Research Directions in Distributed Systems.

Paul McKee, Ian Marshall, and Ian Henning

Summary
This article considers the technical characteristics of current distributed systems technology as exemplified by CORBA. In the context of large-scale globally distributed applications, potential problems are highlighted, and areas for further research and development are suggested.

1. Introduction

Distributed computing systems have been proposed as a viable technology for the provision of highly scaleable, high performance applications. A specific example of this is the World Wide Web. This operates over a heterogeneous combination of computing platforms and communication networks. Interoperability is assured via standard protocols, IP at the network layer and HTTP at the application layer. In a more general sense, the aim of producing a generic platform to support the needs of a large proportion of distributed application developers has been the goal Object Management Group (OMG) and realised in their Common Object Request Broker Architecture (CORBA) specifications. Here the object oriented (OO) software engineering approach has been combined with the remote procedure call (RPC) to produce a 'software bus'. The approach attempts to provide a set of common services for application developers whilst hiding a great deal of the complexity which distribution brings. Services are accessed via API's described via a common interface definition language (IDL). A number of products are now maturing supported on a range of target platforms which offers good prospects for heterogeneous interworking.

There is no doubt that CORBA represents an excellent attempt to solve a very difficult set of problems. However the increasing use of networks such as the Internet to support seamless applications between people, communities, and businesses serve to form a set of changing requirements which suggest further developments may well be necessary. Electronic business for example is likely to be a key application area. Use in such an environment spanning diverse computing, organisational and business enterprises will fuel the demand for the sharing of information and resources. As with the WWW, such applications are expected to become Global in extent. For use in such environments, some of the characteristics required of a supporting distributed platform are likely to be;

- service guarantees covering performance, transactions, and security
- support for mobility both of the client and server parts of the application code
- management for performance, reliability and growth
- an ability to support the location of objects and resources from a global 'pool'
- a series of distributed information stores to reduce network traffic

This list is by no means exhaustive, and it must be noted that new applications and technologies will no doubt appear and add to it.

In this paper we consider the suitability of existing technology to meet some of these evolving requirements in the context of large scale, globally distributed applications. Our aim is to highlight some of the issues and so to stimulate debate and research. In particular in section 2 we discuss the characteristics of existing distributed operating systems using CORBA as the exemplar[1]. We consider some of key areas of communications styles, support for interworking, scalability, mobility, management, and security. Our analysis suggests some significant limitations for large scale and multi-enterprise use, and these are highlighted. Potential solutions are identified and future directions for research highlighted.

2. Systems issues

Distributed computing systems are becoming increasingly important in the realisation of distributed applications. There are number of commercial offerings in this area particularly CORBA from the OMG [2], DCOM from Microsoft [3], and RMI from Sun [4].

JAVA RMI offers the advantage of being tightly coupled to the Java language system, but as such is a closed system that locks users into a single language technology. New languages offering features not found in Java will emerge, and will be unable to interoperate with existing systems. However, in recognition of the need for cross language interoperation Java RMI can make use of the IIOP protocol to link with CORBA. Therefore it can be classed alongside CORBA in any comparison.

DCOM is attractive because it works with all Microsoft products and MS are the dominant operating systems supplier. However the standard is not well documented, unstable and proprietary to one manufacturer, although bindings are available for modern languages.

The current state of the art is more fittingly represented by CORBA [1]. It is language independent, produced by an international committee composed of representatives from industry, often advised by academia, and is designed to support a range of high-level services. The architecture involves communication through an object request broker (ORB) which provides a mechanism for transparently communicating client requests to target object implementations. However identifying CORBA as representative of the state-of-the-art does not mean that this will become the dominant system in the marketplace, Microsoft has much commercial leverage.

[1] Although reference is made to CORBA, many of the same comments are applicable to most existing distributed operating systems.

In this section we focus on a number of key aspects of distributed systems (CORBA) with the aim of identifying weaknesses, we then attempt to suggest some solutions.

