THE PROVISION OF RELOCATION TRANSPARENCY THROUGH A FORMALISED NAMING SYSTEM IN A DISTRIBUTED MOBILE OBJECT SYSTEM

By

Katrina Elizabeth Falkner, B.Sc.(Ma. & Comp. Sc.)(Hons)

September 22, 2000

a thesis submitted for the degree of

Doctor of Philosophy

in the department of computer science

University of Adelaide

Contents

List of Tables

List of Figures

Abstract

Mobility in distributed object systems is useful as it can provide such properties as load balancing, code to data movement, fault tolerance, migration to stable storage, and autonomous semantics. In a widely distributed system, these properties are important as they can help alleviate latency issues and increase performance within the system. Additionally, they provide more flexibility in the programming of distributed systems by relaxing static location restrictions. Location transparency removes the need for client objects to explicitly know or define the location of a server object when communicating. If a server object is capable of migration, relocation transparency maintains reference validity throughout the migration.

Several models for providing relocation transparency exist, including the home location, forwarding location, and broadcast models. This thesis proposes a model that uses a distributed registry system and dynamic reference updating to provide location and relocation transparency. A registry system is used to provide location independence by resolving a location independent *name* to a reference that can be used by a client. A naming system is used to provide correct binding and production of names within the naming restrictions of the system.

The thesis proposes that the choice of naming system within a distributed or mobile object system has a large effect on the system's ability to support efficient transparent object relocation. This thesis proposes that a formal analysis of naming systems enables the selection of an appropriate naming system for a distributed or mobile object system given the object system's naming, distribution and transparency requirements. This thesis presents a new classification scheme for naming systems, based on analysis of a broad spectrum of naming systems.

A classification of existing mobile and distributed object systems with respect to existing naming models is provided. It is shown that the current models need to be refined and extended to completely and correctly classify the example systems. This thesis proposes extensions and refinements that enable correct and complete classification of mobile and distributed object systems with a need for transparency. The extended naming model is

then used to describe a naming system that is capable of implementing any naming system classifiable by the extended model. A classification of a naming system to support the proposed model of location and relocation transparency is presented.

A distributed ORB system is designed and implemented to support the distributed namespace and generic naming system implementation. The distributed ORB system is hierarchically structured and is capable of adapting in response to node failure. This ORB system is used to support client and server object integration in the DISCWorld metacomputing environment. The ORB system is used to provide migration, replication and cloning services to the DISCWorld metacomputing environment.

A qualitative analysis of the generic naming system and the DISCWorld ORB system is performed. A comparison between the proposed model for location and relocation transparency and existing models is also presented. This comparison shows that the proposed model exhibits better location and relocation performance within the DISCWorld environment. The distributed nature of the ORB system and namespace provides a scalable nature in terms of namespace size, the number of objects within the system, and the frequency of location and relocation requests.

Declaration

This work contains no material which has been accepted for the award of any other degree or diploma in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text.

I give consent to this copy of my thesis, when deposited in the University of Adelaide Library, being available for loan and photocopying.

Although DISCWorld is a joint development of the Distributed & High Performance Computing Project group and other students, the work reported here on transparent mobility support and naming systems within DISCWorld is my own work.

Katrina Elizabeth Falkner, B.Sc.(Ma. & Comp. Sc.)(Hons)
September 22, 2000

Acknowledgments

This work was carried out under the Distributed High Performance Computing Infrastructure Project (DHPC-I) of the On-Line Data Archives Program (OLDA) of the Advanced Computational Systems (ACSys) Cooperative Research Centre (CRC) and funded by the Research Data Networks (RDN) CRC. ACSys and RDN are funded by the Australian Commonwealth Government CRC Program.

I would like to thank my supervisors Michael Oudshoorn and Ken Hawick for their comments on this work and their advice throughout my time as a PhD student. Michael Oudshoorn deserves special thanks for all of his help, especially with the preparation of this thesis, and his comments and interest in my work from when I was an Honours student until now.

Many thanks to Chris Barter, Professor at Adelaide University, for his support throughout my time at Adelaide University. Thanks also to Paul Coddington, Kevin Maciunas, Dave Munro, Cheryl Pope, Francis Vaughan and Tracey Young.

Thanks also to past and present members of the DHPC group, who have made my time as a PhD student an enjoyable experience: Duncan Grove, James Hercus, Heath James, Kim Mason, Jesudas Mathew, Craig Patten, Andrew Silis, Darren Webb and Andrew Wendelborn.

I would like to thank my family for their constant support throughout my studies. My parents believe that, with education, a person is able to do anything they wish. With their help and support, and the guidance of my sister Julie, this has been possible for me.

Most of all, I would like to thank my husband, Nick Falkner, whose support and friendship has been invaluable. Providing me with a fantastic and supportive environment, reading my thesis drafts, and keeping me motivated were just part of the daily routine.

Katrina Falkner.
September 22, 2000.
Adelaide, Australia.

Chapter 1

Introduction

The object model is one of the most powerful and commonly used programming paradigms. Distributed object systems require objects to remotely communicate, performing coordinated computation. Migration of objects is an important aspect of distributed object computing as it can help alleviate the effect of object or node failures, and communications latency.

