Home Migration Using a Structured Peer-To-Peer Overlay Network*

Erik Klintskog[1], Per Brand[1], and Seif Haridi[2]

[1] Distributed Systems Lab., Swedish Institute of Computer Science, Kista, Sweden,
{erik, perbrand}@sics.se
[2] IMIT - Royal Institute of Technology, Kista, Sweden
seif@it.kth.se

Abstract. This paper presents the design and implementation of a decentralized home-migration protocol for the Distribution SubSystem(DSS) middleware. The DSS provides generic distribution support for shared data structures in open distributed systems. Previous approaches for migrating homes, such as forward pointers, broadcasts, and centralized directory services are known to have disadvantages. We propose using a structured P2P system to store the location of migrated homes. This enables seamless migration of homes, without the need for lengthy forward pointer chains that degrade performance and robustness. Nor is a dedicated data-base that requires administrative effort needed. The presented design depicts how the self-organizing aspects of peer-to-peer computing can be used to construct a fault-tolerant, scalable, efficient, distributed home-location service to enhance middleware functionality.

1 Introduction

This paper describes a decentralized home-migration protocol that uses a structured peer-to-peer (P2P)[1] overlay network to locate migrated homes. The structured P2P network is constructed by the nodes participating in a distributed application. The resulting service inherits the prominent properties of structured P2P systems; self-organization, low administrative load, high guarantees, good scalability, and resilience to failures. The home-migration protocol is made a component of the generic component-based coordination framework of the Distribution SubSystem(DSS) [2, 3] middleware.

Home-based protocols are simple to implement and often used to grant seamless access to migrating entities[4, 5]. A mobile entity records its current location at a home, usually located at the process where the entity was created. A proxy, that provides seamless communication with the mobile entity, sends messages to the home. In turn, the home forwards the messages to the current location of the entity. The coordination framework of the DSS makes use of the Home/Proxy approach. Each distributed entity is controlled by a consistency protocol, executed over a coordination network of proxies, a home[2, 6], and a current location. In many home based protocols the home and the current location are co-located, but it need not be that way.

* This work was partially supported by the Information Society Technologies program of the European Commission, Future and Emerging Technologies under IST-2001-33234 PEPITO, and by grants from the Swedish Agency for Innovation Systems (Vinnova)

The home-based approach with a fixed home is a simple solution to keep track of mobile data. It requires no administrative effort in the form of dedicated machines or a distributed directory services, as in [7]. A process can, without additional coordination with other processes, create a new home-based protocol instance and make it available for other processes to use. However, the fixed location of the home is a major drawback. Since all communication between a proxy and the current location is directed over the home, a non-optimally placed home causes unnecessary latencies. This is known as the trombone effect [8]. Furthermore, when the node a home is located at halts, the home is lost, preventing further current location access. A natural extension is to be able to relocate the home. A home migrating protocol for open distributed systems should fulfill the following properties:

1. Every proxy should be able to communicate with the home, modulo network failures, even after migration.
2. It must be possible to relocate the home reasonably often.
3. There should not be any dependencies on third party nodes, in particular not on old home locations.
4. Adding home-migration should require minimal additional administration.
5. It should scale in respect to number of nodes, number of entities, and number of proxies.

The fixed-location-of-the-home problem has only been addressed in Distributed Shared Memory (DSM) systems for clusters of workstations. The proposed solutions rely on properties associated with clusters, e.g. efficient broadcast and stable nodes. Chung et al[9] defines three approaches to consistent home-migration in a DSM system. First, *broadcasting*, when a proxy detects it has an outdated home location it locates the home by sending a broadcast to all members of the distributed system. Second, *forward pointers*, when a home migrates, a pointer to the new node is left at the old node. Third, *directory*, a dedicated directory keeps track of the current location of the home.

The fact that *broadcasting* is known not to scale [10] violates requirement #5. The *forward pointers*[11] approach violates requirement #3, and to a certain extent #5(when the chains become long). Directory services based on distributed data-bases violates #4 and to some extent number #5. DNS-like approaches[7] violates #4 and deals poorly with frequent location changes(#2).

The structured peer-to-peer (P2P) systems organize nodes in overlay networks that provide highly available distributed hash tables (DHT)[12]. The self organizing and scaling properties of these P2P systems make it possible to realize the *directory* approach without violating requirements #4 and #5.