2.1 Communication styles

One of the overall goals of distributed object oriented computing is that from the programmers viewpoint there should be no essential distinction between objects that share an address space, and objects that are on two machines with different architectures located on different continents. In a conventional distributed operating system an embedded communication infrastructure is provided to support inter-object interaction. In CORBA, as in most other distributed operating systems, the complexity of the communication infrastructure has been hidden from the application designer who therefore has no control over the choice of mechanism. In practice most CORBA implementations only provide a single mechanism with a limited range of interaction styles. In order to communicate between objects (or programmes) at different locations the programmer uses procedure calls to a set of interfaces declared in a common interface definition language. This approach enables programmers to create distributed systems without needing to learn about networking, since procedure calls are a natural part of all programmes. The programmer writes exactly the same code for any type of call, and relies upon the system to take care of delivery.

CORBA is based on the use of remote procedure calls (RPC) [5]. RPC systems are intended to make cross address space function calls look like local function calls. Applying this within the object oriented programming model allows papering over not just the marshalling of parameters and the un-marshalling of results, but also the location and connection to the target objects. There is thus a single paradigm of object use and communication no matter what the location of the objects might be. Whether a given objects invocation is local or remote is purely a function of the implementation of the objects being used, and could easily change from one invocation to another if no user defined state is stored. This approach will prove effective in many situations. However, there are many cases where the default communication mechanisms will lead to poor performance. In such cases it would be better to encourage programmers to acquire sufficient knowledge to make a more appropriate choice, and provide the tools to enforce the choice as part of the distributed operating system.

Figure 1 depicts a hypothetical situation where a rich mixture of communication styles would be appropriate. In this simple example the performance of the underlying infrastructure may well vary considerably, for example loss and latency cannot be predicted if the Internet is used between enterprises. For time critical applications this may prove problematic. Latency may even be an issue over high performance networks in some demanding cases. In a wider context both computer and telecommunications networks are migrating towards a multi-service paradigm where variable degrees of quality of service will be available. Such capabilities should be made available to the application programmer to use as befits the situation. Moving up from the low-level infrastructure the interaction styles between objects may also vary. Simple connection oriented communication may be adequate in many circumstances but a connectionless 'fire and forget' service with delivery guarantees may also be useful.

