VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui

School of Mathematical and Computing Sciences
Computer Science

NOMAD: An Infrastructure for Mobile Applications

Kris Bubendorfer and John H. Hine

Technical Report CS-TR-02-17
November 2002

School of Mathematical and Computing Sciences
Victoria University
PO Box 600, Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Email: Tech.Reports@mcs.vuw.ac.nz
http://www.mcs.vuw.ac.nz/research

VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui

School of Mathematical and Computing Sciences
Computer Science

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341, Fax: +64 4 463 5045
Email: Tech.Reports@mcs.vuw.ac.nz
http://www.mcs.vuw.ac.nz/research

NOMAD: An Infrastructure for Mobile Applications

Kris Bubendorfer and John H. Hine

Technical Report CS-TR-02-17
November 2002

Abstract

Mobile code is a powerful paradigm for the creation of distributed software. Applications constructed of mobile code are able to execute more efficiently, exploit heterogeneity and dynamically adapt to changes in their environment. This paper describes the NOMAD (Negotiated Object Mobility, Access and Deployment) architecture, a platform that provides a distributed systems infrastructure to support applications constructed of mobile code. The challenges faced by such applications include: the discovery of the resources required for execution, the establishment of rights to access these resources, and the ability of an application to locate and coordinate its component parts. These issues are addressed by three complementary platform elements. Firstly NOMAD embodies an electronic Marketplace through which applications locate and obtain the resources they require. Secondly, applications form contracts with the Java virtual machines (Depots) on which they execute, specifying the quality of service to be provided. Lastly, NOMAD provides a global location service, to track the migration of code throughout the system and enable the various components of an application to coordinate to satisfy their collective goals.

1 Introduction to NOMAD

NOMAD consists of loosely coupled cooperating virtual machines, named Depots, that host distributed applications. Each Depot independently provides a collection of local resources ranging from CPU cycles through to consistency protocols, which are made available to applications in return for payment. Applications utilise these resources to provide services to their clients, and negotiate to alter their resource profile as the demand for their services changes (Fig. 1).

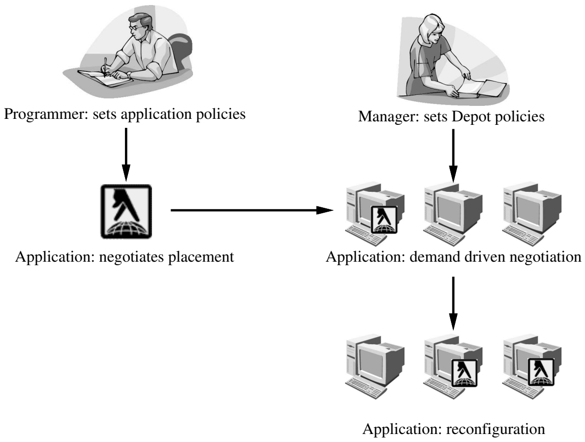

Figure 1: *High level view of NOMAD*

NOMAD provides the global infrastructure that integrates Depots and applications. In particular, the provision of migration support, a location service and an impartial resource negotiation mechanism [2]. The objective of this paper is to provide an overview of these primary infrastructure components within the NOMAD architecture.

1.1 Economic Resource Management

Traditional resource management has been undertaken by the operating system. In this role, it has the conflicting goals of optimising the resource allocation for the entire system (*globally*) and for each application (*locally*). This is difficult as the operating system is neither aware of, nor privy to, the needs or goals of each application. A solution to this problem is to separate the local optimisation from the global optimisation, that is, the application must negotiate for the resources it requires from the system on which it wishes to execute [30].

This is similar to the problem of allocating finite resources over an independent population — as faced by all human societies. For example, a farmer wishes to get the best price for his wheat, while the baker wishes to pay less. The solutions to this and related problems have been refined by many years of human competition and form the basis of economics [18]. This well founded and understood technique of *resource pricing* has long been applied to the problem of allocating finite computational resources [32].

The economic model is also powerful enough to provide a mechanism, with which to solve a number of other problems relating to the behaviour of entities within the system. One such problem in open mobile systems is how to discourage mobile objects from maliciously (or otherwise) wasting or over-using resources on a host they are visiting. This is especially a concern when the host and object have different administrative domains or ownership. Rather than designing a system that predicts the entire range of possible actions an application might take, and then permits or forbids those actions depending on the probable impact on the system, financial incentives [6] can be used to influence the behaviour of entities within the system. An example of this is the denial of service attack which, if the instigator is charged for the actual resources consumed in the attack, becomes infeasible.

The incentives in the construction of the economic system are designed such that it is in an entity's *best interest* to behave in a desirable way [27]. Infractions cost real currency, and as a consequence badly behaving entities suffer a real financial loss and are unable to utilise additional resources once their currency is expended. This is not a complete panacea, as the system must still ensure that the entities within the system are protected from other forms of interference, and that the payment and negotiation systems are secure. Nonetheless, utilising an incentive model is a rational choice and motivates the use of a market approach to resource management in NOMAD.