A *mobile object* is an object that is capable of moving, or being moved, readily from one place to another. Mobility within a distributed object system allows mobility-enabled objects to move freely between processing nodes within the distributed system. Mobility within a distributed object system supports:

- load balancing (by moving objects from heavily loaded nodes to lightly loaded nodes),
- fault tolerance (by moving objects to new nodes in response to node partial failure),
- locality of data access (by moving objects to the vicinity of the data they require) and
- scheduling and monitoring of the distributed system (by allowing monitoring objects to move about the distributed system assessing load and scheduling requirements).

Mobility can be used to hide latency within a widely distributed system by providing specialised communications styles [45], such as moving an object to a remote site to perform computation and then recalling the object to extract result values in place of a traditional remote invocation in client/server systems. Additionally, mobility in a distributed object system allows flexibility in the programming of distribution by removing static location definition restrictions.

Mobility has been examined at many different levels: from thread or task-level mobility within multiprocessor systems [102], and process or object mobility within a distributed system [128, 130] to host mobility within a disconnected network [6, 93, 146]. One of the most important issues when dealing with mobility within an object system is that of locality and reference management [129,130,158]. Objects within a mobile object system may have

references to other objects within the system. When these objects move, it is important that any references to them remain valid and that any referencing object always be able to contact the mobile object regardless of the mobile object's location.

An object reference that does not expose the location of the referenced object is said to be *location transparent*; a location transparent reference can access a remote object in the same manner regardless of the remote object's current location or state. Location transparent references have been implemented in several distributed object systems [83, 86, 109] and allow distributed communication to be programmed without exposing or requiring location specification. Transparency decreases complexity in distributed programming by taking away the need to define object locations, but incurs additional communications overhead as the location transparent reference must be resolved or evaluated to a direct reference to the remote object [47, 113, 162].

As stated by Shapiro *et al* [164], *"it is essential for performance that a stub contain the actual address of its target"*, where a stub is a type of reference. However, this requirement leads to difficulties when the embedded location becomes invalid through object migration. An object reference that does not require knowledge of the mobile object's location is said to be *location independent*.

An object that can change location without affecting other objects is said to be *relocation transparent*. In a relocation transparent system, a client does not have to explicitly update its references in any way as all relocation is performed transparently. An object that can additionally change location without performing an explicit migratory action is termed *migration transparent*.

An additional problem in a distributed object system is that of initially obtaining the location of, or a reference to, an object. This task, and that of maintaining a reference, is often performed by *naming* an object in some way and making that name known to potential referencing objects. Saltzer states that *"Names for objects are required so that programs can refer to the objects, so that objects can be shared, and so that objects can be located at some future time."* [158]. A name can take the place of, or be a part of, an object reference. This causes the location transparency and location dependence of the reference to be dependent on the name's location transparency and location dependence. For example, suppose an object reference is a combination of its object or procedure name, its host and optionally some unique identifier for that host. This produces a globally unique name, however it is neither location transparent or location independent.

The mechanism of matching a name to an object is termed *name binding*; the mechanism of obtaining an object or object reference from a name is termed *name resolution*. A *naming context* is the domain in which the names are understood. A name is always resolved with respect to some naming context. A naming context may be a local context such as a single node or a cluster, or a global context may be defined. A *naming model* defines how binding

and resolution is to be performed and how strong or weak restrictions on these actions will be. A naming model is implemented by a *naming system*.

Many distributed object systems utilise a global or local naming service of some form to identify services registered within the system. This naming service provides the ability to locate a service from a name, to link or bind a service to a name and potentially to track a service once linked to a name. Distributed object systems do not have an inherent need for transparent names, as once a name has been resolved or a reference has been established, the reference is indefinitely correct unless server failure or shutdown occurs. In this form of system it is difficult to support mobility as references are neither location or relocation transparent.

Many mobile object systems do not support a naming service or relocation transparency and, as a result, provide no way of registering or tracking the identity of a mobile object. If it is to maintain consistency, a mobile object system that allows external references to its objects has a requirement for a globally unique name that is immutable for the lifetime of the object. Once an object has been created and other objects have references to it the name of the object must be globally unique to permit tracking throughout the object system. A mobile object system that wishes to provide this, and similar, forms of consistency and also provide location and relocation transparency must provide a naming system of some kind to manage its namespace.

This thesis proposes a model of location and relocation transparency that uses a distributed naming system. An expressive naming model is used to resolve names to independent, location transparent references. The naming system is also used as a framework that provides mobility in a location and migration transparent manner. This framework can be extended to provide other services such as transparent replication and service cloning. Names in this model act as object references and have variable, hierarchical contexts, leading to some names being part of a global context and others part of a local context.

Binding of names to objects was initially studied in depth by Saltzer [158, 159] with respect to naming objects within operating systems (at the level of addressing schemes to support dynamic binding and file systems)[1]; and has been examined in the implementation of many mobile object systems [41, 62, 102, 162]. Bayerdorffer [15, 16] defines a model of name binding that is applicable to concurrent object systems. This model defines several properties that can, additionally, be used to classify distributed and mobile object systems. Bowman *et al* [26] present a formal model of name resolution. These models are defined in more detail in Chapter 3, and are used to classify existing distributed and mobile object systems according to their name binding and resolution schemes in Chapter 4.