1.1 Contribution

This paper contributes in three areas of middleware design. First, the integration of a structured P2P system in the DSS middleware. The P2P system is used to create a distributed, self-organizing, highly available, storage of generic data. Nodes of a distributed application form an overlay network that maintains the storage in the face of concurrent joins, leaves, and failures.

Second, the design and implementation of a decentralized home-migratory component for the home-based protocol framework of the DSS. The component makes use of a structured P2P system for locating the home and has no dependencies to old home locations, i.e. can be used to transfer shared data structures from processes that are to be taken down.

Finally, we believe the design to be general enough to be applied to any home-based protocol. This includes not only the mobile-ip[4] and the home-based release-consistency protocol[8], but also protocols commonly not classified as home based, e.g. RMI[13].

2 Extending the DSS Middleware

DSS[3] provides generic distribution support for shared data structures in open distributed systems. It is realized by a layered architecture, designed to provide a shared data abstraction to programming systems. This section describes how the DSS is augmented with a highly-available name-based repository service. The repository, called the *DSS backbone*, is implemented using the structured P2P system DKS[12]. The DKS system is used to automatically organize DSS nodes in an overlay network that provide an efficient, fault tolerant, distributed hash table (DHT).

The DSS backbone provides a background service for a DSS node; its resource usage should have minimal effect on the functionality of the DSS. The structured P2P system we choose, DKS, has two advantages. It makes use of the correction-on-use technique[12] for routing table updates that reduces bandwidth consumption and provides stronger lookup guarantees because of atomic joins and leaves[12].

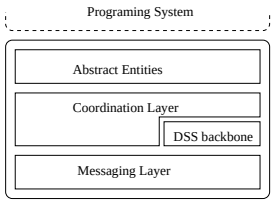

Fig. 1. The internal layout of the DSS middleware.

The layout of the DSS, with the DSS backbone, is depicted in Figure 1. The topmost layer, the *abstract entity layer* provides a generic data structure interface and efficiently hides the internals of the middleware. The second layer, the *coordination layer*, is responsible for coordinating movement and replication of shared data structures according to a given consistency model (different kinds of home-based-protocols are implemented by this layer). The bottom layer, the *messaging layer* [14], provides communi-

[3] The DSS is implemented in C++ and available for download from `http://dss.sics.se`

cation support for the middleware. The DSS backbone service is located conceptually above the messaging layer and makes use of the messaging layer for its communication.

2.1 The DSS Backbone

Each DSS node (i.e. process) of a distributed application participates in a DKS network and together they construct the *DSS backbone* service (see Figure 2). The backbone is available as a basic service component for the protocols of the coordination layer. The service is realized as a variant of a DHT. Items in the backbone are active objects and not just passive immutable objects as typical in a DHT. The backbone supports insertion and removal of objects. In addition messages can be sent to the active objects, but there is no support for retrieval. Note that all the services are asynchronous and location-transparent. Thus it is possible to communicate, by name, with all objects that have been inserted in the backbone from all nodes that are members of the same DSS backbone.

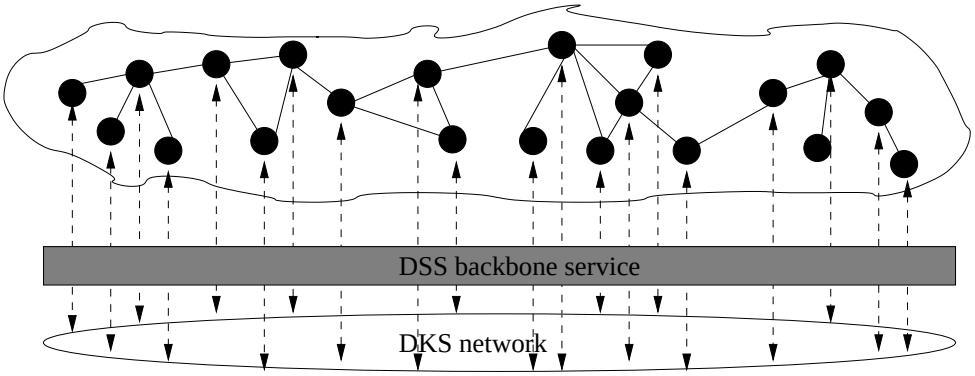

Fig. 2. Organization of the DSS backbone. One DKS network spans all nodes (black circles) of a distributed application. Inter-application communication is mediated by the shared data structures depicted by the solid lines between the nodes. In addition, the processes are all members of the DKS network, not in order to perform point-to-point communication, but in order to get access to the backbone service.

