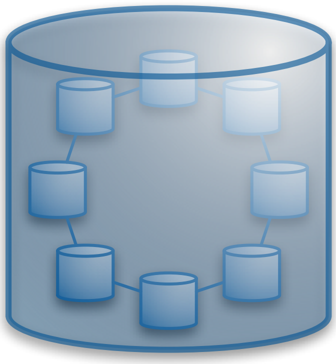
A Distributed Low-Level Data Structure for Composing Scalable Concurrent Web Services



Master Thesis in Computer Science

Submitted by
Michael Kussmaul
kussmaul@nix.ch
Zurich, Switzerland
Student ID: 96-716-055

Department of Information Technology
University of Zurich, Switzerland
Prof. Dr. Klaus R. Dittrich

Supervisor
Dr. Suo Cong

November 17, 2004

Abstract

English - Distributed storage/retrieval of information is becoming increasingly crucial and is used extensively in various kinds of applications, such as web search engines, real-time systems, high performance computing, grid computing, and distributed file systems, to achieve higher throughput (enabling parallel access to the data), and more robust performance (redundancy in case of failures). While various alternatives have been proposed, this work will primarily focus on low-level mechanisms, and will introduce a new approach for a distributed key-value storage based on a b-tree implementation.

Deutsch - Verteiltes Speichern/Abfragen von Information wird je länger je mehr wichtiger und wird in verschiedensten Anwendungen benutzt. Beispiele sind Suchmaschinen, Echtzeitsysteme, Hochleistungsrechner, Grid-Computing und verteilte Dateisysteme, um höheren Durchsatz (ermöglicht parallelen Zugriff auf Daten) und Robustheit (Redundanz bei Ausfällen) zu ermöglichen. Während es verschiedene Möglichkeiten gibt, wird sich diese Arbeit auf grundlegende Mechanismen konzentrieren und einen neuen Ansatz einführen, der auf einem verteilten Assoziativspeicher (key-value) Mechanismus einer B-Baum Implementation basiert.

Document Revision:	183
Document Date:	2004-11-16

Proofread by http://www.proofreading.de

Contents

1 Introduction

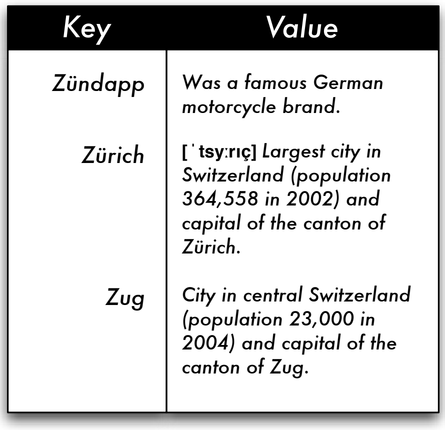
1.1 Situation

The increasing amount of data [18, 70] and the increasing level of access to those data has changed the way in which data is stored.

At the time when most basic systems (such as file-systems, databases and email systems) were designed, the amount of data they needed to store was not that great and everything could be handled by one single computer (e.g. a mainframe [81]). Over time, the amount of data has grown exponentially, and today's systems need to handle large amounts of data. Additionally, the data is stored in multiple places, and systems access this data from different sources. The number of users of such systems has also grown exponentially.

In today's world, a monolithic system, which only runs on a single machine, will not be able to cope with the amount of data and users accessing the system. Over time, most systems have moved from a centralized approach to a distributed one, which should allow for load-sharing the data and accessing requests from a broader range of machines.

It is interesting to note the similarities between those systems: The underlying storage mechanism is based on a key-value architecture for most of those systems. This means the system stores the data in chunks and stores each of them under a certain key. This key allows the user to retrieve the data again afterwards. The approach is similar to a conventional encyclopedia, where the keys are the items and the values are the definitions, as shown in Figure 1.



Key	Value
Zündapp	*Was a famous German motorcycle brand.*
Zürich	[ˈtsyːrɪç] *Largest city in Switzerland (population 364,558 in 2002) and capital of the canton of Zürich.*
Zug	*City in central Switzerland (population 23,000 in 2004) and capital of the canton of Zug.*

Figure 1: Key-Value Pairs in an Encyclopedia

1.2 Problems

There are several ways to actually distribute a system and most of the methods are highly dependent on the actual environment and usage case, e.g.:

- **File Systems** - Transformed from local file systems to network file systems (e.g. NFS [83]) and towards storage area networks, where the actual storage can be distributed to several machines.
- **Databases -** Transformed to a more distributed approach, e.g. by splitting the database tables into several chunks (partitioning). Depending on the architecture, they allow distribution to a few or several hundred machines.

While most of those systems already try to solve the problem of distributed storage, they have done the distribution on a high-level; the fundamental underlying structure has not been dramatically changed. This could soon be a limit, as the data continues to grow exponentially and a more radical and general approach for distributing the data could be necessary.

So much more radical an approach would be a low-level distribution already at the key-value layer. The system on top of it would then already be abstracted from the complexity of a distributed system, as the access to its data would still be the same key-value approach as before, with the difference that those key-values are now already distributed among multiple nodes. Also, more general frameworks could be implemented this way, by offloading all storage needs to a worldwide distributed storage network, so all large scale systems could access data from it, just by using a well-specified interface around a key-value principle.

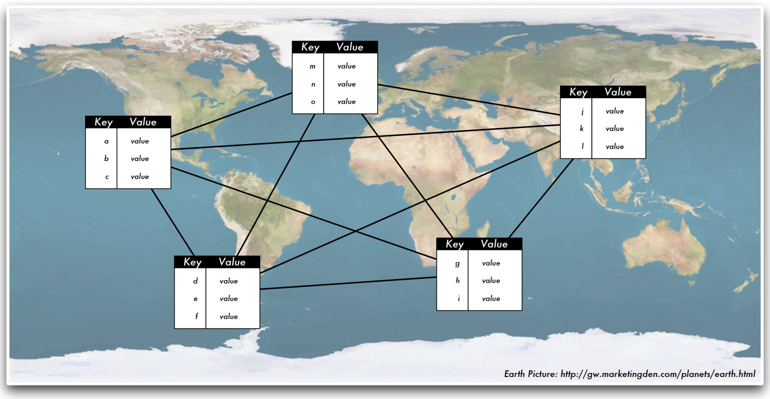


Figure 2: Distributed key-value pairs form a worldwide storage network.

Currently, great efforts to solve this problem are being made in the research community. For the more relaxed case of unsorted key-value (hash tables) architectures, some algorithms and implementations already exist (they are called DHT: distributed hash tables) and are already being used in some special applications, such as file-sharing, peer-to-peer networks, but are not suited for other applications (like databases) where the sorted access to data is important.

For the more interesting case of distributed *sorted* key-value architectures, not much has been done so far. There are some research project working towards solving this problem (e.g. P-Rings [14], Boxwood [49]) but aside from some private prototypes, nothing “real” has been released to the public yet.

1.3 ERT (Extended Root Tree) Approach

The goal of this work is to build a prototype of a system that provides a distributed storage for *sorted* key-value pairs. It can then be used for different kind of applications (e.g. distributed databases, file systems, web applications).

ERT (Extended Root Tree) is a proposed solution for such a distributed sorted key-value architecture and will be introduced in detail in section 3. It is based on a conventional balanced tree (b-Tree) architecture, which has been extended by a separate handling of their root nodes (Extended Root Tree).

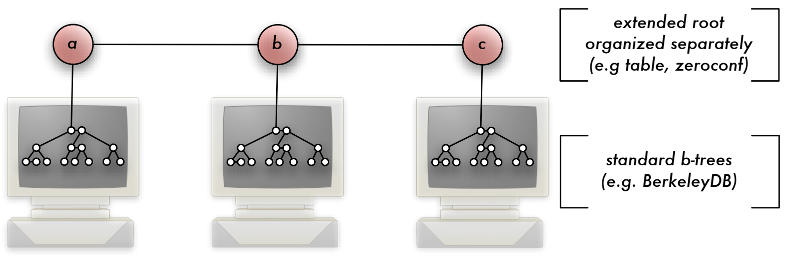

a
b
c
extended root organized separately (e.g table, zeroconf)
standard b-trees (e.g. BerkeleyDB)

Figure 3: Extended Root Tree

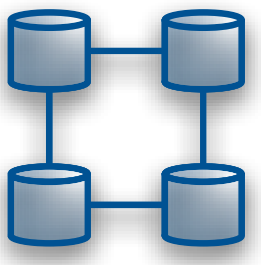
This data structure, which provides distributed storage and a simple interface around its key-value architecture for small/medium/large systems, can be used as an underlying framework for building different kinds of applications on top of it.

The prototype implementation of ERT is hosted at `http://www.nix.ch/kussmaul/ert` and has been released under a BSD open source license [36].

1.4 Structure

The work can roughly be grouped into 3 main parts: The first part describes the theoretical background, the second part describes the chosen approach and the third part the implementation details. Additionally there is a section about "Evaluation" and an ending section, the "Conclusion".

- **Distributed Data Storage -** Describes state-of-the art methods for managing distributed data. It will first give an overview of general distribution techniques and describe the key factors for a successful distributed system. It will then describe methods for distributing data tailored to requirements for small/medium/large systems. The next part will describe local methods for storing data, which are already being used in monolithic systems and are still needed for distributed systems as well. The last part will then discuss performance aspects of a distributed system, such as concurrent access to data and high throughput systems.
- **Overview of ERT (Extended Root Tree) -** Describes the chosen approach to build such a distributed data structure on a more theoretical level.
- **Implementation of ERT (Extended Root Tree) -** Describes the current JAVA implementation of the built prototype and is meant as a program documentation to understand the underlying implementation details.
- **Evaluation** - Describes the benchmarking of the prototype and strengths and weaknesses found during testing and implementation.
- **Conclusion -** Provides a summary and a short outlook of the chosen approach.



2 Distributed Data Storage

2.1 Introduction

2.1.1 Nature of Data

Storage and retrieval of data is used everywhere and is becoming more and more important. In today's world we see the following tendencies:

- **Data Growth** - As we live in an information age, the amount of data (conventional, digital) is increasing rapidly. Studies [18, 70] have tried to get an overview of how much data is produced each year and estimate that in the year 2002 alone, more than 5 terabytes of data were produced [70]. This includes data stored on paper, film, magnetic and optical mediums. The biggest part of the data growth has come from magnetic storage. Digitization is also becoming more common: Old movies are being re-released in digital form (DVD, cinema), and books are being digitized (E.g., Google.com provides a service for full-text searching of books [29]).

- **Data Distribution** - Data needed for a specific task is often not centralized anymore; it is distributed and stored at several locations, even around the world. In the future, this tendency will continue to grow. As for certain data, specific requirements for storage are needed: High-value data (such as permanent data for banking institutes and telecommunications firms) is stored at special secured sites with access control and protection against power failure or other disasters. In contrast, data of lower value (such as temporary data for product downloads, web space) is more commonly stored at places, which provide the most value for the money. Some examples of distribution are:

  - **Personal Data** - More and more people already work on several computers at different locations, such as on a computer at work, at home and perhaps on a mobile device (such as laptop, palmtop or mobile phone). Some data reside on a specific computer, but there is a trend to ”synchronize” the data among all devices, or to allow access to important data from all devices (such as emails, contacts, appointments).

  - **Business Data** - Since companies have been growing, there is no centralized data store anymore; the business data is distributed to several places, as described above. Also, company data is often not stored exclusively inside the organization itself, but is distributed among subsidiaries or partner companies. Some data, such as financial data and address data, is often bought or leased from intermediate companies.
  - **Government Data** - As governments are large organizations which have separate departments for each specific task, they produce data that reside in different places.
  - **Scientific Data** - For large experiments, the data produced can no longer be processed locally (e.g. CERN [12], Folding@home [57]), and methods for distributing the data for remote calculation will be developed.

- **Data Finding** - Storing the data is only one aspect. Of course the data needs to be found and retrieved again later. While web search engines allow for more or less accurate searching of public data on the Internet, new trends are focusing on more personal data searching techniques, which function in the same way as existing web search engines (like Microsoft WinFS [50], Apple Spotlight [4] or Google Desktop Search [28]). There is also a trend towards a more structural storage of data for easier finding. Keywords are: semantic web [52], metadata [27, 82], relational data [86].

  - **Personal Data** - With the increasing amount of personal data (emails, multimedia data, documents) stored at multiple places (work, home, mobile devices), finding the necessary data is becoming more crucial and complex. Today finding information on the Internet is easier than finding data on a personal computer. As most users store the data in an unstructured way, the common methods for finding data - indexing the data before searching, as in web search engines - are normally used.
  - **Business Data** - Growing complexity and dependency on information technology is not easy to overcome. Companies normally use a more structural approach in managing data (in contrast to the personal data methods described above) and try to store data together with metadata or along with other information, even in a relational database. This allows for more exact searching capabilities.
  - **Government Data** - For several tasks, it is necessary to combine different data sources to get the desired result. For instance, if the immigration bureau needs to exchange its data with other countries, the data and metadata must be stored in a structured way. For exchanging those data with other authorities, standards for exactly how the data should appear are needed. The trend here is to have open standards for data exchange.
  - **Scientific Data** - The data itself is not always the interesting part. Oftentimes, data needs to be further analyzed and processed to get the desired

results. So finding the data means finding interesting relations between data. Here the data itself has to be very well-structured to allow automatic post-processing and also well-described so it is transparent and others can follow the experiments. Often for smaller projects, the data is not shared with other groups and it might be sufficient not to describe the data completely, but only to structure it for the needs of the project itself. For larger experiments, the data is shared with multiple groups and strict standards need to be put in place to allow other members to find the necessary data.

Generally speaking, if you need accurate access to data (e.g. finding *all* bank accounts within a specific criteria) the key is to structure the data in a precise way. If you are satisfied with inaccurate access to data, other heuristics can be used to give access to unstructured data (e.g. general indexing of plain text, as done by Google or others).

Quintessentially, it could be argued that:

"Storing data is easy - finding the data afterwards is the challenge."

2.1.2 Managing Data

For the storage and retrieval of data, there are several approaches, which can be distinguished into two main fields (as already described above):

1. **Structured Data** - This data has some sort of defined structure. This can be additional data describing this data (metadata [27, 82]) or other methods for structuring, such as grouping data and describing relations between data (relational model [86]). This structure can be added manually, e.g. by a person who has knowledge about how to structure a certain piece of data, or automatically, by using some heuristics to classify the data. Both methods have pros and cons, as a manual approach is always biased in favor of the person who does the manual classification and automatic categorization will suffer from inaccuracy because of limitations of the algorithms used. Examples are: XML files with a well-defined schema definition, relational databases and categorized articles.

2. **Unstructured Data** - This is data which has no structure. This can be data of any kind, which is mixed together with no relation between. Examples are: A storage medium with an arbitrary number of files of any kind, such as Word documents or pictures.

There are several approaches to managing this kind of data. Some of them allow for searching at different levels, while others only store the data and allow the user to browse all of it. Some examples (sorted from small to large systems) are:

- **Arrays and similar** - These are low-level, used by the application developer: These contain unstructured data and only the specific application can understand and use it. Even the computer memory (RAM) has some kind of array-like structure. The only way to access the data is by asking for the data at a

specific address. Example:
“Give me the data at position 5 of the array”

- **Key-Value Storage -** This is low-level, used by the application developer: The data can already be slightly structured: Each data item has a describing key, which then holds the corresponding value. Depending on the key-value relation, the data is more or less structured; e.g. if the key is only a random number, then it probably does not add any description to the data itself. In contrast, if the key contains a valuable description, such as “name” it adds some metadata to the data. Generally, there are two types of key-value storage mechanisms:

  – **Unsorted -** The keys are unsorted. This means the data is not ordered. This is mostly implemented as a “hash table”. Example:
  “Give me the data for the key ’Zurich”’

  – **Sorted -** The keys are sorted according to a criterion (e.g. alphabetically, numerically). It is mostly implemented in a tree-like structure, with which the ordering is preserved at insertion time. Additionally to unsorted access, it allows queries such as:
  “Give me all data for the keys which start with ’Z”’

  For a more detailed description of key-value mechanism, see section 2.3.

