A RESOURCE ADAPTATION FRAMEWORK FOR REFLECTIVE MIDDLEWARE

Nikos Parlavantzas, Geoff Coulson, Gordon Blair

Distributed Multimedia Research Group, Computing Department, Lancaster University, LA1 4YR,UK

contact: [parlavan, geoff, gordon]@comp.lancs.ac.uk

ABSTRACT

It is now well established that next generation middleware platforms must facilitate the management of QoS. Supporting resource adaptation is useful for QoS management, but this issue is not adequately addressed by current adaptive and reflective middleware architectures. The paper describes a framework for supporting resource adaptation by providing first-class representation of activities and generic interfaces for inspecting and controlling the resources allocated to activities. The paper also discusses the application of the framework in our reflective middleware architecture as well as in the .Net environment.

1. Introduction

Middleware platforms aim to simplify the development of distributed applications by resolving distribution and heterogeneity problems. Although mainstream middleware technologies have been successful in the domain of enterprise applications, it is now widely recognized that they fall short of supporting applications with quality of service (QoS) requirements, such as timeliness and capacity requirements. The important characteristic of such requirements is that their satisfaction depends on unpredictable and changing environmental conditions.

Managing QoS properties in the face of environmental variations demands that the middleware platform is highly adaptable. For example, the platform may support the selection of protocols that suit the needs of a particular network. One desirable form of adaptation is *resource adaptation*; that is, providing facilities for both monitoring and controlling the resource usage of activities within the system. Information on resource usage is useful for making QoS management decisions; for example, knowing the available network bandwidth can drive the selection of an appropriate audio compression algorithm. Controlling resource usage is useful for realizing QoS management decisions; for example, increasing the CPU time allocated to processing requests from a specific client can decrease the response time experienced by the client.

Significant progress has been made recently in the design of QoS-aware and reflective middleware platforms, which can be flexibly modified to accommodate environmental variations [Kon,02][Blair,98]. However, the issue of resource adaptation has received little attention. As a result, middleware and QoS programmers have to rely on ad-hoc, resource-specific interfaces in order to perform resource adaptation. In our opinion, a simpler and more systematic solution is needed. To this end, we are proposing a framework that simplifies resource adaptation by supporting first-class representation of activities and providing uniform interfaces for inspecting and controlling the resources allocated to activities. The framework is a part of OpenORB [Coulson,02a], our reflective component-based middleware architecture, but it is designed to be generally applicable. Indeed, we are currently applying the framework within the .Net remoting architecture. Note that our framework builds on and generalises our earlier work on resource adaptation, which was tightly bound to a previous OpenORB implementation [Duran,02].

The rest of the paper is structured as follows. First, section 2 presents the design of the resource framework together with an example of its use. Following this, section 3 discusses the application of the framework in both OpenORB and .Net remoting. Subsequently, section 4 discusses related work, and the last section presents our conclusions.

2. Resource framework

2.1 Goals

The goal of the resource framework is to support resource adaptation. In refining this goal, two additional requirements have been identified. First, the framework must be *extensible* to capture diverse types of resources at different levels of abstraction, such as memory resources, processing resources (e.g., threads, virtual processors), communication resources (e.g., network bandwidth, transport connections) and device

resources (e.g., disks). Second, the framework must provide *maximum control* to its users with regards to resource adaptation; this flexibility is necessary for increasing the applicability of the framework.

2.2 Resource model

The framework relies on a generic resource model that describes the concepts of *resources* and *tasks* and how they are related. Specifically, a *resource* is a runtime entity that offers a service for which one needs to express a measure of quality of service; the need for specifying quality of service reflects an underlying bounded quantity (e.g., bounded memory capacity) [OMG,02]. A *task* is defined as the unit of computation to which resources are allocated and resource usage is charged. Note that the task concept is orthogonal to concurrency concerns and a task may be associated with one or more threads of control. Tasks are also orthogonal to the component-based structure of the system.