Items in the Backbone The DSS provides a globally unique naming schema based on the unique identities of DSS nodes. A name is represented by a *NetIdentity* data structure that can be sent between nodes. Each active object, called a *backbone item*, stored in the *DSS backbone* is accessed using its associated *NetIdentity*. At creation, the *backbone item* is inserted into the *DSS backbone* that then automatically transports the item to the process that is to host the item. From this point in time, the item is accessible from any node of the *DSS backbone* given the *NetIdentity* of the item.

Joining the Backbone In order to gain access to the DSS backbone a node must first join the associated DKS network. The DSS exposes interfaces for creating a backbone, creating backbone textual tickets and to join the backbone, using such a textual ticket. A textual ticket contains enough information for finding the node that created the ticket, and to join the associated DKS network.

Being Member of a Backbone A member of the DSS backbone is responsible for a set of *backbone items*. The logic of the DHT specifies, at each point in time, which nodes are responsible for which items. As nodes join and leave the overlay network, distribution of the items over the nodes of the DHT continuously changes. Obviously being a member of a DSS backbone and the DKS network comes with a cost in communication and particular in open connections. In the DKS system communication overhead is lower than in most other P2P systems since periodic-stabilization is not used[12]. The DSS is designed to efficiently handle a large number of open connections simultaneously[14], thus the extra connections needed by the DKS system pose litle problem.

The Backbone Interface A DSS backbone instance is represented in the DSS by the `DssBackbone` class. The class has three methods, one that inserts- , one that sends a message to- and one that removes a *backbone item*. A *backbone item* is represented by an instance of the `BackboneItem` class. An instance of the class must implement two interfaces. The `m_messageReceived` method, over which messages to the item are received and the `m_transferService` that returns a marshaled representation of the item. The later method is called when an item is moved between nodes of the DSS backbone. A pseudo description of the classes is shown here:

```
class DssBackbone{
    void m_sendToService(NetIdentity, Message*);
    void m_insertService(NetIdentity, BackboneService*);
    void m_removeService(NetIdentity);
};

class BackboneItem{
    virtual void m_messageReceived(Message*,
                                   DSS_Environment* env)=0;
    virtual LargeMessage* m_transferService() = 0;
};
```

3 Home Based Coordination Framework

The home-based protocols implemented by the DSS are characterized by a home, called the *coordinator*, a set of reference holders, called the *proxies*, and a current location[4].

[4] In some protocols there may be many current locations in the sense of process that hold a consistent copy of the data structure.

The proxies acts as interfaces to the protocol, and exposes a set of operations that can be performed on a shared data structure. For any given home-based-protocol instance, the proxies and the coordinator form a virtual network, called the *coordination network*. Each coordination network is uniquely identified by a *NetIdentity*, assigned to the coordinator at creation and passed to each proxy. A set of proxies and one coordinator is depicted in Figure 3.

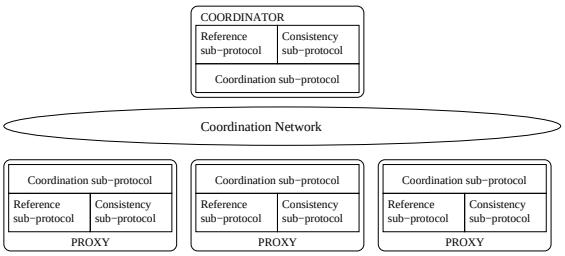

Fig. 3. An example of a coordination network of three proxies and one coordinator.

In order to enable simple protocol development and protocol customization, a home-based protocol is realized by three sub-protocols[5] running in parallel[2], depicted in Figure 3. The *coordination sub-protocol* define the communication infrastructure, i.e. how proxies and the coordinator communicate. This includes locating the coordinator and potentially moving the coordinator. The *consistency sub-protocol* controls the current location and defines how the shared state is accessed, i.e. locally or remotely. The *reference sub-protocol*[6] defines the logic that will eventually allow the coordination network to be completely dismantled, e.g. garbage collection. The later two sub-protocols makes use of the first sub-protocol for inter coordination network communication. The *coordination sub-protocol* and *reference sub-protocol* are independent of entity type, while the *consistency sub-protocol* is dependent[2].