- **File Systems -** From the low-level point of view, the data is structured in some way, e.g. the file system knows at which position on the disk a specific file is stored and in which sub-directory it is located, but mostly this information is useless for high-level usage; e.g. to find a file which contains a certain word, all the data on the file-system needs to be browsed. Example:
“I have to browse the file system to find the file I need”

- **Databases -** Contain structured data. Furthermore, the database requires that only data that conforms to this structure can be inserted. Before filling a database with data, the designer has to decide how the data should be structured. Changing the structure afterwards is not an easy task, as adding a property mostly means updating all existing data items with these new properties. For accessing the data, there are sophisticated methods, which allow for fast and accurate searching of data. Example:
“Give me the data which has the property ’city’ set to ’Zurich’ and which has the property ’income’ greater than ’100,000’ Swiss francs”

- **Web Search Engines -** Depending on the architecture, these are based on databases or have their own techniques for storing the data. The data can be the web address of a page or even an archived copy of this web page [30]. To allow searching for unstructured data, the search engine maintains a huge index for the web pages to search. The algorithm to index and finally rank the index is unique to each search engine. Example:
“Give me the link of a web page which is relevant for my word, ’Zurich”’

- **Peer-To-Peer Systems -** This is another unstructured way to store data: Several computers are connected to each other and each acts as a server and client at the same time and are able to retrieve some data. There are some large systems, such as the older Gnutella [24] network. Mostly those systems only allow for inaccurate, rudimentary and very slow searching capabilities (normally only on file names or some limited predefined properties such as file size or type of music encoding used, but all these additional properties do not have to be included all the time). Example:
  "Give me the data which has to do with 'Zurich'. This will not return all data, as some cannot be found, and can even return data which has nothing to do with 'Zurich"'

As already noted in the introduction, most systems described above are internally built on top of a key-value storage architecture and could therefore profit from a more general approach to do the distribution already on the key-value layer. Such systems are:

- **File Systems -** The base for most file systems is a sorted key-value storage architecture. Examples include: Microsoft NTFS [84] file system, Linux ReiserFS [54], Linux Ext2/3 [10], Apple HFS(+) [80] and others.
- **Databases -** Here also, key-value based solutions are commonly used as their underlying storage architecture, e.g. for building an index or allowing for sorted access to data [78].
- **Web Search Engines -** As some of them are based on databases or file systems, key-value kind storage algorithms are also found here, e.g. for the creation of the index [9].
- **Peer-To-Peer Systems -** Most of them are built on top of an unsorted distributed key-value architecture [24].

2.2 Distributed System

According to Tanenbaum [69]:

> "A distributed system is a collection of independent computers that appears to its users as a single coherent system."

As it appears, this definition is quite difficult to implement, so normally the following weaker definition [44] is used:

> "A distributed system is a collection of independent computers that are used jointly to perform a single task or to provide a single service."

Building a distributed system will always have some trade-off between different desirable properties. E.g. a system that is fully secure will probably not be fully efficient [43]. Also for specific use cases not all distribution problems need to be solved. Especially in our case, to build a distributed balanced tree, not all aspects need to be addressed. General important goals in a distributed system are (taken from different sources: [21, 22, 32, 43, 48, 60, 61, 62]):

- **Transparency** - For some applications it is necessary to hide from the user the fact that the system is distributed, so the user thinks he is working on a local system. There are different kinds of transparencies:

  - LOCATION - Users unaware of location of resources
  - MIGRATION - Resources can migrate without name change
  - REPLICATION - Users unaware of existence of multiple copies
  - FAILURE - Users unaware of the failure of individual components
  - CONCURRENCY - Users unaware of sharing resources with others
  - PARALLELISM - Users unaware of parallel execution of activities
  - OTHERS

- **Flexibility** - Depending on the need, a distributed system should be extensible, so additional components/services can be added/changed. Interface specification should be open, to allow easy addition of new features or enhancement of interoperability. Flexibility can also have an impact on other factors, such as reduced performance and security.

- **Reliability / Availability / Redundancy** - To enhance availability, load-sharing can be implemented with additional redundant data, so if part of the system fails, the system is still fully operable and after the failure, the whole system should stabilize again. There is also the problem that in a distributed system we have more components that could potentially fail, so to increase reliability, some recovery methods need to be implemented to bring the system back into a consistent[1] state after a partial failure. Reliability/Availability/Redundancy can have an impact on performance, flexibility, scalability, concurrency and basically all other elements.

- **Performance** - There are multiple measurements of performance, such as: Response time, throughput, system utilization and network capacity used. Any system will benefit from performance enhancements. In distributed systems, performance can have an impact on other desirable properties, such as transparency, flexibility, reliability and security.

[1]There are some objections that for a large distributed system, full consistency can't be reached and that, therefore, some ideas need to be implemented so the whole system will work, even if it is not in a fully consistent state (e.g. voting mechanism to check whether result can be trusted, multiple querying to see if the same result is retrieved).

- **Scalability** - "A system is said to be scalable if it can handle the addition of users and resources without suffering a noticeable loss of performance or increase in administrative complexity" [B. Clifford Neuman].
  There are three metrics in a scalable system. Depending on the needs, some of them are more important than others:

  - SIZE - Number of users and resources
  - GEOGRAPHY - Distance between users and resources
  - ADMINISTRATION - Number of organizations that exert administrative control over parts of the system

  Scalability can have an impact on performance. Sometimes a distributed system only provides better performance if a certain initial size of the system is being reached[2]. Scalability influences reliability, availability, redundancy and flexibility.

- **Concurrency / Parallelism** - Components in a distributed system are normally concurrent by design, as each component can perform its own calculation (those can be independent of or dependent on other components). Another aspect is the synchronization of information across the system. Concurrency has an impact on reliability, availability and redundancy as well as on performance.

- **Security** - A distributed system is more complex than a similar centralized one, so for components which need to exchange information, there is the risk that malicious components can harm the whole distributed system. Security has an impact on reliability, performance and scalability.

2.2.1 Distribution Methods

There are many methods to build a distributed system. Below, some strategies from large database vendors are discussed, how they distribute their data to multiple nodes. As those product portfolios are huge, and also because information is restricted, not all aspects are shown here. Currently the topic "distribution" is very hot, especially with regard to grid computing, and as vendors are trying to get into this area, the competition is quite tough.

Generally, a full-blown database (in contrast to our approach of a low-level database) allows several approaches to actually achieve distribution. E.g. the distribution could be done at table level, SQL level, or even higher at application level.

To compare the three databases, papers from [33, 37] helped give a general overview, although one of them is sponsored by IBM.

[2]Often a distributed system containing two nodes is slower than a comparable centralized system.

2.2.1.1 Oracle - Oracle Corporation Oracle's solution for distributing data is based on using a "shared-disk" architecture [37]. In the most current version (Oracle 10g), Oracle is going into the direction of grid computing, but uses the common former distribution technique (RAC - Real Application Cluster) as the foundation of their grid offerings. RAC is a cluster database with a shared cache architecture that runs on multiple machines attached through a cluster interconnect and a shared storage subsystem [51].

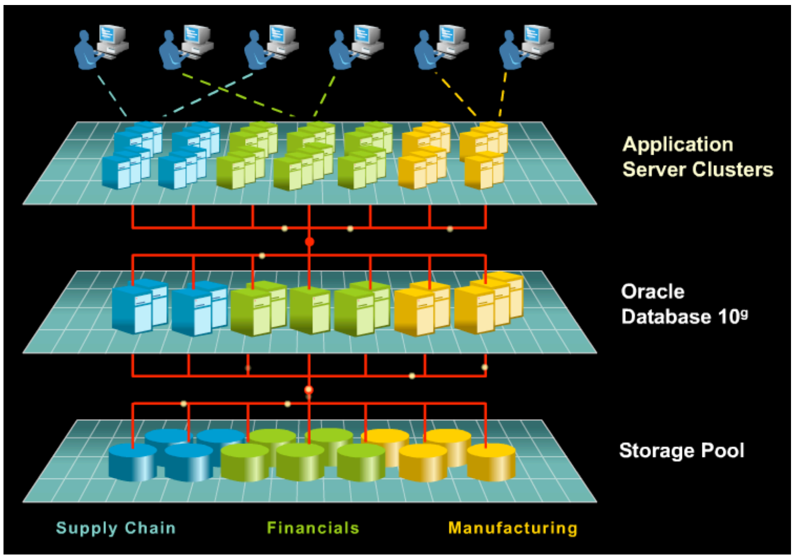


Figure 4: Oracle 10g Grid Overview (taken from [56])

Using a cluster architecture has some limitations, as all machines need to be connected using a cluster-interconnect (a fast path from machine to machine); so distribution is limited to a high-performance LAN/Memory interconnect. Those machines can't be distributed over a larger area (e.g. the whole Internet). Also, depending on the cluster architecture used as a foundation, there are some more limitations. From our own experience in the telecommunications field where we deployed clustered systems, we have found that most architectures can only scale to a small number of nodes inside one single cluster (8-32 nodes per cluster [38]). The advantage of such a system is their high performance capability, because all machines are within a very fast network, but the limitation is the small size of such a system and the inability to distribute the cluster itself over large distances - so all nodes participating in a cluster must reside at the same location.

Using a storage subsystem (also known as shared storage, storage area network) is limited to a local distribution of the system, as access to this storage system has to be fast. The disks can't be spread over a large distance. Also, building a shared storage system can be quite complex, because it needs to have several mechanisms built in for concurrent access, e.g. a distributed locking manager, and therefore Oracle is limited to systems, which actually support those features. See Appendix A on page 89 for an overview of two example architectures of such a storage subsystem.

The major benefit in using a shared disk/cluster architecture over a monolithic system is the increase in availability of such a solution. If one node goes down, all the storage is still accessible and another node can take over working on the same data.

Oracle has other offerings to distribute part of the database or the whole thing to larger areas. This is done in a replication manner using “Oracle Streams” to have multiple databases in sync over a bigger distance or using partitioning (explained below).

2.2.1.2 DB2 - IBM Corporation In contrast to Oracle’s offering, DB2 follows the “shared-nothing” approach: Each node owns and manages its own resources, including the data it has access to and the DB cache [37]. This approach was chosen to address the lack of good shared disk support (e.g. distributed locking mechanisms) in Linux, UNIX and Windows. Using this approach, IBM is claiming to reach “infinite” scalability to at least 1000 nodes [33]. Generally, this technique is closer to our b-tree distribution method, where we also plan to have “independent” nodes.

To distribute data in a DB2 environment, it mainly uses partitioning (MDC - Multi Dimensional Clustering [34]) to spread a table across multiple partitions, residing on multiple disks or systems (so a relational database table will be split horizontally into several chunks and those are then placed into different nodes).

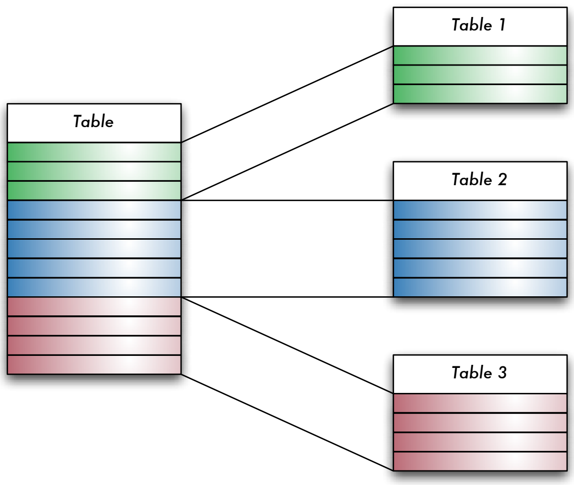


Figure 5: Table Partitioning

This also allows for parallel executing of queries, as every node can process its own data and only the result will then be combined. Generally this technique is also available in Oracle.

2.2.1.3 SQLServer - Microsoft Corporation In contrast to Oracle and DB2, SQLServer does not support clustering of machines by itself. It is limited to one machine, although it can be a multi CPU machine. Distribution of data is only possible using “federated databases” (linking distributed, heterogeneous databases together using software), [33] e.g. by having a database installed in each country and then running an application which will query each database independently and combine the results. This approach is similar to partitioning, but in the case of SQLServer, the user has to do it by himself in the application accessing the database. This approach has several limitations. E.g. in case of renaming parts of the table, this renaming has to be propagated manually to all participating database nodes.

2.3 Local Data

Regardless, for monolithic or distributed systems, they need to store data. As each machine only has direct access to its own hardware, the actual low-level storing is always done locally[3]. This means that the actual algorithm to store the data can be the same for monolithic machines or nodes of a distributed system.

There are several algorithms for storing the key-value pairs. In this section, we will mainly focus on algorithms for sorted storage of key-values and will further focus on the most commonly used algorithms.

2.3.1 Binary Search Tree

A binary search tree [76] is a special form of a binary tree [77], in which all nodes are inserted in an ordered manner: If you pick one node, all sub-nodes to the left always contain values less than this node, and all sub-nodes to the right contain values greater than this node itself, as shown in Figure 6. This data structure is the base for most others, which just add functionality to it.

[3]The terms distributed/monolithic have no exact boundary, as one could argue that a single machine itself has distributed components: CPU accesses the memory over an internal bus-system and accesses the disk over an external cable, for example. Nevertheless, in this work distributed means connected using a network interconnect to each other. All other bus-systems or internal communication paths are regarded as part of a monolithic system.

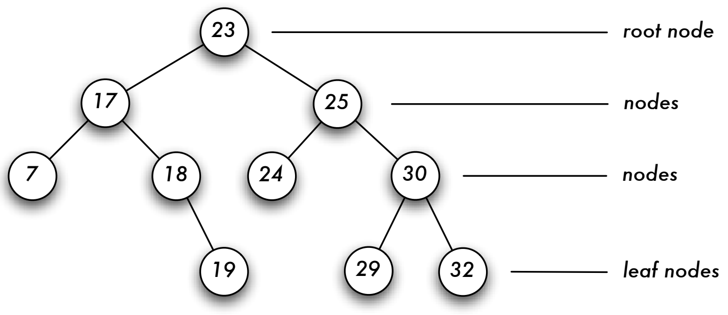


Figure 6: Binary Search Tree

As a binary search tree does not rebalance itself during insertion of new elements, the tree might, in the end, not look like a tree but more like a linked list, where all nodes are chained after each other. In this case, the tree is highly unbalanced.

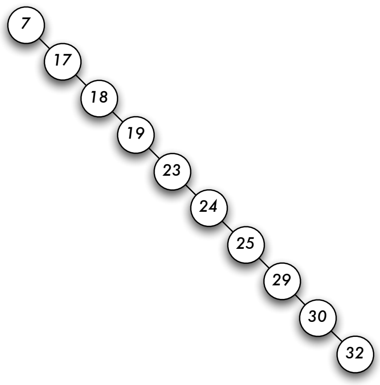


Figure 7: Highly Unbalanced Binary Search Tree

This will limit the search performance of such a tree, as for the worst case scenario described above, the search-algorithm basically has to look up all nodes of this tree and is of order $\mathcal{O}(n^2)$, where n is the number of nodes of the tree.

2.3.2 Self-Balancing Binary Search Tree

A self-balancing binary search tree [87] is a special form of a binary search tree in which the algorithm tries to limit the shortcoming of a normal binary search tree by implementing a method to prevent the unbalanced growth of the tree. The height of the tree is automatically kept to an optimal minimum. Depending on the actual implementation, the worst-case lookup time can then be reduced to an order of $\mathcal{O}(\log n)$, where again n is the number of nodes of the tree.

Some data structures that use a self-balancing binary search tree are:

- **AVL Tree -** Invented by Adelson-Velskii and Landis [73], 1962.
- **Red-Black Tree -** Invented by Rudolf Bayer [85], 1972.
- **Splay Tree -** Invented by Daniel Sleator and Robert Tarjan [88], 1985.
- **B-Tree -** Invented by Rudolf Bayer [74], 1972. This is the most commonly used form of a self-balanced binary search tree. There are also a variety of slightly different algorithms which all are based on the original b-tree.

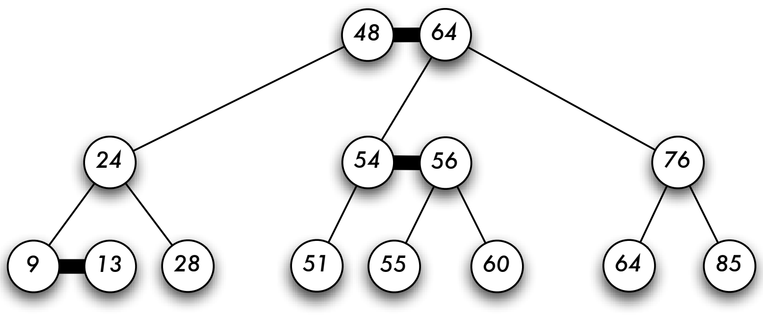


Figure 8: B-Tree

As shown in Figure 8, all leaf-nodes have the same height of 3 and always need the same amount of time for lookup. In contrast to a normal binary search tree as shown in Figure 6, the height depends on the nodes: Nodes 7 and 24 have a height of 3, while the other leaf-nodes have a height of 4. Therefore, the lookup for each leaf-node takes a different amount of time. Hence, the tree could be even worse - a few leaf-nodes could have a much greater height, for instance (as seen in Figure 7).

2.3.3 Skip List

This data structure is also commonly used and is based on parallel linked lists and was invented by William Pugh [58], 1990. Instead of only linking the successors/ancestors, crosslinks from one element to another arbitrary one are possible.