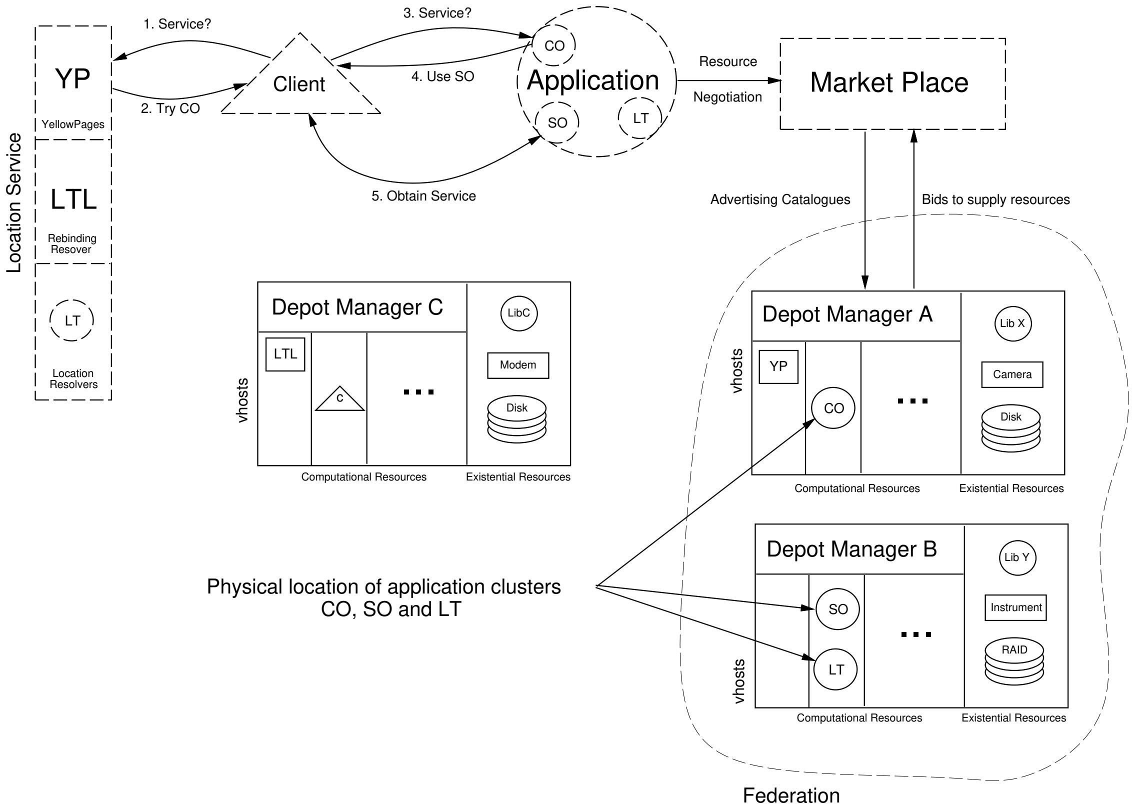

Figure 2: *Entities within the NOMAD system.*

1.2 Structure and Composition

An application within NOMAD is a logical grouping of mobile code that performs one or more services (functions) for some client (another application, user etc.). An application can range from a single mobile agent, implemented within a single piece of code and offering a single service, through to a collection of many pieces of mobile code offering many different services. The distinction is that an *application* is an autonomous unit of administration and identity, that is, the parts are cohesive and managed collectively.

Clusters form the unit of addressing and mobility within NOMAD, as many language based objects such as integers and strings are too small to be independently mobile. Each application is responsible for: initiating negotiation for resources, directing the distribution of its *own* clusters, handling unrecoverable failures, and maintaining part of the location service. Applications provide services to clients, employing the resources of one or more Depots, and may themselves be clients of other applications.

Each application controls its degree of distribution by placing and replicating its clusters to achieve QoS goals such as reduced latency to clients, redundancy and concurrent processing. The application negotiates with the Depots via the NOMAD Marketplace for the resources that it requires. For their part, Depots use policies to price their resources and to organise such things as the balance of work amongst a federation of Depots, or the quality of service ranges that individual Depots offer. A federation of Depots reflects common ownership, or administration of the Depots.

Figure 2 gives an overview of the major components within the NOMAD architecture. Each Depot [3] consists of a Depot Manager and a set of managed virtual hosts (vHosts). Management policies dictate how each Depot will react to specific circumstances: how it will ensure levels of service and how it interacts with Depots within and outside of its federation. The Depot Manager is also responsible for responding to negotiation, charging, and arranging the hosting of the global NOMAD infrastructure components. A vHost is a virtual machine on which application code executes. A vHost is responsible for managing physical resource use, security, fault detection, and communication.

The remaining components shown with dashed lines in Fig. 2 are logical entities providing services. The Application represents a service that is supplied to a client. This application has the minimal components: a Contact Object (CO), Service Object (SO) and Location Table (LT). These are described in sections 2.1 and 2.4 respectively. The clusters physically reside on vHosts of Depot A and Depot B. The Marketplace provides a set of services allowing Applications to discover available resources and to contract with Depots for the provision of those resources. Finally, the location service provides dynamic binding to mobile objects and the initial discovery of services — by providing a reference for an appropriate contact object. This is shown in steps 1 through 5, where the client (the triangle) obtains a service from an application (the circle).

The combination of the consistent local infrastructure provided by the Depot architecture, and the global infrastructure provided by NOMAD services, combine to form a consistent and interlocking environment for distributed mobile applications.

2 NOMAD Design Overview

NOMAD services execute on the network of Depots and are responsible for providing a consistent global infrastructure to applications and Depots. This infrastructure includes resource negotiation, location services and a payment service. Resource negotiation includes mechanisms for performing negotiation, ensuring the integrity of negotiations and ultimately, policing contract compliance. The location service unifies the location service components, maintained by individual applications, into one cohesive global structure.

Section 1 presented a general introduction of the various components and their interactions within NOMAD. This section provides a more detailed exploration of the design of these components, starting with the concept of the application, the construction of a Depot, the design of the Marketplace, and lastly details on the operation of the location service.

2.1 Mobile Applications

An application's services to clients are delivered by a set of service objects, but the first point of contact between an application and its clients is the published application contact object. Globe has a similar construct termed a contact-point [33]. However the contact-point is part of the distributed object structure, rather than a separate entity, and is severely limited in functionality.

