Up Prev Next

DepotNet: Support for Distributed Applications

Kris BUBENDORFER <kris.bubendorfer@mcs.vuw.ac.nz>
John H. HINE <john.hine@mcs.vuw.ac.nz>
Victoria University of Wellington
New Zealand

Abstract

The Internet is evolving in terms of application and structure. The "network" applications of the 1980s have been superseded by increasingly complex Web-based applications, and the presence of Web proxies and Java applets are early indicators that the Internet is becoming a globally distributed system. This paper presents the design of DepotNet, a support system for distributed applications. The goal of our design is to facilitate the development and deployment of complex distributed applications.

DepotNet provides a consistent worldwide view, supports distributed applications based on lightweight objects, and decreases the time to prototype these applications. DepotNet provides a collection of higher-level services normally associated with distributed systems, such as communication, consistency, naming and location, and security. However, applications still retain control over the location of objects and the resources used. DepotNet recognizes the heterogeneous relationship between applications and infrastructure and addresses the need for an economic model to support payment for the infrastructure's services. Finally, DepotNet provides a plug-and-play interface to facilitate the rapid development and deployment of new services.

Contents

Introduction

The convergence of computing and communication as exemplified by the Internet has been widely touted. In particular, the rapid growth of the World Wide Web has made many realize the potential for global services and applications. What impact has this had on system design?

The Internet differs from the distributed system models studied by researchers for the past 15 years. Key issues are scale and the lack of a single controlling organization. Distributed applications provided by diverse organizations will potentially provide thousands of services reaching millions of customers around the world. While many of the issues involved in building these globally distributed applications are similar to those faced in traditional distributed applications, novel solutions are required for this new context. The current environment requires a unifying architecture capable of building truly global systems.

Applications must retain ultimate control over the question of the resources they consume and the quality of service they require to meet the needs of their clients. Our system provides support by way of services such as object storage, remote execution, transparent multi-cast communication, and security of information. Significantly, it also provides an economic model in which costs may be charged for services, allowing the application to make decisions about which resources to consume and at what rate.

Consider the following example. Many services use the Web for delivery, despite the lack of a clear economic model to support this. We ask, "Who should pay the cost of a proxy cache?" One argument says the client, as cache hits reduce traffic costs and latency. Another argument says the object provider, because the provider wishes the object to be readily available to clients. The answer for most of us would depend on the nature of an object, perhaps being prepared to pay for a cached copy of a current news item but expecting the provider to pay for a cached advertisement.

We believe this is the direction of future development. The Internet will support a many-to-many relationship between application service providers and users (customers). This will happen on a very large scale, though not all applications will operate within the same homogeneous environment.

This paper presents the design of DepotNet. DepotNet provides a high-level, unifying architecture that allows globally distributed applications to be constructed from lightweight objects that interact with the supporting infrastructure to achieve cost-effective delivery. DepotNet will provide services such as multi-cast communication, relocation and replication, consistency, persistent storage, security, and naming services to these distributed applications. DepotNet recognizes the heterogeneous relationship between applications and infrastructure and addresses the need for an economic model to support payment for the infrastructure's services. Finally, DepotNet provides a plug-and-play interface to facilitate the rapid development and deployment of new services.

Our goal is to reproduce the role played and the benefit achieved by the standardization of the Unix API. This standardization meant applications could be developed that would then be available on many vendors' systems. This was successful enough that Unix grew to dominate the server market. We envision a similar service interface at a higher level that recognizes the convergence of communication and computing.

Related work

Our vision of a unifying environment is shared by AT&T's Geoplex [1]. Geoplex is presented as network convergence software that will assist the merging of telephone and Internet networks. While Geoplex aims to simplify the development of services, they continue to focus on the end-to-end communication characteristic of the client-server model rather than the truly distributed application. We expect services will be distributed with different users receiving service from widely distributed replicas of the service provider.