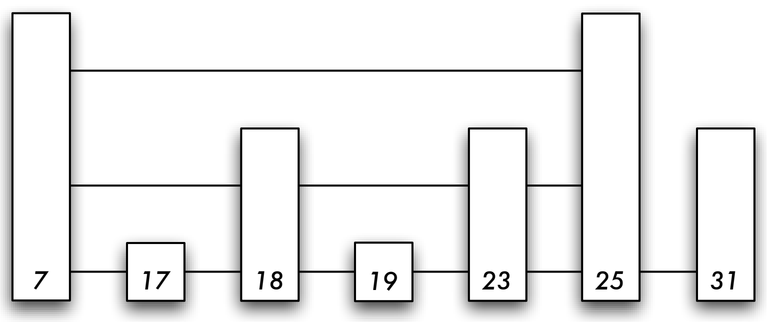


Figure 9: Skip List

It has advantages in parallel computing where the re-balancing (or re-linking) can be done independently of the other nodes: Each element can decide by itself to whom a link should be established, without global communication. Nevertheless, it has some shortcomings, as there is still a possibility (although very low) that at the end the skip-list is just one huge linked list, which would then have a worst-case lookup order of $\mathcal{O}(n^2)$. But in normal circumstances, this algorithm can achieve a lookup performance similar to a balanced tree of $\mathcal{O}(\log n)$.

2.4 Distributed Data

While good implementation of balanced trees for centralized architectures exist, we are looking for a solution on how to distribute sorted key-value like structures (e.g. balanced trees) among multiple nodes, to gain higher throughput, as well as redundancy. With this approach some new problems, which are normally not found on a centralized system arise:

- **Data Balancing -** Data is now located on different computer nodes, so a mechanism that will distribute the whole database over all participating nodes has to be built. Depending on the needs, some nodes can hold more data than others (e.g. if they have larger disks, more processing power) or data should be evenly distributed among all nodes.

- **Data Reallocating -** A database is highly dynamic, so it has to move its data automatically among other nodes if needed. (E.g. a lot of inserts could lead to a growth in some limited number of nodes, so data has to be transferred to other nodes to gain good balancing data). This is related to data balancing.

- **Communication Between Nodes -** Depending on the design decision, internal communication between participating nodes will be needed (e.g. to reallocate data). Those have to be reduced to a minimum, or at least be on local order only. E.g. there is only communication between “neighboring” nodes, not broadcasting to all nodes.

- **Access Time Concerns** - In a distributed environment, all accesses (e.g. searches, inserts, deletes) eventually go over a network connection, depending on architecture: An access-request could even be delegated from one node to another, which could have an impact on access latency and throughput.

2.4.1 Small-Sized Systems

This section describes some methods for smaller systems of how the data could be distributed and still allow for sorted key-value access.

2.4.1.1 Paging B-Trees One promising approach is the use of paging b-trees [55], where sub-trees are relocated to different nodes. To prevent splitting the tree into too many small sub-trees, the distribution is done on the depth-level of a tree. E.g. if the whole tree has a depth of 4, then splitting is done in such a way that 2 depth fields are always grouped together. This also solves the problem of locality, because now local/similar data is most probably on the same node and not spread to other nodes as well. Experiments have shown that the overhead in querying these paging b-trees is around 5%, while the overhead for inserting is around 50%, which seems quite high. It seems it is better suited for applications, which read more from the tree than write to it.

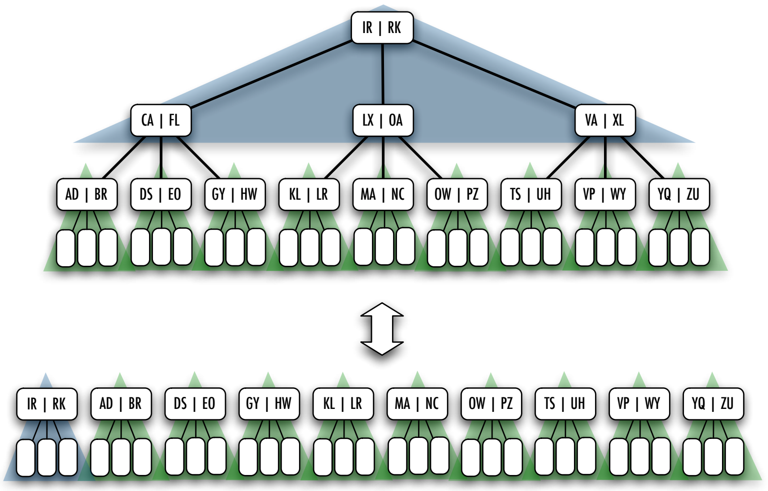


Figure 10: Example of a b-tree split into several parts and put on multiple nodes (taken from [55])

2.4.1.2 Concurrent B-Link Trees A more theoretical work on distributing b-trees is done by Johnson/Colbrook [39]. As a foundation, a concurrent b-link tree [47, 63], which, according to the authors, has the highest performance of all available b-tree algorithms, is used. This is done with fine granular locking mechanism in case of concurrent access on the b-link tree. Further b-link trees help in finding the data, even when parts of the upper tree are missing, because all leaves are linked to each other, so in case of a node failure or a short-local inconsistency (e.g. during a balanced tree re-organization process) the data can still be found.

Because all b-tree queries normally start at the root node, Johnson/Colbrook [39] proposed to replicate the root to all available nodes to have better throughput. The same is done for sub-trees, so in the best case the request will never leave the first chosen node and will find the data without hopping to different machines.

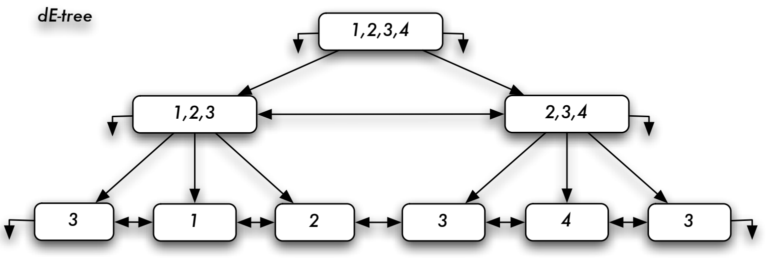


Figure 11: Shows the replication of sub-trees to get higher throughput (taken from [39]). The numbers show, on which CPU this part of the tree is stored. E.g. the root tree is stored on CPU 1, 2, 3 and 4.

2.4.2 Medium-Sized Systems

The difference between small and medium sized systems is not very big. Most people consider a system medium sized if it is too large for huge data exchanges (e.g. it does not use a shared disk approach) but it is still efficient enough to broadcast messages to all members of a system. Mostly those systems are also located at the same site and are not distributed over the whole world.

2.4.2.1 Boxwood Project This project is part of a Microsoft research program [49] to build a distributed storage architecture. It focuses on building a distributed file system on top of a distributed data structure. To solve this problem, they first want to build a distributed balanced tree architecture and then use this as a foundation for their file system. Unfortunately, no source code or architectural overview, on how they actually want to build it, has been released so far.

2.4.3 Large-Sized Systems

Large systems are considered Internet-scale systems, which also have the characteristics of unpredictable failure of nodes and communication between them. Such systems are mostly built around a structure common to P2P (Peer-to-Peer) networks or have a hierarchical approach (e.g. such as DNS systems).

2.4.3.1 Structured Peer-to-Peer Overlays Structured Peer-to-Peer (P2P) overlays are the foundation of highly distributed P2P networks and currently a hot topic. In contrast to older P2P networks like [24], those new approaches [23, 40, 59, 66, 90] focus on true decentralized mechanisms, scalability, self organization, higher reliability and better performance. They allow routing to location-independent names (e.g. “route message to *key*, where *key* is located on a node-not-yet-known in the network). On top of such an architecture, higher-level operations, e.g. a distributed hash table (DHT), decentralized object location and routing (DOLR), group anycast/multicast (CAST) and possibly, also a distributed balanced tree (DBT) can be built.

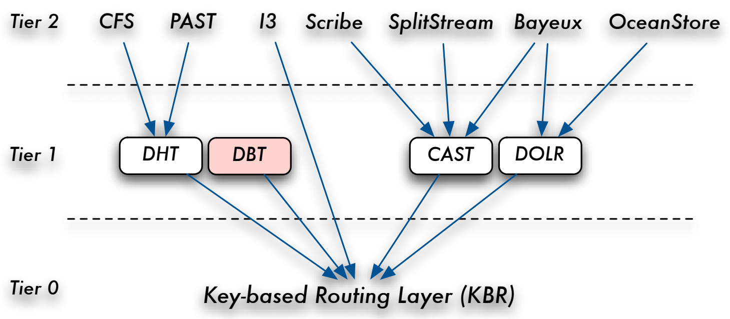


Figure 12: P2P overlay (key-based routing layer) with several services on top of it and applications using those services (taken from [17])

There are some projects going on which are working on a distributed low-level data-structure called Distributed Hash Tables (DHT), which are commonly used in Peer-to-Peer network architectures and are also found in some research projects related to distributed storage. The difference between hash tables and b-trees are that hash tables only allow unsorted access to keys (e.g. a lookup of a range of values is not possible) and b-trees have sorted keys. It seems that for distributed storage (e.g. file-based access, Peer-to-Peer networks) a DHT is enough, as several projects [8, 42] are using it. Nevertheless, no implementation of a sorted distributed data structure has been found so far.

All those P2P overlay projects focus on large-scale distribution, and therefore try to minimize global communication, so we can learn from them:

- How those projects do data balancing.
- There are some techniques to build a b-tree on top of a P2P overlay (e.g. [13]). With this approach we could use an already proven overlay implementation and add a b-tree to it. It has to be shown whether there are drawbacks. E.g. data balancing could be worse or access time could be less than perfect.
- There exist already a DHT implementation on top of those P2P overlays. Perhaps we could find a way to build a distributed b-tree on top of such a DHT.

2.4.3.2 Sorted Distributed Hash Tables The main advantage of using a balanced tree instead of a hash table is the fact that all keys are sorted. Besides using a b-tree, there are other techniques, e.g. those based on hash tables, to get access to sorted keys (some are described at [89]). With those methods, we could use existing DHT implementations and build a system, which has the same functionality as a distributed balanced tree.

One proposed data structure, which would provide the full functionality of a distributed balanced tree are skip graphs [7]. Skip graphs are based on skip lists [58] (an alternative to balanced trees, see section 2.3.3) and enhance them with DHT to get a truly distributed balanced tree. Nevertheless, Yalagandula [89] showed that skip graphs do not seem to scale well.

Another is Squid [64], who basically allows multidimensional sorted keys, but here also Yalagandula [89] showed that this method does not scale well.

Yalagandula then proposes his own approach by combining Tries [1, 25] with hash tables by first filling the data into a Trie and then mapping this structure onto physical nodes using DHT techniques. Although the author plans to implement a prototype of his idea, no such implementation has been found so far.

2.4.3.3 P-Tree (P2P Tree) Instead of splitting a single balanced tree across all nodes, there are also ideas to have multiple independent b-trees and assign them to each available node. One such an approach is described in [13], where they introduce P-trees (Peer-to-Peer trees) and some kind of semi-independent B+-trees (a flavor of the common b-tree). This is a mixture of having one full tree distributed over all nodes and having several independent trees.

Each node will build up some part of a global index where the actual data can be found. This index does not need to be fully consistent; in case there are some failures, the search will take just a bit longer. The main disadvantage of this structure is that the data, which can be stored on one node, is very limited (e.g. only one single item of data can be stored - this limitation can only be overcome by starting multiple virtual nodes on each real node, but even with this way, the amount of data to be stored is very limited). This data structure can be seen more as an indexing framework, where only a very limited amount of data needs to be stored at each node. Or the node itself is the interesting data to be found, so this data structure could be used for node discovery, e.g. as a “replacement” for DNS. (It would then be called DDNS, Distributed DNS).

2.4.3.4 P-Ring (P2P Ring) To overcome the limitations in P-trees, an extension has been proposed: P-ring[14]. With this approach, it is now possible to store large amounts of data on each node and the current limitation only seems to be that the amount of data stored on each node is expected to be roughly the same. But also here, no prototype has been released to the public so far.

2.5 Performance Aspects

When designing a distributed system, special care needs to be taken with performance issues. Generally, a distributed system will send some information between each of its nodes. Depending on the actual architecture, such messages could be:

- **State Information -** General alive messages, state-change information or messages needed for synchronization of the overall system wide state.
- **Managing Information -** Such as information about joining, leaving, or statistics.
- **User Information -** The actual user messages, the data packets that need to be processed by the system.
- **Overhead Information -** As each network layer, including the application protocol, adds some housekeeping information to each message, the overall message grows in size.

Sending messages and accessing data remotely is always very expensive, as in current network topologies (e.g. Ethernet, IP) transmitting information between nodes is slow, compared to local access methods, as shown in Figure 13. This is especially true if the nodes are distributed on the Internet, where the network latency is quite high compared to the other methods.

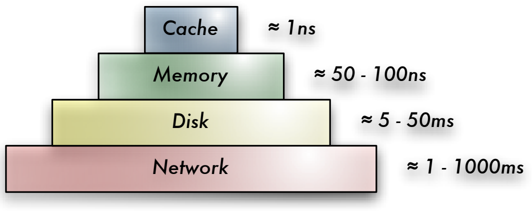


Figure 13: Access Time Differences (taken from [53])

2.5.1 Minimize Disk / Network Traffic

The goal is to minimize disk- and network-related traffic, as those have the biggest impact on slowing down a distributed system. As disk access is always done locally, it is not only a problem in distributed systems but also in basic monolithic ones, and several implementations for local key-value storage have already addressed this issue (e.g. the BERKELEYDB [65] implementation of a balanced tree implements advanced caching and disk access procedures to further reduce disk access). For discussing performance issues on a distributed system, minimizing and optimizing network traffic becomes the main issue. Basically this problem is split into two main parts:

- **Problems you can control** - These are issues which can be reduced by choosing a good design and implementation of the distributed systems. Such things are:

  - **Algorithms** - Using efficient algorithms probably has the highest effect on reducing network traffic - the goal is to reduce all unnecessary traffic and also limit the amount of data to be exchanged. This has to be done not only for the network-related part, but also for the underlying architecture. E.g. if the system can also correctly operate in a not fully consistent state, the number of messages related to consistency can be reduced. Or by taking advantage of parallelized algorithms, where the actual workload can be split into multiple subtasks, which could then be processed independently, the latency of each request can be reduced.
  - **Broadcasts** - Instead of sending requests to all members of a system, the message could be sent only to a subset of members (multicast) or the number of such messages could be reduced in favor of more personalized messages (point to point).
  - **Caching** - Requests could be cached, so if a similar access needs to be processed, it could be found in the cache and would not need to be sent out to the network.
  - **Overhead** - Each network layer adds some administrative overhead to a message - generally this is out of your control, but by wisely choosing the right network medium (e.g. TCP or UDP) and the serialization coding (to convert internal data to binary data) and not adding much overhead with your own application, this overhead can be reduced.

- **Problems which are out of your control** - These are issues which arise in the network and are not in control of the application itself. These issues harm the overall system performance, as the system needs to react to such failures with additional administrative effort (e.g. by implementing fault tolerance mechanisms). Some characteristics of problems are (taken from [11]):

  - **Failures** - Network components can fail at any time. This is especially true for larger systems, where the probability that some parts of the networks are down is higher than on smaller systems. Besides network failures, nodes participating in the distributed system can fail for any reason.
  - **Latency** - The time a network message needs to pass from one node to another can vary, and this is especially true for larger systems, or if the transport channel is also used for other traffic (e.g. a company wide intranet, which transports emails, web page requests and also the communication messages for the distributed system).
  - **Throughput** - The amount of data which can be sent during a specific time will also vary, especially for larger systems where the network paths have different characteristics with regard to speed.

2.5.2 Asynchronous Processing

A general problem for systems which access different components which all have different access time constraints (as shown in Figure 13) is that faster components could process much more data but are slowed down because they need to wait for slower components. E.g. the disk or network part of processing will slow down the CPU because it has to wait until the data is stored to disk or sent over the network and waits for a response. The CPU could do other work at the same time instead of waiting for responses.

There are a few methods for overcoming these problems [72]. What they all have in common is that they don't wait for a full request to be processed. They accept new requests and try to execute them while waiting for the slower components to be finished.

2.5.2.1 A Thread per Request This method basically assigns a thread for each incoming request, to process it. As the number of requests increase, the number of running threads increases as well, as shown in Figure 14. Generally, each thread processes the whole request from start to finish by itself and then sends back a response to that request. Similar to old manufactures, where the employee is building a product from start to finish.

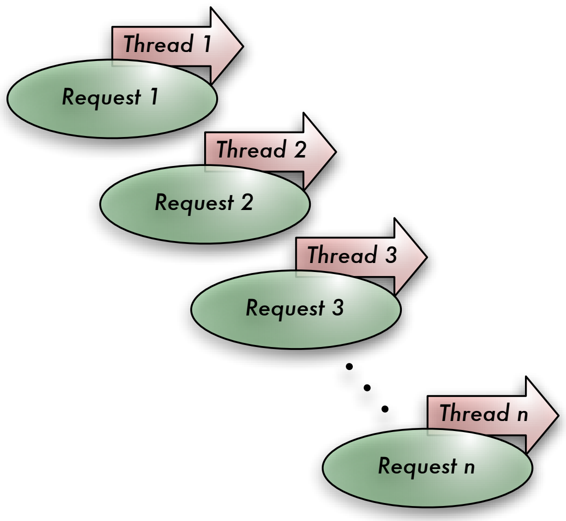


Figure 14: A Thread per Request

This method allows for the processing of multiple requests at the same time. Even when one request takes a long time to process, another thread can be processing another request. This method has the drawback that a lot of threads are running at the same time and the operating system needs to switch thread context quite often. As the number of threads increases, the administrative overhead to manage those threads also increases and overall performance suffers.