Tasks are dynamic entities and are organised into a hierarchy whereby a task is created in the context of a parent task; the hierarchy serves to delineate computations/resources that contribute to the same goal. Resources are also organized into hierarchies whereby a higher level resource builds on lower-level resources (e.g., a user-level thread builds on kernel-level threads, which build on processors). A resource is associated with a single task but the association is dynamically controllable. For instance, a thread (resource) can be transferred to a different task to reflect the fact that it performs work contributing to another goal.

2.3 Resource framework overview

The resource framework defines the following roles played by participating components: A *resource manager* creates *resources* of a given type, and a *task manager* creates and terminates *tasks*. *Resource users* access and manipulate resources and tasks. Moreover, the resource framework defines rules that govern the cooperation between these roles. Specifically, resource managers have the responsibility to enforce resource control and task-based allocation and accounting. In addition, they must keep track of the dynamic associations between resources and tasks. The task manager acts as the access point for obtaining resource allocation information and also offers support functionality.

By assigning most of the responsibilities to resource managers, this design encapsulates the large diversity in resource types, resource management mechanisms, and policies, and satisfies the extensibility requirement. By directly exposing tasks and resources to resource users through low-level primitives (see next section), this design also satisfies the maximum control requirement. For instance, the design is not concerned with hiding task boundaries or imposing a fixed component-to-task mapping (e.g., that a component always runs in a single task). Such constraints are added on top of the framework together with appropriate support functionality (e.g., support for task switching).

2.4 Resource framework details

A structure diagram that shows the roles and interfaces defined by the framework is shown below. The following sections provide more details on the roles and interfaces.
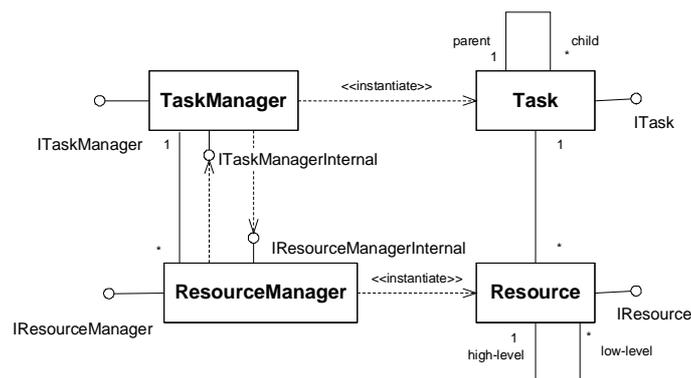
2.4.1 Resource Manager

A resource manager exposes the IResourceManager interface to resource users and the IResourceManagerInternal interface to the task manager. It uses the ITaskManagerInternal interface of the task manager.

```
interface IResourceManager: IUnknown {
        HRESULT Allocate ([in] OLECHAR* spec, [out] IResource**);
        HRESULT AllocateToTask ([in] OLECHAR* spec , [in] ITask* task, [out] IResource**);
        HRESULT GetLowerResourcesOf ([in] IResource high*, [out, size_is(,*max)] IResource**,
```

```
                [out] long* max);
                HRESULT GetManagedRersources ([out, size_is(,*max)] IResource**, [out] long* max);
}
```

IResourceManager offers a generic operation for resource allocation (Allocate) that takes as an argument a resource specification and returns a resource object (see below). The resource is implicitly allocated to the task within which the client executes (i.e., the task associated with the invoking thread). The AllocateToTask operation is similar but explicitly accepts a task object as an argument. IResourceManager also offers operations to return the set of managed resources as well as the set of lower-level resources that a given managed resource builds upon. Resource managers will normally offer supplementary, resource-type-specific interfaces that expose additional functionality.

