Quintet, Tools for Reliable Enterprise Computing

Werner Vogels, Dan Dumitriu, Mike Pantiz, Kevin Chipawolski, Jason Pettis

Department of Computer Science, Cornell University[†]

vogels@cs.cornell.edu

Abstract

This paper describes Quintet, a system for developing and managing reliable enterprise servers. Quintet provides tools for the distribution and replication of server components to achieve guaranteed availability and performance. It is targeted to serve the application tier in multi-tier business systems, with components constructed using Microsoft COM. Quintet takes a radical different approach from previous systems that support object replication, in that replication and distribution are no longer transparent and are brought under full control of the developer.

1 Introduction

In corporate settings the general enterprise computing systems are becoming more and more organized as distributed systems. These systems are critical to the corporate operation and a strong need arises for making these systems highly reliable. The first step in addressing these needs has been taken by industry: based on their experiences with dedicated cluster environments, new cluster management software has been developed that targets off-the-shelf enterprise server systems. In general commercial cluster products provide functionality for the migration of applications from failed nodes to surviving nodes in the system. Although this offers some relief for systems such as web servers, databases or electronic mail processors, it does not facilitate the development of systems that capable of exploiting the cluster environments in all its potential.

A new research project at the Reliable Distributed Systems group at Cornell addresses the problems of building reliable enterprise systems. The project, dubbed Quintet, focuses on development and runtime support for components that make up the application tier of multi-tier business systems. In our target systems this layer is constructed out of servers build as collections of COM components. Components developed using the tools provided by Quintet are able to guarantee reliable operation in a number of ways, and the system is extensible in that new mechanisms and interfaces can be added.

The project is concerned with research into two areas: in the first area the quest is for what kind of development tools are needed to build reliable distributed components for enterprise computing, with a focus on efficiency, simplicity and ease of use. The second research area concentrates on the infrastructure needed for reliability management on the high performance cluster systems providing the component runtime environment.

This paper first provides some background on the way Quintet views issues surrounding reliability and distribution transparency. This is necessary to understand the design choices that have been made. The section following this provides an overview of Quintet's functionality and the solutions that can be build with the Quintet tools. After a description of the target environment and relation between Quintet and MTS, the paper describes in detail, the major components that make up Quintet.

2 Reliability

Component reliability in Quintet addresses two aspects of distributed computing: high-availability and scalable performance. The first is concerned with that given a limit to the number of node failures, the system guarantees that the remaining set of nodes continues to provide the required functionality. The second aspect ensures that the system, using adaptive methods, distributes the load over available resources to guarantee optimal performance.

Reliability in Quintet is described using a Quality of Service specification. When a new component is added to the system, the administrator describes the reliability requirements of the component, which are input for the runtime system and for the component class factories. The specification can be changed on-line and the system can be requested to reconfigure accordingly.

The most obvious approach for providing high-availability and scalable performance is to replicate components over several server nodes and to provide client fail-over and load balancing to achieve the

[†] Quintet is part of the research performed by the Reliable Distributed Computing Group at Cornell, and is supported by DARPA/ONR under contract N0014-96-1-10014 and by Intel Corporation and Microsoft Corporation.

reliability goals. Although this is an approach that certainly can be used in Quintet, several more tools to design the distribution of server components, beside active replication [3], are offered. The designer has a full range of synchronization, replication, persistency, data sharing & consistency, checkpoint & logging, coordination and communication tools available to construct components that are distributed in a fashion that exactly match its reliability requirements.

3 Transparency

A decade of building large distributed systems in industrial settings [4, 12] has a shown serious mismatch between the available tools for constructing reliable systems and the requirements of professional system builders. Many of the problems can be reduced to the fact that tools builders were trying to achieve the transparent insertion of their technology, while system builders needed full control to achieve acceptable performance or efficient management.

The pinnacle of transparent operation can be found in the attempts to provide fully automatic object replication. By using language or ORB features the replication of objects could be automated without the need for any changes to the objects, while using state machine replication. Products such as Orbix+Isis[6] and projects such as Electra [7] have been successful in implementing these techniques, but their success in the hands of users was very limited.