There are variations to this approach, such as limiting the number of threads and only allowing a few threads to run at the same time. Additional work, which cannot be processed because no idle thread is available, will be put into a queue and every time a thread has finished its work, it will retrieve the next pending request from its queue. This allows for more control over the number of parallel threads running on the system.

Using a multi-threaded approach, great care must be taken in synchronizing shared resources. If different threads want to access the same resource (e.g. file system) locking must be put in place. This leads to lower performance, as this synchronization/locking overhead needs more CPU resources. In addition, the complexity of the program can increase, as multiple threads could access each component at the same time.

2.5.2.2 Event-Driven Architecture Instead of letting a thread process the whole request from start to finish, event-driven architecture basically runs around events. Events are more finely granulated than a whole request. E.g. a request is a full-blown application-related task, such as: "Fetch web page with name seda.html", while an event is just: "I have a TCP-packet on my network interface", so the whole request is split into multiple events, and for each event type there is a responsible component, which processes it and also generates new events which are then processed by other components. The communication between those components is separated by event-queues; this allows for nicely decoupled components. Similar to more modern assembly lines, where each employee is only building a part of the whole product - this is normally much faster than old manufactures.

The number of threads is not controlled by the components themselves, but by a scheduler component, which gives the thread of execution to those components that have full event-queues.

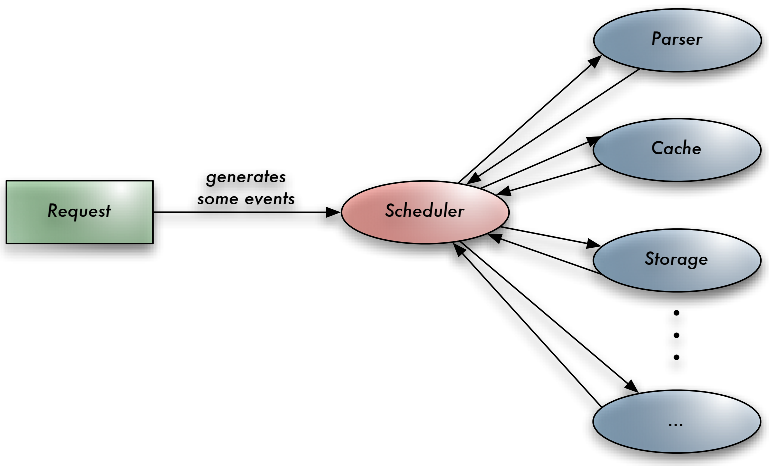


Figure 15: Event-Driven Architecture (taken from [72])

As Figure 15 shows, an incoming request will trigger one or more events, which are then passed to the responsible event-queues for processing (e.g. the request first needs to be parsed). This component itself will trigger new events for further processing (e.g. the parser will now check whether the request could already be answered with information from a cache).

A possible implementation of such an event-driven architecture is SEDA [72] (Staged Event-Driven Architecture), which is used in several applications already. Beside providing access to an event-driven architecture, it also provides additional useful functions for an application programmer and implements a sophisticated thread pooling mechanism which allows for controlling the number of threads for each component (called stage in SEDA) depending on system load.

As we will see later, the proposed prototype ERT also uses this kind of architecture and borrows some ideas from the SEDA approach (e.g. the components only communicate through queues and never call methods directly).

3 Overview of ERT (Extended Root Tree)

When sorted access to data is needed in a centralized environment, b-trees (balanced trees) and their similar flavors are commonly used. We will now discuss how such a data structure can be scaled to a distributed environment to provide the same functionality as a centralized b-tree. The method discussed here extends the algorithms found on balanced trees or other local sorted key-value algorithms with additional functionality to get the desired distribution. One can say that the data structure in the end is not a distributed b-tree, but solves the same problems and gives the same interface to key-value data:

- It allows access for exact match results (e.g. lookup of a single key).
- It allows range queries (e.g. lookup of a range of keys).
- It implements *put(key, value)*, *get(key)*, *getNext(startKey, amount)*, *getPrev(startKey, amount)* and similar interfaces to set and get data from the distributed data structure.

The proposed high-level architecture is designed to scale from small systems to very large systems. The main idea behind this is to reuse as much as possible already available technologies and software, to lower the risk of having to build a completely new system. This also has the advantage that if new trends or new problem-solving techniques come up, it should be possible to incorporate them into this system. Another point is that for different requirements (e.g. smaller systems, larger systems) some components can be tuned for their specific environment, without changing other parts. Such a framework is shown here (similar frameworks can be found in [15, 17]):

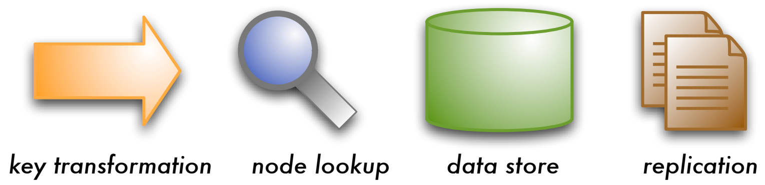


Figure 16: The different components for building a distributed data structure.

The next part of this article shows an overview of how everything works together, and the following part goes into a bit more detail about each component.

3.1 Basic Concepts

The main benefit of a distributed b-tree over a distributed hash table is the sorted access to its data. One idea would be to store the keys out-of-order (unstructured) and only to restore the order during query-processing time. But because the aspect of sorting is the fundamental feature of the whole system, it makes sense to sort the keys at insertion time, so at all times, the whole data structure is in a sorted state (structured).

Considering we have a user key-space $[A, Z]$, this range has to be partitioned and distributed among all participating nodes[4]. Distribution can be done at key-level, so each single key is managed on its own (e.g. [13]), or we can group some keys together (*“key-group”*) and then manage the whole group as one single entity. A *key-group* is like a very small range of the whole user key-space and all keys inside such a *key-group* $a \subset [A, Z]$ are ordered and are responsible for exactly this assigned range. No other *key-group* $b \subset [A, Z]$ contains a key which is also contained in *key-group* a (so $a \cap b \cap \ldots = \emptyset$). Each *key-group* has a unique label (id, hash-key) to identify it[5]. Furthermore, a range of continuous *key-groups* will be called *key-partition* e.g.: $\alpha = [a, d]$. As an overview of how the mapping of a user key to an internal key (= *key-group*) and the mapping to a node is done, take a look at Figure 17. Figure 3 will show, how such a structure is distributed among multiple nodes.

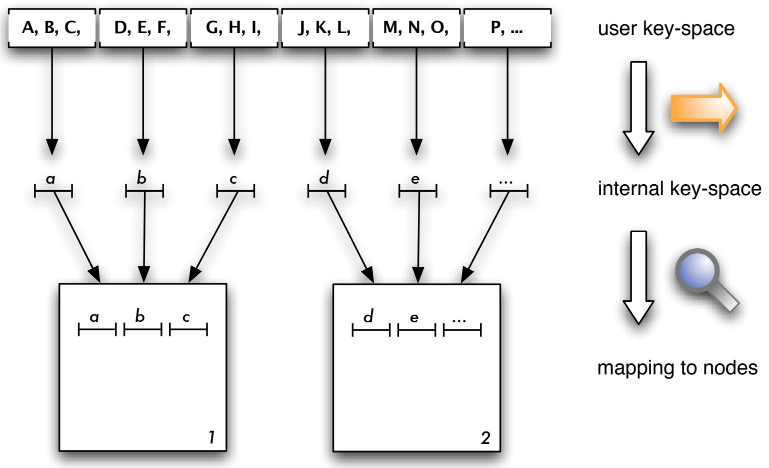


Figure 17: Mapping of user key-space → internal key-space → nodes.

[4]At this point, we won’t look at data replication. So if one node goes away, all data for which this node was responsible are no longer accessible.

[5]In this paper, elements of the user key-space are written in capital letters (e.g. A, B, C) and *key-groups* are labeled with lowercase letters (e.g. a, b, c) and *key-partitions* are labeled with Greek letters (e.g. α, β, γ).

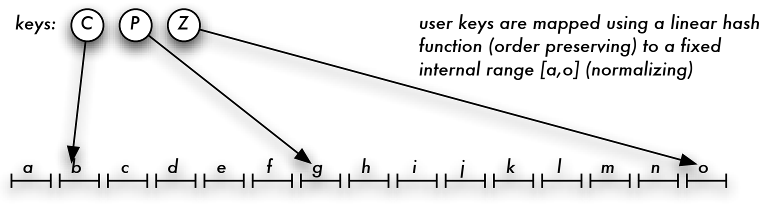

keys:
C
P
Z
user keys are mapped using a linear hash function (order preserving) to a fixed internal range [a,o] (normalizing)
a b c d e f g h i j k l m n o

Figure 18: Transformation of keys to our internal key-space.

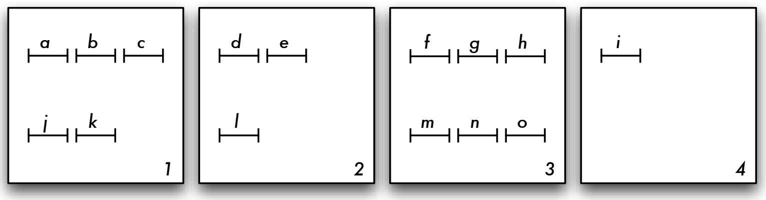


Figure 19: Range $[a, o]$ is distributed among 4 nodes. Node 1, 2, 3 hold two *key-partitions*, node 4 only one.

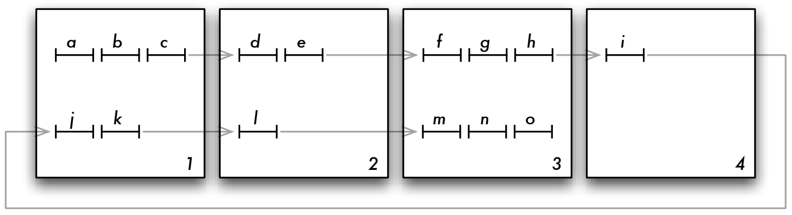


Figure 20: All sub ranges are linked to their successors and predecessors.

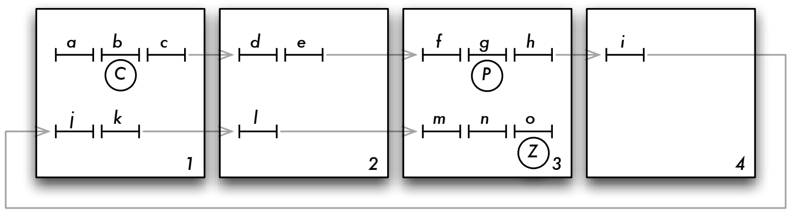


Figure 21: User keys C, P and Z are stored on the corresponding nodes.

As the user of such a distributed data structure wants to have full flexibility with the keys, he might use alphanumeric, large range or small range, but the keys can’t be used as received from the user. They must be transformed to an internal key-range (example in Figure 18).

Assuming we have a whole internal key-range (= *key-partition*) of $\alpha = [a, o]$, then each participating node will be responsible for one (or more) non-overlapping sub ranges of the whole key-range (see Figure 19: E.g. node 1 consists of sub ranges $\alpha = [a, c]$ and $\vartheta = [j, k]$) and all nodes will know where the successor and predecessor of their sub ranges is (Figure 20).

The transformation process depends on the size of the system. For example, on a small 4-node system, it would be possible to have 4 large *key-groups* and just establish a 1:1 relationship between *key-groups* and nodes, so each node is solely responsible for one single *key-group*. On a large system, there are many more *key-groups* and each contains only a very small subset of the whole user key-range.

To retain the original user key, it is stored on the node itself (Figure 21).

The commonly used operations are described here:

3.1.1 Lookup / Storing of Key-Value Data

1. The user key is transformed to the internal key-space using a linear hash-function.

2. The corresponding node for this internal key is now searched using already proven P2P lookup methods (P2P overlays, distributed hash tables [17, 23, 40, 41, 66, 90]) or for smaller systems a simple lookup-table (static table, Zeroconf implementation à la “Rendezvous” [2, 3, 35] or like a DNS system).

3. The corresponding node will receive the request and will return/store the data.

3.1.2 Lookup / Browsing a Range

1. The lower end of the user key is transformed to the internal key-space using a linear hash-function.

2. The corresponding node is now looked up.

3. The node returns all values in the requested range.

4. The node forwards the requests he cant fulfill to the next node (successor) and so on.

For optimization, range queries could also start at both ends, so the lower-end and the higher-end are executed in parallel. This will require some kind of transaction ID, so if the two split and processed meet at one node (e.g. in the middle of the range), this node will know that it has already processed this query and will stop.

3.1.3 Node Insertion

1. The new node will randomly send out some requests to get the load of some nodes.
2. The new node will then contact the node with the highest load to migrate some data.
3. The new node will then register its successor and predecessor. (This can be done in the way b-link trees register new tree nodes [39]).
4. The new node will then officially be the holder of the *key-group*.

3.1.4 Node Failure

1. If a node leaves the system (planned or unplanned), the links to its successor and predecessor will be broken.
2. Its successor and predecessor will search each other and will link together. The range of the lost node will no longer be available (unless there was a data replication mechanism in place), but the rest of the distributed data structure will still be functional.

3.2 Key Transformation

The main idea behind key transformation is that our implementation will have an already pre-defined key-range for the internal key-space. Depending on the technology used for the node-lookup, this can be a very large integer space. If we used “Bamboo” [59] as the underlying node-lookup infrastructure, the internal key-space could be integers from 0 to 2^{160}. At the same time, the user key-space could be phone numbers ranging from 1,000,000,000 to 9,999,999,999 and therefore would need to be mapped to our internal key-space. Note that not the whole internal key-space needs to be used. In this example, the phone number range could be mapped 1:1 into our internal key-space or, better yet, to an even smaller internal key-space, because otherwise each *key-group* would only contain a single entry[6].

This transformation needs to keep the ordering of the user key-space, which is why a linear hashing function needs to be chosen. The hashing function does not need to provide perfect ordering. If the hashing function only provides a proximity result (e.g. it finds that the requested key should be in *key-group* a but is in fact in b), then the corresponding node, which holds the *key-group*, will forward the request to its successor node to query there. Of course, this will have an impact on performance, so the better the hashing function, the better the lookup performance[7].

[6]E.g. the range 1,000,000,000 to 9,999,999,999 could be mapped to 0 to 1,000. This way each *key-group* would hold up to 9,000,000 numbers and we could still scale up to 1,000 nodes.

[7]Forwarding data to the successor node only needs to be done for range-queries. For exact queries, the hashing function will never be inaccurate, because when the hashing function decides in which *key-group* to store the value, it will be the same as when the user queries for the same value for a lookup.

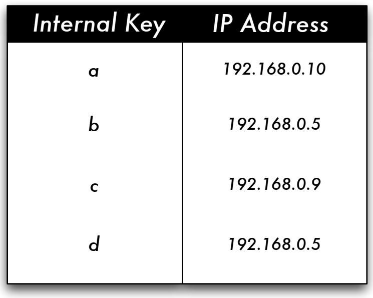
3.3 Node Lookup

Once we get the internal key for the request, we have to look up the corresponding host, which contains the requested key. All *key-groups* are distributed among all participating nodes. As a general rule, if we have several *key-groups* located at the same host, it is preferable that those *key-groups* form a closed range of the internal key-space (this is called *key-partition*) (e.g. it is preferred to have *key-groups* a, b, c, d on one node and not a, k, r, z. This will help prevent unnecessary jumps from one node to the other during range queries).

As mentioned above, depending on the size of the system, different approaches can be taken:

3.3.1 Small-Sized Systems

For smaller systems, a static routing table can be put in place, e.g.:



Internal Key	IP Address
a	192.168.0.10
b	192.168.0.5
c	192.168.0.9
d	192.168.0.5

Figure 22: Static Routing Table

3.3.2 Medium-Sized Systems

In a more dynamic environment, in which nodes are constantly added/removed, DNS-like approaches or other local-network methods like Zeroconf (also known as “Rendezvous”) [2, 3, 35] can be used. With these methods, every node broadcasts its internal key on the network, so others can easily find it. Zeroconf claims that their broadcast mechanism is resource limited and will not overload the network with unnecessary broadcast messages. Nevertheless, this mechanism only works on the IP broadcast scope and is therefore limited to the local network, as broadcast messages are not forwarded by a router to a foreign network.