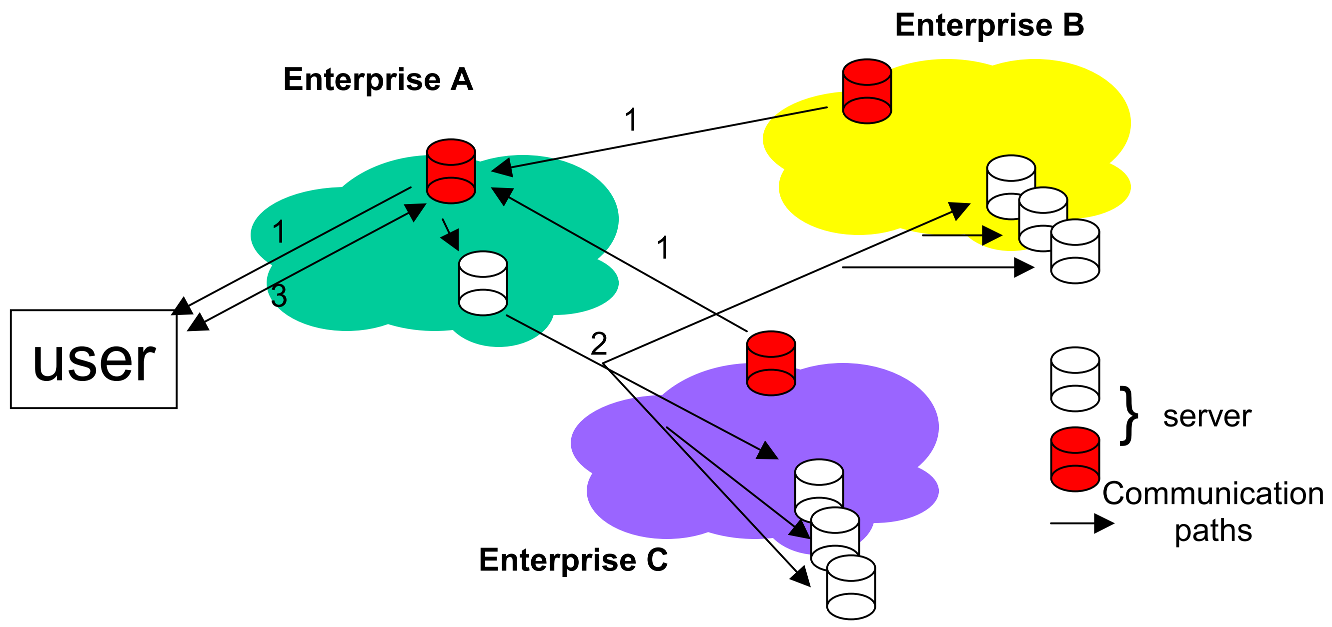

Figure 1 Examples of different communication styles; type 1: unicast across enterprise boundaries, type 2: multicast to communicate with multiple objects associated with a session, and type 3: low delay, low latency messaging.

In general there are differences between distributed and centralised programmes as a local member function call and a cross continent object invocation are clearly not the same. One of the main differences concerns concurrency. A centralised programme typically has a single thread of control. When centralised programmes do support concurrent execution the threads communicate using shared data supported by monitors or semaphores to allow synchronisation or mutual exclusion. A distributed programme on the other hand is always multi-threaded. Whether more than one thread is simultaneously active depends on the original motivation for distribution. In a conventional client server model a single conceptual thread passes between servers and interactions almost always follow a request-reply model. This type of application is well suited to the conventional interaction style supported by CORBA and other current distributed operating systems, and the use of the underlying communication mechanism can be very efficient (in a good CORBA implementation). However, where distribution has been used to exploit parallelism in the task and spread the processing load over multiple processors, it will normally be more effective to use a message based interaction style. Messages enable programmers to invoke remote objects without waiting for the response (and wasting local processor cycles). If the distributed operating system does not support more than one interaction style programmers are forced to model those they require in terms of the dominant model, this is often error prone and obscures the programmers original intent.

It should be clear by now that a range of communication options may well prove essential. Typical examples of these might include one to one; one-many, broadcast; streamed or message based; connection or connectionless; synch or asynch, delay/jitter sensitive/insensitive, reliable/unreliable, secure, authenticated etc. From an application programmers viewpoint it would be an advantage to be able to more appropriately engineer methods of communication to match a variety of environmental conditions. In this section we consider the CORBA communication infrastructure, and review some of the alternative approaches that are emerging.

Prior to the introduction of the messaging specification [6], CORBA provided three communication models:

- Synchronous two-way: in this model a client request is forwarded to a target object and the client then waits for a response. While it is waiting the client thread that invoked the request is blocked and cannot perform any other processing. This behaviour results in unsatisfactory performance for constrained applications.
- One way: A one-way invocation is composed of only request with no response. The original intention was to use one way calls over unreliable transport protocols such as the user datagram protocol UDP. The addition of one-way invocations promises only best effort semantics, thus the ORB need not report an error if the one way fails. This interaction mode was not clearly defined by the OMG and there may be variations between different vendor implementations. However, most ORBs implement one-way calls over TCP as required by the standard IIOP protocol specification, and is thus not non-blocking. Moreover the mechanism is inherently unreliable, and a reliable one-way call is often needed at both transport and request level.
- Deferred synchronous: A client sends a request to a target object and then continues its own processing. The client ORB does not block the calling thread until the response arrives, instead the client can either later poll to see if the target object has returned a response, or it can perform a separate blocking call to wait for the response. The deferred synchronous request model can only be used with the dynamic invocation interface DII, which is both difficult to implement and slow.

Thus essentially CORBA only offers one communication method, which is a uni-cast method. Similar observations may be made of the DCOM communications structure that is based on the DCE RPC.

One way in which the performance of large scale distributed systems can be improved is the removal of the assumption that all function calls must be dealt with in the same way [7]. In a large-scale network latency must be taken into account; obviously a local member function call and one that crosses continents will be quite different. To do this a platform is required which provides a communications architecture capable of supporting multiple communication styles between objects over multiple Transport Protocols. These styles must include support for reliable asynchronous and multi-cast communication. Thus the Developer would be enabled to define a task or network specific interaction style, and would not be restricted to the current 'one size fits' all method.