It all observed systems [12] it turned out that server developers *want* to worry about replica configuration, intervene in failure detection or enabling explicit synchronization between replicas. There was only a small class of server applications where the designer did not care about the impact of replication, and most of these involved server replicas that needed no access to shared resources and were not part of a larger execution chain. The majority of systems, in which transparent replication is used, become more complex, suffer reduced performance and potential incorrect behavior traceable to the lack of control, within the application, concerning how replication is performed. There is a strong analogy with starting additional threads in a previously single threaded program, where the designer is not aware of the added concurrency.

In addition to the transparency limitations just described, automatic object replication exposed problems in the area of efficiency. Apart from the fact that the state-machine replication is a very heavyweight mechanism when used with more than two replicas, a generic replication mechanism needs to be conservative in its strategies and may be very limited in terms of available optimizations. When allowing the developer control over what and especially when to replicate, optimizations can be made using the semantics of the application.

These observations have resulted in that in Quintet distribution in all its aspects is made explicit. Although there are many tools available to help with operations such as replication, the developer decides, what, where and when to replicate. Some of the management and support tools, such as the shared data structure toolkit, rely on generic replication, but the developers that use them are aware of the implications of importing this functionality.

The decision to give full control to the developer is in strong contrast with the current trends in reliable distributed object research, were transparency is still considered the Holy Grail [12]. These systems [8,9,14] experience the same limitations that the serious use of our systems exposed. Only by restricting the useable model will these systems be able to support developers in a consistent manner. Quintet does not restrict the traditional programming model in any way and provides the developer with more tools to do his/her job.

4 Quintet goals

Quintet targets the development of client/server computing in multi-tier enterprise systems, where there are reliability requirements for the servers. In the prototype system, the servers are implemented using Microsoft COM component technology.

The central research goal of Quintet is to find the collection of tools that is most useful for the developers of reliable components. Given that this is not an area were past experience can drive the selection of these tools, the project is started with building a limited set of essential tools and interfaces. Iterative, based on user feedback, the tool collection is changed to meet the real needs. One of the major reasons for targeting COM based server environments, is that there is a perceived need for reliability and the project is very likely to get valuable feedback from the user community to ensure the much needed improvement cycles. An overview of the initial tool set appears in a later section of this paper.

The server components developed with Quintet need to be COM aware as all services offered are only available through COM interfaces. Although the majority of client/server interaction is envisioned to be DCOM based, the system can support client/server communication based on RPC, sockets or integrate an IIOP-bridge. In all cases however the components are implemented as COM classes and instantiated through COM class factories.

In a traditional DCOM client/server system the event that triggers the instantiation of a component in that of a *CreateInstance* call at the client system. For reliable COM server components the rules for instantiation can be more

complex and are often based on the reliability QoS specification for the particular component. For example server components can out-live client connections to ensure real-time volatile state replication, where the component is only made persistent and decommissioned after no new client connection was made within a given time period.

In the selection of the initial set of tools it is assumed that instances of the same component have a certain need for cooperation. The basic communication tool for example is pre-configured to provide a component with primitives to communicate with all other instances of the same component and to receive membership style notifications. Quintet based replicated components are not forced to maintain identical state, the developer chooses when and what to replicate, to which components. How optimistic (or pessimistic) the state replication strategy is, depends on application tradeoffs, and can be adjusted on the fly. Cooperation in Quintet is not limited to components of the same class. Different components, can transfer state, synchronize, vote for leadership, use shared data structures and use the communication tools in explicit manners, etc.

Given that all distribution is explicit, a major concern in Quintet is that the exposed complexity could make the development task more hazardous, yielding systems that are more error prone and thus implicitly defeating the reliability goals. The tools and interfaces are designed with care to match the existing COM programming practices as much as possible, making the transition for developers as simple as possible. Recently, in good Windows tradition, experiments have started with programming Wizards to try to assist in the more complex tasks.

The second major goal of Quintet is to build an efficient runtime environment to support the development of complex tools. In Quintet new algorithms for scalable lightweight object membership, fast distributed synchronization, efficient component migration, are being prototyped. The Core Technology (QCT), which implements the underlying communication system, is designed with high-performance cluster communication interfaces in mind. More details can be found in the section that describes QCT.

5 Relation with MTS

Although the Microsoft Transaction Server (MTS) is concerned with offering solutions to server components with a different set of requirements, Quintet has in its implementation some solutions that are similar to MTS. The way the component management service is the container server for the components it manages and the way it maintains contexts for each component instances are similar to the way MTS manages its components. The similarity is based on that this is the correct way of managing COM objects.