3.3.3 Large-Sized Systems

Obviously for a large-scale systems, other mechanisms need to be considered, even non-centralized ones (e.g. for current Peer-to-Peer systems such as Gnutella [24]). In such systems, lookup becomes the most critical part of the system, as it is normally the slowest. Recently, there have been many projects working on a distributed lookup architecture (sometimes also called “Peer-to-Peer overlays” or “P2P overlays” for short) [15, 17, 23, 40, 41, 59, 66, 90]. Some of them could be used for our approach.

Depending on the underlying system we choose, additional optimizations can be implemented. E.g. sometimes the key-space is not uniformly distributed - there are some spots (“data-spots”) with a lot of data and some with no data at all (“data-holes”). In such a sparsely occupied key-range, not all *key-groups* need to be pre-allocated. They could be allocated as soon as some data fell into this *key-group*. This way, during an insertion of a new key, the system asks the nearest node to also add this new *key-group*. (E.g. if you already have *key-groups* a and b on node 1, now a new key, which would fall into *key-group* c, should be added. The system will ask node 1 to also hold *key-group* c.) This approach has two advantages:

1. The *key-groups* do not need to be pre-allocated, so use of the nodes is made more efficient.
2. Allows better scalability, as newly added nodes can efficiently be filled with new keys without having to migrate data from existing nodes.

To accomplish this feature, the underlying structure needs to support the routing of lookups for non-existent *key-groups* to the nearest existing *key-group*. (E.g. if *key-group* a and f exist, and then a lookup for *key-group* c is done, the underlying structure needs to route this request to either a or f. This only has to be done for range-queries. For exact queries, the system can immediately return, as the key does not exist).

3.4 Data Store

Each node has its own storage where all the keys of these *key-groups* are stored (see section 2.3). For efficiency reasons, each node only needs to have one single storage, and all key-ranges can be stored in the same place. As this storage also needs support for sorted access to keys, conventional b-tree (e.g. BERKELEYDB [65]) or highly concurrent b-link trees [39, 47, 63] can be used.

To make the system as flexible as possible and to allow further extensions, each key-value pair can have one or more pieces of additional metadata information, which is stored alongside the data. Such metadata information could be:

- Permission flags, so access to this data can be restricted to some users
- Time and date, transaction ID of last change
- How many times the data item has been accessed already
- Locks for transactions
- A flag if actual data is locally stored or stored on another machine of the network

This metadata is also used by the system itself for monitoring, development and tuning purposes. Beside having metadata and counters on data level, also other counters should be put in place on node level, e.g. the amount of network traffic coming in or out, how many queries the node already has answered, etc. This allows for monitoring of the whole system during testing, finding the best way to implement all its features. Later it can also be used for self-maintenance, e.g. to detect an overloaded system or cache frequently used data.

Each request coming to the node should have a “unique” transaction ID, so if the whole query is split in parallel and would reach a node twice - for whatever reason - the node will know that he already processed this query and does not need to do any further action. Those transaction IDs do not need to be stored for a long time, and can be removed after some time.

3.5 Replication

So far, we have not discussed replication of data in such a system, as there are several aspects and granularity to look at. E.g. replication could be used as an active/standby mechanism, so the replication is only used if the master is not available anymore, or in an active/active manner, where writing/reading could be done on either master or replica.

3.5.1 Active / Standby

This is the simplest approach and would only give the advantage of having a backup copy in case the master is not available anymore. Additionally, several replicas could be used to achieve even higher redundancy in case of failure. This method would not give any performance advantage of having multiple data sources for the same data, so access to its data could not be load-shared between master and replicas.

But the advantage is that this approach would not need to block write requests until all replicas could be updated, so this is rather simple to implement.

To include such mechanisms into our distributed data structure, a simple approach can be used: The original *key-group* acts as master and in case of an insertion/update of a key-value it propagates these changes to its first replica, and this first replica then propagates the changes along to the second replica (e.g. $a \rightarrow a' \rightarrow a''$). Alternatively, the master could also update all its replicas (e.g. $a \rightarrow a'$ and $a \rightarrow a''$). This could be done synchronously (the master waits to send back the response until all replicas have been updated, and then returns its status) or asynchronously (the master immediately returns its status, and updates its replicas later).

The master and its replica always send some tokens to probe their availability[8]. Alternatively, replica 2 could also only probe replica 1, in case of a hierarchical setup.

If the master is unavailable, the replicas will fail to send its probe tokens and the next available will be elected as master. If we have an underlying node-lookup

[8]Only the replicas have to send such a token. The master itself never needs to send a token to its replica, as he knows their status anyway, when an update fails.

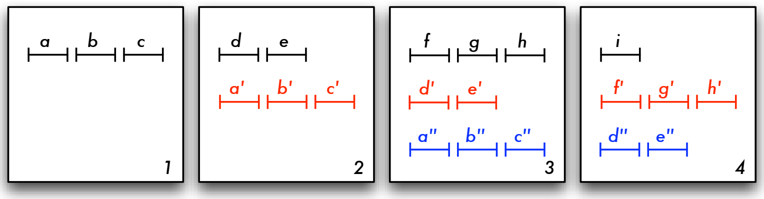


Figure 23: Replication: A master and two replicas.

architecture as described in section 3.3.3, the node-lookup will automatically find the correct replica, as it cannot find the original host and will just forward the query to the next available *key-group*. (E.g. if a lookup for *key-group* a fails because node 1 is not reachable, the underlying node-lookup architecture tries to find the next available *key-group*. In Figure 23 this would be *key-group* d located at node 2, where the first replica a' is also located.)

If the old master is available again, it will contact the current active replica to update its value and will then act again as normal master.

3.5.2 Active / Active

With Active/Standby, we gain redundancy in case of a node failure. With Active/Active, we could additionally gain load-sharing, although this method is more difficult to implement.

If some data that fall into the same *key-group* is "hot" and needs to be accessed from several clients at the same time, a potential bottleneck will occur, as only one single node is responsible for this data. Here replication could also help, as master and replicas would allow access to their data at the same time. Using this method, synchronization and locking between master and replica must be put in place, so all update requests must be atomic, and during a write/update, no client is allowed to access the same data for read/write. Depending on the transactional need, there are possibilities to tune the locking behavior, but nevertheless there is always some amount of administrative cost to update all replicas.

Generally we can assume that writing data will be slower, as all replicas need to be updated and during that time no other client is allowed to access the same data. Each time before updating the data, the master has to set locks on each replica and can then update the values.

Reading from those data can then be done in a load-balanced manner. The node-lookup mechanism will randomly (or any other load-balance mechanism can be used) forward the queries to either the master or his replicas.

Instead of using a separate key-space for all replicas, the node-lookup will just add a fixed prefix to its internal *key-group*. This prefix has to be large, so the chance of putting the replica on the same node as the master is minimized. Assume a prefix of 1000 and look at the master *key-group* with ID "35". Doing a lookup of "35" will return node 1. As *key-group* “36” is probably also on node 1, we now take our prefix of 1000 and then look up *key-group* “1035”. This one is most probably not on node 1, and on this node we could store our first replica. The second will be on the node with *key-group* “2035”, and so on.

4 Implementation of ERT (Extended Root Tree)

This section will describe the low-level and implementation details for the distributed data structure. It will introduce the main concepts of the application and explain why a specific approach has been taken. There are two main roles a user of the system can have:

1. **End User** - This is the user of the Application Programming Interface (API) of the system.
2. **Administrator** - This user configures the system and does administrative tasks (like adding new nodes and removing nodes).

It should be noted that in some circumstances, both roles could be executed by the same person.

According to section 2.2, the system will try to fulfill the requirements for a distributed system. The following list summarizes the goals achieved or those still open for the implementation of ERT:

- **Transparency** - The current design tries to be as transparent as possible to the end user. It is important to know that the administrator of the system knows that the system is distributed, as he needs to configure the nodes. On the other hand, the end user API (Application Programming Interface) abstracts the fact that the system is a distributed one. There are different kinds of transparencies:
  - LOCATION - The end user is unaware of the location of a resource; he just has an interface to get/put/delete values and does not know where the actual data is placed (local/remote).
  - MIGRATION - The end user is unaware if resources have been migrated or are migrating at the time the user sends requests. In addition, migration does not alter the resource name or anything else.
  - REPLICATION - The end user is unaware that data exists as multiple copies distributed among some nodes. Also, the end user has no influence to stop/start migration for a particular data item. Replication can be

turned on/off by the administrator of the system. In the current version, replication is not implemented, but the design has already taken replication into account.

  - FAILURE - Depending on whether replication is enabled/disabled, the end user will not see any disruption in service. The failure rate also depends on the degree of replication and the overall size of the system.
  - CONCURRENCY - The end user is unaware of other users of the system. The current implementation does not support transactions. E.g. every change an end user performs on the system is immediately available to other users; there is currently no BEGIN/COMMIT/ROLLBACK for transactions.
  - PARALLELISM - The end user is unaware of parallel execution of his requests.

- **Flexibility** - The system tries to be as flexible as possible with respect to performance. The three main points, which will give flexibility, are:

  - SERVICE CONCEPT - The system is designed around services: Each node has some services running, and these are responsible for certain requests (e.g. a data store service is responsible for storing/retrieving data). Those services are decoupled from each other and the only interfaces to the outside world are data items, which can be sent to it. Each service has to subscribe for keywords it is listening to (e.g. the data store service subscribes to the keywords “get”, “put” and “delete”).
    This allows extensions with new services, without having an impact on the design or structure of other parts of the system. (A detailed description can be found in section 4.1).
  - PLUGIN CONCEPT - Some services allow for further extension. E.g. the data store service allows for extensions with plugins to have different kinds of backend storage (file, database or others). The plugin structure has been designed to allow transparent access: The system itself does not know which exact plugin is in use; it just accesses it through a well-defined interface. (Detailed description can be found in section 4.2).
  - MESSAGE CONCEPT - The messages passed between the services are extensible as well. It defines a key-value concept, so each service can add new keys if it needs to. (A detailed description can be found in section 4.3).

- **Reliability / Availability / Redundancy** - For enhanced reliability and availability, the data can be replicated to other nodes. For write access (put, delete) the request is always routed to the master of the responsible data, and this master then transfers the data to its replicas. For read access (get), the requests are load-shared among all master/replicas to enhance performance. To circumvent the problem of possible inconsistency, the routing is done at several levels, and if for some reason a request is routed to the wrong node, this node will relay the request to the correct one. In current implementation, no replication has been implemented yet, but the design allows for adding this functionality later.

- **Performance -** To gain optimal performance, all services are decoupled and run inside their own thread of execution. Together with other services, the system can run multi-threaded and can achieve performance gains on multi-processor machines. Also, the system is designed to allow asynchronous execution. This means all requests sent from one service to another one (local or remote) do not wait for an immediate response and continue to work with other data until the response eventually comes in. This is a very important aspect to allow high performance processing.

- **Scalability -** The scalability is highly dependent on the actual implementation of the communication/directory component (see section 4.4.3). From the design point of view, the system can scale to a very large number of nodes, as the system has been designed to hold as little status information as possible on the node itself (e.g. the algorithms do not store status information for other nodes).

- **Concurrency / Parallelism -** As the system is already distributed to multiple nodes, it already does parallel processing of data. To allow concurrent access to the system, there are two possibilities:

  1. The end user attaches his own application to a node that also stores data.
  2. The end user attaches his own application to a new node that does not store any data.

  In the former case, the node is a member of the distributed data structure and also a member of the end user application. In the latter case, the node is only a member of the end user application and connects to the system over the network. In both cases, the end user has concurrent access to its data.

- **Security -** No security counter measurements are currently in place. This implies that a malicious user could harm the whole system, e.g. by sending administrative messages, which could result in bringing down the whole system. However, the current design allows for extending the system with security measurements (such as encryption and authentication of messages as an extension to the sending/receiving entity).

Figure 24 shows the overall design with all its components (Services, Plugins, Messages with their queues and supporting components).

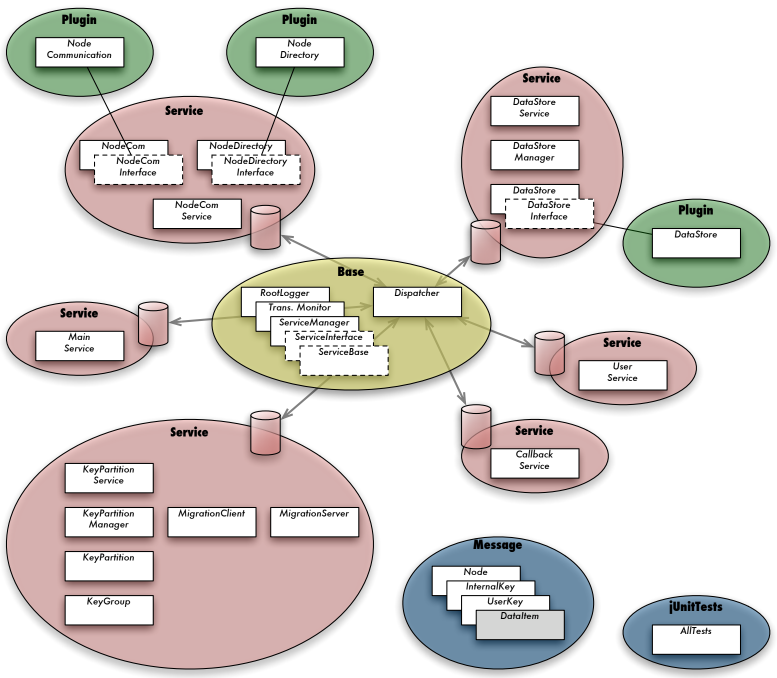


Figure 24: ERT System Overview

4.1 Service Concept

The system consists of several services, and each of them solves a well-defined part of the whole process. This allows for grouping functionality together and decoupling the system. It also allows for extending the system in the future with new functionality, which is not yet known or is application specific. Such extensions could be:

- A database application could implement a SQL [16] parser service, which would then use the underlying b-tree as a storage layer.
- A conversion service, which allows XML-based requests to be parsed and processed.
- A security service, which would grant/deny access on special criteria.

Generally such services only need be implemented if the standard user access, which only gives get/put/delete access to data, is not enough and access to lower-level parts are necessary to extend the functionality beyond the simple get/put/delete interface.

While services are not allowed to exchange data directly, there is a dispatcher, which will act as an intermediator between all services (see section 4.1.1 for details). The dispatcher decides where a specific request has to be forwarded; according to the supplied keywords, every service has to subscribe beforehand at the dispatcher. Using the service concept has the following advantages:

- **Decoupling -** Each service has a well-defined role and interface. A service can never call another service method directly. The interfaces between services are thread safe queues, and only the main component - the dispatcher - can put/get a workload from/to each service. Therefore, the only interface of a service is his queue, where he receives the workload from the dispatcher. All completed work is then sent again to the dispatcher, which then decides what to do with the work (e.g. send it to another service to process further).

- **Enhanced Performance -** Each service runs independently from the others. In the current implementation, each service runs within its own thread of execution, which scales with the number of CPUs of the underlying machine. Generally, it is not a good idea to have a large number of threads running at the same time, as each context switch of a thread will cause an overhead on the machine and will slow down the performance. However, with the current number of services, not many threads are running. Because the generation of the threads is done within the dispatcher, a further enhancement could be the introduction of a thread pool, where a predefined amount of threads are passed on to the busiest services.

- **Avoiding Threading Problems -** There is no limit how long a service needs to complete a certain task (e.g. the data store service takes longer to process, as it has to store data to disk, while other services have shorter processing time), so a multi-threaded system is necessary. Designing a well-functioning multi-threaded system is not an easy task (see section 2.5), as concurrent access to data is difficult to handle and the needed synchronization overhead will slow down performance.
  The service concept will solve this problem, as each service only runs within one single thread of execution. This way, the design and implementation of a service becomes easier, as it only has to be designed with a single threaded approach in mind, and no attention has to be paid to synchronization or concurrency issues. The only part of the system that is really multi-threaded is the dispatcher itself, as it will receive requests from all services and therefore has several threads of execution. But the dispatcher is a very small part of the system and consists only of a handful of lines of code and has been tested extensively (with respect to performance and accuracy).

- **Extensibility -** Each service has keywords assigned, which will act as routing indication for the dispatcher (e.g. the data store service will respond to “get”, “put” and “delete” requests). This allows for easy extension, as new services just need to assign new keywords for their functionality and can be put in place without changing other services or the core system. Services could even be added/removed during runtime, just by registering/unregistering the keywords at the dispatcher.

4.1.1 Structure

Because every service has some basic thing in common, they all extend from an abstract service base (ServiceBase class). This class already implements features needed for message handling and registering service keywords.

It makes no sense to have several instances of the same service running, therefore every service needs to be implemented as a singleton [20, 26].

For a structural overview, see Figure 25.

