurich
Winterthurerstr. 190, CH-8057, Zurich

{mukhija | glinz}@ifi.unizh.ch

Abstract

Traditionally, applications are developed with an implicit reliance on the stability of their execution environment and available resources, while little or no support is provided for the runtime adaptation of application behavior in case of any instability encountered. But such an approach proves futile for more dynamic environments, such as those encountered in self-organized mobile networks, wherein any form of reliance on the runtime computing environment of an application would be highly optimistic.

The Contract-based Adaptive Software Architecture (CASA) framework, described in this paper, addresses the need to equip an application with the ability to dynamically adapt itself in response to changes in its execution environment. This implies that an application is able to meet its functional and/or non-functional commitments even when its runtime computing environment changes. The framework builds on the idea of specifying resource requirements and adaptation behavior of applications in application contracts.

1 Introduction

Software applications for dynamic distributed computing environments are faced with two challenges: limited resources and unreliable availability of resources. The former challenge is primarily due to the limited communication resources available when using wireless networks; moreover the ever-reducing size of mobile nodes restricts the amount of local resources that can be integrated into them. The latter challenge of unreliable resource availability is due to uncertain variations in load and unanticipated resource failures. It is especially profound in the case of self-organized mobile networks, popularly known as mobile ad-hoc networks, as these networks offer a very flexible way of operation, wherein nodes

*The work presented in this paper was supported (in part) by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322.

Proceedings of the 3rd IEEE Workshop on Applications and Services in Wireless Networks (ASWN 2003), Berne, Switzerland, July 2003, pp. 275-286.

are free to join or leave a network community or travel within the network, without any prior intimation. Such flexibility, obviously, comes at the cost of highly dynamic topology of the network and thus less reliability on the available resources.

Conventional approaches of software development do not account for the instability of resources: the applications are developed with more or less rigid resource requirements. Such an approach works reasonably well for computing environments where dependability on the available resources is quite high. But for more dynamic environments, where fluctuations in resource availability are very frequent, this approach fails. Hard-coding some degree of adaptability into the applications is a tedious and rather limited solution to the problem. Recent attempts to enhance middleware for providing adaptability services are also limited in scope and lack flexibility (see Section 5 on related work).

Developing applications for such dynamic environments requires a fundamental shift in the approach towards development. Unlike the traditional approaches, the new approach for software development should ideally make no prior assumptions about the resources that will be available to an application, while at the same time the application should be prepared for all possible resource availability scenarios. This, in effect, implies that applications should be made dynamically adaptable in response to changes in their execution environment. This problem needs to be handled in two parts. Firstly, an application should be able to detect changes in its runtime computing environment and the resources available to it. And secondly, the application should be able to adapt its behavior in response to such changes, so that it can continue to operate in the new environment, probably at a different level of performance and/or functionality.

Even with such an approach, it will sometimes be necessary to suspend an application in case of a significant drop in the availability of a critical resource. But, in general, applications will be able to carry on with their execution for a wide range of resource availability scenarios. Needless to say, such adaptive applications would outlive those that have strict resource requirements and are not adaptable.

As an example, let two remote applications be collaborating for an emergency coordination project that requires the monitoring application to send latest images of maps of an affected area to be analyzed and used by the back-end support application. Now, in case of a drop in bandwidth available on the path between the two applications, the monitoring application will need to switch to another configuration that sends maps with reduced details or sends them less frequently; and accordingly the back-end application will need to switch to a configuration that is able to work with the maps with reduced details or with stale maps. Or alternatively, the monitoring application may simply switch to a configuration that sends the data computed form the maps, at the expense of more CPU cycles, instead of directly sending the maps. While the former is an example of reduced performance due to a drop in available resources, the latter is an example of an increase in usage of another resource (CPU in this case)

without sacrificing the performance of the application as such.