The IResourceManagerInternal interface provides operations to return the resource type and the resources associated with a given task. Note that according to the framework, the resource manager is responsible for maintaining the association between allocated resources and tasks. Resource managers can obtain information about a task's ancestors using the ITaskManagerInternal interface. This information is useful to realise various allocation policies. For example, when a task does not have a necessary lower-level resource to support a new resource creation, the resource manager may search for the required resource in one of the task's ancestors. As abnother example, the resource manager may realise a hierarchical resource allocation scheme, whereby the resource allocation of a child task is constrained by the allocation of its parent (e.g., the resource manager deducts an amount of resources from the parent task and adds it to the child).

2.4.2 Resource

A resource represents a resource allocation performed by a resource manager and is associated with a specific task. It minimally offers the IResource interface with operations to return the associated resource manager and task. IResource also has generic operations to release the resource, get/set the associated QoS of the resource and return the higher-level resource manager that is currently using the resource (if applicable).

```
interface IResource: IUnknown {
                HRESULT GetResourceManager([out] IResourceManager**);
                HRESULT GetTask ([out] ITask**);
                HRESULT Release();
                HRESULT SetQoS( [in] OLECHAR* requiredQOS);
                HRESULT GetQoS( [out] OLECHAR** offeredQoS);
                HRESULT GetHigherResourceManager ([out] IResourceManager** );
                HRESULT TransferTo ( [in] ITask*);
}
```

Resources can be transferred from the current task to another task with the TransferTo operation. Resource managers may enforce resource type-specific policies with regards to resource transfers (e.g., a resource may be transferred only to child tasks).

2.4.3 Task Manager

The task manager exposes the ITaskManager interface to both resource users and resource managers, and the ITaskManagerInternal interface only to resource managers. It uses the IResourceManagerInternal interface of resource managers.

```
interface ITaskManager: IUnknown {
                HRESULT CreateTask( [out] ITask**, [in] OLECHAR* taskName );
                HRESULT GetCurrentTask( [out] ITask** );
                HRESULT SetCurrentTask( [in] ITask*);
}
```

The CreateTask operation creates a new task with an optional name as a child of the current task, i.e., the task associated with the invoking thread. The new task is initially empty of resources. The GetCurrentTask operation returns the task associated with the current thread. The SetCurrentTask migrates the current thread to a new task; this is realised by invoking the TransferTo operation on the current thread.

The ITaskManagerInternal interface offers operations for navigating the task hierarchy and is used by resource managers to realise task-based allocation policies (e.g., hierarchical resource allocation). Note that this interface should not be exposed to normal resource users in order to enforce the safety constraint that each resource user has access only to its current task or its children.

2.4.4 Task

A task offers the ITask interface. The GetResources operation takes as an argument the resource type and returns the resource objects of the specific type that are allocated to this task. The task manager implements this operation by forwarding to the IResourceManagerInternal interface of the appropriate resource manager. The Terminate operation releases the allocated resources and deletes the task. Importantly, the children of the task are also recursively terminated. For safety reasons, the Terminate operation should only be called from the

context of the same task or its parent. Finally, the GetChildren operation allows navigation to the children of this task.

```
Interface ITask: IUnknown {
        HRESULT GetResources( [in] OLECHAR* resourceType, [out, size_is(,*max)] IResource**, [out] long *max);
        HRESULT Terminate();
        HRESULT GetChildren( [out, size_is(,*max)] ITask**, [out]long* max);
}
```

2.5 Example of framework use

Consider a middleware platform that supports a QoS-aware audio streaming service (e.g., a CORBA A/V Streams implementation or an OpenORB media streaming binding type). When a binding is about to be established at the audio consumer site, a *binding factory* component carries out static QoS management functions (QoS mapping, QoS negotiation, admission control) in order to decide on appropriate resource allocations (e.g., in terms of memory, CPU processing time and network bandwidth). Next, the factory creates a new task t1, allocates the appropriate resources to it and instantiates a *local binding* component that is configured to execute within task t1. The local binding, in turn, creates a socket, a media filter graph for processing audio (e.g., containing a decompressor and a renderer) and a thread for receiving data from the socket and passing them to the filter graph. Graph construction and use incur further resource usage, which is transparently charged to t1 (e.g., the renderer allocates buffers for storing incoming audio data and creates a periodic thread for writing data to the device).