The one size fits all approach to communication currently used in CORBA is being challenged by a number of initiatives from both industry and universities. For example Win Sock version 2 which includes support for multiple protocols on a plug-in or pile-on basis, offers Transport Protocol independence allowing the user to choose the protocol by the services they provide, and includes support for multi-point and multi-cast communication [8]. ILU from Xerox [9] also includes support for multiple protocols, and can reasonably be expected to influence the approach of the IETF HTTP NG working group in which Xerox are represented. In the CORBA world the Orbacus ORB [10] includes support for different protocols. Approaches typified by Flexi-Net from ANSA [11] and the Regent framework from Imperial College [12] introduce even greater flexibility by allowing dynamic protocol stack assembly, even at run-time. This added flexibility allows the user to make optimum use of advanced network capabilities as they become available. It further allows options such as compression and encryption only when they are required, which can produce a consequent improvement in efficiency. The availability of these flexible protocol choices would in no way replace the standard RPC, these would still be available for use in applications where performance was not paramount, but for performance critical applications the extra flexibility would be a great benefit.

2.2 Support for inter working

Future large-scale distributed systems will contain machines of every type, with the broadest range of operating systems and network connections possible. The machines will possess different architectures, and will be programmed using different programming languages. Applications will consist of collaborations between objects supplied by different parties, co-operating to achieve the desired end result. A designer will specify the objects that comprise a system. Some of the objects may be specified by a third party, and may not be located and bound into the system until run-time. In this scenario, the designer will need to specify his functional and non-functional expectations of the object in such a way that the specification can be used at run-time to check whether an appropriate object has been located. The object provider must also specify the functional and non-functional properties of his object in such a way that they can be unambiguously checked at run-time.

Everything in CORBA is represented as a service that is accessed via an interface specified in IDL, commonly referred to as an API. Interworking in CORBA is based on invoking standard APIs using a standard message Protocol GIOP [13]. In most cases this maps onto a set of wire protocols via IIOP. Should the user wish to inter-work with a non-CORBA system a translation bridge must be supplied that will translate message formats, location information, transmission protocol etc, and act in a bi-directional way to translate any response. The example in table 1 illustrates some problems that can arise with this approach. An IDL specification describes the syntax of an object's interface, including parent classes, exceptions, attributes and operation signatures. Unfortunately, the meaning of the interface is only defined for a particular context, and the context does not form part of the interface specification. In other words the interface does not specify the objects behaviour (i.e. its functional and non-functional properties). In practice, when developers are co-located, and objects are being developed concurrently this may not seem a problem. However, when the developers are separated by organisation, location or time, it can lead to great difficulties since a full specification of behaviour may not exist, and even if it does it may not be correct. If selection of an object is to be performed automatically at run-time the problem is even worse, since any specification must be machine readable. Clearly an API based exclusively on IDL will not guarantee interoperability. Since APIs are normally considered to be the programming interface only and not the associated specification we could say that APIs do not guarantee interoperability, they merely facilitate it by providing a template for programmers. APIs are thus necessary but not sufficient.

Interface Definition	**Subtract(float a, float b) return float r , string ERR**	
Inputs	**Returned value**	**Comments**
a =15 b=10	**r = 5**	**As expected**
a =15 b =10	**r = -5**	**Subtract subtracts a from b**
a=15 b =10	**Err = parameter out of range**	**Either neg returns not allowed Or max size of a or b exceeded**
a =15 b =10	**r = 5 but on 10th invocation Err = server out of memory**	**Function is allocating memory and not releasing it**
a = 15 b =10	**r = 5 but response only returned after 3min wait**	**Process is running at very low priority**
a =15 b = 10	**r = 5 then next time r = -5**	**Version change in subtract**

Table 1. Potential problems with API's

If we refer to table 1 all of the identified problems may be alleviated by access to a complete, manual based, specification. However, we must determine that we are using the correct manual for the object instance we are currently operating on. In automated scenarios a manual would in any case be unavailable since human readable manuals are not normally machine readable. To avoid these problems every object in the system should have its properties precisely defined in an appropriate language. To facilitate automated usage the language should be strongly typed, with fully enumerated variables (i.e. like a protocol specification). The definition should include items like the objects dependencies on other objects, the resource requirements of the object, the permitted parameter ranges, the intended semantics of the parameters, the quality of service offered etc. as required by the complexity of the object being described. This definition must be tightly associated with each instance of every object, as it will contain much information that is instance specific. One possible solution would be to use containers (like XML or Opendoc) and embed the definition into the top level container as object metadata. Not only would this tie the object to its specification (metadata), it would also provide a standardised metadata query interface.