The CASA (Contract-based Adaptive Software Architecture) framework, presented in this paper, provides an integrated framework for the development of adaptive applications, while the CASA run-time system takes care of providing 'resource awareness' and 'dynamic adaptability' to the applications in a transparent manner. Different application domains may have different service parameters of interest. For example, multimedia applications may be interested in service parameters such as latency and jitter, while some other applications may be interested in service parameters at a higher level of abstraction such as timely response and dependability, and still others may be interested directly in resource requirements such as memory space and processor cycles. CASA provides an integrated approach to include all kind of service parameters across different application domains within the same framework.

Applications residing on autonomous nodes of a self-organized mobile network negotiate a service agreement with their peers. The application-domain-specific service requirements agreed upon for an application are mapped to the corresponding resource-level requirements. The underlying CASA run-time system strives to satisfy the resource requirements of the application by proper resource allocation and management techniques. If significant changes in the resource availability occur, due to load variations or resource failures, the components of the concerned application are dynamically reconfigured by the CASA run-time system to suit the changed execution environment. Dynamic reconfiguration of components is carried out in a seamless manner, that is, without taking down the system. The adaptation policy of an application is specified in the so-called application contract. The application contract is expressed in the Contract Specification Language (CSL), developed as part of the CASA framework.

The approach offered by CASA is flexible, as the level of adaptability can be tailored to the application's requirements. The level of adaptability depends on the number of alternative configurations provided for an application. Moreover it is extensible, as the level of adaptability as well as the policies of adaptation can be extended anytime, by integrating more alternative configurations and updating the application contract accordingly, to make it more sensitive to environmental changes.

The rest of the paper is organized as follows. Section 2 describes the constituent entities of the CASA framework. Section 3 links these entities together to explain the working of CASA. Section 4 provides a brief description of contract specifications. Section 5 gives an overview of related work. Finally Section 6 concludes the paper and indicates the future direction of our work.

2 CASA (Contract-based Adaptive Software Architecture) Framework

The overall framework of CASA is as illustrated in Figure 1. Adaptive applications reside on distributed autonomous nodes that form ad-hoc networks. At

run-time, when the peer applications decide to interact, they negotiate a service agreement amongst themselves. The underlying CASA run-time system utilizes proper resource allocation and management techniques in order to satisfy service commitments of individual applications. In case of a mismatch between resources requirements and availability, the CASA run-time system carries out a dynamic reconfiguration of application components according to the adaptation policy specified in the application contracts.

The details of each of the constituent entities of the CASA framework are described in the following sub-sections. (We use the term "entity" to refer to components of the CASA framework, in order to avoid confusion with the term "component" used for application components).

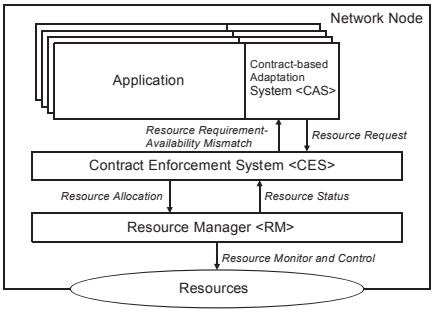


Figure 1: CASA Framework

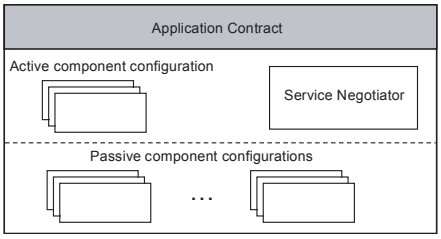


Figure 2: Application Structure

2.1 Applications

The internal structure of an application is as shown in Figure 2. CASA supports component-oriented development of applications. To support adaptation, alternative component configurations of an application need to be provided by the application developer, such that each one is best suited to particular resource conditions. Providing such alternative configurations for an application forms the backbone of our adaptive software architecture. A component configuration here implies the set of components constituting the application. Alternative component configurations may differ in just a few of their constituent components, while many other components remain the same across the configurations. Mutually replaceable components in alternative configurations must belong to the same type. Belonging to the same type implies that components conform to the same functional interface, but differ in their implementations – that is, in their resource requirements and probably functional and/or performance characteristics. Many of the constituent components of an application may be standard components and be reused in the integration of various diverse applications in the same or other domains. Thus the effort spent in developing different implementations of the same component, each suited to a different execution environment, will be compensated for by the amount of reuse of the component.