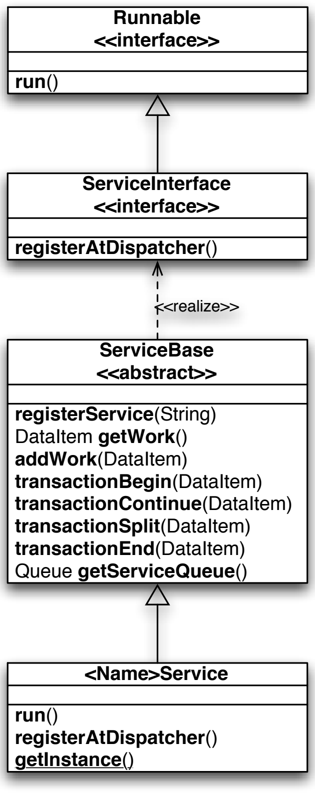


Figure 25: Class Diagram for the Service Concept

An actual implementation of a service already gets the functionality of the service base class:

- **run** - Must be implemented in the actual service and must contain the run loop of the service. Typically, it consists of a *getWork* command and will then process the request. After completion, it will process the next request and so on.

- **registerAtDispatcher** - Must be implemented in the actual service. During initialization, this method will be called to register all its keywords at the dispatcher. It normally contains a list of *registerService* requests for all assigned keywords.

- **registerService -** With the *registerAtDispatcher* method, a list of those *registerService* requests will subscribe the service keywords at the dispatcher. It is not required to register the keywords during service initialization - this can be done at any time - but it is recommended to do it at the beginning, because if the dispatcher receives a request for an unknown keyword, it will just discard it.

- **getWork -** Will take the oldest request from the service queue. If the queue is empty, the thread will go into a waiting state until some work is pushed into that queue.

- **addWork -** Will put some work into the dispatcher queue. This method can be used if the work is of the "shoot-and-forget" kind and does not need to be handled as a transaction; therefore, no answer needs to be sent back.

- **transactionBegin -** Will put some work into the dispatcher queue. This method can be used if the work is a request that needs a response. If you use this method, the system makes sure that the response is routed back to this service. This also stays true if the request needs to be sent to another node. The response will always find its way back to its originating service.

- **transactionContinue -** Will put some work into the dispatcher queue. This method can be used if the request is received by this service but needs further processing by another service. You can do processing on this chunk of data and then pass it to another service. The response to this request will then be sent directly to the service, which originated the actual transaction.

- **transactionSplit -** Will put some work into the dispatcher queue. This method is a special case of a *transactionContinue* method. It will also relay the work, but is able to relay it to several services at once. This is useful if you need to spread some information to several nodes (e.g. if a "put" request needs to be stored on the master and also on all replicas, then the "put" request can be split and directly sent to all required nodes). A transaction monitor at the dispatcher of the initiating transaction will make sure that the final response of that transaction is collected from all sources and, if all sources report "successful operation", it will send back a "successful operation" message to the initiating service. If only one source sends back an "unsuccessful operation" message, the whole transaction is sent back as "unsuccessful" to the originating service.

- **transactionEnd -** Will put some work into the dispatcher queue. This method needs to be used if it is the final response to a request and the response should be sent back to the originating service.

- **getInstance -** This must be implemented in the actual service and should return the only instance of this service. The service has to be implemented as a singelton [20, 26], and the *ServiceManager,* which will load the service, will use this method to get its service instance using standard JAVA introspection mechanisms. There is currently no other way in JAVA to force implementing a static method (such as forcing to implement a normal method using JAVA interfaces). In any case, you would get a runtime exception if the *ServiceManager* tried to load this service but could not find this *getInstance* method.

The complete processing of requests from one service to another is shown in the following Figure 26

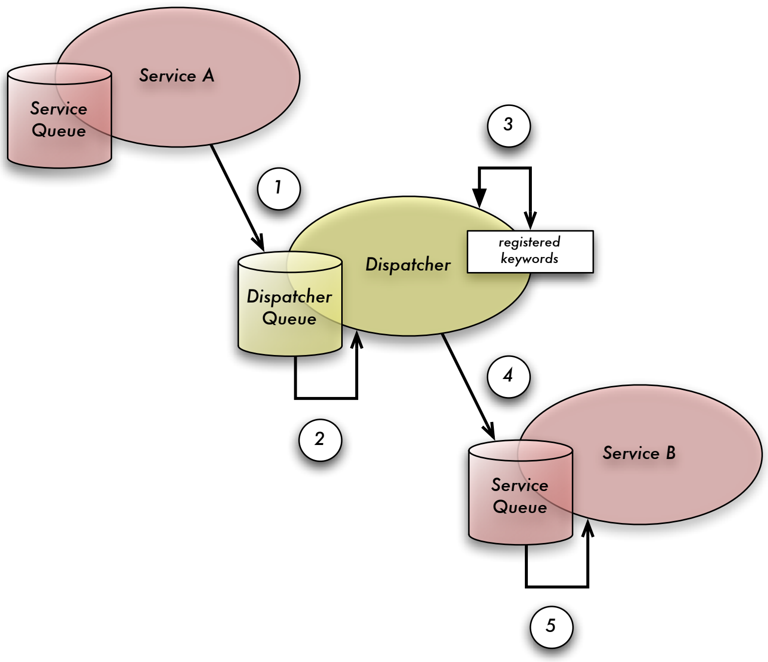


Figure 26: Processing of a Request

1. Service A sends a request to the dispatcher (either as *addWork, transactionBegin, transactionContinu*e or *transactionEnd*).
2. Dispatcher will pick up the oldest entry of its queue and start processing it.
3. Dispatcher will lookup in its registered keywords list to find where it has to put the request.
4. Dispatcher will find a suitable Service B and puts the request in its queue.
5. Service B will pick up the oldest entry of its queue and start processing it.

4.1.2 Usage

- **Creation -** The instance is created by the *ServiceManager*, which manages all services that should be loaded during application startup. This *ServiceManager* will use the *getInstance* method to get the single instance of this service.

```
static public synchronized DataStoreService getInstance() {
  if (null == instance) {
    instance = new DataStoreService();
    instance.dataStoreManager = DataStoreManager.getInstance();
    logger.info("DataStoreService started");
  }
  return instance;
}
```

Figure 27: Creation of a Service

- **Registering -** The *ServiceManager* uses *registerAtDispatcher* to register all keywords for this service at the dispatcher. The service is then set up to receive requests for those keywords. There is one special keyword, “default”, which can be implemented by one and only one service, which will then be chosen by the dispatcher, if no service for a particular keyword has been found.

```
public void registerAtDispatcher() {
  registerService("putdb");
  registerService("getdb");
  registerService("getnextdb");
  registerService("getprevdb");
  registerService("deletedb");
  registerService("getsizedb");
}
```

Figure 28: Registering at Dispatcher

- **Run Loop -** After the initialization phase, the *ServiceManager* will execute the run loop of the service, and the service will do the actual work.

```
public void run() {
  Thread.currentThread().setName("ERT.KeyPartitionService");
  while (true) {
    DataItem dataItem = getWork();
    String operation = dataItem.getOperationType();
    if (operation.equals("get")) {
      keyPartitionManager.get(dataItem);
    }
    ...
  }
}
```

Figure 29: Run Loop

- **Responding -** After or during the processing, one or more new requests/responses are sent to the dispatcher. This can be done inside the service itself, or within subcomponents of the actual service.

```
DataItem response = dataStoreManager.get(dataItem);
transactionEnd(response);
```

Figure 30: Responding

4.2 Plugin Concept

Besides extending the functionality with new services, some core services have the possibility to change their functionality. Currently there are two services which have a common plugin structure: *NodeCommunicationService* and *DataStoreService.* Extensions to services could be:

- New routing mechanisms for messages (e.g. Zeroconf based [35], P2P-overlay based [15, 17])
- New messaging methods for communication between system instances (e.g. JAVA Remote-Method-Invocation, XML based, ASN.1 based [6, 45])
- New storage mechanisms for actual data (e.g. file based, database based)

The main idea behind the plugin concept is to allow for exchanging parts in an already implemented service with other plugin components. Instead of building a complete new service from scratch, extending existing services with plugins reduces the development time. Using the plugin concept has the following advantages:

- **Reducing Development Efforts -** By re-using the already available service component with all its business logic, the time to develop and test a new service can be reduced, as only part of the plugin itself needs to be written and

tested. The plugin itself needs to implement a JAVA interface and has to conform to its method declaration. The plugin will then be loaded by the service itself as a delegate/proxy (see delegate/proxy pattern in [19, 26]). This means the service or the whole system does not know which particular plugin is in use.
Also, the plugin can use the already proven base classes, such as logging, concurrency packages and data structures.

- **Extensibility -** As you would use the service concept if you wanted to make structural changes to the whole system (e.g. by introducing new keywords, functionality), you can use the plugin concept to make smaller changes to already developed services.

- **Abstraction -** As the system itself is not aware of the actual plugin implementation, the whole system will benefit by the reduction of complexity that needs to be handled.

4.2.1 Structure

The actual structure is dependent on the service and the interface it provides for plugins, so the number and signature of the methods can change for each service. For a structural overview, see Figure 31.

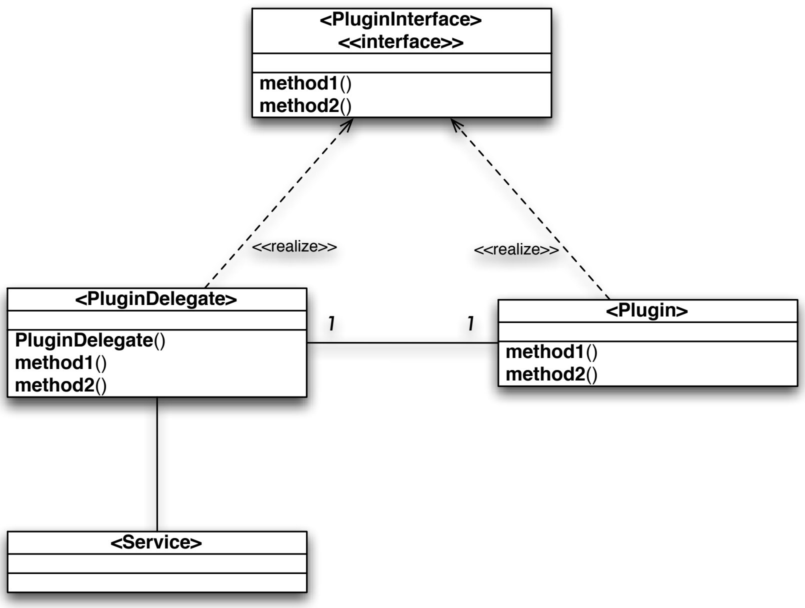


Figure 31: Class Diagram for the Plugin Concept

- **method1 / method2 -** The interface specifies which methods should be part of the *Plugin*. Those methods need to be implemented at the *Plugin*. The

PluginDelegate will also implement those methods, but only as an envelope to the actual plugin. This means the implementation of the *PluginDelegate* will use the same method in the *Plugin* instead and therefore acts as a proxy (see section 4.2.2 for details).

- **PluginDelegate -** During instantiation of the *PluginDelegate*, it will load the actual *Plugin* using JAVA introspection mechanisms. In the current design, the *Plugin* needs to have the same name as the *PluginDelegate*, and it needs to reside inside its own sub package (see JAVA code for examples).
  The *Service* will only access the *PluginDelegate* to actually reach a plugin. This abstracts the actual plugin implementation from the rest of the system.

4.2.2 Usage

- **Creation -** The *PluginDelegate* needs to load the actual implementation of the *Plugin*. This is done in the constructor of the delegate, as shown below. The delegate is then initialized (e.g. by setting properties, which are needed for setup).

```
static private String plugin = "berkeleyje";

public DataStore(String instanceId) {
  try {
    delegate = (DataStoreInterface)
                    (Class.forName("ch.nix.ert.store."
                    + plugin + ".DataStore").newInstance());
    delegate.setDatabaseId(instanceId);
  } catch (InstantiationException e) {
    e.printStackTrace();
  } catch (IllegalAccessException e) {
    e.printStackTrace();
  } catch (ClassNotFoundException e) {
         e.printStackTrace();
  }
}
```

Figure 32: Creation

- **Delegating -** If a method is accessed, the delegate acts as a proxy and forwards the request to the actual *Plugin*.

```
public DataItem getDataItem(DataItem data) {
        return delegate.getDataItem(data);
}
```

Figure 33: Delegating

- **Implementation -** The actual implementation of the functionality inside the

Plugin.

```
public DataItem getDataItem(DataItem data) {
  //extract key from data
  long key = data.getUserKey().getId();
  //check, if connection to database is already open
  if (myDatabase == null) {
    this.openDatabase();
  }
  try {
    // Create a pair of DatabaseEntry objects. theKey
    // is used to perform the search. theData is used
    // to store the data returned by the get() operation.
    DatabaseEntry theKey = new DatabaseEntry();
    DatabaseEntry theData = new DatabaseEntry();
...
```

Figure 34: Implementation

4.3 Message Concept

The current architecture is based on message passing. All services pass messages along to other services. Those can be on the same node or on a remote node. The basic idea of this message concept is to introduce a common way for intercommunication between services, which also allows for future extension. Such extensions could be:

- New yet unknown services need specific data, which needs to be transported over the already deployed network.
- In addition to the raw key-value data from users, other properties such as permission-related attributes and metadata might be introduced.
- New communication protocols (e.g. switching to XML, ASN.1 [6, 45]) could be introduced; this should not affect the implementation of the services.

As the future is unknown, the messaging should be decoupled as much as possible from the actual implemented services, so additional attributes could be added without affecting already developed services (backward, forward compatibility [75, 79]). Generally, using the message concept has the following advantages:

- **Backward Compatible -** New versions of ERT should understand old messages, as the messages themselves are based on a key-value mechanism. In newer versions, new keywords can be introduced, but the old keywords are still recognized. A system with a mixture of old and new systems is possible and in the case of a rolling update (an update with zero downtime) unavoidable.
- **Forward Compatible -** Old versions of ERT should understand new messages, again because the old key-values are still present in the new messages. Of

course, this has the limitation that features introduced in the new version are not available on the old system. However, because the message concept will still understand the grammar of the message, it will not alter the message itself, but just not react to new attributes. This is also important during a rolling update of a system, where it is unavoidable that at a certain time, multiple versions of the system are running at the same time.

- **Extensibility -** Because of the key-value nature (See section 4.3.1 for details), new attributes can be introduced at any time. Even when new attributes are introduced, the communication component is still capable of sending the messages, even if they contain new/unknown attributes. The same is true for services: Each service only looks at those attributes it is interested in and will not fail if new attributes are introduced.

- **Abstraction -** The structure of the actual message is hidden for the developer; he does not know how the message is built. The message itself is a lightweight object, which basically contains an array. The keys are integers and the values can be of any kind. Making the values as lightweight as possible, e.g. using only strings, integers or simple objects to speed up the processing speed, is encouraged. This is important, as each sending and storing activity will serialize the message object and serializing complex object structures can be slow.

4.3.1 Structure

The message concept is based on a class named *DataItem*, which consists of an array and the idea that every attribute is represented by a key-value pair:

- **Key -** The key is the representation of the attribute. It is of type *int* (integer) and in current implementation all keys are on the same level. There is no namespace. Every attribute is identified by its unique key. In the future, as the attributes might grow, a nested approach in which keys are grouped and nested together could be necessary.

- **Value -** The value is the attribute content. It is of type *Object* and therefore can be any kind of information. Having only simple objects is encouraged, as there is a performance penalty to convert complex objects to the actual binary representation needed for storing and communication between nodes.

To make it easier for the developer, all key-value attributes have accessor methods, which help setting/getting/deleting attributes. It is favorable to use these accessor methods, because using the array directly will always result in a return value of type *Object*, which would need to be cast to the needed object by hand afterwards. Using accessor methods does this for you.

For a structural overview, see Figure 35

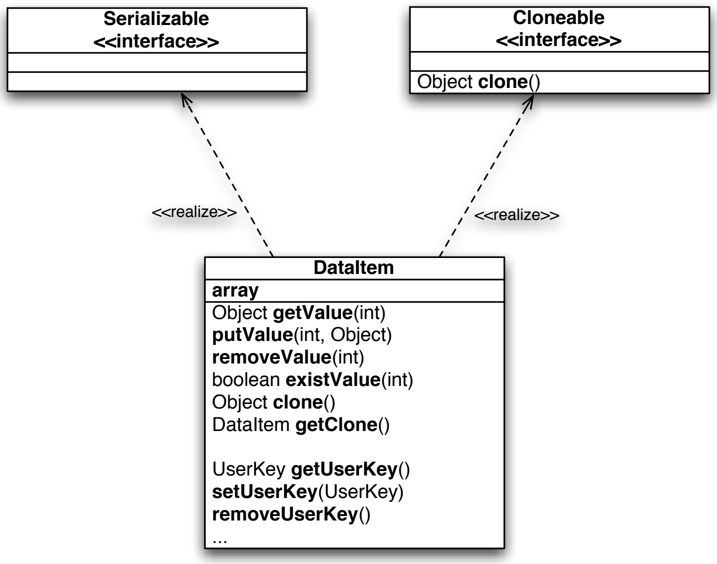


Figure 35: Class Diagram for the Message Concept

- **Serializable -** There is no method to implement for the *Serializable* interface. Nevertheless, *DataItem* needs to implement it, so this class can be automatically serialized for use in data store and communication layer as needed.