Every sub-protocol instance is implemented as two components, a home and remote component. An instance of the remote component is located at each proxy of a coordination network and communicates with an instance of the home component located at the coordinator. The *coordination sub-protocol* provides the other sub-protocols with a proxy-to-coordinator interface; the location of the coordinator (or the home) is effectively abstracted away. This is depicted as the boxes inside the proxies and the coordinator in Figure 3. The home-based framework of the DSS is not primary focused on migrating entities; it provides seamless access to the home, in order to simplify *consistency sub-protocol* development.

[5] The sub-protocols were called strategies in [3, 2, 14]

[6] The *reference sub-protocol* was called the *memory management strategy* in [3, 2, 14].

4 A Migrating Coordination Sub-protocol

This section describes a new *coordination sub-protocol* that allows migration of the coordinator (remember that the coordinator is similar to the home in a home-based protocol). As long as the *coordination sub-protocol* fulfills the interface and the functionality required, every existing *consistency* and *reference sub-protocol* can be used in conjunction with the new sub-protocol. Thus, constructing one migrating-home protocol, has the effect of enabling a multitude of different shared-data protocols with non-fixed homes.

The assumption in home-based protocols is that the coordinator is migrated seldom in respect of the traffic over its coordinator network. The primary requirement on a *coordination sub-protocol* is that every proxy is eventually able to contact the coordinator, modulo failure. Each proxy caches the last known location of the coordinator that is used when passing messages to the coordinator. If the coordinator has moved from the cached location, this is reported to the proxy by the DSS; the proxy is said to have *discovered* that the coordinator has moved. Upon discovering a move, the proxy invalidates its cache and contacts the DSS backbone in order to get the current location of the coordinator. Newly created proxies, created by receiving a proxy description from a remote node, and is initialized with the cached location from the sending node.

For the remainder of this section we will only consider the migrating-coordinator sub-protocol, proxy and coordinator refer to the remote and home instances of the *coordination sub-protocol*. It is important to keep in mind that the migrating coordination sub-protocol can be combined with a wide variety of reference and consistency sub-protocols to produce a large set of complete entity protocols – all with migrating homes.

4.1 The Remote-Instance

Messages within the coordination network (proxy-to-proxy, proxy-to-coordinator) are sent together with the location of the cached coordinator. References to proxies passed between nodes are also attached the latest known coordinator location. When a remote-instance receives such a location for a coordination network it already knows about, it will check which is the most recent and adjusting the cache if necessary. The location of a coordinator is described as a pair of a process identity and a sequence number, referred to as the *coordloc*. The sequence number describes how many times the coordinator has migrated and is used to order different *coordloc* structures.

All of this can be seen as an optimization. It is not strictly necessary to send *coordloc* structures over the network, newly constructed proxies could query the *DSS backbone* instead.

4.2 Implementing the Directory

The current location of the coordinator is stored in the distributed directory provided by the DSS backbone. Each migrating-coordinator instance maintains a specific backbone-item that holds the *coordloc* that describes the current location of the coordinator. This backbone-item is called the Coordinator Location Repository (CLR).

The CLR is created and inserted in the DSS backbone when the coordinator is first created. The *NetIdentity* of the coordination-network is used to identify the CLR instance. Since the coordinator and the proxies of the coordination network locally store the *NetIdentity* of their coordination network, all can reach the CLR over the DSS backbone interface. The coordinator updates the CLR when it migrates. When a remote-instance discovers that it has an outdated *coordloc* it asks the CLR for an update. Both the coordinator and the remote-instance communicate with the CLR over the messaging interface provided by the `DssBackbone` class.