As shown in the internal structure of an application in Figure 2, there is one active component configuration while there may be several passive component configurations. As is evident from their names, the active configuration is the one that is currently being executed, while passive configurations are the ones that are not part of the current execution. This is just a logical representation, as in practice the majority of the components will be the same across active and passive configurations; so it will be only a few components that will be passive, and not a complete configuration. The other two significant constituents of an application, namely the application contract and the service negotiator are described below.

The Application Contract: The application contract of an application is divided into so-called operating zones. The operating zones of an application contract are distinguished by the service level provided and/or expected by the application in a given zone. Switching between the operating zones of an application contract implies significant differences in the level of service provided and/or expected by the corresponding application. Each zone, in turn, contains a list of valid alternative component configurations for that zone, and their corresponding resource requirements. Component configurations are specified by the list of names of their constituent components. Alternative component configurations within a zone offer and expect, more or less, the same level of service (as they belong to the same operating zone) but differ in their resource requirements. The first configuration listed in a zone is treated as the most preferred configuration for that zone, while others are substitutes subject to resource availability conditions. Application contracts are expressed in the Contract Specification Language (CSL).

The Service Negotiator (SN): Each application contains a Service Negotiator (SN) component that is responsible for negotiating the service level (also referred to as quality of service or QoS in the literature) to be offered to and/or expected from its peer applications, on behalf of its host application. A self-organized mobile network is essentially a peer-to-peer network, wherein the applications offer services to other peer applications and at the same time use services provided by the other applications, and thus they do not play strict roles of clients or servers. The SNs of the peer applications use a service-agreement protocol to arrive at a mutually acceptable service agreement.

A mapping module within the SN maps the service parameters that it has negotiated with its peers to the appropriate service zone. The mapping rules have to be supplied by the application developer, although for the standard components used, there may be automated tools to generate customized mapping rules for applications. The selected operating zone, obviously, corresponds to the component configurations that are able to satisfy the service commitments of the application.

2.2 The Contract-based Adaptation System (CAS)

The Contract-based Adaptation System (CAS), which is part of the CASA run-time system, is a standard application-independent entity that is responsible for carrying out dynamic adaptation on behalf of its associated application. The CAS submits resource requests of its associated application – as specified in the application contract corresponding to the selected operating zone – to the underlying Contract Enforcement System (CES). If there is a mismatch between the resources requested by an application and those that can be made available to it, the CAS carries out dynamic adaptation of the application by replacing the current component configuration with the one that has resource requirements compatible with the available resources.

While carrying out dynamic adaptation, the CAS takes into account the need for state transfer between components of the same type. Moreover the adaptation is carried out without taking down the system and the integrity of existing transactions is maintained.

2.3 The Contract Enforcement System (CES)

The Contract Enforcement System (CES) is also a part of the CASA run-time system but, unlike the CAS, the CES is a central entity responsible for satisfying resource requirements of all applications running on its host node. The CES is kept up to date about the current resource status by the Resource Manager (RM), and is responsible for making resource allocation decisions in order to satisfy resource requirements of requesting applications. In making resource allocation decisions, the CES needs to take into account the relative priorities of the various requesting applications, particularly when there are not enough resources to satisfy the requirements of all the applications.