Dynamic QoS management functions (QoS renegotiation, QoS adaptation) are then performed by a *manager* component, which interacts with the local binding using both a local binding-specific management interface and the generic resource framework API. The former provides operations to perform parametric (e.g., changing audio formats) and compositional modifications (e.g., changing the filter graph configuration) on the local binding and emits related events (e.g., "high packet loss" event). The latter allows inspection and modification of the resource configuration associated with t1. For instance, it allows the manager to inspect the amount of CPU time received by the binding or to receive a "memory use exceeded" event when an imposed memory limit is reached. Moreover, it allows the manager to dynamically change network reservations or priorities of individual threads. Note that the uniformity of the resource API allows the manager to control the behaviour of independently developed and dynamically combined components (e.g., media filters). The combined use of the two management interfaces enables a wide variety of "resource-aware" management policies to be provided. Importantly, this design decouples resource allocation and adaptation (handled by the binding factory and the manager respectively) from the binding's functional, resource-using behaviour (encapsulated in the local binding component), which simplifies the development of the latter.

The framework enables similar managers to be positioned at every hierarchy level of a platform. Indeed, due to the genericity of the resource framework, one can develop managers with no hard-wired knowledge of middleware internals, which can monitor resource utilisation and even perform unanticipated, dynamic changes. For instance, such a generic manager could redistribute resources among tasks to reflect changes in the importance of application activities.

3. Framework applications

OpenORB [Coulson,02a][Parlavantzas,02] is a middleware architecture that supports the development of a wide range of configurable and reconfigurable middleware platforms, each targeting different application domains and underlying infrastructures (e.g., platforms for real-time, collaborative, or multimedia applications deployed on workstations or PDAs). The role of the resource framework in this architecture is to offer a uniform and simple interface for resource adaptation. The extensibility of the framework with regards to resource types is particularly useful since platforms have diverse resource management needs. For instance, a platform for mobile computing may need to manage a battery life resource along with other traditional resources.

The current OpenORB platform contains resource managers that expose the following types of resources: scheduler contexts, user-level threads, memory contexts, buffers, network contexts, and transport service access points (tsaps). A *scheduler context* is a resource that represents an allocation of CPU processing time. *User-level threads* are resources that build on scheduler contexts. Specifically, a scheduler context's processing time is distributed among a number of threads according to a scheduler context-specific scheduling policy (e.g., earliest deadline first). A thread is mapped to a single scheduler context at any given time but the mapping may change. Similarly, a *memory context* represents an allocation of memory, which is distributed among a number of *buffers*. Finally, a *network context* represents a logical share of access to the network from the point of view of the host process; *tsaps* are resources that build on network contexts.

With regard to tasks, the current platform employs a separation into *middleware tasks* and *application tasks*. The former are associated exclusively with the middleware implementation and represent shared activities, such as book-keeping, event logging, or dynamically linking a plug-in. Application tasks are associated with middleware users. Importantly, when an application component invokes a middleware service, the middleware implementation spawns one or more subtasks of the application task to represent middleware processing performed for the benefit of the application. Those subtasks encompass the necessary resources to support the requested service (e.g., sockets and buffers needed for establishing a binding to a remote object). Note that the creation of multiple tasks has only a small performance overhead due to the use of a lightweight task-switching mechanism. This involves migrating a thread to a different task without rescheduling and trades off precision of resource accounting for efficiency.