Two other mechanisms in Quintet have identical counterparts in MTS: Security is implemented using a role based management system and long running components can be temporarily *retired* without notifying the connected clients. The role-based security was chosen based on a research decision and its similarity to the MTS solution can be seen as accidental. The *retire* operation was added to Quintet, based on the argumentation by the MTS architects that memory consumption by long running components is the limiting factor in scaling component servers such as MTS and Quintet. We do not have any experiences that support this claim, but the arguments seem reasonable and by implementing the facility Quintet can be used to research this issue.

6 Target environment

Quintet is designed to function on a collection of server nodes, organized into a cluster, with some form of cluster management software offering basic services such as node addressing, node enumeration, object naming and basic security. The prototype implementation of Quintet uses the Microsoft Cluster Service (MSCS) [11], LDAP accessible naming service (active directory) and the standard NT/DCOM security mechanisms (LanManager).

Currently Quintet assumes cluster sizes of 4 to 16 nodes. Although nothing in its design prohibits the use of larger sized clusters, the distributed algorithms used in the Core Technology are optimized towards clusters of this size. The implementation is modular in the sense that the Core Technology components can be replaced if the need for that arises. A fundamental assumption in the construction of the system is that the intra-cluster communication can be performed an order of magnitude faster than the client/server interaction.

Although a first concern of Quintet is correctness of the services it offers, providing scalable performance is important second goal. A related measurement project is started in which MSCS and DCOM are thoroughly analyzed to understand the performance boundaries of these technologies [13], and to be able to offset Quintet introduced overhead and costs correctly.

7 System overview

Components developed with Quintet are available on the server nodes through application servers (Quintet Component Manager) that are configured to export the components through the traditional component registration channels. Instantiation requests arrive at the servers, which are responsible for the loading and unloading of class factories, and tracking component instances.

A variety of different styles can be used in developing reliable components, all depending on the particular reliability requirements of the application. Components can be longer running, actively replicated components, where each new client connection only triggers the instantiation of some client state. Or each new instantiation request can result in the creation of two instances at different nodes that collaborate in a primary/backup fashion.

In general the class factories implement client management and replica instantiation, while the components implement the replication strategy. In implementing each of these task the developer is assisted by Quintet functionality. Quintet provides default implementations for general cases.

The current Quintet prototype consists of seven major building blocks

1. *Core Technology*. The communication system on which the component manger and the component runtime are based. It provides membership and multicast communication functionality.
2. *Server Component Management*. Provides the registering and loading of the server components. Manages component placement, fault monitoring and handling, security and basic system management.
3. *Server Component Development*. The basic tools for the developer to construct the server components
4. *Server Component Runtime*. Tool implementation and management, is part of the component manager.
5. *System Management Tools*. A collection of tools for administrators to monitor and manage the system and its individual components.
6. *Client Runtime*. Mechanisms to support connections to potentially replicated components by regular DCOM clients. Support for failover to alternative component instances upon failure.

Each of the different areas is described in detail in following sections.

7.1 Core Technology

Quintet Core Technology (QCT) is the basic building block for the server management and component runtime. It is a lightweight implementation of a Group Communication Service, specifically targeted towards high-performance clusters. It uses MSCS style addressing and makes use of some of the nodes management features of the MSCS management software [11]. It used this information to locate other Quintet component managers, and to determine which network interfaces to exploit for intra-cluster network communication.

QCT is designed to run over both reliable and unreliable interconnects and is optimized towards user-level communication interfaces such as VIA [10] and U-Net [2]. However until stable, commercial strength versions of these interfaces are available, QCT primarily uses traditional network communication as not to compromise its reliability goals. The low-level message handling interfaces make extensive use of asynchronous message transfer and NT completion ports to optimize interaction with the network.

QCT offers *Virtual Synchrony* [5] guarantees on its communication primitives, ensuring the ordering of messages in relation to membership changes and *atomicity* on all message delivery. The communication interface provides a *multicast* primitive to send all members in a group and a *send* primitive to address a single member. Messages sent with the multicast primitive can be send with either the basic guarantees (atomicity) or can be extended with a *total order* guarantee ensuring that all members see all messages in this group in the same order.

QCT provides an internal interface, mainly used by the component managers (see next section). The components and class factories see a higher level interface for communication. To make the system scalable and not overuse the heavy weight virtual synchronous membership for each instantiation of a component, a lightweight component membership mechanism is layered over the basic system.