If – initially or at anytime during the execution life of an application – there is a mismatch between resources requested by the CAS (on behalf of its associated application) and those that can be allocated to it, the CES informs the CAS about the resource requirement-availability mismatch, also specifying the values of resources that can be allocated to it. The mismatch may occur because of scarcity or abundance of resources. Resources might become scarce due to increased load or resource failures, with the result that the currently available resources are not sufficient enough to meet the demands of all requesting applications. Resources might become abundant because of reduced load or restoration of some resources that failed earlier, with the result that more resources can be allocated to an application than specified in the resource request. A resource request submitted by a CAS specifies the range of desired values for each resource, as well as options to inform the CAS about any possible degradation below the desired value and/or improvement above the desired value for each resource. The ranges of acceptable values for the individual resources, specified in the resource request, allow the CES the flexibility to operate within the given ranges. The CES dynamically adjusts resource allocations within specified resource ranges, based on the current actual resource usage patterns of applications. In effect, the

CES provides a first level of absorption mechanism in the event of degradation in the resource availability by reallocating existing resources among applications so that high priority applications can carry on with their execution without much interruption, and only the low priority applications need to be adapted.

2.4 The Resource Manager (RM)

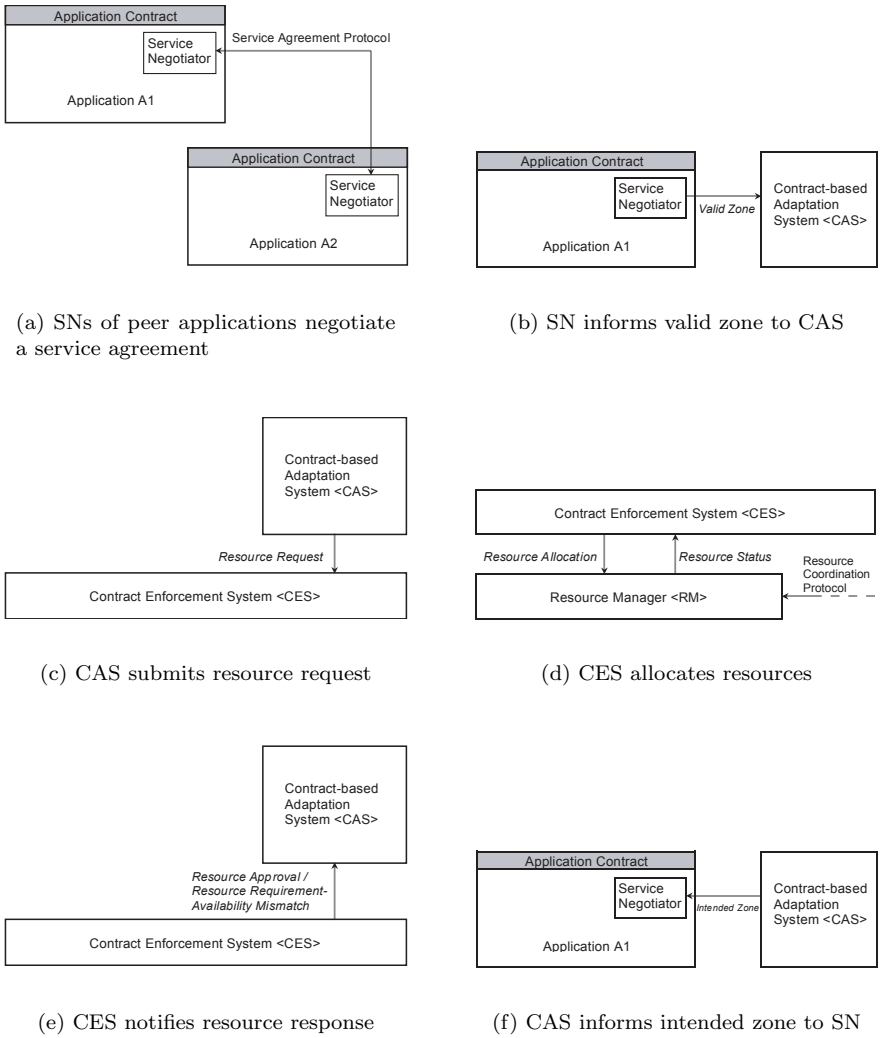
The Resource Manager (RM) monitors the value and availability of resources and keeps the CES updated about the current resource status. Monitoring the changes directly at the resource level shortens the turnaround time from the change in resource conditions to adapting the application, as compared to when measuring the actual level of service being available at the receiving end, before taking an adaptation decision. However, to avoid a premature reaction to temporary fluctuations in the availability of a resource, the RM may suitably delay updating the CES. Resources include memory, processor occupancy, communication budget etc. The RM carries out reservation of resources as governed by the resource allocation decisions of the CES. It further ensures that applications operate within their allocated resource limits, so that resources allocated to an application are protected. For distributed resources, the RM works in coordination with the RMs of other participating nodes using a resource-coordination protocol.