[1] A detailed study of naming and binding relative to a distributed directory structures with focus on client caching is given by Terry in [174].

The classifications presented in Chapter 4 are used to highlight deficiencies in the formal models and outline additional elements or changes to the models that increase their suitability for classifying the example systems. Chapter 5 presents the defined extensions to the models and uses these extensions to classify pertinent systems and to define the extended naming model used within this thesis.

Distributed name services have been studied in the context of distributed file systems [35, 158] and global distribution systems [112]. The need for scalability in the name service requires the name service to be distributed over the processing nodes within the system rather than be centralised [174], however this distributed nature introduces coherency and update problems [35, 112]. These issues have been discussed by Lampson in great detail [112]; additionally by Cheriton and Mann [35] and Terry [174]. A centralised name service for a widely distributed object system introduces a central point of failure and is also a bottleneck for communications; a distributed name service alleviates these problems but forces the system to either pass around large amounts of data (to keep replicas of a logically centralized name service) or to pass requests through some hierarchy. The design issues involved in designing a scalable and coherent distributed name service form part of this thesis.

This thesis explores support for object mobility within distributed object systems, specifically that found in systems based on an Object Request Broker (ORB) model. An ORB system is a form of distributed object system that utilises an intermediate well-known broker object to provide and manage references on a small scale. An ORB does this by providing mappings between names and objects, supporting facilities for name binding and name resolution. ORBs are an interesting platform as the model is easily extensible and provide mechanisms for integrating additional support services into a distributed or mobile object system, such as security or trading services [136].

This thesis extends the naming models provided by Bayerdorffer [15, 16] and Bowman *et al* [26] to be more suitable for classifying distributed object systems with mobility. New characteristics applicable to the classification of distributed and mobile object systems are defined and existing systems are classified according to this extended model. An expressive naming model based on these extended characteristics is outlined and forms the basis for an implementation of location and relocation transparency within a distributed ORB system. An implementation of a distributed ORB using this expressive naming model to support object mobility and location and relocation transparent references is described. Analysis of this implementation shows that the additional overhead in maintaining transparency is outweighed by the benefit of robust reference management and the scalability of the extended naming model and the distributed ORB system.

1.1 Models for Relocation Transparency

Several solutions have been suggested to maintain the validity of existing references within a distributed object system. Mechanisms have been proposed based on the concept of a *home* location; a home location is updated whenever an object moves and serves as the client contact point for referencing. A home location can also be responsible for a form of "descriptive" naming in that it keeps track of any additional information or status, such as whether the object is idle, busy or blocked. The home location model has been used to support location transparency in [12, 34, 49].

One common extension to the home location idea to to use a mobile home location [61, 116]. This results in a chain of homes or *forwarding* locations leading from the original home location to the current location of the mobile object. A client reference may point to any link in the chain. The forwarding location model has been used in [4, 6, 84].

A fragmented object [46, 122, 123] model is often chosen as the distribution model in a distributed object system. A fragmented object is one with several components that may be distributed over nodes within a distributed system, while appearing to be a single entity to external clients. A fragmented object consists of several types of component:

- a set of fundamental components that make up the application,
- client interface components, existing on the client's system, that support communication within the fragment,
- an interface between fragments called the group interface, and
- low-level shared fragments, known as connective objects, that enable communication between other fragments.

By having the server object distributed over several nodes, including client nodes, the client can access the server abstractly regardless of the location of the fundamental components. Multiple client interface fragments are distributed to the sites of clients to allow this form of fragmented access. A client interface fragment can then be used as a named reference and is responsible for maintaining reference validity and for any transparent or independent locality that the model may support.

Fragmented objects are similar to concepts used in RPC [20, 22], CORBA [138], DCOM [27] and Java RMI [167]. The fragmented object model differs from these in that it provides support for connective objects and group interfaces that enables multiple levels of fragmentation, while the other systems provide only one level of fragmentation: separating client interface fragments and the remaining service fragments. Fragmented objects also differ in that they can provide multiple client interface fragments dependent on the client.

A client interface fragment is often termed a *stub* [22, 138, 167] and can be viewed as a simple proxy of the server object within a client/server communication. The stub has

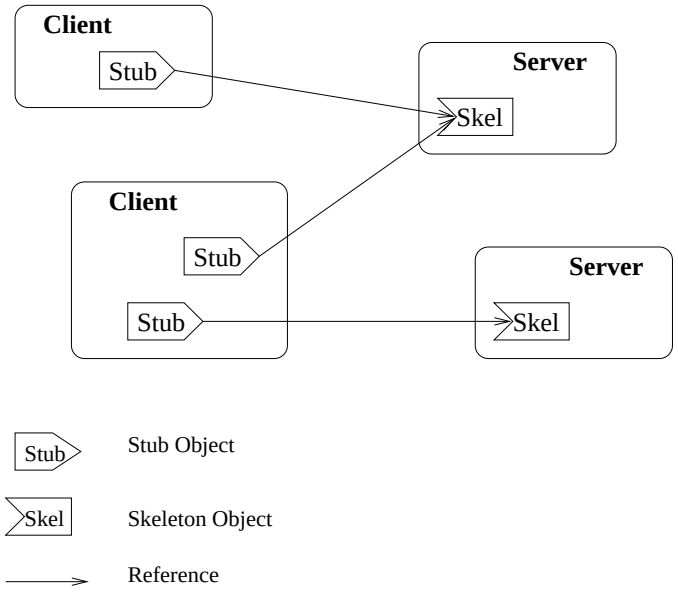

Figure 1: A stub-skeleton style communication.

identical method signatures to the server and, hence, provides an abstract local interface for all server methods available to the client. The stub is responsible for performing the actual remote invocation and also may be responsible for maintaining reference transparency and/or reference independence. On the server side additional code (often termed a skeleton [22, 70, 83]) is used to translate the remote invocation from the stub to a local invocation on the server object. Figure 1 shows a typical client/server system that uses stubs and skeletons to perform remote invocation.