Given access to a precise strongly typed description of an object, it is relatively straightforward to provide high-level programmatic interfaces to a wide range of network protocols, without affecting the interoperabilty of the protocol. Because the protocols are simple the correctness of their operation may be proved formally. Any system that provides an endpoint for such a protocol can communicate with any other system implementing the same protocol using the well defined vocabulary of messages defined by the protocol. Thus, a promising approach for the future could be to base the services of a distributed processing environment on a rich set of protocols with proven operation and so increase the potential for successful inter-working. Rather than develop a confusing array of distributed services, which don't interwork well (as has happened in CORBA), future distributed operating systems should exploit protocol based development wherever possible.

There is already some progress in this general direction based on the CORBA meta-object facility and protocols are being developed as a part of HTTP-NG [14], ILU from Xerox [15] and JINI from SUN [16].

2.3 Scalability

In the context of a future large scale distributed system scalability may be simply stated as the ability to add applications, users and computing nodes to the system whilst maintaining the specified level of service. Obviously adding users will eventually require the addition of extra machines as will the addition of extra applications, but the scalability requirement means that the required performance can be regained by adding computing power. Architecturally the system must avoid any system function whose performance degradation depends on the number of system users.

Even though a central tenet of distribution is scalability, within CORBA retains a reliance on centralised information stores for such things as the name service, the trading service and implementation repositories. It must be acknowledged that the implementation repository has a somewhat mixed effect: it is advantageous that only the objects that are in use are live. Nevertheless in all three cases the centralisation of data represents a potential performance bottleneck and a single point of failure. There are architectures described for the Federation of both trading and naming services that assume the existence of some form of centralised intelligence in the system which requires the user to specify the connection graph and search order.

Centralisation is not the only scalability issue in CORBA. The communication protocol in CORBA is also inefficient [17] giving rise to a high message overhead on certain operations, such as trading using dynamic properties. Use of the security and transaction services also expands the size of messages, particularly when a transaction or security relationship spans a number of objects. The message grows at each step with the inclusion of transaction or security context information. The management of connections is undefined in the CORBA specification, each IIOP connection requires a TCP connection, and if the TCP connection fails the IIOP session cannot be recovered. The use of increasing numbers of API's in CORBA also affects scalability, every time a new service is added, additional API's [18] are introduced adding to an already heavy ORB implementation.

In order to improve the scalability of future distributed systems we must remove the need for any form of centralised information repositories and reduce the impact of adding additional services that further complicate the operating system. It would be desirable if an element of information locality could be included in any search for a service, as this would reduce communications overhead. It would also be desirable if information location were handled locally, and largely independently. One way of achieving this would be the use of structured multicasts to announce and request object information. The use of hop count limits or time to live limitations would maintain network usage at reasonable levels. This multicast approach is currently used in SUN's JINI platform in the discovery and join part of the specification. Softwired's IBUS [19] also uses multicast technology. Multicast is used extensively in existing group management systems such as HORUS and ENSEMBLE [20].

A recent IETF submission involving Microsoft, Inktomi, RealNetworks and SUN for a Web Proxy Auto-discovery protocol [21] proposes an escalating strategy of resource discovery, based on existing internet protocols, to find a nearby web proxy server. Using simple, functional and efficient mechanisms resource discovery is used to obtain information for the automatic configuration of web clients. Combining local discovery mechanisms with a cache overlay to store information could effectively distribute the information discovery overhead. The caches can also share information between them using cache digests as proposed by Rousskov [22], increasing the knowledge of the system.

There will be a requirement for any practical system to access and mix objects that are part of different existing systems in different locations. Therefore the concept of a universal set of globally unique names becomes important. This allows named objects in different systems to be treated in a common way. The scalability of a global naming system will also be greatly improved following the work on URC's and URN's at the W3C [23].