3 Working of CASA

The Service Negotiators (SNs) of peer applications negotiate a service agreement using a service-agreement protocol (cf. Figure 3a). The mapping modules, present in each of the SNs, then map the service parameters agreed upon to the appropriate operating zones for each application. The SN of each participating application then sends the information about the negotiated operating zone to the Contract-based Adaptation System (CAS) associated with the application (cf. Figure 3b). Based on the given operating zone, the CAS looks up the first component configuration specified in the application contract under that zone, and forwards the resource requirements corresponding to this configuration as a resource request to the Contract Enforcement System (CES) (cf. Figure 3c).

Based on the resource request submitted by the CAS and the resource status updated by the RM, the CES makes resource allocation decisions (cf. Figure 3d). If the resource request can be satisfied, the CES notifies the resource approval to the CAS and instructs the RM to reserve the required resources. But if (initially or at anytime during the execution life of an application) there is a mismatch between the resource request and resource availability, CES triggers CAS about the resource requirement-availability mismatch (cf. Figure 3e).

If the resources requested by the CAS are allocated (indicated by resource approval), it activates the corresponding component configuration. Otherwise, if the CAS is informed by the CES about a resource requirement-availability mismatch, the CAS looks for an alternative configuration that has resource requirements compatible with the available resources within the current zone in



(a) SNs of peer applications negotiate a service agreement

(b) SN informs valid zone to CAS

(c) CAS submits resource request

(d) CES allocates resources

(e) CES notifies resource response

(f) CAS informs intended zone to SN

Figure 3: Working of CASA

the application contract. If the CAS finds one, it switches to the new configuration and renews its resource request to the CES. In case the CAS cannot find an alternative configuration within the current zone, it searches for one in other zones. If the CAS finds a suitable configuration in another zone that has resource requirements matching the resource availability, it sends information to the SN about the zone number of the new configuration, to which it intends to switch (cf. Figure 3f).

The SN, in turn, sends information to its peer applications about the new

level of service that it can offer (obviously, the mapping module first reverse maps the operating zone to service parameters). Once the new service level is approved, SN gives the signal to the CAS to go ahead with switching to the new configuration and renew its resource request to the CES.[1] Since the new configuration may differ from the old one in just a few components, the CAS activates only those components that are new and passivates the components that are not required in the new configuration. Since components of the same type conform to the same interface, the remote application code interacting with the old components will normally not be affected by the adaptation. However, the peer applications may also need to adapt (by changing their operating zones) based on the changes in service level being offered. Note that while switching between configurations within the same zone, there is no need to inform the SN or the peer applications. This is because alternative configurations within the same zone offer almost the same level of service. A crucial factor in deciding granularity of operating zones is how much the difference in service level is considered minor and acceptable (grouped under same zone), beyond which the peer applications need to be informed (grouped in different zones).

If the peer applications do not approve the new service level being offered, or the CAS is unable to find any appropriate component configuration in response to the mismatch trigger (which means none of the configurations specified in the application contract matches the existing resource conditions), the application will need to be suspended or terminated. An executing application may also need to change its operating zone voluntarily, for example due to a change in user's preference or due to some performance considerations etc. In this case, SN first informs its peer applications about its intended change to the new service level; and on approval, informs the CAS about the new zone number to switch to. The rest of the cycle proceeds similar to the one when an initial request is made. Similarly, when an application is informed by its peer application about a change in the service level provided by the latter to the former, the former may need to change its operating zone as well, to match the new service conditions.