A further alternative used to manage location referencing is that of stub-scion pairs [164] (SSPs). A stub is maintained as the object reference on the client side and a scion is maintained for each stub on the server side; a stub-scion pair is produced for each object reference created. When an object reference is moved, a new stub-scion pair is created to produce a chain of pointers through which the correct reference can be obtained.

The way in which the stub-scion technique differs from that of forwarding locations or a home location is that a stub within a stub-scion system may have multiple references to the referenced object. A stub will contain a *strong locator* reference which will always lead to the destination object and potentially multiple *weak locator* references that are not guaranteed to complete but may contain a shorter path. A strong locator references through the chain of SSPs which will eventually resolve to the server object. A weak reference will directly reference the last known location of the server, encouraging efficient communication. Once the server has relocated, the weak reference become invalid. A weak locator may be returned as part of an invocation through a strong locator reference, and can be used as a direct path for future invocations.

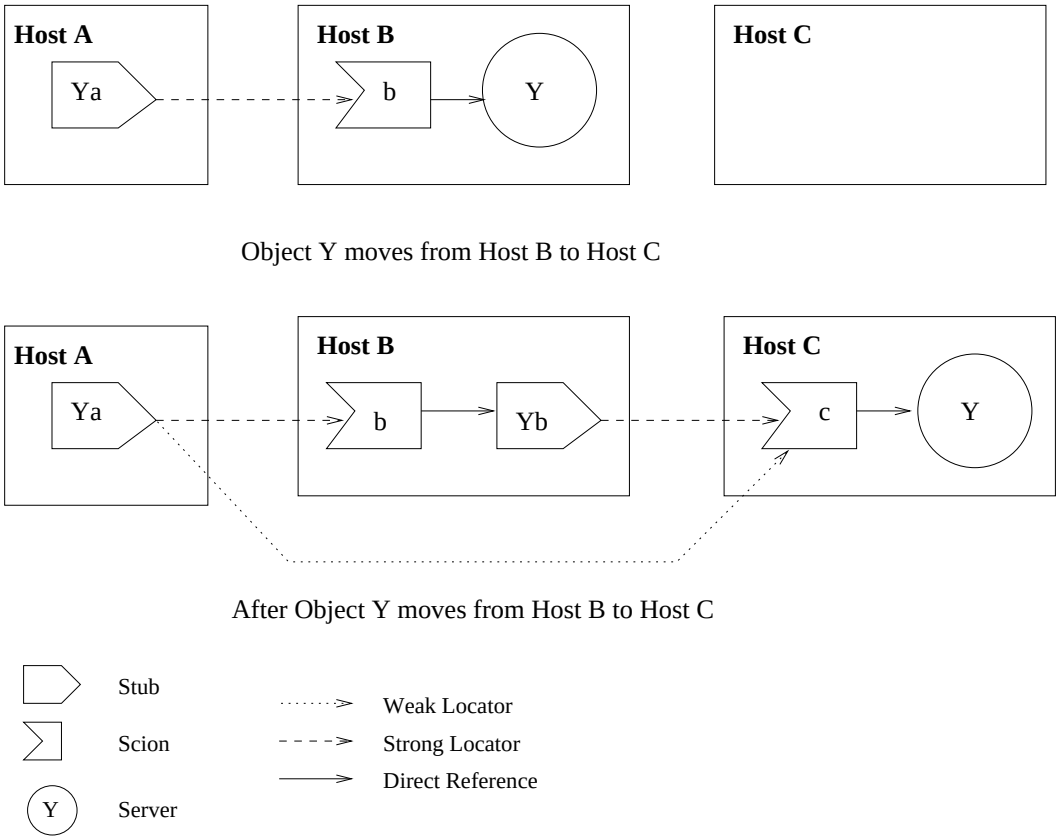

Figure 2: Reference management using stub-scion chains.

Figure 2 shows an example of the stub-scion mechanism where Object Y relocates from Host B to Host C, leaving updates of its location in the form of a SSP chain. Instead of a single forwarding object or home object, scion objects are maintained for each existing reference enabling a form of reference counting. Shapiro *et al* [164] use the stub-scion mechanism to support a distributed acyclic garbage collection system where the invocation protocol is used to provide suggestions of weak locators. Stub-scion chains have been used to support reference management in Hobbes [121] and work by Baggio [7].