- **clone -** This allows cloning the object. Sometimes this is needed if you want to split the transaction (see section 4.1.1 for details) into multiple pieces and every piece contains the same *DataItem* (e.g. to replicate the very same data to several nodes). It is also needed if multiple references of the same *DataItem* are sent around inside the same node, as then multiple threads might modify this *DataItem* at the same time. Cloning can help in these circumstances. Generally, the *DataItem* is implemented in a thread-safe way, so even multiple threads could eventually modify the same *DataItem*. Nevertheless, some services may not allow the *DataItem* change at the same time by another service and therefore need to implement cloning.

- **getValue -** Gets a value for a given key. The key is always of type *int* and this method returns an object of type *Object*. Using accessor methods for the keys instead is encouraged, as they provide optimized versions on accessing the actual value.

- **putValue -** Assigns a value for a given key. The key is always of type *int* and value of type *Object*. Using accessor methods for the keys instead is encouraged, as they provide optimized versions on storing the value.

- **removeValue -** Removes a given key. The key is always of type *int*. Using accessor methods for most of the keys instead is encouraged.

- **existValue** - Checks for the existence of a certain key. The key is always of type *int* and will return a *boolean* (false/true).
- **getClone** - This is a more convenient method to clone a *DataItem*, as it already returns an object of type *DataItem* (as clone itself only returns a generic *Object*).
- **get / set / remove** - These are accessor methods for the most common used keys. Currently all used keys have such accessor methods and using them is strongly encouraged, as they already provide basic conversion to the final object type.

4.3.2 Usage

- **Getting Values** - There are two ways of getting values: One is by using the general method, and the other is by using the accessor methods.

```
Node originator = (Node) dataItem.getValue(
          OperationInterface.DIRECTORIGINATOR);
```

Figure 36: Getting Values - General

```
Node originator = dataItem.getDirectOriginator();
```

Figure 37: Getting Values - Accessor Method

- **Setting Values** - There are two ways of setting values: One is by using the general method, and the other is by using the accessor methods.

```
Node node = new Node();
dataItem.putValue(
          OperationInterface.DIRECTORIGINATOR, node);
```

Figure 38: Setting Values - General

```
Node node = new Node();
dataItem.setDirectOriginator(node);
```

Figure 39: Setting Values - Accessor Method

- **Removing Values -** There are two ways of removing/deleting values: One is by using the general method, and the other is by using the accessor methods.

```
dataItem.removeValue(OperationInterface.DIRECTORIGINATOR);
```

Figure 40: Removing Values - General

```
dataItem.removeDirectOriginator();
```

Figure 41: Removing Values - Accessor Method

4.4 Components

This section will describe the most important components, which form the ERT system.

4.4.1 Key Partition Service

This service is responsible for deciding if this node can handle an incoming request or if it should be relayed to another node. Furthermore, it handles all data migration requests and in future versions of ERT will also handle replication. In current implementation, the *Key Partition Service* listens to the following commands:

- **put -** Will first check to determine whether this node is responsible for this key-value pair. If so, it will store the key-value by using the *Data Store Service* (see section 4.4.2) method: putdb. If the *Key Partition Service* decides it is not responsible for this key-value pair, it will try to relay the request to the node it thinks is responsible for it.

- **get -** Will first check to determine whether this node is responsible for this request. If so, it will retrieve the value for this key by using the *Data Store Service* (see section 4.4.2) method: getdb. If the *Key Partition Service* decides it is not responsible for this key-value pair, it will try to relay the request to the node it thinks is responsible for it.

- **getnext -** Will first check to determine whether this node is responsible for this request. If so, it will retrieve the value for the next key by using the *Data Store Service* (see section 4.4.2) method: getnext. If the *Key Partition Service* decides it is not responsible for this key-value pair, it will try to relay the request to the node it thinks is responsible for it.

- **getprev -** Will first check to determine whether this node is responsible for this request. If so, it will retrieve the value for the previous key by using the

Data Store Service (see section 4.4.2) method: getprev. If the *Key Partition Service* decides it is not responsible for this key-value pair, it will try to relay the request to the node it thinks is responsible for it.

- **delete -** Will first check to determine whether this node is responsible for this request. If so, it will delete the corresponding key-value pair by using the *Data Store Service* (see section 4.4.2) method: deletedb. If the *Key Partition Service* decides it is not responsible for this key-value pair, it will try to relay the request to the node it thinks is responsible for it.

- **nodeload -** Will return the current load of this node. The return value is a number between 0 and 1. 0 means: Node is idle and 1 means: Node is under heavy load.

- **firstnode -** During startup phase, this command indicates that this node is the first node on the whole network. This implies that this node does not try to probe for other nodes during startup and will assign the whole data range (all possible keys) to itself.

- **join -** During startup phase, this command indicates that this node should be part of an already existing network. It will then probe some random nodes to find out which node currently has the highest load, and will then ask this node to migrate some data from it.

- **migrationprobe -** Is similar to the join command. During runtime, the system will probe its neighboring nodes (those nodes which are responsible for the next higher or lower data range) every few minutes to see if they have comparable load. If it finds that the neighbors are more heavy loaded compared to its own load factor, it will start a migration. This way it can be assured that the overall load is evenly distributed among all participating system nodes.

- **migrationstart -** This request will start a migration (see section 4.5.4). As each node can only be part of one migration at a time, it will first check to determine whether this node is already doing another migration. If not, it will respond with the data range to migrate.

- **getmigration -** This request will actually transfer the data from one node to the other.

- **migrationend -** If the migration is finished and all necessary data has been transferred to the other node, this request will indicate that the data has been successfully migrated. The two participating nodes will then adjust their *Node Directories* and carry out other administrative actions.

4.4.1.1 Responsibility The *Key Partition Service* will always check to determine whether the local node can handle the incoming request. If not, it tries to relay the request to another node, which it thinks is responsible for. Of course, the newly chosen node again checks the responsibility and will further relay the request until the correct node is found. Under normal circumstances, in which the node directory is up-to-date, no relaying is necessary. Relaying only happens if one node directory has temporary outdated information.

4.4.1.2 Migration The *Key Partition Service* is responsible for data balancing and will migrate the data automatically if necessary. During startup of a new node, it will probe a few random nodes to find out which is the most loaded node and will then start migrating some data to the new node. After this first migration (the new node joins the network), the nodes will continuously watch their neighboring nodes, and, if their load starts to climb, request a migration to relieve overloaded nodes. See section 4.5.4 for more details.

4.4.1.3 Replication In current version of ERT, no replication feature has been implemented. Replication already has been prepared in the design of the application and should be straightforward to implement, as the logic should be about the same as for migration.

4.4.2 Data Store Service

This service is responsible for actually storing the data permanently, e.g. to disk. It uses the *Plugin Concept* described above to allow replacing the actual data store with another implementation. The following commands are supported:

- **putdb** - Will store a tuple (key-value) to the storage. The outcome of this operation will be sent back by returning true/false.
- **putdbnoack** - Will store a tuple (key-value) to the storage. There is no response sent back, so there is no guarantee that the tuple was successfully stored.
- **getdb** - Will retrieve a tuple (key-value) from the storage. The response will contain the complete key-value pair.
- **getnextdb** - For a given key, it will respond with the next available key-value pair. E.g. if your database contains the following key-value pairs: (1→DATA1), (4→DATA4), (12→DATA12), a *getnextdb* request for key 1 will return the tuple (4→DATA4).
- **getprevdb** - For a given key, it will respond with the previous available key-value pair. E.g. if your storage contains the following key-value pairs: (1→DATA1), (4→DATA4), (12→DATA12), a *getprevdb* request for key 4 will return the tuple (1→DATA1).
- **deletedb** - Will delete a tuple (key-value) from the storage. The outcome of this operation will be sent back by returning true/false.

- **getsizedb -** Will return the current amount of tuples (key-value pairs) of the datastore.

In the current implementation, only a BerkeleyDB Java Edition [65] storage is available, but other plugins can be built and used instead.

4.4.2.1 BerkeleyDB

BerkeleyDB [65] is an implementation of a balanced tree (b-tree) and is available from Sleepycat Software. Two different implementations are available: BerkeleyDB and BerkeleyDB Java Edition. The former is written in C-language and is a proven product, used in multiple commercial and non-commercial applications. There are Java bindings, through the Java Native Interface (JNI), so Java applications can use the C version of this implementation. The newer BerkeleyDB Java Edition is a re-write of the whole data-store in Java language. There is no need to install any C libraries anymore in the system, as this version is now 100% Java.

Some general features of BerkeleyDB are (taken from [65]):

- **Open Source -** The sources for BerkeleyDB are available for download and BerkeleyDB can be integrated into your own open source software for free, although if you want to include it in your own commercial applications (where you don't want to make the sources available), a license has to be bought.
- **Small Footprint -** In contrast to full-blown databases (e.g. Oracle, DB2), BerkeleyDB only needs very limited space on disk for the executable, and the memory footprint of the loaded executable is very low.
- **Thread-Safe Library -** BerkeleyDB is completely thread-safe; therefore, multiple threads can be read/written to the database at the same time. This feature is currently not used, as this *Data Store Service* only runs within a single thread.
- **File System Integration -** The database itself is stored on a normal file system. There is no need for a special partitioned hard disk or other storage medium.
- **Large Databases -** BerkeleyDB supports databases up to a size of 256 terabytes and key-values up to 4 gigabytes in size.
- **Transaction -** BerkeleyDB allows transaction. This means that several operations can be performed on the database and they will only be visible to others if the whole transaction is executed. Otherwise, BerkeleyDB will roll back to the state it was in before the transaction started.
- **Check Points -** If BerkeleyDB or its hosting application crashes, BerkeleyDB automatically brings the database into a consistent state after the next startup.
- **Hot Backup -** A backup of the database can be done during normal operation. There is no need to shut down the application to do a backup of the data.

4.4.3 Node Communication / Directory Service

This service is responsible for sending requests to other nodes and consists of two main components:

1. **Node Communication -** This part is responsible for sending direct requests to nodes. Such requests can be administrative in nature (e.g. status messages) or application related (e.g. get/put/delete requests). The general dispatcher (see section 4.1) tries to route the messages, and if he can't find the required service on the local node, he will put it on the *Node Communication Service* and this service will then send out the request to the correct node.

2. **Node Directory -** As the *Node Communication Service* needs to know which node is responsible for which data range, the *Node Directory* maintains a list (this list can be real or virtual; e.g. for smaller systems, the list could be stored on each node. For larger systems, the list can be distributed over all or parts of the nodes) and just responds to requests like "give me the IP of the node which is responsible for key a".

Both components are implemented following the *Plugin Concept*. Depending on the needs, both components (communication and directory) could be implemented as one single large component if needed, or separated.

The following commands are supported:

- **send -** Will send out this data item. Depending on the routing indication of this data item, the service will first do a lookup in its *Node Directory* or if an IP is already supplied in the data item, it will send it directly to the corresponding node.

- **default -** The dispatcher (see section 4.1) will automatically send all requests for which it does not find a local responsible, to a service which subscribes the "default" keyword. The *Node Communication Service* will then receive all requests that cannot be handled by the local instance and therefore need to be transferred to another node.

- **publish -** If there is a new key-group to be added to the local instance (so the local instance will get a new data range assigned), the *Node Directory* needs to be called with this command, to update its directory. It will then pass this information to its *Node Directory* plugin, which will carry out the necessary steps to publish this information, e.g. to send a broadcast to all other nodes, telling them that this node is now responsible for this new data range.

- **unpublish -** If the current node is no longer responsible for a certain data range, it needs to send this message to the *Node Directory*. It will then pass on this information to its *Node Directory* plugin, which will carry out the necessary steps to unpublish this information.

In the current implementation, two communication plugins (RMI, NETTY2) and one directory plugin (JMDNS) are available.

4.4.3.1 RMI This communication implementation on top of the standard JAVA Remote Method Invocation (RMI) [67] is a rather slow but straightforward method for sending messages from one node to another. It basically uses two main parts for sending messages:

- **Serialization -** A RMI message is basically a full JAVA object that is being sent over the network. Before the sending can be done, RMI does a serialization of the object into a byte stream. That means all relevant data of an object (instance variables, static variables) are converted to a flat byte stream, including administrative data to de-serialize the data stream afterwards. RMI automatically uses an internal serialization mechanism, so the application developer does not need to write its own. He can, however, by implementing special serialization methods (e.g. for better serialization performance).

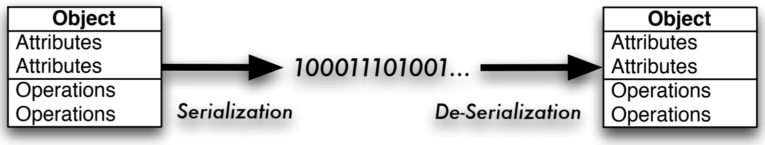


Figure 42: Serialization, De-Serialization

- **Sending -** After a byte stream has been generated through serialization, RMI tries to deliver it over the network.

The main use of this plugin is to have a running communication layer very quickly, as most implementation details are already included in the RMI library. As a downside, the performance is quite low. A test between two machines showed that this mechanism allows around 80 messages per second to be sent over the network. The main bottleneck is the serialization mechanism, which takes the most time.

4.4.3.2 NETTY2 This communication implementation on top of NETTY2 [46] uses the new JAVA NIO-package (New I/O) [68], which claims to support superior network performance. The main difference between the old networking libraries and the new are: The old was using one thread for each listening network socket, while the new is only using a user-specified number of threads for all network sockets. This is especially useful for servers, which need to listen to many concurrent network channels from different nodes.

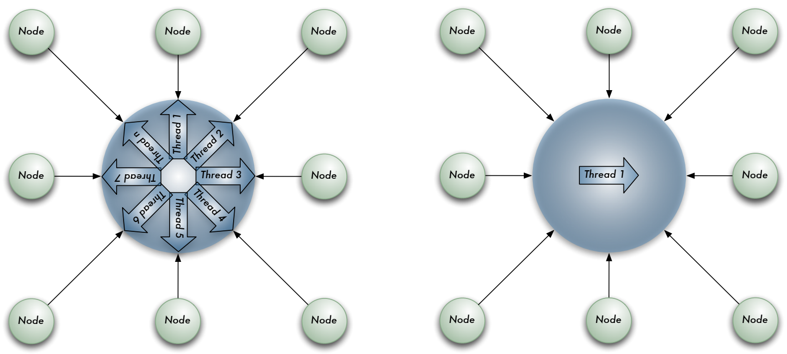


Figure 43: Old vs. New JAVA Networking

The main advantage of this plugin is its better performance. A test between two machines showed that this mechanism allows around 700 messages per second to be sent over the network (around 10 times the performance of RMI). As NETTY2 or NIO are only libraries for doing raw network transfer, the downside is that the application developer needs to build its own serialization/de-serialization mechanism. This has been done for ERT, which implements an efficient way to encode/decode messages in a way, similar to ASN.1 [6, 45].

4.4.3.3 JMDNS This is a directory implementation on top of JMDNS [71], which conforms to the Zeroconf/Rendezvous [2, 35] specification.

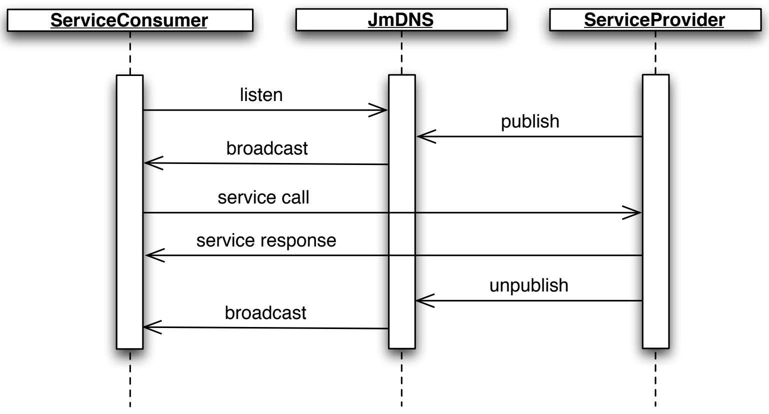


Figure 44: JMDNS Message Flow

It is responsible for maintaining the directory of ERT and holds a list, which part of the data is stored on which node. It implements a broadcasting mechanism to transmit changes to its directory to other nodes. As changes to the data range do not occur very often, this mechanism does not need to be very efficient. Also, the problem of propagating time is solved, as the system works even if one node has outdated information about the other nodes. In this case, the lookup just takes a bit longer, as the node that received the request and does not feel responsible will just relay it to the correct one.