2.4 Mobility

One of the requirements for any large scale distributed system will be an increased level of support for mobility both of users and terminal equipment. The future user may reasonably expect to log onto any terminal anywhere in the world and be able to access their normal range of applications. To ensure that they continue to receive an acceptable quality of service, processes may need to be relocated closer to the point of use. Developments in mobile access technologies and mobile terminals will mean that a wide variation in access bandwidth and end terminal capabilities must be catered for together with the interference and fading problems that may be encountered over wireless links.

In common with all distributed operating systems CORBA hides the network from the developer, consequently it provides no support for such non-reliable networks and services. Location transparency is not always a desirable property as there are many instances when location based information would be very useful to mobile users, and information describing the location of events such as failures and load peaks would aid the management of large scale distributed systems. Moving objects without breaking references is also problematic. If we consider servers formed by the collaboration of several different processes every object must register with an object adapter, which acts as an interface between the implementation and the ORB. It is also responsible for managing object references, call handling, and registration of the objects with the implementation repository. In order to move the server all objects registered with the object adapter must move together, they must register with the new repository giving rise to potential consistency and scalability problems. A footprint is left in the old repository, which again impacts the potential scalability and causes poor performance.

Adding support for mobility to a distributed operating system will require a number of fundamental changes to the conventional model as typified by CORBA. One of the design cornerstones of distributed systems is location transparency, but in order to efficiently support mobile users and code it is desirable to selectively choose to implement location transparency or not. It will also be necessary to resolve the requirements from any request in terms of choosing between objects or an identical object if closer. This will of course require a more complete meta-data description of each object to aid in the resolution of this choice. Another aid to the support of mobility in distributed systems would be the use of a stateless programming model for server objects again helping to resolve the fundamental "same or identical" object question.

This selective inclusion of functionality is, as previously mentioned, available in the layered protocol stacks built for the REGENT system from Imperial College. The specification for Winsock 2 also includes a number of layered constructs for specifying quality of service and it would be interesting to determine if the same approach would allow selective transparency.

2.5 Management

In a future large-scale distributed platform that, as previously described, supports mobility the location of objects and users will be dynamic. In this section we are principally concerned with management of the objects running on the platform. The management of applications will be a concern of the application developer. We are therefore interested in the issues surrounding the location, activation, use and movement of objects within the platform. The system needs to evolve in response to changing requirements, and we also need to be able to describe how objects should respond to their changing environment. This may be achieved using policies. Policies describe possible behaviour options, and constraints on use of the options, and are often expressed as obligations, authorisations and prohibitions.

In almost all commercial CORBA implementations there is little support for management. Tool support is lacking and there are very few standard interface specifications that allow management to be added on. Available management is centralised and proprietary, it has a monolithic architecture and consequently will not inter-work across different vendor implementations. Problems are also encountered with the exact specification of management rules in a CORBA system. One of the first challenges any management system needs to tackle is that of the CORBA infrastructure itself because it will be a large object based system. To achieve this it seems reasonable that the rules and policies should be objects. However when we come to manage the running system the rules and policies will be attributes of the system objects and may need to be handled purely as text strings. This fundamental dichotomy needs to be resolved.

The management of future large-scale distributed systems must be decentralised. The network delays and congestion involved in communicating information to any centralised management entity make this approach unattractive. A key challenge for any distributed management scheme will be how to manage the scheme, as it will be a large distributed application too. Many of the management functions will need to be devolved locally, and may be autonomous. To achieve this local policies will be needed for all objects in the system, both hardware and software, to determine their action when faced with events. This policy set may form part of the meta-data associated with every object. It would also be desirable to devolve application management to the end user of that application wherever possible, thus spreading the management overhead, and allowing each user of the system to have an essentially unique solution. This in many ways mirrors the current work on active networks allowing users to inject code into the network to achieve a greater degree of application customisation. The description of management policies is still very much a topic of active research, as is the local implementation of policies. On any particular node in the system there will be policies relating to the nodes actions and policies relating to the objects actions. The overall behaviour from such a composite collection of local policies will need to be understood.

2.6 Security

Use of existing CORBA implementations between enterprises presents a range of problems depending upon the particular Firewall implementation. IIOP can use any port and these interactions are bi-directional which can be difficult with application layer firewalls. The object communication end point in the IOR contains information about the host and port for the communication, and betrays the LAN based origins of CORBA. Tunnelling over HTTP is possible but this potentially compromises the security of the firewall. Servers are also highly vulnerable to ill-formed requests that cause stability problems. Overall there is a lack of a credible solution that limits the applicability of CORBA in the commercial world.