A central registry has also been proposed by systems such as Gardens [154], which treats a distributed system as a closely linked parallel system. Each reference is treated as if it were part of a shared memory system where offsets are managed as part of reference access; this results in additional overhead but removes any required reference updating and also any naming requirement. However, this type of system is only suitable for a restricted set of distributed object systems where the structure of the system is known. V-System [124] introduces a search-based mechanism where references are constructed of (*logical host id*, *local index*) pairs. When an object moves, the logical host is duplicated and then set to the new physical host address. As existing references become invalid, their entry in the mapping cache becomes invalid and a request is broadcast to the network for the new logical host id.

1.2 A Proposed Model for Relocation Transparency

The model for location and relocation transparency proposed in this thesis relies on a distributed and scalable registry system which acts as both a mobility service and a naming service.

Location transparency is provided through the use of a distributed naming service. A fragmented object model is used, with some location dependent information kept within the client interface fragment as cached data only. When this data becomes invalid, new location information can be obtained from the distributed naming service [51].

Three types of reference are recognised within the proposed model:

- Connected: references given to a client and currently being actively used.
- Unconnected: references given to clients but currently unconnected to a service.
- Unknown: references yet to be handed out.

A reference that has been enabled and used for remote access is a connected reference. Relocation transparency is provided for connected references as the server object is able to send update messages directly to the client upon migration; reconnection can be established without any loss of messages or requests. This category of reference is the most common to be found within the proposed system.

Unconnected references are possible as, within a fragmented object model (or even a stub model), there is a delay between the return of the client interface fragment and activation of the reference. References are only activated upon method invocation to minimise the number of currently connected clients requiring migration updates. This time delay may coincide with a migration of the object. In this category, there must be a facility for reference updating upon a failed reference activation.

A client that has a name, but this name has not yet been resolved to a reference, has an unknown reference. This object system has no way of knowing that this future reference exists and is unable to maintain or verify the name's validity. In this case, the only requirement on the name service is to maintain validity within itself, *i.e.* its own knowledge of references must be maintained at all times.

Unconnected references and connected references are similar to unrefined handles and refined handles as introduced in the refined fragmented object model [46]. In the refined fragmented object model, handles are client interface fragments that contain information on how to contact an object. A handle can be unconnected due to the time delay between reference transmission and connection. Handles are connected immediately when the client receives them, not at a future communication point. Similarly to handles, unconnected and connected references are capable of being freely copied within the distributed system.

Figure 3 shows an example of the proposed location mechanism at work. Object A initially has an unconnected reference to object B (i), which is then connected using a client

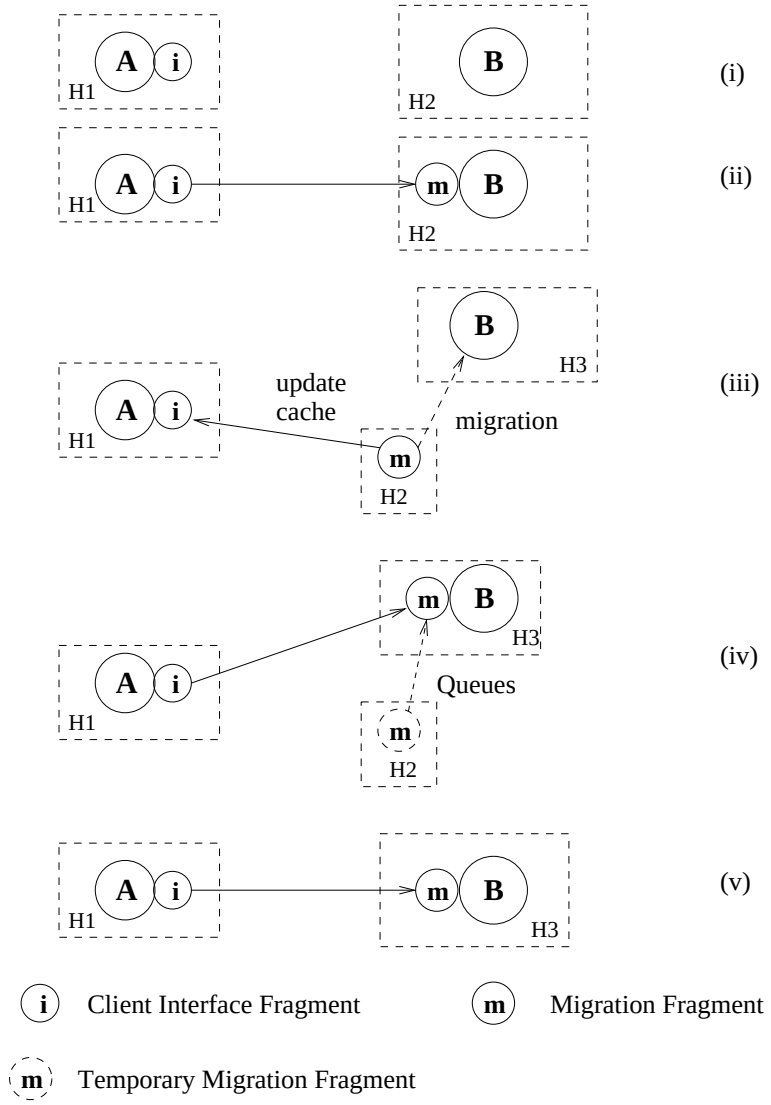

Figure 3: Location and relocation within the proposed update model.

interface fragment and a *migration fragment* (ii). The migration fragment is responsible for handling and updating connections and disappears after a completed migration. It does not act as a forwarding pointer at any stage. When object B relocates, object A's reference is updated (iii); after migration is completed and request queues have been sent to the new location, object A can reconnect and resume communication (iv). In (v) the client has a newly connected reference to the new location of B.