The contact object negotiates[1] with clients, and determines how the required service is to be delivered. This client-service negotiation is different from the resource negotiation provided by the NOMAD negotiation service. A contact object for a service may, for instance, offer different levels of service, gold, silver and bronze — for which it will charge accordingly.

Once negotiation has been successfully completed, the contact object returns a service object interface to the client. Using this interface, the client then transparently interacts with the service object regardless of the service object's actual location.

This distinction between contact objects and service objects is required to address the dynamic configuration of an application and its ability to meet Quality of Service (QoS) requirements. Both service and contact objects are potentially replicated within the application, and the use of formal contact objects allows the application to distribute the client load over its service objects, or indeed redeploy or create additional service objects on demand.

This separation of contact and service objects has considerable advantage in the design of the location service. It permits a clear partition between service discovery and location resolution, reducing the impact of mobility on the location services.

2.1.1 Clusters

A cluster rather than an object is the NOMAD unit of mobility. An application composed of objects must be able to manipulate all of its objects. In most object oriented languages this implies being able to locate and bind to quite small objects such as Integers. For distribution, this is generally too fine-grained, and the smallest meaningful unit for mobility will consist of one or more objects which provide some encapsulated functionality. Such an encapsulated grouping is a *cluster*.

Each cluster inherits NOMAD functionality supporting mobility and resource negotiation. The cluster also provides an address space that allows references within the cluster to be handled by standard language mechanisms. References outside the cluster, even within the same host machine, take place via proxies.

A cluster also provides an address space that allows references within the cluster to be handled by the implementation language's mechanisms. A new object is created within a cluster and remains within that cluster for its lifetime. (A clone of the object may be created in a different cluster.) A cluster may or may not contain threads of its own.

Objects in different clusters are in different address spaces. NOMAD provides mobile remote method invocation (via a proxy) so that an object in one cluster may invoke a method on an object in a different cluster, independently of either of the clusters' physical locations. Proxies are generated *on-the-fly* by the NOMAD system, placed within the client's address space and rebind transparently (on the next invocation) after the target object moves.

2.1.2 Distribution

An application may be structured in a variety of ways. It might maintain a pool of service objects, of which the closest is used to service a client, or it may create replicas on demand. These structures require the distribution process to occur at different times, *preemptively* and *on demand.*

As an application controls its own distribution it is also expected to meet client QoS demands. Moving closer to the client or a fixed resource is a well recognised technique to improve performance [1, 13], although in these days of ever increasing bandwidth some of these arguments are less compelling [28]. There is nonetheless an opportunity to harness a

[1] It is expected that the majority of negotiations will simply involve stating the one time fee, followed by acceptance from the client.

new application paradigm, and new ways of constructing and funding distributed systems and infrastructure.

2.2 Depots – Resource Providers

A Depot is a pool of resources, a resource manager and a set of vHosts, and is typically a single physical machine. The pool of vHosts is managed by a Depot manager, which executes on its own vHost. This Depot structure is illustrated in figure 3.

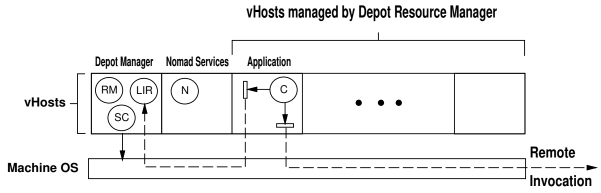

Figure 3: *The Depot is a pool of vHosts, Depot Manager and resident NOMAD services.*

The Depot manager contains a Resource Manager (RM) which manages and allocates the vHosts in the pool, a Statistics Collector (SC) which collects and distributes machine resource statistics for all the managed vHosts, and the local interface repository (LIR). Resource management and negotiation are implemented via policies, as described in section 2.2.1.

This figure also shows the second vHost reserved for the exclusive use of NOMAD services, such as the location service. The client application (C) is shown performing a remote method invocation on the LIR in the Depot Manager's vHost, and a second invocation on an object on some other Depot.

The LIR contains references to services such as the location service, Marketplace and Resource Manager which may be required by clients executing on the vHost. The LIR executes in its own cluster on the Depot Manager vHost, and an interface on it is available in each cluster in each vHost. The LIR is local to a Depot and cannot be accessed from outside the Depot.

Each vHost provides a consistent, secure execution environment along with value added services such as transparent method invocation, fault detection and basic mechanisms supporting mobility and persistence, as shown in figure 4. All references external to a cluster are accessed via proxies resident in the cluster and therefore in its namespace. Objects of the same application, on the same vHost, but in different clusters also require proxies. This allows one cluster to to move to another vHost, on the current Depot or another, without interrupting an interaction with another cluster.

2.2.1 Policy Model

Each Depot is controlled by management policy which dictates how the Depot will react in specific circumstances: how it will ensure levels of service, its allocation policy, its negotiation policy, and how it interacts with Depots in and outside its federation. The Depot architecture separates mechanism and policy. Policy can be applied through a number of mechanisms such as: finite state machines, policy languages or standard general purpose programming languages.

2.3 The Negotiation Service

Markets make an ideal tool for economic resource management.

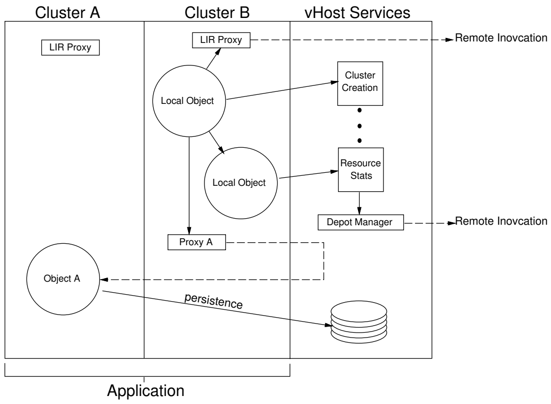

Figure 4: *One vHost with two clusters.*

> *Market price systems constitute a well understood class of mechanisms that under certain conditions provide effective decentralisation of decision making with minimal communication overhead — Wellman [37].*

Indeed, such systems have been used for the allocation of computational resources since 1968 [32]. Markets are particularly appropriate in situations where software is spread across a distributed system, serving different clients and pursuing different goals [18].