Apart from OpenORB, the resource framework is also been applied to the .Net remoting system, a middleware platform for supporting distributed objects [Microsoft,02]. The platform provides numerous extensibility points, such as pluggable channels and formatters, message filters and custom properties, but it does not offer any special support for resource adaptation. To address this limitation, we are extending the resource management services exposed by the .Net infrastructure to enable them to participate in the resource framework. Specifically, we are currently focusing on thread and file management services as realized within the Shared Source CLI implementation. At the same time, we are extending the remoting system to take advantage of the resource framework. Specifically, we are relying on the built-in extensibility points to inject code that creates tasks and thus enables the uniform resource adaptation interfaces to be used. The goal is to the support the development of a QoS-aware .Net remoting system, which will better address the timeliness and predictability needs of .Net distributed applications.

4. Related Work

The framework builds on previous research on reflective resource management at Lancaster University [Duran,02]. This work focused on high-level analysis and design of resource management in distributed systems and provided support in terms of architecture description languages. Tasks are regarded as invocation sequences and can span address spaces. The framework proposed in this paper focuses instead on simplicity and wide applicability and assumes higher-level features are implemented separately; moreover, tasks are independent of concurrency concerns.

Generic abstractions for resource management are provided by many flexible middleware platforms, such as FlexiNet [Hayton,98], TAO [Schmidt,99], and 2K/dynamicTAO [Kon,01]. However, these platforms do no provide generic support for resource inspection and control. QuO [Zinky,97] and CIAO [Wang,03] are QoS-enabled adaptive middleware platforms that promote a separation of concerns between application logic and QoS provisioning logic. DotQos [Weis,03] is an ongoing project that aims to enable QoS provision on top of .NET's remoting infrastructure. Research such as QuO, CIAO and DotQos can benefit from the proposed resource framework because the uniform resource adaptation interfaces contribute to the composability and reuse of QoS provisioning logic (cf. the manager object in section 2.5).

The task concept is similar to a number of proposed OS abstractions for resource ownership, such as *resource containers* [Druschel,99] and Rialto *activities* [Jones,95]. However, those approaches target exclusively kernel-level resources, and there is no support for extensibility with regards to resource types. The need for resource monitoring and control has also been recognized in the context of the Java platform. JRes [Czajkowski,98] is a resource control library for Java that supports per-thread resource accounting and limiting. J-SEAL2 [Binder,01] is a secure mobile agent system that provides a resource control API for a fixed set of resource types (memory and CPU resources). RAJE [Guidec,02] reifies an extensible set of system and higher-level Java resources but does not currently provide a task abstraction.

5. Conclusions

In this paper, we have presented a resource framework that aims to simplify resource adaptation within middleware architectures. Specifically, the framework provides a uniform adaptation API based on the abstraction of a task, a part of the system's computation whose resource usage can be monitored and controlled independently. The API enable users to inspect and control the distribution of resources among different activities and can thus be viewed as a meta-interface (or, MOP) for reifying resource allocation. The use of low-level, generic interfaces supports extensibility with regards to resource types and increased applicability while maintaining understandability and ease of use. Moreover, we have outlined the application of the framework within the OpenORB reflective middleware architecture and discussed ongoing work to apply the framework in .Net remoting. We believe that the framework provides significant support for building resource-aware middleware systems. To validate this claim, we have additional plans to apply

the framework in both reflective middleware for programmable networking environments [Coulson,02b] and flexible Grid platforms.

References

[Binder,01] Binder, W., Hulaas, J., Villazón, A., and Vidal, R., "Portable Resource Control in Java: The J-SEAL2 Approach", ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA-2001), Tampa Bay, Florida, USA, October 14-18, 2001.

[Blair,98] Blair G.S., Coulson G., Robin P. and Papathomas M., "An Architecture for Next Generation Middleware", Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'98), Davies N.A.J., Raymond K. & Seitz J. (Eds.), The Lake District, UK, pp. 191-206, 15-18 September 1998.