The Globe Project [2] comes from the distributed systems community. Globe shares the view of large, global, distributed applications. A novel feature is its location service, which will be compared with

DepotNet's service below. Globe has a more rigid view of its applications, with each application consisting of a set of "local objects" constructed from control, replication, semantic, and communication sub-objects. Globe does not address the issue of different service providers or payment for resources used.

DepotNet architecture

Figure 1 shows a DepotNet consisting of a logical network of Depots. Collections of Depots are managed by different organizations. Individually, each Depot manages a set of finite resources and provides a locus of coordination and computation for mobile, active objects. These individual Depots are networked to create DepotNet, a federation of cooperating Depots. Ownership or management of a Depot has no explicit architectural recognition in DepotNet.

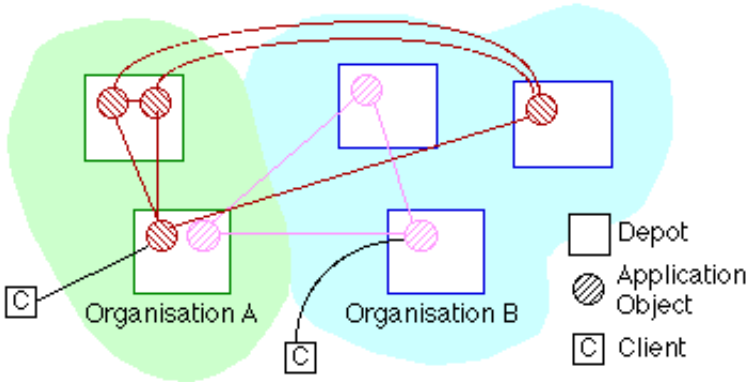

Figure 1: Applications and depots

A typical application is expected to consist of a large number of widely distributed objects. These objects use the services provided by DepotNet, such as communication, consistency, replication, and relocation, as well as those provided by individual Depots such as storage, computation, and location. An application may now consider how it can best use the services of DepotNet to satisfy its clients. For example, the application may negotiate with different Depots for placement of its component objects. Individual objects will be "housed" by a Depot, which will provide it with a range of resources in agreement with a contract. Each application will negotiate with the various Depots for the resources that it requires. In most cases this negotiation will be carried out by the specific object that requires the resource.

Applications offer services to external clients. The clients do not explicitly use the services of the Depots, but use standard network connections to application objects. A client that wishes to obtain a service from an application will negotiate with a *contact object*. The contact object is responsible for negotiating quality of service, payment, etc. Once the quality of service has been agreed on, the client will receive the service from a different object in the application -- a *service object*, which is quite probably replicated or relocated to suit the client. This approach gives the application the greatest flexibility in distributing it's client load.

Responsibilities

A goal of DepotNet is to support lightweight applications to allow rapid development and deployment of new application services. This requires DepotNet to provide most of the algorithms found between the IP network layer and the actual application algorithms. On the other hand, we wish the distributed application to execute as effectively as possible. This requires that the application be sufficiently aware of its environment to carry out various optimizations and the potential for an application to extend the

Depot environment.

The allocation of responsibilities between a Depot and an application has been a key feature of the design process. The current division of duties sees the Depots with the following responsibilities:

- Persistent, reliable storage of objects.
- Provision of a set of communication primitives that support both point-to-point and group communication.
- Provision of a set of consistency algorithms.
- Location of objects.
- Security for both information and object integrity.
- Fault detection and reporting through exceptions.

As well as the function for which the application was written, it also retains responsibility for

- Determination of the relative placement of objects
- Determination of the degree of replication required.
- Selection of communication and consistency algorithms.
- Extension of provided algorithms where required.

These divisions are flexible. As we will see, the design of the Depot Kernel facilitates applications taking on greater responsibility where this is deemed worthwhile.

Transparency

Another key issue in the design of the DepotNet architecture has been transparency. Many authors have argued for "total transparency" for distributed applications. Neither the application nor its objects are aware of the location of any objects.