4 Contract Specification

Application contracts and other informational entities required to support the CASA framework are specified in the Contract Specification language (CSL). CSL is an XML-based specification language developed as part of the CASA framework. The reason for using CSL is to specify application contracts and other informational entities in a standard and uniform manner that is independent of the application implementation language or platform. This uniformity helps to achieve transparency in dynamic adaptability of applications.

First, we have to define ranges for all managed resources.

[1] For certain applications, it may not be required to inform peer applications explicitly about a change in the service level being offered. For such applications, SN can simply confirm the CAS to go ahead with the change, without initiating the process of informing peer applications and seeking their approval.

```
<resource_range>
   <resource name=R1 unit=M1>
      <value from=V1 to=V2 range=1/>
      <value from=V2+1 to=V3 range=2/>
      .
      .
   </resource>
   <resource name=R2 unit=M2>
      .
      .
   </resource>
   .
   .
</resource_range>
```

Figure 4: Resource Range Table

```
<app_contract app_name=A'>
   <zone number=1>
      <config number=1 comps=C1,C2,…,Ck
          resource_tuple=(N1)(N2)…(Nn)/>
      .
      .
   </zone>
   <zone number=2>
      .
      .
   </zone>
   .
   .
</app_contract>
```

Figure 5: Application Contract

Resource Range Table: An example of the format of resource range table is given in Figure 4. The purpose of the resource range table is to distribute possible values of each resource appropriately into ranges, and allocate a range number for each such range. The resource range table is needed for calculating Resource Tuples (see below).
Resource Tuple: The resource requirements of each component configuration are specified in terms of a resource tuple. A resource tuple is a sequential representation of desired values of resources, where the desired values are expressed in terms of range numbers from the resource range table.
Application Contract: An example of the format of the application contract is given in Figure 5. As is clear from the format, an application contract is divided into zones, identified by their unique numbers. Each zone in turn consists of a list of alternative component configurations and their corresponding resource requirements, specified in terms of resource tuples. Each component configuration is specified as a list of its constituent components.

For a detailed description of these and other informational entities, please refer to [4].

5 Related Work

Recently there have been several attempts to enhance middleware services to account for the QoS requirements of applications. Real-Time CORBA [7], and its implementation in TAO [8], focus on achieving end-to-end predictability for the real-time CORBA applications. However, it provides no means for an application to explicitly negotiate its resource requirements or to adapt its behavior. Work on Quality Objects (QuO) [10] extends CORBA to provide QoS for CORBA object invocations. Although QuO mentions a broad range of application-specific QoS parameters, it does not offer an integrated framework for handling all QoS parameters of interest in a unified way.

There are a few other approaches that attempt to control QoS at the middleware or system level, such as the Odyssey architecture [6] and Reflective Middleware [1]. But they do not offer any mechanisms for reconfiguring the applications themselves in case of changed execution environment. Work on Adaptive Resource Allocation (ARA) [9] provides models and mechanisms to

enable adaptive resource allocation for the applications with dynamically changing resource needs. Some other approaches are restricted to providing efficient resource management techniques in order to satisfy QoS requirements, including the Globus Architecture for Reservation and Allocation (GARA) [3] and the Darwin project [2].

The approach advocated by the $2K^Q$ system [5] talks about functional adaptation in response to QoS changes, and it shares the same goals as our CASA framework. However, it provides a centralized control over adaptation policies for the complete distributed system, whereas in CASA the applications at every discrete node can adapt individually, as per their own adaptation policies. Since self-organized mobile networks consist of autonomous nodes that form ad-hoc networks, independence in deciding an application's own adaptation policies is significant. For a detailed discussion on related approaches see [4].