A migration fragment is created for each connected client. This means that direct communication and updating of connected clients can be performed, while unconnected and unknown clients are unknown to the server and must perform relocation through the name registry system. Relocation transparency is maintained by providing migration and

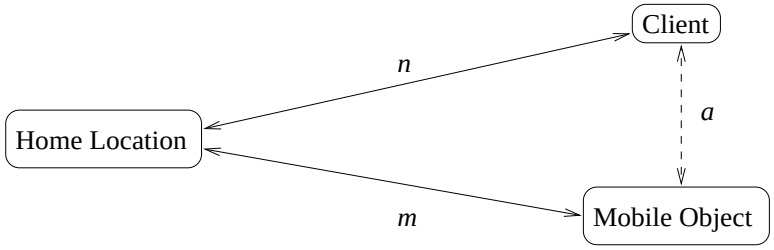

Figure 4: The trombone problem.

client interface fragments which perform any additional update tasks and location querying. No residual objects are left after object migration.

Interaction with the distributed name service is required when cached location information becomes dirty without an update from a connected server. Relocation can be performed by requesting a reference or a reference update (cached information only) for a name matching the server. A name is maintained within the client interface fragment, and consists of the object name and a unique identifier that identifies the communications channel. This identifier is used to reconnect and claim queues of migrated requests. These mechanisms are described in more detail in Chapter 6.

1.3 Comparison of the Models

One of the potential problems of the home location model is that the home location acts as a central point of failure. If the home location fails, all existing object references and any future object references become invalid. This also introduces a communications bottleneck. An additional problem is that there may be large communications overhead unnecessarily incurred if the mobile object and client are both separated by large distances (in terms of latency) from the home location, but close to each other. This is commonly known as the *trombone* problem [43, 152].

Figure 4 shows an example of the trombone problem where a client has to communicate through a mobile object's home location where the distance between the objects causes a large latency. It would be more efficient if it could communicate directly with the mobile object. The values n and m in Figure 4 represent the latencies between the home location, and the client and mobile object respectively, a represents the latency between the client and mobile object through a direct communication, where $a \ll n, m$.

In a forwarding location model, a client may have any of the forwarding pointers as their first reference to the mobile object, alleviating the bottleneck and central point of failure issues. However, for initial clients the list of forwarding pointers to visit may become arbitrarily large and potentially cyclic. A failure of a member of the chain will invalidate any references to earlier chain members. Stub-scion chains incur the same cost as the

forwarding location model for the first invocation, however a shorter path can be returned or *piggy-backed* as part of an invocation response to produce a direct reference (consisting of a single SSP) for future usage.

Location management using home location and forwarding pointers are commonly used methods in the areas of mobile object, mobile process and mobile host systems [129, 130]. The problems in these methods exist due to the presence of residual objects within the object system. The use of residual objects results in out-of-date location information being used throughout the object system; additionally this information is required by clients to locate server objects. A reference that has to contact a home or forwarding location is not location independent and in some systems, not even location transparent [138].

The model proposed within this thesis makes use of a fragmented object model with migration fragments existing only for connected references. These migration fragments do not act as a chain, a single migration fragment is used as a direct reference. Migration fragments also allow updated location messages to be forwarded to the client. Unconnected references can update their cached location hints by contacting the registry system. The time between the provision of a reference and its connection is client dependent; in a system where migration is infrequent this delay will be small (minimising relocation possibilities) hence making this case uncommon.

Where relocation is common but client access is rare, the home location model does not suffer from bottleneck issues. When the scale of distribution is small, the home location model is ideal as the effect of tromboning is also removed.

In the case where relocation is uncommon but client access is frequent, the forwarding location model and, more so, the SSP chain model can be efficient as the length of the chain in each case will be small.

The proposed model is suited to the case where relocation is common and client access is frequent. References can be updated transparently for connected clients without an increasing chain of forwarding locations. Bottleneck issues are reduced due to the distributed nature of the registry system and direct referencing. All models benefit from a smaller scale of distribution due to the reduced latency costs and reduction of any potential trombone effect.

1.4 The DISCWorld Metacomputing Environment

The proposed model of location and relocation transparency based on top of a distributed registry system is implemented as part of the Distributed Information Systems Control World (DISCWorld) metacomputing environment [78, 82, 104].

The DISCWorld metacomputing environment provides middleware support for distributed systems. It supports web-based client access to legacy and specialised services running within the DISCWorld system. DISCWorld supports dynamic reconfiguration and

adaption in response to data access requirements and processing load within a data-intensive high-performance system. Additional services that DISCWorld provides are meta-data resource discovery, data transport, process scheduling and monitoring services.

Several high-performance legacy applications have been developed to support Geographical Information Systems (GIS) within the DISCWorld framework [37, 38, 103]. These services have been integrated into the DISCWorld metacomputing environment and provide a mixture of mobile and host dependent code; and code implemented for high-performance and low-performance systems.