[Coulson,02a] Coulson, G., Blair, G.S., Clarke, M., Parlavantzas, N., "The Design of a Highly Configurable and Reconfigurable Middleware Platform", ACM Distributed Computing Journal, Vol 15, No 2, pp 109-126, April 2002.

[Coulson,02b] Coulson, G., Blair, G.S., Hutchison, D., Ueyama, J., Ye, I., Lee, K., and Surajbali, B., "NETKIT: A SoftwareComponent-Based Approach to Programmable Networking", Computing Dept. Internal, Report, Lancaster University, 2002.

[Czajkowski,98] Czajkowski, G., and von Eicken. T., "JRes: A resource accounting interface for Java". In OOPSLA-98, pages 21-35, New York, USA, Oct. 18-22 1998. ACM Press.

[Druschel,99] Druschel, P., Banga, G., and Mogul, J.C., "Resource Containers: A New Facility for Resource Management in Server Systems", Proc. Third Symposium on Operating Systems Design and Implementation (OSDI '99), New Orleans, LA, February 1999.

[Duran,02] Duran-Limon, H., Blair, G.S., "Reconfiguration of Resources in Middleware" In the 7th IEEE International Workshop on Object-oriented Real-time Dependable Systems (WORDS 2002), San Diego, CA, January, 2002.

[Guidec,02] Guidec F., Le Sommer, N., "Towards resource consumption accounting and control in Java: a practical experience". ECOOP'2002, (Workshop on Resource Management for Safe Language).

[Hayton,98] Hayton, R., Herbert, A., and Donaldson, D., "Flexinet: a flexible, component oriented middleware system", Proceedings of the 8th ACM SIGOPS European Workshop: Support for Composing Distributed Applications, Sintra, Portugal, 7-10 September 1998.

[Jones,95] Jones, M.B., Leach, P.J., Draves, R.P., Barrera, J.S., "Modular Real-Time Resource Management in the Rialto Operating System," Proc. of the Fifth Workshop on Hot Topics in Operating Systems, May 1995.

[Kon,01] Kon, F., Yamane, T., Hess, C., Campbell, R., and Mickunas, M.D., "Dynamic Resource Management and Automatic Configuration of Distributed Component Systems", Proceedings of the 6th USENIX Conference on Object-Oriented Technologies and Systems (COOTS'2001), San Antonio, Texas, January, 2001.

[Kon,02] Kon, F., Costa, F., Campbell, R., and Blair, G., "The Case for Reflective Middleware", Communications of the ACM., Vol. 45, No. 6, pp. 33-38. June, 2002.

[Microsoft,02] Microsoft, .Net Home Page, http://www.microsoft.com/net

[OMG,02] Object Management Group, UML Profile for Schedulability, Performance and Time final adopted specification, OMG Document ptc/02-03-02

[Parlavantzas,02] Parlavantzas, N., Coulson G., Blair, G.S., "An approach to building reflective component-based middleware platforms", MSRC Summer Research Workshop, Cambridge, U.K., September 9-11, 2002

[Schmidt,99] Schmidt, D.C., and Cleeland, C., "Applying Patterns to Develop Extensible ORB Middleware", IEEE Communications Magazine Special Issue on Design Patterns, April, 1999.

[Wang,03] Wang, N., Schmidt, D., Gokhale, A., Gill, C.D., Natarajan, B., Rodrigues, C., Loyall J.P., and Schantz, R., "Total Quality of Service Provisioning in Middleware and Applications", Microprocessors and Microsystems, special issue on Middleware Solutions for QoS-enabled Multimedia Provisioning over the Internet (Paolo Bellavista ed.), vol. 27, no. 2, pp. 45-54, March 2003.

[Weis,03] Weis, T., Ulbrich, A., Geihs, K., DotQoS project website: http://www.dotqos.org/

[Zinky,97] Zinky JA, Bakken DE, Schantz R. "Architectural Support for Quality of Service for CORBA Objects", Theory and Practice of Object Systems, Jan 1997.