4.3 Migrating the Coordinator

The coordinator can only migrate to a process that holds a proxy of the same coordination network. Every proxy provides an *attract* interface that migrates the coordinator to the process of the proxy. The interaction between the proxy that requests migration, the coordinator and the backbone unit that holds the current coordinator location is shown in Figure 4. The initiating proxy sends a migration request messages to the coordinator (1). The state of the coordinator is encoded in the migrate message sent to the requesting proxy (2). Upon reception of the migration message, a new coordinator is created, with a new *coordloc* (3). The initiating proxy is the only proxy that knows the new location of the coordinator. In order for the location discovery mechanism to work properly, a message with the new *coordloc* is sent (4) over the backbone to the CLR of the coordination network (5).

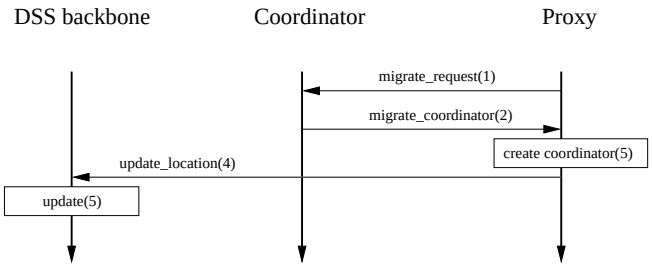

Fig. 4. A sequence diagram depicting the interaction between a proxy, the coordinator and the CLR when moving the coordinator to the process of the proxy. Note that the box labeled “create coordinator” indicates that the Coordinator has moved to the process of the proxy.

The coordination network has no coordinator present while the migrating message travels over the network (2-3). Furthermore, the repository holds outdated information regarding the coordinator location from (2) to (5). The coordination network will in effect be suspended until (5). However, this suspension only affects the threads that accesses the data structure associated with that coordination network. Other data structures (distributed or non-distributed) are not affected.

4.4 Discovering the Coordinator Location

The messaging layer can fail to deliver a message from a proxy to its coordinator for two reasons. Either because no coordinator for the coordination network is present at the destination process or the destination process does not exist. Both cases are reported to the proxy and treated as if the proxy holds an outdated *coordloc*. Detection of an outdated *coordloc* is done lazily; only when a proxy communicates with its coordinator will it verify whether its *coordloc* is correct. The proxy invalidates its cached *coordloc* when it is found to be outdated. This prevents further communication with the coordinator until the cache has been validated again. Every message sent to the coordinator is queued. The cache is validated when a more recent *coordloc* than the currently held is received. The *DSS backbone* is used to find the current location of the coordinator. When a new *coordloc* is finally received, the queued messages are sent to the new coordinator location.

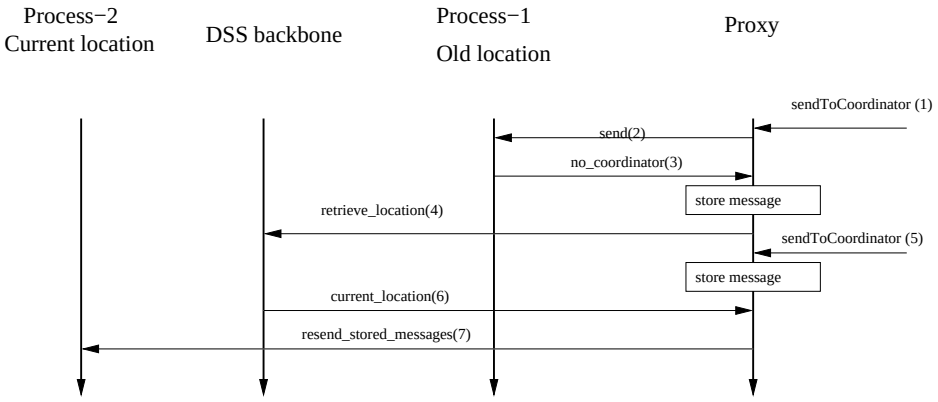

Fig. 5. The sequence diagram depicts how a Proxy detects that it points to an old coordinator location. The Proxy communicates with the CLR to get the current location of the coordinator.

Figure 5 depicts a proxy that discovers that its coordinator has moved, and retrieves a new location. The *coordination sub-protocol* receives a message addressed to the coordinator from the *consistency sub-protocol* (1). The message is sent to the process the current location points to (2), and is returned, indicating the location to be outdated (3). The returned message is stored, and a `retrieve_location` message is sent over the backbone network to the CLR (4). The *consistency sub-protocol* passes a new message to the *coordination-sub-protocol* for delivery to the coordinator (5). Since the proxy holds an invalid coordinator location, the message is stored. The CLR sends a message back directly to the proxy containing its current *coordloc*(6). The proxy will now send all queued coordinator messages (7) to the new process. It is possible, due to race conditions, that the CLR is not yet updated, in which case the proxy can only wait and send a new request after an appropriate time-out strategy.