Each component is automatically a member of its *ClassGroup*, which provides membership notification and communication to all instances of a single component class. All the class factories of the same component class, present at the different component managers see membership change notifications whenever a component is instantiated or destructed. The class factories also see membership changes whenever a new instance of the particular class factory joins the system. The components only see changes in the component membership, not of the factories, and the components only receive membership updates if they explicitly register for it. The virtual synchronous membership agreement algorithm is only run in case of the failure of an object manager, or when a class factory at a component manager is unloaded.

Components can make use of the group communication interface outside of the ClassGroups by using self-defined groups and names. In this case the component can choose to either use the lightweight component membership or the more heavyweight low-level QCT interface.

7.2 Component Management

The *Quintet Component Manager* (QCM) is the central unit in the management of the reliable components. The functionality of the manager includes: loading of component libraries, starting of class factories, performing

security checks, client administration & configuration, failure handling, dynamic load management and system administration. The manager contains the Core Technology and the runtime for the tool collection.

QCM is registered at each server node to implement all the component classes it manages, resulting in that the Service Control Manager at the node routes regular DCOM instantiation requests for the components to QCM. The method by which a client receives information about which node to contact for its instantiation request depends on the particular client technique used, which are described in the section on the client runtime. The class factories for the requested component are expected to collaborate on providing a hint to the QCM at which node the instantiation is preferred. This information is relayed to the client moniker or the proxy process. If the class factories suggest "don't care" the QCM makes a decision based on the QoS spec for the component.

For each component instance the QCM maintains a shadow object (context object in MTS terms), where the object references, returned to the client, refer to. The shadow object contains administration, statistic and debug information. Longer running components, with a low method invocation frequency, can be request to persist their state and then destruct themselves. At the next method request, directed to the shadow object, the component is reloaded from the saved state. This mechanism can not be used for all types of components, as for example components engaged in active replication can not be decommissioned.

It is possible to migrate active components to other nodes in the system, and there are two mechanisms from which the component state at the new node can be recreated: the component can implement an IMigrateState interface or the manager can forcibly use the checkpoint and reload mechanism. Requests from clients that are not yet updated with the new location of the migrated component are forwarded based on information in the shadow object. The shadow object is garbage collected after the component is destructed.

Each node in the cluster runs one or more component managers. How many managers run at a node depends on the particular component configuration. If a component is considered to be unreliable, for example during a development phase, its potential failure is isolated from the rest of the component system by running the component in a separate address space attached to a private manager. In this if a component crashes it will not cause the failure of the other components. For scheduling and network efficiency it is desirable to have a single component manager per node, controlling all components at that node. The first QCM process runs as an NT service, any additional processes are started by the first QCM process. The primary QCM process is under control of an MSCS resource monitor to ensure automatic restart in case of failure.

7.3 Component Development Tools

Quintet offers a collection of tools for the developer to use for the construction of the class factories and the components. The following is a short list of the most important tools.

Basic membership & communication. As already described in the section on Core Technology, each component and its class factory are automatically a member of its *ClassGroup*. The components can register for receiving membership change notifications, when it provides its ConnectionPoint interface for receiving messages. Next to the use of the *ClassGroup* the component is free to create, join and leave other groups and communicate using those groups. The component, however, cannot leave its *ClassGroup* other than through destruction.

State maintenance. A component can implement a shared state interface and register this interface with the object manager. State update can be performed manually by a component notifying the component manager that it now want its state transferred to all other components that have registered the same component state interface. If needed, any synchronization before the state update is to be handled by the component itself. The component manager retrieves the state from requesting component and updates all components in total order.

The developer can also choose for an automated version of state update. Components notify the QCM whenever the state has changed, and whenever they are in a position to receive the state update. As soon as all components have signaled their availability the component manager updates the state. Any conflict resolution needs to be implemented by the component state receive routines. An example where this kind of automation is useful is when using a primary component with a collection of hot standbys.

Shared data structures. To support simplified state sharing strategies Quintet offers a collection of data structures (hash table, associative sets, queues, etc.) that can be shared among component instances. The object managers implement the runtime for this and ensure that component instances which share interface references to a shared data structure, always have access to a local copy. Updates to the data structures are guaranteed to keep all replicas in a consistent state.

Voting. A component can *propose* an action, on which the participating components *vote* to accept or reject. More complex algorithms for quorum techniques, barrier synchronization, distributed transactions and leader

election are implemented using this basic voting interface. This is similar to the services in [1].