Another impact of the existing IDL based approach to interface specification, is the lack of any additional information regarding the source, reliability and resource requirements of any object offering that interface. A more complete description of the object is required. To illustrate this we can use an example from Active Networks, where third party code is intended to run on network nodes. If the resource requirements of the code are available as a specification, and these limits are breached, the object is clearly faulty and may be terminated. This enables code to be run without an onerous testing requirement. The additional resource usage data would also be useful in the support of mobility, as it would be useful to know if a node could support the demands of an object, before the object is moved to that node. Performance management would also be helped by such a facility.

3. Conclusions

It can be seen from the examples discussed in this paper that current distributed operating systems work well for small/medium-scale distributed applications operating over reliable, and well dimensioned network and computing infrastructures. However weaknesses are exposed when consideration is given to very large-scale applications, and particularly to their use over wide area networked systems exhibiting a diverse range of inter-system capabilities and performance characteristics. Although many of the examples in this paper have been developed with reference to CORBA, similar observations may be made for the other candidate systems. Having discussed some of the weaknesses of current systems we identify areas for research to allow the development of distributed operating systems that will be useful for some of the future large-scale applications. Characteristics of such systems will include;

- greater flexibility in the task specific choice of communication style
- autonomous or local management with increased end user participation
- greater support for mobility of users and code
- objects will have an associated meta data description

Inter working will be based on provably correct simple protocols, files will be stored close to the originator and cached everywhere else, and such a cache hierarchy will be used for information discovery. Over all such a system should be dynamic and responsive to events.

In order to be able to achieve this vision future research priorities in distributed systems must include;

- providing a range of protocols in dynamic stacks
- support for different communication styles such as multicast
- location of objects and resources using local mechanisms
- local mechanisms for management and policy definition
- a language capable of unambiguously describing both functional and non-functional attributes of the objects in the system.

Building distributed systems is difficult, and existing products such as CORBA and DCOM offer significant benefits. But building global distributed systems is very hard. There are significant problems with existing systems and we need fresh approaches which embody more flexibility and allow less 'isolation' from the capabilities of the underlying hardware and network.

References

1. John Bates, "The State of the Art in Distributed and Dependable Computing", A CaberNet Sponsored Report, October 1998.
2. www.omg.org
3. www.microsoft.com/com/dcom.asp
4. www.java.sun.com/products/jdk/1.2/docs/guide/rmi/spec/rmi-title.doc.html
5. www12.w3.org/History/1992/nfs_dxcern_mirror/rpc/doc/Introduction/Abstract.html
6. Douglas C. Schmidt and Steve Vinoski, Programming Asynchronous Method Invocations with CORBA Messaging, C++ Report, SIGS, Vol. 11, No 7 2, February, 1999
7. Jim Waldo, Geoff Wyant, Ann Wollrath, Sam Kendall, "A Note on Distributed Computing" Sun Microsystems Laboratories Inc, TR-94-29, November 1994
8. www.sockets.com/winsock2.htm
9. http://pubweb.parc.xerox.com/hypertext/ilu/index.html
10. http://www.ooc.com/ob/
11. http://www.jungle.bt.co.uk/projects/ansa/
12. http://outoften.doc.ic.ac.uk/~np2/regent/regent.html
13. http://www.mitre.org/research/domis/reports/UNO.htm
14. http://www.w3.org/Protocols/HTTP-NG/
15. http://www.parc.xerox.com/istl/projects/http-ng/
16. http://java.sun.com/products/jini/specs/index.html
17. http://www.cs.wustl.edu/~schmidt/corba-research-performance.html
18. http://www.objectwatch.com/issue14.htm
19. http://i-gate.softwired.ch/products/ibus/
20. http://simon.cs.cornell.edu/Info/Projects/Ensemble/index.html
21. http://eggplant.rte.microsoft.com/wpad//

22. http://squid.nlanr.net/Cache/CacheDigest/
23. http://www.w3.org/Addressing/Addressing.html