We do not accept this argument. We believe that experience with Internet-based applications has shown that the designers of Emerald [3] were correct and that benefits ranging from load balancing to reduced communication costs can be derived from the explicit placement of objects. Caches are a straightforward example. We argue that there are places where transparency is a significant benefit and others where it is undesirable. In DepotNet objects may be placed explicitly `At`, `Near`, `Away` (from), or `Between` selected reference points. Once this has been done communication among objects is transparent with respect to the location of any of the objects

We now look at several of the key design areas in more detail to elaborate on the discussion above.

The Depot

Each Depot is a separate entity, but relies on a set of agreed protocols to cooperate with other Depots. The other Depots may or may not be under the same management. The leading theme of our design was to keep the Depot system structure as flexible as possible, allowing a plug-and-play approach. This led us to emphasize libraries to provide services and implement protocols and to support the concept of extensions to allow applications to enhance the services of the Depot. A consequence of this approach was a minimal kernel.

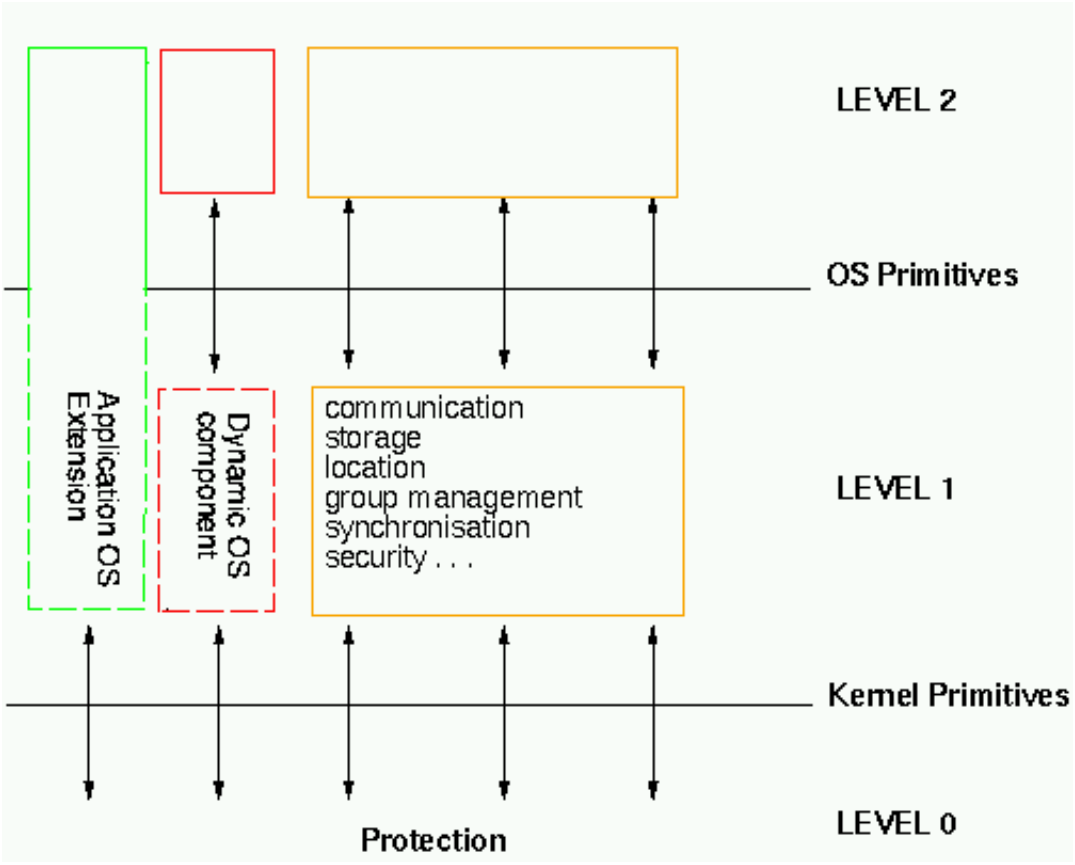

Figure 2: The Depot system structure

Figure 2 shows the resulting structure of the Depot system. The prototype system runs as an application on a Unix system. It is easiest to think of the various components in three levels. At level 0 is a minimalist kernel in the fashion of [4] and [5]. The kernel's responsibilities are limited to resource allocation, basic protection mechanisms, and object storage. All objects permanently reside in the virtual memory of the Depot.