6 Concluding Discussion and Future Work

The CASA framework enables dynamic adaptation of applications in response to changes in their execution environment. Dynamic adaptation is achieved through runtime reconfiguration of the components of an application, according to the adaptation policy specified in the application contract. Adaptive applications are able to best meet their functional and/or performance commitments even in dynamically changing environments, and thus have a longer execution life than their non-adaptable counterparts.

A limitation of our approach is that it places a lot of responsibility on the application developer in developing alternative component configurations for the application to suit different resource conditions that can arise during the execution life of the application, and in generating the application contract accordingly. In doing so, an application developer may have to take difficult trade-off decisions in deciding between various alternatives. But it also provides enough flexibility to the application developer to tailor the adaptation policy of an application specific to its needs. Moreover with standard COTS components widely used for realizing applications, automated tools for analyzing alternative component configurations and generating application contracts are envisaged, to make the application developer's job easier. Of course, the actual effort spent on making an application adaptive, and the resulting amount of adaptation, would depend upon factors such as criticality and re-usability of the application.

The framework provides the possibility to extend the adaptation structure later by widening the scope of reconfiguration, as the need arises. And it offers an integrated support to handle a wide range of service parameters across different application domains. Moreover, the application independent characteristic of the CASA run-time system helps to achieve dynamic adaptability in a transparent manner. Thus it expands the applicability of the framework to all kinds of applications that are faced with the challenge of unreliable resource availability, and have alternative component configurations to offer in the face of it.

CASA is still in its evolutionary phase. We are currently developing the de-

tails of the distributed service-agreement protocol and the resource-coordination protocol. We are also developing a priority model for various contending applications to help CES make justified resource allocation decisions. The priority model will also include some kind of charging mechanism whereby applications may be charged for reserving the resources. In addition, we are working on the implementation of a prototype to study feasibility and performance issues related to the CASA framework.

References

[1] L. Capra, W. Emmerich and C. Mascolo. Reflective Middleware Solutions for Context-Aware Applications. *Proc. of 3rd Intl. Conference on Metalevel Architectures and Separation of Crosscutting Concerns*, 2001.

[2] P. Chandra, A. Fisher, C. Kosak, T.S. Eugene Ng, P. Steenkiste, E. Takahashi and H. Zhang. Darwin: Customizable Resource Management for Value-Added Network Services. *Proc. of 6th Intl. Conference on Network Protocols*, 1998.

[3] I. Foster, A. Roy and V. Sander. A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation. *Proc. of 8th Intl. Workshop on Quality of Service*, 2000.

[4] A. Mukhija and M. Glinz. *CASA - A Contract-based Adaptive Software Architecture Framework*. Technical Report, IFI, University of Zurich, 2003. `http://www.ifi.unizh.ch/groups/req/ftp/papers/CASA2003.pdf`

[5] K. Nahrstedt, D. Wichadakul and D. Xu. Distributed QoS Compilation and Runtime Instantiation. *Proc. of 8th Intl. Workshop on Quality of Service*, 2000.

[6] B.D. Noble, M. Satyanarayanan, D. Narayanan, J.E. Tilton, J. Flinn and K.R. Walker. Agile Application-Aware Adaptation for Mobility. *Proc. of 16th ACM Symposium on Operating System Principles*, 1997.

[7] Object Management Group. *Real-Time CORBA Specification*, 2002. `http://www.omg.org/`

[8] I. Pyarali, D.C. Schmidt and R.K. Cytron. Techniques for Enhancing Real-Time CORBA Quality of Service. *IEEE Spl. Issue on Real-Time Systems*, 2003.

[9] D.I. Rosu, K. Schwan, S. Yalamanchili and R. Jha. On Adaptive Resource Allocation for Complex Real-Time Applications. *Proc. of 18th IEEE Real-Time Systems Symposium*, 1997.

[10] J.A. Zinky, D.E. Bakken and R.E. Schantz. Architectural Support for Quality of Service for CORBA Objects. *Theory and Practice of Object Systems*, Vol. 3(1), 1997.