State persistence. Quintet offers a persistent object store to the components from which the components can be initialized. The mechanism is used for crash recovery, system startup and the decommissioning and restart of long running components. Components are not automatically persistent, they use a checkpoint and logging interface to explicitly persist their state.

7.4 Component Runtime

Many of the tools Quintet offers to the developer have a significant runtime component to them. The majority of the current tool collection is implemented uses the facilities offered by QCT. Except for some management modules, the communication between tool instances runs over the same heavyweight communication endpoint as the *ClassGroup* of the component that uses the tool. The lightweight addressing space is divided such that class factories, component instances and tools can be addresses separately and messages can be multicast to the appropriate subsets. This sharing is an optimization, when the runtime detects that a tool instance is use by different components, it creates its own heavyweight endpoint.

An example of a tool with a major runtime component is the one that implements *shared data structures*. The runtime implements the data structures itself, the distributed access, the consistent updates, and the replica & location management to ensure that components always local read access.

To ease the development of new tools, a small support toolkit was built that implements basic data types, message handling and the serialization of basic data types through QCT.

A tool that does not make use of the QCS facilities is the *Persistent Object Store*, which uses the checkpoint and logging facility offered by MSCS [11]. This ensures that the data is always available to the nodes that are active in the cluster as MSCS terminates minority partitions that have no access to the shared Quorum resource. The Quorum resource is a shared disk, which also stores the checkpoints and logs.

7.5 System Management

Experiences show that complex server systems such as Quintet are only as useful as the system management tools and interfaces that accompany it. Without these tools the system becomes painful to use, difficult to monitor and diverts the attention of the developer from the most important task: developing robust components.

The key system management tool is a traditional Win32 explorer style application, which can be used for all the administrative tasks. The management tool is developed as a traditional COM client/server application, with the server functionality implemented in the Component Managers. There are command line counterparts of the tool, but they are geared towards the use in shell scripts. Several tool instances can be running at the same time. The tools are augmented with failure detection mechanisms outside of the scope of COM to ensure timely failover to another component manager, in the case of a manager or node crash. The RPC timeout mechanism in COM is very tolerant but a 30 seconds delay is unacceptable for Quintet purposes.

The management tool provides developers with the possibility to add components to cluster through drag and drop, and specify the Reliability QoS spec and the security information for the new component. The tool can be used to control manager and component configuration, and runtime control tasks, such as component migration, can be performed manually.

The Component Managers contain several methods to monitor the operation of system and to monitor components on an individual basis. Information ranging from statistics to individual method invocation and results can be monitored and displayed in the management tool.

The tool can be extended on a per-component basis with component specific control and management functionality (see the section on extensibility).

7.6 Client Runtime

Client access to the component instances is still a major research issue. Currently two approaches are under investigation: In the first approach the client is failover aware and uses a reconnect mechanism to hookup with an alternative component instance. The second approach leaves the client unaware of the new situation and uses a local proxy process through which all calls to the component replicas are routed.

Although prototypes of both approaches are implemented and running, the difficulty is in determining whether all cases are handled correctly. Real COM client/server system often have very rich interaction patterns and some of the pre-built MS support components implemented using ATL or MFC introduce additional levels of complexity.

The failover aware approach has its limitations as COM insists on making distribution of the component fully transparent and treats each component as if it is local, providing no failure information about the distributed case. Quintet's support here consist of a set of monikers that internally interact with the component managers to connect to a selected node and are also responsible for the selection of new nodes after failure occurs. The developer in the design of the client needs to catch these failures and use the moniker to reconnect. This approach has only

been made to work in C/C++ COM environments and have not found their counterpart (yet) in VB and Java.

The most promising of the two approaches is the one that uses a local proxy to implement the client/server interaction. The proxy receives component information from the component manager and registers itself at the local node as implementing these components. The Service Control Manager at the local node then routes all create requests to the local proxy. The proxy interacts with the Component Manager to create the correct forwarding path and upon server failure or component migration adjusts the path accordingly. The proxy is efficient in that it inspects the stack of the incoming RPC call to do some cut-through routing without the need to implement the interface locally and fully execute each RPC invocation. As always Connection Points (COM's version of callbacks) make the whole mechanism more complex, and are handled with support from the Component Managers. The Managers keep track of this kind of full duplex connection, and ensure that upon client handoff to another component instance, the component that now handles the client has the right client routing information.