Level 1 is composed of a library of services. Depot core services are always resident and present in all Depots. Individual Depots may offer additional services that can be dynamically loaded on demand. In the prototype, the kernel and each service runs in disjoint address spaces for security. Communication is done using IPC.

At level 2 we find application objects that are currently resident on this Depot. In general, these objects obtain services from level 1 by communicating with the threads that implement level 1 services. Such objects cannot directly invoke the methods of kernel objects. This approach allows our application objects to maintain minimal address spaces, removes the need to maintain copies of library functions in each object, and ensures the relocation of an object is as efficient as possible.

Alternatively, an application may install a system extension into level 1 of the Depot system. This extension can invoke kernel methods and communicate with other level 1 threads. This mechanism supports experimentation with new library services and also allows an application with special requirements to directly access the kernel functions.

The current prototype supports the following services in the core of the Depot system:

Location
: An application is generally composed of objects distributed over a large number of Depots. This service keeps track of the location of an application's objects. It is discussed in more detail in the

following section.

Group management

Groups are a powerful construct by which applications can express control over assorted far-flung objects. They are frequently used in distributed applications as a means of assisting communication and consistency algorithms and simplifying the programming process[5]. This service allows application objects to create, join and leave groups, to locate the members of a group, etc. In conjunction with the location service it allows the location of a group's members to be determined.

Communication

This service provides direct point-to-point and multi-cast communication. Groups are used for the multi-cast communication. Varying semantics are available for the point-to-point communication. The service maintains connection-oriented communication paths between mobile objects, ensuring that communication is transparent to the movement of objects.

Replication, synchronization, and consistency

Many distributed applications depend on replicated objects and/or the maintenance of an agreed level of consistency among copies of data. This service provides for the replication of objects in a way that establishes a consistency requirement among the replicas. The necessary degree of consistency is then maintained throughout the lifetime of the objects.

Security

Responsibility for security is divided between level 0 and level 1. Individual Depots will host large numbers of objects from various applications and interact with other Depots, not all under the same management. This requires a security system capable of ensuring that each object plays the role intended and no other. We believe the OASIS architecture[7] is an appropriate mechanism for developing this and are currently working on the design.

The DepotNet location service

In a large distributed environment, a client wishing to contact an application to obtain a service must first locate that application. In the DepotNet context the application will be active on some set of Depots but will not be known to all. Further, DepotNet applications are expected to configure themselves for the most efficient delivery of services to clients. We anticipate that the configuration of a DepotNet application will be highly dynamic as it relocates or replicates objects in response to client demand. This level of mobility and replication precludes the use of object identifiers incorporating physical addresses as is the situation with uniform resource identifiers (URIs).

Scaling is a substantial problem that must be addressed by any naming system. Globe addresses scaling by keeping a single authoritative record for each object and employing a heuristic to optimally locate that object[8]. Pointer references provide access to the record from anywhere within the Globe system. As the scale of a distributed system grows, so does the number of hosts, clients, applications, and objects. The potential distance between the outlying parts of the system and any of its applications also increases. The combination of scale and high mobility presents a significant problem to the design of a global location service.

Past approaches to this problem have used a hierarchical structure, based on either geography [8] or organization [9]. Our approach is significantly different and motivated by the fact that each application is responsible for the distribution of its own objects. We distribute the work between a name server (White or Yellow Pages), the Depots, and the applications themselves.