Markets are an ideal choice for NOMAD, with distributed decision making, minimal overhead, well understood theory and the abstraction of resources as currency. This permits the NOMAD system to charge for the use of resources, prevent unbounded resource attacks as with denial of service [38], and represent priority. A Market provides a forum in which buyers and sellers meet (resource discovery) and negotiate to perform optimal allocations with minimal overhead, mutual selection and decentralised decision making between buyers and sellers.

In Nomad, as applications direct distribution, they must therefore also initiate negotiation for the resources they require. Depots take on the competitive role of bidding to supply their individual resources.

2.3.1 Contracts

There are two types of contracts formed in NOMAD, the contract resulting from a resource negotiation between a Depot and application, and the contract between a client and an application providing a service.

- **Application to Depot**: The NOMAD negotiation service is used to negotiate a contract that contains an agreed set of resources and QoS specifications. How the Depot will satisfy these QoS requirements is an active and complementary research field [7, 5, 10, 23].

- **Client to Application**: This contract is negotiated through the application's contact object using application specific QoS specifications. The application may redistribute its service objects to meet the client's QoS requirements which may require the application to negotiate with Depots for additional resources.

The NOMAD resource negotiation service is only concerned with the negotiation of contracts between applications and Depots. The requirement for a contact object in each application provides a natural place for the negotiation of a contract between client and application.

2.3.2 Selection of an Auction Protocol

For use in a computational economy, like NOMAD, an auction protocol must lead to optimal allocations with minimal overhead. In this section we look at the suitability of different types of auction protocols.

There are four main types of auction protocol identified by Vickrey [35]; the English, Dutch, Sealed-Bid, and what has since become known as the Vickrey auction protocol. The English auction is the conventional open forum, ascending price, multiple bid protocol. The Dutch auction is an open forum, descending price, single bid protocol. The Sealed-Bid, or tender, is a closed forum, single bid, best price protocol in which all bids are opened simultaneously. The Vickrey auction is similar to the Sealed-Bid auction, except that the winning (highest) bid then pays the amount of the second highest bid.

The most significant result in auction theory is the **revenue equivalence theorem** [35]. This states that all four auction protocols yield the same expected return in private value[2] auctions. In addition, the strategies and payoffs associated with the Dutch and Sealed-Bid auction protocols, even for non private-value auctions, are the same. That is, the Dutch and Sealed-Bid auction protocols are strategically equivalent, in all valuation models, as only the winning bid matters and no information is revealed during the auction process.

In private value auctions the English and Vickrey auction protocols produce the same allocations (at the same prices), where the bidder who values the item most wins it, but do so with different strategies. However, the English and Vickrey auction protocols are not equivalent in non-private value auctions, as the open outcry nature of the English auction protocol provides additional information to the bidders, which can then alter their valuations. Milgrom and Weber [17] show that in correlated value[3] auctions, this enables a variation of the English[4] auction protocol to generate greater revenue than the Vickrey, Dutch and Sealed-Bid protocols when there are three or more bidders.

If a certain strategy pays a player the highest playoff, regardless of other players' strategies, then that strategy is known as a dominant strategy.

Neither the Dutch nor the Sealed-Bid auction protocols have a dominant strategy. The dominant strategy for the English auction protocol is to bid a small increment over the current bid price and stop when the private value is reached. The dominant strategy for the Vickrey auction protocol is to bid the true value of the item. Truthful bidding is the dominant strategy for the Vickrey auction protocol, which has two important and beneficial side effects:

- bidders reveal their values accurately allowing for globally efficient allocations, and
- bidders need not waste efforts in attempting to counter-speculate other bidders.

Counter-speculation wastes computational resources and introduces considerable complexity into the system, while not improving overall allocations.

In addition, there is the obvious efficiency of submitting a single bid, based on the stable description of lot[5] and auction. That is, not only does the bidder submit just a single bid - but they do not need to be kept up-to-date about the current state of the auction in a timely and reliable fashion, as is necessary with the English and Dutch auction protocols. This minimises the number of messages exchanged during an auction.

[2]In private auctions the valuation depends solely on the bidders own preferences. In contrast, the valuation in a common value auction depends entirely on the other bidders values of the item.

[3]Correlated valuations are a combination of private and common values.

[4]When the protocol is **open exit**, that is, bidders are aware when each bidder ceases to bid.

[5]A *lot* is an individual set of goods in an auction, which are bid on as a unit.

The Vickrey auction protocol[6] is a one shot Pareto-optimal [24] technique that is highly efficient in terms of messages and allocation. This suggests it is ideal for machine to machine negotiation and is the auction protocol selected for use in Nomad.

There are however, seven limitations concerning the applicability of the Vickrey auction protocol to computational systems: bidder collusion, the lying auctioneer, possible release of sensitive information, lying in non-private value auctions, lower revenue in non-private value auctions, sub-optimal allocation and lying in interrelated auctions, uncertainty of valuation and wasteful counter-speculation. These limitations have been identified in the literature [24][26][35], and are probably amongst the reasons why the protocol has been abandoned in some recent projects.