8 Extensibility

The base system is extensible by developers in a number of ways:

1. Component specific management modules can be added to the management tools.
2. Component specific debug support can be added to component managers.
3. Runtime support for new tools can be added to the component managers.
4. General component management can be added to the component managers.

The extensions to the management tools and the component managers are implemented as COM components and loaded on demand though the regular configuration information in the NT registry. Each extension type needs to implement a predefined interface through which the component can be initialized and given access to environment it executes in.

To enable the development of these extensions, the component manager and the administration tool export a set of interfaces that can be used by each extension to implement its functionality.

9 Current Status

There are a number of major research issues that have not been worked out yet. The most important issue is that of inter-component dependencies, the obvious solution seems to hint to the use of component groups and manage the groups as single units, but as yet it is uncertain what the best way is to express the dependencies and how to manage them at runtime. Whether the current approach to client runtime is the right one, remains an open issue until there is more experience with larger scale systems using this technology.

Quintet is a system under development. At this moment a prototype is implemented and in is use for the development of a set of applications that are to stress the system to its limits.

Building support tools for reliable systems puts unusual pressure on the robustness of the system before it can be released for alpha & beta tests. Any form of instability of the system is counter-productive. Many more development cycles will need to be spent before Quintet will have the robustness such that it is ready for experiments within industrial settings. The first public alpha distributions are expected in September of 1998.

References

[1] Badovinatz, P., Chandra, T.D., Gopal, A., Jurgensen, D., Kirby, T., Krishnamur, S., and Pershing, J., "GroupServices: infrastructure for highly available, clustered computing", unpublished document, December 1997

[2] Basu, A., Buch, V., Vogels, W., and von Eicken, T., "U-Net: A User-Level Network Interface for Parallel and Distributed Computing" *Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP)*, Copper Mountain, Colorado, December 3-6, 1995

[3] Birman, K.P., *Building Secure and Reliable Network Applications*. Manning Publishing Company, and Prentice Hall, 1997

[4] Birman, K., "Reliable Multicast Goes Mainstream", Bulletin of the Technical Committee on Operating Systems and Application Environments (TCOS) Spring 1998 (Volume 10, Number 1)

[5] Birman, K., and Renesse, R. van, “Software for Reliable Networks”, in *Scientific American*, May, 1996

[6] Landis, S.,and Maffeis, S. "Building reliable distributed systems with CORBA," *Theory and Practice of Object Systems*, John Wiley & Sons, New York, 1997. 5.

[7] Maffeis, S., "Adding group communication and fault-tolerance to CORBA," in *Proc. Usenix Conf. on Object-Oriented Technologies*, June 199

[8] Narasimhan, P., Moser, L.E., and Melliar-Smith, P.M., "Exploiting the Internet Inter-ORB Protocol interface to provide CORBA with fault tolerance," in *Proceeding .of the 3rd Conference. on Object-Oriented Technologies and Systems*, Portland, OR, June 1997.

[9] Singhai, A., Sane, A., and Campbell, R.H., "Quarterware for Middleware", *Proceedings of the International Conference on Distributed Computing Systems*, Amsterdam, The Netherlands, 1998.

[10] The Virtual Interface Architecture Specification 1.0, http://www.viarch.org

[11] Vogels, W., Dumitriu, D., Birman, K. Gamache, R., Short, R., Vert, J., Massa, M., Barrera, J., and Gray, J., "The Design and Architecture of the Microsoft Cluster Service -- A Practical Approach to High-Availability and Scalability", *Proceedings of the 28th symposium on Fault-Tolerant Computing*, Munich, Germany, June 1998.

[12] Vogels, W., van Renesse, R., and Birman, K., "Six Misconceptions about Reliable Distributed Computing", *Proceedings of 5th SIGOPS European Workshop*, Sintra, Portugal, September 1998.

[13] Vogels, W., Dumitriu, D., Agrawal, A., Chia, T., and Guo K., "Scalability of the Microsoft Cluster Service", *Proceedings of the 2nd Usenix NT Symposium*, Seattle, August 1998.

[14] Wang, Y., and Lee, W., "COMERA: COM Extensible Remoting Architecture", *Proceedings of the 4th Conference. on Object-Oriented Technologies and Systems*, Santa Fe, NM, April 1998.