5 Discussion

The migrating coordinator sub-protocol described in this paper is not a solution to random process crashes that can affect the coordinator process, but is a solution for graceful shutdown of a process. If the shutdown of a process is known at forehand, all coordinators located at the process can be migrated to other processes. Since the discovery of the new location of the coordinator is not dependent on the process that is taken down, the coordinator is still accessible.

The described implementation supports relocation of the coordinator by attraction to the process where a proxy exists. Attraction can be used to program the oposit of attraction, *repulsion*. Repulsion of a coordinator can be realized by sending a proxy reference to a remote node and at the remote note using attract on the coordinator. This is easily realized at the programming system level.

The migrating coordinator sub-protocol caters for many interesting home-based protocol instances. For example, combining the mobile-state *consistency sub-protocol* with migratory coordinator sub-protocol, the resulting protocol allows for local access of the state and can simultaneously adapt to changes in usage pattern. Moving the state between two proxies requires communication with the coordinator. Thus a misplaced coordinator will dramatically affect the latency for each state request. However, the migrating coordinator sub-protocol allows for relocating the coordinator if it is suboptimal placed. Combining remote-state *consistency sub-protocol* (implements a variant of RPC/RMI) with the migrating coordinator sub-protocol is another interesting example. The remote-state protocol sends requests to the coordinator that executes them locally. When moving the coordinator, the state is moved as well. Interesting to note is that the choice of *coordination sub-protocol* does not affect the functional properties of a *consistency sub-protocol*.

We ruled out broadcasting in the beginning of the paper. However, this is somewhat simplistic, if location changes are extremely rare broadcasting might be advantageous. The break-even point will of course depend on how broadcasting is implemented. In particular, the DKS provides a broadcast service which is reasonably efficient and scalable[15]. It would be interesting to investigate an adaptive scheme that changes between the directory and broadcasting approach depending on the frequency of coordinator relocations.

The migrating coordinator sub-protocol is implemented in the DSS, but the approach is applicable to any home-based protocol. It is built on a three assumptions, existing in the DSS. First, the proxies and the home share a common unique name. Second, it is possible to detect if a home has migrated. Third, messaging is reliable, modulo crash failure.

6 Related Work

Home migration for Distributed Shared Memory (DSM) systems has attracted a lot of research [9, 8, 16] with the focus of improving performance and scalability. The proposed solutions make heavy use of techniques that are possible to realize efficiently only on clusters of workstations, but not applicable to open distributed systems. The focus is on when to move the home, not how to find the home after migration.

The Globe/Core system [17] is a home-based remote object system that allows for home migration. A look-up for a proxy description is performed both when receiving a reference to a shared object and when discovering that the home has moved. Similar to the migrating coordinator sub-protocol, a look-up facility called the Globe Distribution Network (GDN)[7] is used. The GDN is hierarchically organized, similarly to how DNS is organized. Thus it requires administration and does not show the same strong self organizing properties as the DKS. Furthermore, it is not incorporated in the Globe/Core, but is realized as an external system.

Zhao et al. suggests using the Tapestry structured P2P system for mobile-home location[18]. Their solution differs in that it uses the overlay network for all communication and not, as in our solution, only when the location has changed since the last use. Consequently, a lot of effort is investigated in organizing the overlay for providing efficient paths to the repositories for a given object.

7 Conclusion

This paper describes a solution to the fixed-home problem associated with home-based protocols. Homes are allowed to migrate, while still being accessible from the set of proxies referring the home. By making use of a structured peer-to-peer system, a distributed, decentralized directory service is realized. The directory service stores the current location of the home, and is used by the proxies to update their cached home location when the home has migrated. The protocol enables for more stable distributed applications since shared data structures are potentially not dependent on the process the home is located at. The implementation of the protocol is done in the home-based framework of the DSS, but the solution is general enough to be used to enhance any home-based protocol. The migrating coordinator *sub-protocol*, the implementation of the protocol in the DSS, can be transparently used by the abstract entity layer and provide support for more stable shared data protocol.