The Vickrey auction is an ideal basis for automation, once its limitations are addressed in the design of the Marketplace that implements the protocol, as is the case in NOMAD. The protocol satisfies the requirement for stability, by having a dominant strategy, while the system is individually rational as truthful bidding ensures that the payoff is not less than not participating. Finally, the system is symmetric, as the same behaviours will result in the same payoffs.

2.3.3 Describing Multiple Requirements

Another challenge is the focus of standard auction mechanisms on the sale of a single good. This is fine when the good is a single unit, such as a soccer ball, but not when the good in question is, say, one sock — which has its greatest value when part of a pair. Likewise, execution resources form an indivisible set, related and conditional upon the availability of each other. Piecewise negotiation of these individual resources will not give any usable result let alone an optimal allocation. After all, it is very difficult to execute when the memory is on a different host from the allocated CPU cycles. This is the combinatorial allocation problem (CAP) [25, 21], in which a set of components have a synergistic value that exceeds the sum of the individual parts. Because of synergistic combinations and substitution effects, bidders have preferences not just for particular items, but for collections of items.

The solution to the CAP is the Generalised Vickrey Auction (GVA) [16], which utilises a single auctioneer that is required to solve the CAP for all resources and all bidders. This requires however, not just a single *NP*-complete computation, but multiple *NP*-complete computations to find the optimal solution. Approximations such as iBundle [20] still require that the auctioneer perform multiple *NP*-complete computations.

Clearly these existing approaches are untenable in practice, and would severely limit the scale of any system in which they were utilised. Our approach is to move the computation of the CAP from the auctioneer to the set of bidders. To describe multiple resource constraints NOMAD utilises a resource description graph (RDG) [2]. An RDG is a single directed acyclic graph which describes an auction, its set of lots and associated compromises. Each sentence (a path to an accept state) through the graph identifies a non-unique set of resources required to fulfil a particular task. Alternate sentences to the same accept state define acceptable compromises. Multiple lots can be specified in a single RDG and a bid over all the lots in an auction, called an entirety bid, can captures Depot synergy. An application may represent a desirable combination of resources by specifying multiple lots in a single RDG. A Depot may bid on individual lots, or may make an entirety bid for all lots included in the RDG. The latter may occur if the RDG expresses a favourable[7] set of resources for the bidding Depot.

[6]It is worth pointing out that even small modifications to the protocol may seriously damage its dominant strategy of truthful bidding. For example, in Miller and Drexler [6, 18] and Spawn [36], the addition of the *escalation* mechanism reintroduces counterspeculation and non-truthful bidding.

[7]For example, a set of resources that minimises the fragmentation of a Depot's resources. This would likely result in a more competitive bid.

The RDG provides a general representation of resources and limits bidding to combinations of resource allocations that deemed to be valid by the creator of the RDG. This restriction means in practice that the Depot is faced with a smaller and better defined problem. Further, NOMAD utilises the combination of RDG and entirety bidding to replace the *NP*-complete CAP with a good approximation enabling multi-component auctions with the standard Vickrey auction protocol.

This approximation does not eliminate all *NP*-complete computations, however, it does reduce the size of the problem set and move the computations from the auctioneer to the Depot (bidder) as intended. The Depots compute their bids asynchronously and the load is widely distributed.

There are two main points of caution that we have identified. Firstly, the size of the RDGs cannot be too large or the load on each Depot could become excessive. Secondly, the edge expressions which compose the RDG must be carefully specified to minimise computation. Malicious exploitation of these limitations could form routes of attack on the negotiation system and must be rigorously policed by both the Marketplace and the Depots themselves.

The reduced *NP*-complete problem at each Depot can be entirely eliminated by further restricting the Depot to:

- making either just a single lot bid or an entirety bid for each auction, and
- waiting on the outcome of that auction before bidding again.

Providing that the size and edge specifications are enforced, the bid computation will be sufficiently tractable making these restrictions unnecessary.

In addition, this approach avoids the difficulty of balancing the books [22] and the computation of the social welfare of the system [16], as the winner simply pays the second price. This further reduces the load on the entire system and particularly on the auctioneer. Privacy is enhanced as Depots need only reveal a single bid value to the Market (auctioneer). In addition, the Market need not parse and understand RDG edge expressions — this evaluation is performed by the Depots interested in bidding[8].

2.3.4 The Marketplace Structure

Accordingly, negotiation in NOMAD is performed in a market, operating on the principles of the Vickrey auction protocol. The Nomad negotiation service is a distributed resource allocation system, which enables applications to negotiate and pay for execution resources. The mechanism is straightforward, allowing computationally bound clients to construct resource descriptions from building blocks, and to negotiate efficiently. Applications requiring a contract construct a description of their requirements using an RDG, as shown in Fig. 5. The Market then constructs a set of catalogues which are distributed generally or only to targeted Depots. Once the auction is over, the Market identifies the winner(s) and generates signed contract certificates that are passed to the application. The application then redeems these contracts at the appropriate Depots and obtains the negotiated resources.

The Marketplace does not seek globally optimal allocations, rather it divides the applications and Depots into geographic and categorical regions. Auctions are advertised between regions via the distribution of catalogues between regional Markets. Bipartite auctions (in which only one Depot is eligible to bid) are simulated within the multipartite auction protocol, in such a way that ensures a target Depot will bid its true value.

[8] A Depot looking to bid will, by inference, have spare resources. Using these resources to evaluate potential clients is sensible.