A distributed application in DepotNet manages the location of its objects and experiences communication transparency. This requires that both the Depots supporting the application and the

application itself know the location of all objects. It would be unnecessarily redundant for both the Depot and the application to maintain this information, and, further, the Depot cannot trust the application to maintain the integrity of this information.

The solution is to store a per application location table (LT) in the Depots. The Depot is responsible for the integrity of the LT and the application is given read only access to the LT. The location table is maintained by the Depots supporting the application as a side effect of invoking kernel distribution primitives. The only direct control the application has over the LT is to request its replication on another Depot. Depots with copies of the location table are responsible for maintaining the consistency of the information. Each Depot holding a location table for an application must also register that association with the name service.

We differentiate between *contact objects* and *service objects* to address the dynamic configuration of an application. A contact object is the address that a client may use to request service from the application, while a service object subsequently delivers that service. This is analogous to a Web site supported by multiple servers that dynamically assigns request to servers. Applications designate contact objects when the objects are registered in the location table.

The steps taken to associate a client with the service offered by an application is shown in figure 3.

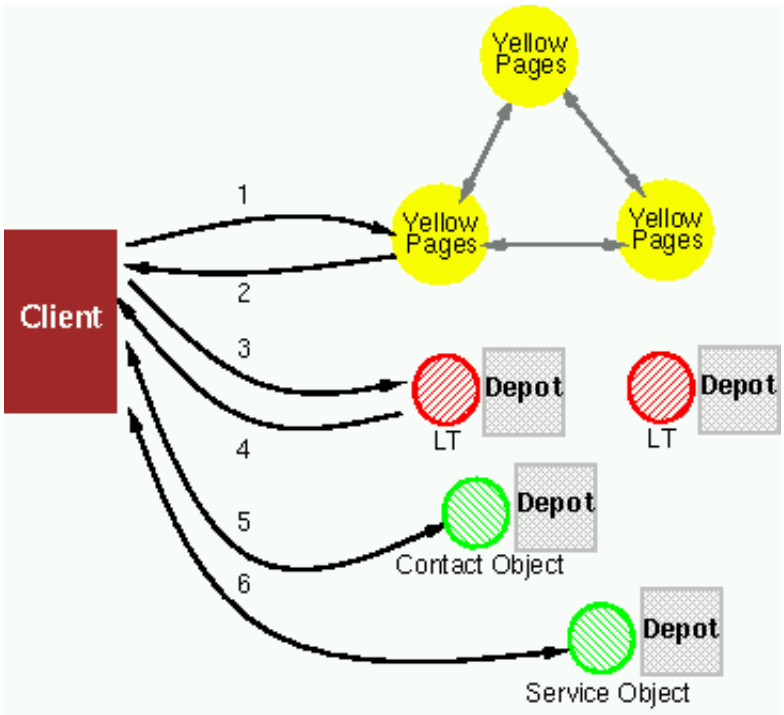

Figure 3: The location sequence for a previously unbound object

1. The client requests the location of the application from the name server.
2. The name server returns the addresses of any Depots holding this application's location tables.
3. The client contacts any of these Depots and requests the address of a contact object for the application.
4. The Depot returns the contact object address.
5. The client uses the contact object to negotiate for service with the application.
6. Following successful negotiation, the client returns the address of a service object for use by the client.

These steps reflect a division of responsibility between the name server, the Depot, and the application. Name server entries bind an application to those Depots that maintain LTs for the application. This

binding is expected to be relatively static compared with the location of either contact or service objects. In turn the LT maintains a binding to the application's contact objects. This binding will change but is located close to the application and is inexpensive to maintain.

Finally, the application itself is responsible for the binding of the client to a service object. This may be as simple as choosing a lightly loaded object able to deliver the service or may be as complex as creating a new object at some location appropriate to serving the client. This is entirely up to the application, though each Depot will provide a library of commonly used algorithms.