1.4.1 The DISCWorld ORB System

The distributed registry system is based on an ORB model, and acts as a name server and a framework for linking additional services into the DISCWorld ORB system [51, 104]. The DISCWorld ORB system is designed for native mobility support; this is implemented on top of the ORB system as an additional service.

An ORB model defines a model for client/server communications utilising an independent third party. Access to this third party may be transparent or it may be direct depending on the system model or even on the types of communications and reference access used within the same model. Examples of well known ORB models are the Common Object Request Broker Architecture (CORBA) [136,138] and Java RMI [70,167]. These are commonly identified examples of ORB systems, however many other distributed systems fit the ORB model, including Globus [54, 57] and Globe [177–179].

Essentially a distributed object system based on an ORB model is one that uses a third party object to provide references and additionally manage name binding and name resolution to some extent within client/server communication. A server will register its service and potentially bind a name to this service (alternatively a name can be provided by the naming system itself) through some facility provided by the ORB. This could entail binding to an external ORB service [136] or accessing internal library code [39, 167]. A client will request a named service from the ORB and obtain a reference, generally in the form of a stub, to abstract over remote communication requirements.

The ORB provides a mapping between a name and the service object and is able to provide stub code or a method to obtain stub code as provided. Figure 5 shows the operation of typical client/server communication within an ORB model. Initially (i), a server object registers its service or name with the ORB. When a client requests name resolution by the ORB, a stub is returned to the client (ii). This stub can then be used to remotely invoke methods on the server by communicating with the server's skeleton object (iii).

An ORB model is useful for dynamic distributed object systems as it acts as a well known contact point through which clients and servers can communicate. A publicised port is commonly used as the ORB contact point. If communications libraries are provided to

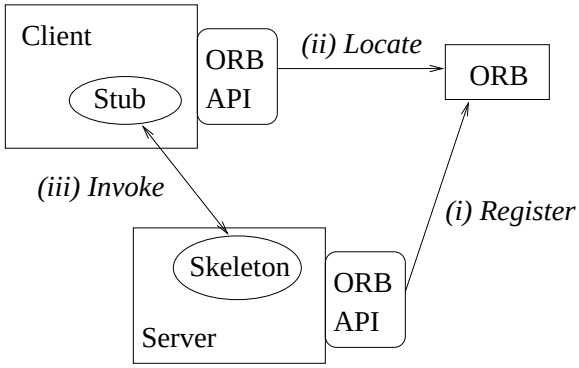

Figure 5: Stub-based client/server communication within an ORB system.

the client and server objects, this enables an ORB system to be accessed without explicit remote communications required by the client and server code.

ORBs are often used as well known access points for additional services provided as part of the distributed object system. Some commonly found services include security services [136], trading services [17, 137] and contextual naming systems [135]. Mobility services have also been proposed [128], while ORB-based systems with inbuilt mobility have also been proposed [131].

The DISCWorld ORB system provides mechanisms for clients within the DISCWorld system to register services, obtain references to services and perform optimised communications. To support scalable wide-area communications, a communications construct has been developed which utilises the inbuilt support for mobility and transparent reference updating (see Section 6.6). The DISCWorld ORB system provides additional service support for object migration, replication and cloning.

The DISCWorld ORB system has a hierarchical directory structure with adaptive components and a distributed nature. Different models for updating these directories are used dependent on the level of the registry within the hierarchy and the expected latency within the system. The DISCWorld ORB's naming model is consistent with the extended naming model detailed in Chapter 5 and provides support for service relocation, name aliasing, name extension and dynamic reassignment of names to objects. This naming model avoids polling of services and provides a consistent communications scheme to support the distributed directory service.

1.4.2 Distribution and Object Models

The distributed object model used within this thesis an example of a fragmented object model. A client interface fragment or stub is used to separate the communications from the application object. Client/server communications then occur through the stub and the group interface fragment. Additional fragments are used to perform the low level

communications within the system and fragments are required to interface to the naming system.

Objects are accessible through predefined, typed methods either locally through direct invocation or remotely through the stub interface. Multiple stub interfaces can be accessed by the system to support multiple protocols and multiple client types.

Objects that can be accessed or referenced remotely by another object must be named. Local access need not be named although this is not prevented. Additionally objects passed or returned in a remote invocation are passed by reference if named, but by value otherwise. Naming is performed on a per-object basis so that multiple instantiations of the same class or code may have different names. An object is named if it is registered with the ORB system or has been exported as part of a remote communication; objects that are only accessed locally are not named. Primitive types that do not exist as objects can not be named.

Names are constructed using a contextual mechanism that supports three contexts. Objects are accessed through their code defined names (such as variable names) within the object's private address space. Local names can be used within the local name registry system that consists of a small collection of nodes involved in a computation; global names can be used universally between name registry systems. An object may have multiple names (global and local) where the local names can only be used within their valid context. User defined contexts beyond these facilities can not be defined.

1.5 Contributions

This thesis discusses models for location and relocation transparency used within existing mobile and distributed object systems. A new model, based on the combination of an explicit update model and an ORB-based distributed directory system, is introduced. Location transparent and independent naming to support this model is provided through a global namespace managed by the ORB-based distributed directory system.