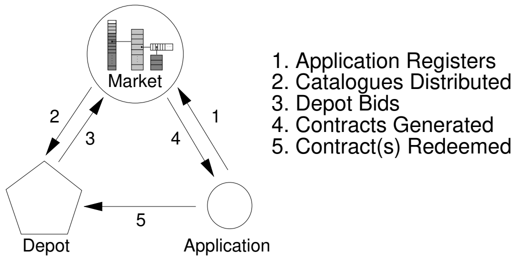

Figure 5: *The Market Protocol.*

2.4 The NOMAD Location Service

Location service research has previously concentrated on architectures with a geographical or organisational basis [19, 34]. Most middleware location services, such as those offered by CORBA and Java RMI, do not address the question of mobility on a global scale, nor solve problems with residual dependencies[9]. Other technologies such as Mobile IP do not address location independence, transparency or residual dependencies.

A location service must track the set of locations of each mobile object throughout that object's lifetime. A seemingly simple task perhaps, but in a large scale distributed system a location service may have to deal with billions of objects, some of which will be moving frequently. Scaling a system to this degree requires wide distribution, load balancing and a minimum of coordinating network traffic. The general capabilities required for the NOMAD location service are: a client should be able to locate and bind to a target application for which it holds only a name, the relocation of an object should be supported in a manner that is transparent to the users of that object, and an application should be able to locate and bind to all of its element objects. In addition, the specific functional requirements of a global location service require that external and higher level bindings should be stable and the algorithms must scale.

2.4.1 Exploiting Application Locality

Existing systems have one thing in common: a reliance on geography — either based on country and organisational divisions as in the DNS, or totally as with the quad tree mapping in Globe. The major requirement of Globe was to provide a geographic organisation is support locality of reference. For highly mobile NOMAD applications this no longer holds true, as application objects are envisioned as using their mobility to move closer to globally distributed clients and resources. Using an organisational basis rather than geographic is problematic for exactly the same reasons.

Instead there is another form of locality which can be exploited — the locality of reference within an application. If some entity interacts with an application object once, chances are that it will interact with it, or a related object from the same application, again.

The NOMAD location service [4] endeavours to take advantage of this and builds the basis of its location service around the concept of the application. That is, each application is wholly responsible for tracking and locating its objects on behalf of itself, the NOMAD system and its clients. In order to do this, the application maintains a set of Location Tables (LTs) that hold the current locations of all of the application's clusters, and consequently their objects. In addition to providing locality of reference, an application's location tables exhibit a high degree of inherent load distribution (they are as distributed as the applications

[9] Residual dependencies negate one of the primary advantages of Mobility: "Dispatch your Agents; shut off your machine." [13]

themselves), and permit the application to tailor its location table to suit its needs and reference characteristics.

2.4.2 Location Service Overview

The NOMAD location service must meet two distinct requirements; the discovery of services and the resolution of out-of-date bindings. Discovery provides an external interface used to establish an initial binding, while rebinding occurs internally and transparently. These two systems act independently and are illustrated in Fig. 6. In addition, this figure highlights the separation between the NOMAD services above the dividing line, and the application components below.

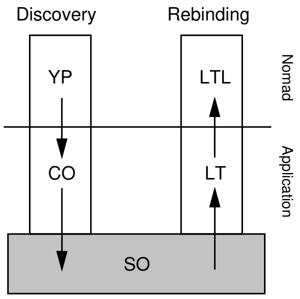

Figure 6: *Discovery and rebinding are complementary parts of the location service.*

Figure 6 shows the various components making up the location service. We have already introduced the contact object (CO) which is a part of the application responsible for negotiating service requirements with a client, and the service object (SO) which is responsible for providing the service. The Yellow Pages (YP) is the NOMAD URN resolver [29], accepting a textual URN and returning a reference to a contact object representing the specified service. The Location Table Object (LT) contains a reference to each of the mobile clusters composing an application and is responsible for tracing their movements. The Location Table exists in its own mobile cluster. The location table locator (LTL) is a well known NOMAD service responsible for locating and tracking the mobile location table's clusters.

Each mobile cluster has a reference to a location table, which is shared by all mobile clusters belonging to the same application. The location table, which is in its own mobile cluster, has in turn a reference to the global location table locator. All objects created in a mobile cluster share the same location table.

As an example, consider a client which holds a textual URN for a service which it needs to use. This requires the discovery facet of the location service. The service's URN, as specified by the client, is resolved, returning a binding to an application specific contact object. The service is then provided to the client via a service object whose reference returned via the contact object.

When an object moves, all existing bindings held on that object become outdated. Resolving these bindings needs the rebinding hierarchy of the location service. Once a binding is outdated, subsequent invocations on that object will fail. The local proxy representing the binding will transparently rebind via a cached reference to the object's cluster's location table. As the location table is also a mobile object, it may also have moved, requiring the rebinding to resort the next level — the location table locator (LTL).

Figure 7 illustrates the division of labour between the components of the naming and location service resulting from this location service architecture. As the load increases towards the bottom of the pyramid, so does the degree of distribution.

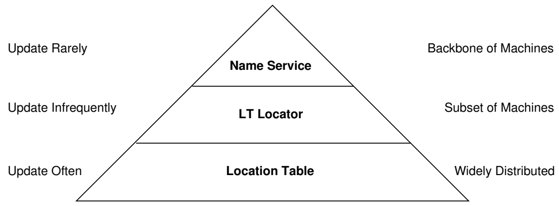

Figure 7: *Division of Labour.*

3 Implementation

A NOMAD prototype has been implemented in Java on top of Flexinet [9]. The current version features implementations of all of the system components described in this paper. The design of the system is not bound to Java or Flexinet, and alternative platforms and language support are being investigated.

3.1 Mobile Application Prototypes

One of the test applications implemented on the NOMAD prototype is a whiteboard service, which serves as a useful illustration of the capabilities of the system. In the interests of experiencing the flexibility of the system with regard to different distributed programming paradigms and the versatility of the contact and service dichotomy within a single application, two version of this application have been written. The first was a standard centralised client-server version of a whiteboard service. In this case the contact and service interfaces where both implemented by a single static agent. In response to registrations from clients, the contact object simply returned a reference to the service object of the agent.