Once a client has bindings for a Depot with an LT, all future location requests from the client can be resolved at the Depot level unless the LT is relocated. The only time the name server is involved is when an LT is created, destroyed, or moved. The majority of the effort in maintaining and serving location information about dynamic widely distributed application objects comes from a highly distributed collection of LTs residing on many different Depots. As the Depots are load balanced, the location service will not place an unreasonable burden on any individual members of DepotNet.

Economic model

We believe the long-term viability of globally distributed applications will depend on the ability of an application to pay for the resources it requires. This will require the application to dynamically negotiate suitable contracts with Depots. Currently, services such as AltaVista, Yahoo, and Geocities are paid for by side effect -- that is, the advertising pays. With an economic model, the consumer pays.

The following are major benefits of a DepotNet economy:

- Payment for infrastructure.
- An application can better match its use of resources to its requirements.
- Dynamic pricing of Depot resources is used to reflect the availability, and can, for instance, be used to balance load.
- Limitation of certain forms of denial of service. Such software will by its greedy nature be short lived as its funds are expended.

The following are the basic requirements to support such a model:

- The ability to obtain information to make a judgment on how to best provide a service.
- The ability to negotiate a contract for the provision of services (e.g., storage, multi-cast, etc.).
- A mechanism to pay.

Conclusion

We have presented an architecture that supports lightweight applications to allow rapid development and deployment of new application services on the Internet. The design of DepotNet allows applications substantial control over their use of resources, placement of objects, levels of replication, and consisteny while providing many of the algorithms usually found between the IP network layer and the application layer.

The Depot system is designed to promote flexibility and a plug-and-play approach to adding new services. It has a minimal kernel supporting a library of services. Applications can both use the provided services and install new services through system extensions.

The DepotNet location service solves the problems of scale with a solution that is novel, low cost, and elegant. In contrast to existing location services we base DepotNet's name hierarchy around applications and distribute the work among name servers, Depots, and the applications. This solution provides fast lookups regardless of an object's location.

DepotNet provides a new approach to application deployment on the Internet. High-level services are provided by an enhanced infrastructure and applications are expected to meet the costs of the services they use.

References

1. AT&T, AT&T Labs Internet Platforms The GeoPlex Project, http://www.geoplex.com.
2. van Steen, Maarten, Homburg, Philip, and Tanenbaum, Andrew S., *The Architectural Design of Globe: A Wide-Area Distributed System*, IEEE Concurrency, to appear. Also available as Report IR-422.97, Computer Science, Vrije Universiteit. (See http://www.cs.vu.nl/~steen/globe/ .)
3. Jul, Eric, Levy, Henry, Hutchinson, Norman, and Black, Andrew, *Fine-Grained Mobility in the Emerald System*, ACM Trans on Computer Systems, Vol. 6, No. 1 (Feb. 1988), pp. 109-133.
4. van Doorn, Leendert, Homburg, Philip, and Tanenbaum, Andrew S., *Paramecium: An Extensible Object-Based Kernel*, in Proc. of 5th Hot Topics in Operating Systems (HotOS) Workshop, Orcas Island, May 1995, pp. 86-89.
5. Engler, Dawson R., *The Exokernel Operating System Architecture*. Ph.D. dissertation, Massachusetts Institute of Technology, 1998.
6. Powell, David, ed., *Group Communication*, Special Section, Communications of the ACM, Vol. 39, No. 4 (April 1996), pp. 50-97.
7. Hayton, Richard, Bacon, Jean, and Moody, Ken, *OASIS: Access Control in an Open, Distributed Environment*, Proc. IEEE Symposium on Security and Privacy, Oakland, CA, May 1998, pp. 3-14.
8. van Steen, Maarten, Hauck, Franz J., Homburg, Philip, and Tanenbaum, Andrew S., *Locating Objects in Wide-Area Systems*, IEEE Communications Magazine, Vol. 36, No. 1 (Jan. 1998), pp. 104-109.
9. Needham, Roger M., *Names*, in Mullender, Sape, ed., Distributed Systems 2nd ed., Addison-Wesley, 1993. pp. 315-327.