This thesis describes the development of a formal naming system classification model based on existing work in the area by Bayerdorffer [15, 16] and Bowman *et al* [26]. Attention is also payed to other naming models preceding Bayerdorffer and Bowmans' work [50, 63, 89, 92, 158, 159]. This naming model is used to define a separation between the naming system and the remaining parts of the system in which it will be used. The naming models defined by Bayerdorffer and Bowman *et al* are used to classify existing mobile and distributed object systems. This classification shows that these models do not correctly or completely classify these kinds of object systems. Extensions and refinements to these models are proposed that enable a complete and correct classification of existing mobile and distributed object systems, specifically with a need for transparency.

Bowman *et al* define a formal model for defining name resolution systems. This formalisation is extended to formally define first Bayerdorffer's name binding model and then the extended naming models. The definition of a formal classification model enables the development of a generic support framework for naming systems.

An implementation of the proposed relocation transparency model within a distributed object system with support for mobility is presented. The DISCWorld ORB system is introduced as a motivating framework for the transparency and naming models. The DISCWorld ORB system provides a global namespace managed by a naming system. The implementation of this naming system is separated from the remaining implementation and is based on, and classifiable by, the extended naming system classification model.

The DISCWorld ORB system supports an Application Programming Interface (API) for client/server access to the registry system. This API allows transparent access to the distributed directory service without requiring client/server knowledge of directory location or structure. The DISCWorld ORB APIs provide support for object migration, cloning and replication. This system provides an API for dynamically constructing mobile services to perform batched remote invocations. This construct enables a sequence of connected remote invocations to be programmed as an itinerary. The mobile object is then responsible for colocating with the required service objects to perform more efficient local invocation.

This thesis describes an implementation of a prototype of the DISCWorld ORB system which supports distributed and scalable object location and relocation. Experimentation with the DISCWorld ORB system has shown that it provides better performance than existing models for relocation transparency in terms of cost performance and scalability. The DISCWorld ORB system also provides support for fault resilience and removes central points of failure.

This thesis makes three main contributions. The first contribution is the development of a formal naming model that is suitable for classifying existing mobile and distributed object systems. The formal definition of this naming model enables support for generic naming systems to be separated from the development of the remainder of the object system. Separation in this way enables the naming model that is used within an object system to be changed and updated according to requirements of the object system.

The second contribution of this thesis is the development of a model to support location and relocation transparency. This model provides relocation transparency using location transparent and independent references and has no central point of failure or bottleneck issues.

The third contribution of this thesis is the development of a prototype distributed ORB system, the DISCWorld ORB system, that supports the proposed generic naming model and the model for relocation transparency. The DISCWorld ORB system is implemented as a distributed, hierarchically structured, ORB system that is capable of fault resilience and adaption. Experimentation with the DISCWorld ORB system shows that the proposed

model for relocation transparency provides better location and relocation performance and is scalable in terms of the number of nodes within the system, the number of names within the global namespace, and the frequency of client requests.

1.6 Thesis Structure

This thesis is divided into three logical parts. The first part introduces the problem and defines the problem area through a literature review. This introductory chapter outlines the area within which the rest of the thesis falls and outlines the main contributions of the work. This chapter motivates mobility in distributed object systems and discusses additionally the need for naming in distributed object systems for client/server communication. It outlines how naming can provide the facility for mobility and transparent locality within a distributed object system. A system is outlined, based on an ORB-style system where an expressive naming model is used to provide location transparent and independent mobility. The benefits of this system are outlined and examined. Chapter 2 examines existing mobile and distributed systems, specifically looking at each system's support for location and relocation transparency and distributed communication.

The second part of this thesis discusses the issue of naming. Chapter 3 presents problems in the area of naming within mobile and distributed object systems. Naming models used to classify naming systems, including the models defined by Bayerdorffer and Bowman *et al*, are presented in both an informal and formal manner. In Chapter 4, these naming models are then used to classify the naming systems of existing mobile and distributed object systems. A taxonomy of the classified naming schemes is presented. This classification is used to highlight deficiencies in the current models. Chapter 5 presents motivation for extending the naming models to provide characteristics suitable for distributed object systems with mobility and distributed systems constructed around an ORB model. It presents the extended model and uses it to form a reclassification of the systems described in Chapter 4. This extended model is then used to define a naming system suitable for location and relocation transparency in a distributed object system with mobility support. An expressive naming system is presented and it is shown how this system can support transparent object mobility by providing a state dependent mixture of location dependent and location independent mappings.

The third and final part of this thesis presents the example system that has been developed as part of this work and shows how this system is used to support the naming models and relocation model developed throughout parts one and two of this thesis. Chapter 6 outlines the system that has been developed to support mobility of objects within the DISCWorld Metacomputing system. It presents the naming model and its classification suitable for use with a mobile object system. Chapter 6 also discusses how the naming model can be extended to a distributed directory system with partially contextual names.

Chapter 7 presents a discussion of the implementation of this system and how tasks such as distributed aliasing were achieved. Chapter 8 presents an evaluation of the extended naming model and its effectiveness within a distributed ORB system supporting mobility. A comparison between the performance (with respect to both relocation cost, location cost and communication cost) using the proposed model for transparency and existing models is presented.

Chapter 9 presents conclusions from this work and outlines future work that is still to be done in the areas of relocation transparency and naming.

Appendices present more detailed descriptions of the policy specification and communications protocols that are used within the DISCWorld ORB system. APIs for client and server access are also provided.