In the distributed version of the whiteboard, negotiation with the contact agent resulted in a new service object being created close to the location of the new client. These service agents act in concert with other client's whiteboard service agents to keep the client views weakly consistent[10]. The contact agent only performed duties involving the initial assignment of the service agents to clients. This prototype could be easily extended to provide additional functionality — in the case of physically mobile clients, each service agent could shadow the movement their client throughout the network.

Liu and Liu [15] describe a prototype distributed meeting scheduler implemented as a mobile agent. The agent travels amongst multiple machines, identified by the owning faculty member, and queries each machine to obtain the owners meeting schedule. NOMAD supports such applications, as the *ownership* of a machine is a constraint that can be specified during negotiation. A similar application was implemented in NOMAD, although in our case it was a test of the technology rather than a useful application in its own right. The application would negotiate for resources on two Depots, and then migrate continously between them, carrying a small payload of data each time.

These prototype applications demonstrate the strength of the NOMAD platform in the creation of distributed software. In particular the ease with which both paradigms were implemented, demonstrating the effectiveness of the supporting NOMAD framework.

4 Related Work

Mobile agent systems comprise a vibrant research field, and a comprehensive survey of these is carried out in Gray et al. [8]. These same authors are responsible for the D'Agents system

[10] Updates are not causally ordered [14].

that shares a market based approach with NOMAD. A number of differences exist between the two systems. NOMAD utilises rebinding proxies for communication, multithreaded agents, application (rather than host) driven negotiation and persistence. Both systems also share sender initiated migration (in our case migration is weak due to limitations of the current vHost implementation) and signed authentication. We do not utilise access control lists, but rather rely on incentive based control within the economic model. Similarities also exist with other systems, such as Aglets [12], NOMADS[31], TACOMA[11] and Globe [34]. However, no other system attempts to create the global infrastructure that is the driving goal of the NOMAD architecture.

5 Summary

This paper has presented the NOMAD architecture. The combination of the consistent local infrastructure provided by the Depot architecture, and the global infrastructure provided by NOMAD services, combine to form a consistent and interlocking environment for highly distributed mobile applications.

The application architecture separates the roles of contact and service objects, and along with the use of application specific location tables, encourages an application to intelligently distribute itself to meet negotiated client QoS requirements. The use of the application to optimise the distribution of the location tables in the location service, limits the impact of the majority of updates to these small and infrequently mobile data structures. The remaining global workload only involves resolving the locations of the location tables themselves, ensuring an implicit ability to scale. Higher level bindings are stable within the discovery hierarchy.

Economic resource management, particularly the market price model combined with financial incentives provides a distributed, tractable computational economy. The design of the Nomad Marketplace addresses each of the limitations of the Vickrey auction protocol, and enables multiple resource specifications within the constraints of tractability. Global coordination between regional Markets and categorical Markets is via the distribution of catalogues which are the fundamental means of resource advertising. This combination of Vickrey auction, resource description graph, Market mechanisms, and catalogue distribution, provides a tractable solution to the problem of global distributed resource allocation.

References

[1] Azer Bestavros. Speculative Data Dissemination and Service. In Stanley Y. W. Su, editor, *Proceedings of the Twelfth International Conference on Data Engineering (ICDE)*, pages 180–187, New Orleans, Louisiana, February 26 - March 1 1996. IEEE Computer Society.

[2] Kris Bubendorfer. *NOMAD: Towards an Architecture for Mobility in Large Scale Distributed Systems.* PhD thesis, Victoria University of Wellington, 2001.

[3] Kris Bubendorfer and John H. Hine. DepotNet: Support for Distributed Applications. In *Proceedings of INET'99, Internet Society's 9th Annual Networking Conference*, June 1999.

[4] Kris Bubendorfer and John H Hine. NOMAD: Application Participation in a Global Location Service. In Ming-Syan Chen, Panos K.Chrysanthis, Morris Sloman, and Arkady Zaslavsky, editors, *Lecture Notes in Computer Science (to appear)*, number 2574 in

LNCS, pages 294–306. January 2003. Proceedings of the 4th International Conference on Mobile Data Management, 21-24 January, 2003, Melbourne, Australia.

[5] Dan Chalmers and Morris Sloman. Survey of Quality of Service in Mobile Computing Environments. Technical Report 98/10, Department of Computing, Imperial College, London, February 1999.

[6] K. Eric Drexler and Mark S. Miller. Incentive Engineering for Computational Resource Management. In Huberman B.A, editor, *The Ecology of Computation*, pages 231–267. Elsevier Science Publishers (North-Holland), 1988.

[7] Pawan Goyal, Xingang Guo, and Harrick M. Vin. A Hierarchical CPU Scheduler for Multimedia Operating Systems. In *Proceedings of the USENIX 2nd Symposium on Operating System Design and Implementation*, Seattle, Washington, October 1996.

[8] Robert S. Gray, George Cybenko, David Kotz, Ronald A. Peterson, and Daniela Rus. D'Agents: Applications and performance of a mobile-agent system. *Software—Practice and Experience*, 32(6):543–573, May 2001.

[9] Richard Hayton and the Advanced Networked Systems Architecture Team. *FlexiNet Architecture*. Citrix Systems (Cambridge) Limited, Poseidon House, Castle Park, Cambridge, CB3 0RD, United Kingdom, February 1999.

[10] David Ingram. *Integrated Quality of Service Management*. PhD thesis, Jesus College, University of Cambridge, January 2000.

[11] Dag Johansen, Robbert van Renesse, and Fred B. Schneider. Operating System Support for Mobile Agents. In *Proceedings of the 5th IEEE Workshop on Hot Topics in Operating Systems*, 1997.