Making use of a structured P2P algorithm as a lookup facility for current home locations is a novel approach. The very attractive properties of structured P2P, like self organization, low administrative load, and low bandwidth consumptions, makes structured P2P a good alternative for extending the range and power of shared-state distributed systems. Instead of investing in expensive persistent distributed data-base, nodes can by self-organization provide the same service, without the heavy administrative burden associated with traditional solutions.

Acknowledgments

We would like to thank Zacharias El Banna, Valentin Messaros and Ali Godshi who commented on earlier drafts of this paper. Furthermore, we would like to show our gratitude toward Luc Onana, Ali Godshi and Sameh El Ansary for providing us with the design of, expertise in, and constructive discussions about the DKS system.

References

[1] Onana, L., Ghodsi, A., Haridi, S.: A framework for structured peer-to-peer overlay networks. Technical Report T 2004:09, Swedish Institute of Computer Science (2004)

[2] Klintskog, E., Banna, Z.E., Brand, P., Haridi, S.: The dss, a middleware library for efficient and transparent distribution of language entities. In: Proceedings of HICSS'37. (2004)

[3] Klintskog, E., Banna, Z.E., Brand, P.: A generic middleware for intra-language transparent distribution. Technical Report T2003:01, Swedish Institute of Computer Science (2003)

[4] Perkins, C.: Mobile IP: Design Principles and Practices. Addison-Wesley (1997)

[5] Van Roy, P., Brand, P., Haridi, S., Collet, R.: A lightweight reliable object migration protocol. In: Internet Programming Languages, ICCL'98 Workshop, Springer Verlag (1998)

[6] Klintskog, E., Banna, Z.E., Brand, P., Haridi, S.: The design and evaluation of a middleware library for distribution of language entities. In: 8^{th} Asian Computing Conference. (2003)

[7] Bakker, A., Amade, E., Ballintijn, G., Kuz, I., Verkaik, P., van der Wijk, I., van Steen, M., Tanenbaum, A.: The globe distribution network. In: USENIX Annual Conference,Freenix Session. (2000)

[8] Ng, M.C., Wong, W.F.: Orion: An adaptive home-based software distributed shared memory system. In: Seventh International Conference on Parallel and Distributed Systems (ICPADS'00). (2000) 187–194

[9] Chung, J.W., Park, K.H., Park, D.: Moving home-based lazy release consistency for shared virtual memory systems. In: 1999 International Conference on Parallel Processing. (1999)

[10] Neuman, B.C.: Scale in distributed systems. Readings in Distributed Computing Systems (1994)

[11] Fowler, R.: The complexity of using forwarding addresses for decentralized object finding. In: Proceedings of the fifth annual ACM symposium on Principles of distributed computing. (1986) 108 – 120

[12] L. Onana, S. El-Ansary, P.B., Haridi, S.: Dks (n, k, f): A family of low communication, scalable and fault-tolerant infrastructures for p2p applications. In: The 3rd International workshop CCGRID2003. (2003) 344–350

[13] Microsystems., S.: The remote method invocation specification (2002) Available from http://java.sun.com.

[14] Klintskog, E., Mesaros, V., Banna, Z.E., Brand, P., Haridi, S.: A peer-to-peer approach to enhance middleware connectivity. In: OPODIS 2003: 7^{th} International Conference on Principles of Distributed Systems. (2004)

[15] Onana, L., Ghodsi, A., Brand, P., Haridi, S.: Self-correcting broadcast in distributed hash tables. In: OPODIS 2003: 7^{th} International Conference on Principles of Distributed Systems. (2004)

[16] Hu, W., Shi, W., Tang, Z.: Home migration in home-based software DSMs. In: Proc. of the 1st Workshop on Software Distributed Shared Memory (WSDSM'99). (1999)

[17] Jansen, M., Klaver, E., Verkaik, P., van Steen, M., Tanenbaum, A.S.: Encapsulating distribution by remote objects. Information and Software Technology **43** (2001) 353–363

[18] Zhao, B., Huang, L., Joseph, A., Kubiatowicz, J.: Rapid mobility via type indirection. In: iptps04. (2004)