[12] Danny B. Lange and Mitsuru Oshima. *Programming and Deploying Java(TM) Mobile Agents with Aglets(TM)*. Addison-Wesley, 1st edition, 1998.

[13] Danny B. Lange and Mitsuru Oshima. Seven Good Reasons for Mobile Agents. *Communications of the ACM*, 42(3):88–89, 1999.

[14] Henry M. Levy and Ewan D. Tempero. Modules, Objects and Distributed Programming: Issues in RPC and Remote Object Invocation. *Software Practice and Experience*, 21(1):77–90, January 1991.

[15] M.L. Liu and Yajun Liu. A Prototype Mobile-Agent Application. In *the 15th International Conference on Computers and their Applications (CATA)*, New Orleans, Louisiana., March 2000.

[16] Jeffery K. MacKie-Mason and Hal R. Varian. Generalized Vickrey Auctions. Working paper, University of Michigan, 1994.

[17] Paul Milgrom and Robert Weber. A Theory of Auctions and Competitive Bidding. *Econometrica*, 50(5):1089–1122, September 1982.

[18] Mark S. Miller and K. Eric Drexler. Markets and Computation: Agoric Open Systems. In Huberman B.A, editor, *The Ecology of Computation*, pages 133–176. Elsevier Science Publishers (North-Holland), 1988.

[19] P. Mockapetris. Domain Names: Concepts and Facilities. Network Working Group, Request for Comments (RFC) 1034, November 1987.

[20] David C. Parkes. iBundle: An efficient ascending price bundle auction. In *Proceedings of the 1st ACM Conference on Electronic Commerce, EC-99*, pages 148–157, 1999.

[21] David C. Parkes. An Iterative Generalized Vickrey Auction: Strategy-Proofness without Complete Revelation. In *Proceedings of the AAAI Spring Symposium on Game Theoretic and Decision Theoretic Agents*, Stanford University, CA, March 2001.

[22] David C. Parkes, Jayant Kalagnanam, and Marta Eso. Achieving Budget-Balance with Vickrey-Based Payment Schemes in Exchanges. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1161–1168, 2001.

[23] Ragunathan Rajkumar, Chen Lee, John Lehoczky, and Dan Siewiorek. A Resource Allocation Model for QoS Management. In *Proceedings of the 18th IEEE Real-Time Systems Symposium*, San Francisco, California, December 1997.

[24] E. Rasmusen. *Games and Information — An Introduction to Game Theory.* Blackwell Publishers, Oxford, 2nd edition, 1994.

[25] M. H. Rothkopf, A. Pekeč, and R. M. Harstad. Computationally Manageable Combinatorial Auctions. Technical Report 95-09, DIMACS, Center for Discrete Mathematics and Theoretical Computer Science, Rutgers, New Jersey, USA, April 1995.

[26] Tuomas W. Sandholm. Limitations of the Vickrey Auction in Computational Multiagent Systems . In *Proceedings of the Second International Conference on Multiagent Systems (ICMAS-96)*, pages 299–306, Keihanna Plaza, Kyoto, Japan, December 1996.

[27] Tuomas W. Sandholm. *Negotiation Among Self-Interested Computationally Limited Agents.* PhD thesis, Department of Computer Science, University of Massachusetts, September 1996.

[28] Detlef Schoder and Torsten Eymann. Technical opinion: The real challenges of mobile agents. *Communications of the ACM*, 43(6):111–112, 2000.

[29] Karen. R. Sollins. Requirements and a Framework for URN Resolution Systems. Internet Engineering Task Force (IETF) Internet-Draft, March 1997.

[30] Neil Stratford and Richard Mortier. An Economic Approach to Adaptive Resource Management. In *Proceedings of the 7th Workshop on Hot Topics in Operating Systems*, Rio Rico Resort and Country Club, Rio Rico, Arizona, March 1999. IEEE Computer Society and IEEE Technical Committee on Operating Systems (TCOS).

[31] Niranjan Suri, Jeffrey Bradshaw, Maggie Breedy, Paul Groth, Gregory Hill, Renia Jeffers, and Timothy Mitrovich. An Overview of the NOMADS Mobile Agent System. In *Proceedings of ECOOP'2000*, Nice, France, 2000.

[32] L. E. Sutherland. A Futures Market in Computer Time. *Communications of the ACM*, 11(6):449–451, June 1968.

[33] Maarten van Steen, Philip Homburg, and Andrew S. Tanenbaum. The Architectural Design of Globe: A Wide-Area Distributed System. Technical Report IR-422, Department of Computer Science, Vrije Universiteit, 1997.

[34] Maarten van Steen, Philip Homburg, and Andrew S. Tanenbaum. Globe: A Wide-Area Distributed System. *IEEE Concurrency*, 7(1):70–78, January–March 1999.

[35] William Vickrey. Counterspeculation, Auctions, and Competitive Sealed Tenders. *The Journal of Finance*, 16(1):8–37, March 1961.

[36] Carl A. Waldspurger, Tad Hogg, Bernardo A. Huberman, Jeffrey O. Kephart, and W. Scott Stornetta. Spawn: A Distributed Computational Economy. *IEEE Transactions on Software Engineering*, 18(2):103–117, 1992.

[37] Michael P. Wellman. A Market-Oriented Programming Environment and its Application to Distributed Multicommodity Flow Problems. *Journal of Artificial Intelligence Research*, 1(1):1–23, 1993.

[38] Jianxin Yan, Stephen Early, and Ross Anderson. The XenoService – A Distributed Defeat for Distributed Denial of Service. Technical report, Computing Laboratory, Cambridge, UK, 2000.