4.4.4 Callback Service

This service is responsible for automatic callbacks within the ERT application. Normally, if you need periodic execution of a code (e.g. to collect statistic information every minute, clean-up), a separate thread that will execute the code from time to time is introduced. This can lead to poor performance, as every such situation would need a separate thread (e.g. if you need to update statistical information every minute, clean up every 10 minutes and poll a server every 5 seconds). *Callback Service* tries to outsource this mechanism to its own service. Additionally the timely execution is always done over the normal service interface, so the callback service never executes code directly; it just puts the callback into the service queue, which requested the callback before. This way, every service can still run within one single thread (see section 4.1), and no synchronization overhead needs to be put in place.

There are two types of callbacks:

- **calllater** - The callback is executed only once after a specified amount of time.
- **callcontinous** - The callback is executed at every specified time (e.g. every minute).

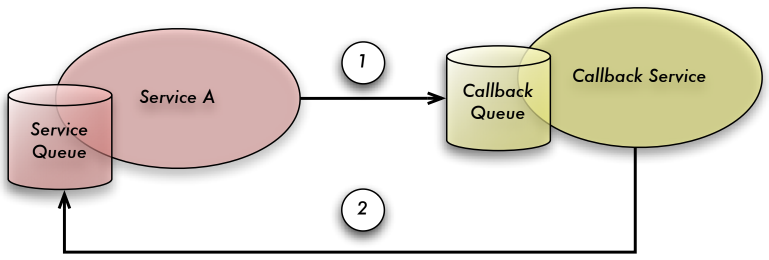


Figure 45: Callback Service

1. Service requests a callback (e.g. a continuous callback)
2. After the time is up, the *Callback Service* puts the original request into the Service Queue[9]

[9]For simplicity, the role of the dispatcher is not shown here. All requests for *Callback Service* or *Service A* normally go through the dispatcher.

4.4.5 User Service

This service is part of the user application; the user of ERT will interact only with this *User Service*. It supports wrapper methods for commonly used functions and will hide the complexity of a distributed system. The methods provided by the *User Service* are:

- **put(key, value) -** Will store the key-value pair and will not block further execution. Whether the request can be executed successfully or not, it will be returned at a later stage using an asynchronous response.
- **putSync(key, value) -** Will store the key-value pair and will block until a response (successful/unsuccessful) is returned by the ERT system.
- **get(key) -** Will retrieve a value for this given key. The request is executed asynchronously, and the response is sent back at a later stage.
- **getSync(key) -** Will retrieve a value for this given key and will block until a response (the value or a status message indicating a failure) is returned.
- **getNext(key) -** Will retrieve the next value for this given key. The request is executed asynchronously, and the response is sent back at a later stage.
- **getNextSync(key) -** Will retrieve the next value for this given key and will block until a response (the value or a status message indicating a failure) is returned.
- **getPrev(key) -** Will retrieve the previous value for this given key. The request is executed asynchronously, and the response is sent back at a later stage.
- **getPrevSync(key) -** Will retrieve the previous value for this given key and will block until a response (the value or a status message indicating a failure) is returned.
- **delete(key)** -Will delete the value associated with this key and will not block further execution. Whether the request can be executed successfully or not, it will be returned at a later stage using an asynchronous response.
- **deleteSync(key) -** Will delete the value associated with this key and will block until a response (successful/unsuccessful) is returned by the ERT system.

The only interface between the user application and ERT will be the interface described above, so a user application never uses other services or other parts of the ERT system.

4.4.6 Main Service

The only responsibility of the *Main Service* is to initialize the system and listen to application requests (such as shutdown, startup, dumping version information).

4.5 Message Flow

This section will describe the most common message flows of ERT. For simplicity, the role of the dispatcher is omitted.

4.5.1 Get Requests

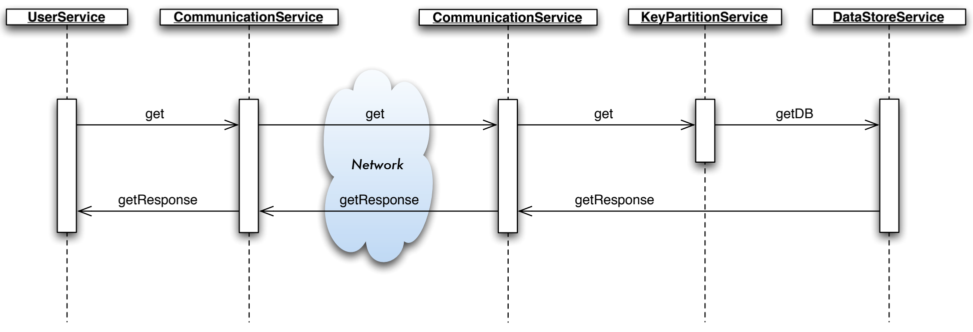


Figure 46: Get Request

The *UserService* will initiate a get request (*TransactionBegin*) and if the request can't already be answered at the current node (e.g. because the node does not contain any data, as it is not part of the storing network and only acts as a user service, or the data is not part of its own data partition), the *NodeCommunicationService* will find the correct node which is the holder of the requested data and will send the request there. If for some reason the *NodeCommunicationService* has outdated information and therefore sends the request to the wrong node, the receiving node will relay the message to the correct one.

The *DataStoreService* will answer the request and end the transaction (*TransactionEnd*) and the transaction mechanism will then route the response to the node that originated it.

Additionally there are other similar get requests not shown in the message flow chart. There are also the *getNext/getPrev* commands, which work similarly to the normal *get* command.

4.5.2 Put Requests

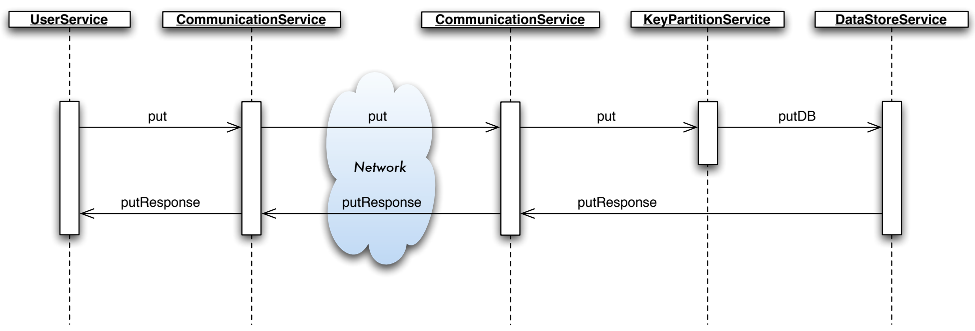


Figure 47: Put Request

Put requests use the same procedure as get requests. A put request normally has no return value. In this architecture, a put request will always return, whether the request has been successfully executed or not. This is important, because in a highly distributed system with unpredictable network latency, it is not certain that the put request can be processed and stored successfully.

4.5.3 Delete Requests

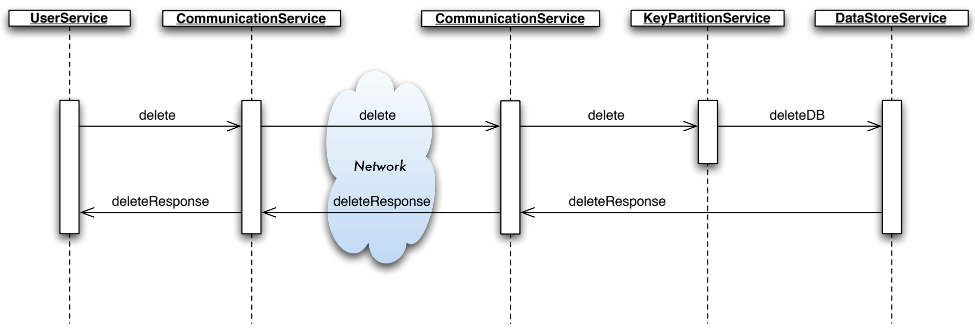


Figure 48: Delete Request

Works similarly to the put request above. It will also return whether the delete request can be executed successfully or not.

4.5.4 Migration Requests

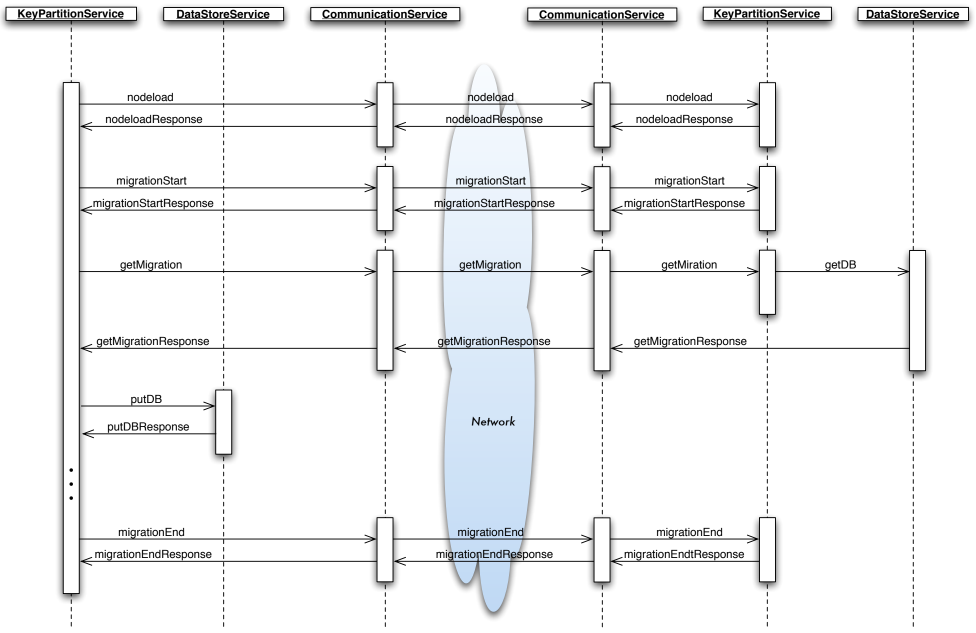


Figure 49: Migration Request

Migration has the most complex message flow and was designed to be as fault tolerant as possible. In addition to the mechanism pictured above, different timeout mechanisms have been put in place so the whole migration can be rolled back if something happens. It is interesting to note that in case of a migration roll-back, no data have to be moved again, as during migration no data is being deleted at the original node and all write/delete requests during migration update both parties simultaneously.

The whole migration process starts by probing the neighboring node at certain time intervals with a node load request. If the node load of the neighboring node is much higher then the local load factor, the node that has initiated the node load request will start a migration request. The other party will respond with a migration request response, which contains the exact amount of data that could be migrated, including the lower/higher boundary of the data to migrate (e.g. range $[p, t]$). The migration initiator will then check the range and choose a sub/whole-range of it and start requesting the data (*getMigration*). Once all data have been transferred, the initiator will end the transaction by sending a *migrationEnd* message.

Note that there is no deletion of data. Even if the migration has been successfully terminated, the data are not deleted immediately. There is a background task running on each node, which will take care of deleting all unnecessary data after some time.

4.5.5 Administrative Requests

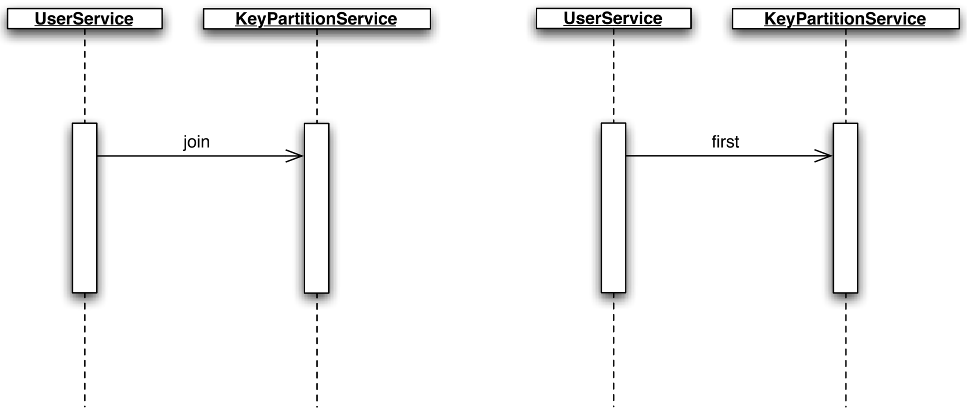


Figure 50: Administrative Requests

The only administrative requests at the moment are requests to join a running network, and for the first node, there is a request to tell the node that it is the first node on the whole network.

5 Evaluation

This section will provide an overview of the tests done with the prototype and will list some issues found or things which have worked out well.

5.1 Benchmark

Benchmarking has been done at the student computer lab at the IFI (Institute for Informatics, University of Zurich), running on multiple machines. All machines have the same hardware and software setup (see CD-ROM for a detailed machine profile, including all installed software and hardware).

- **Hardware -** PowerMac G4, dual CPU, each running at 450 MHz with 256-640 MB of RAM.
- **Software -** MacOSX 10.3.5 running JAVA version 1.4.2_05-141.3 and HotSpot client VM 1.4.2-38. ERT was running in version 165 and was configured to use the NETTY2 (see section 4.4.3.2) communication layer.
- **Network -** All machines are connected using switches on a 100 Mbit/s (Fast Ethernet) network.

To perform the tests, the machines are divided into three roles:

- **Collector -** A single machine, who will collect all the data received from the load generators and will calculate the overall load.
- **Load Generator -** Those machines generate the load to the test machines. They run the same ERT software, but do not perform any storage on their own. They just send out requests, e.g. a *put(key, value)*, and wait for their responses. The number of responses per second is sent to the collector every 10 seconds. This figure is used to calculate the overall performance of the whole system. It is important to note that not the amount of requests sent out per second is recorded (as this is much higher), but only the amount of responses returned.

- **Test Machine -** These machines process the requests received from the load generator and send back the result. There is no measuring of the processed number of requests, as this is done solely at the load generators.

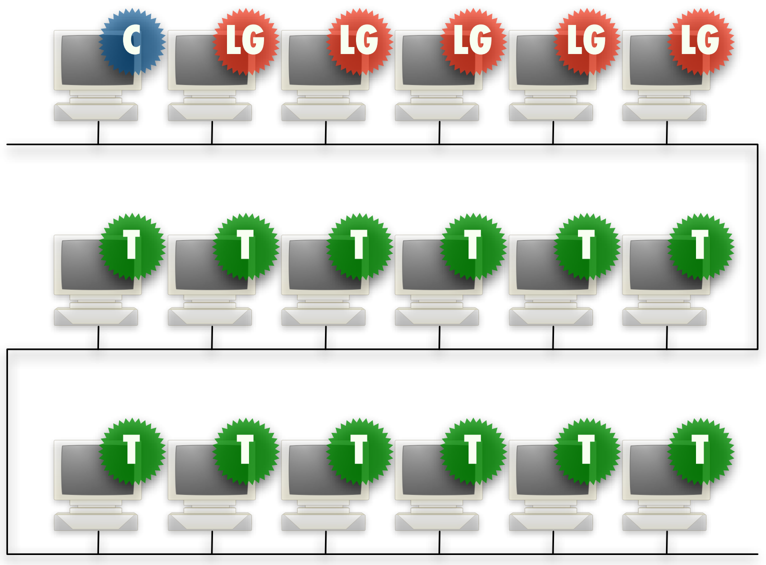


Figure 51: Benchmark Setup: Collector (C), Load Generators (LG) and Test Machines (T)

In the benchmark printed below, 1 collector, 20 load generators and up to 42 test machines are used. The load of all 20 load generators is added up to get the overall load.

Execution of the benchmark has been automated and is always done in the same manner:

1. The database on each node is cleared.
2. ERT is started on all test machines.
3. Waiting until the test machines are stabilized and all data-ranges have been distributed equally to all nodes.
4. Collector is started and is ready to receive results.
5. Load generators are all started simultaneously and start sending requests. To prevent overloading the system, the load generators start with a small load and will increase their load every second, until the maximum possible throughput is reached.
6. This setup runs for a few minutes and is then stopped.

7. The results from the collector are noted down. Always the highest three numbers in a row are averaged and used for plotting the data below.

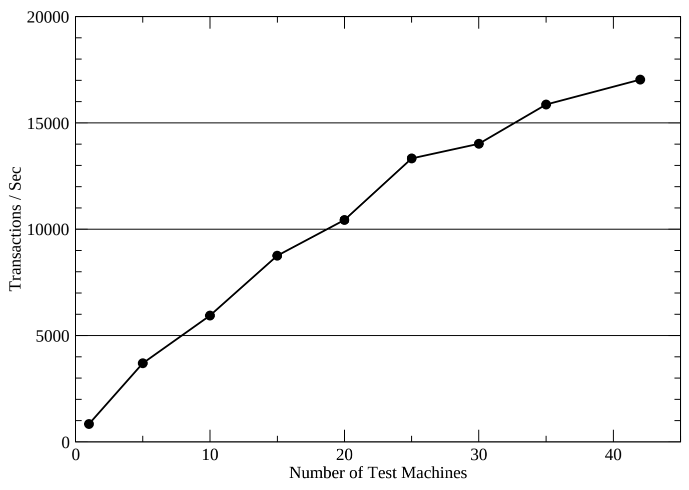


Figure 52: Benchmark: Random put, 10 bytes of data

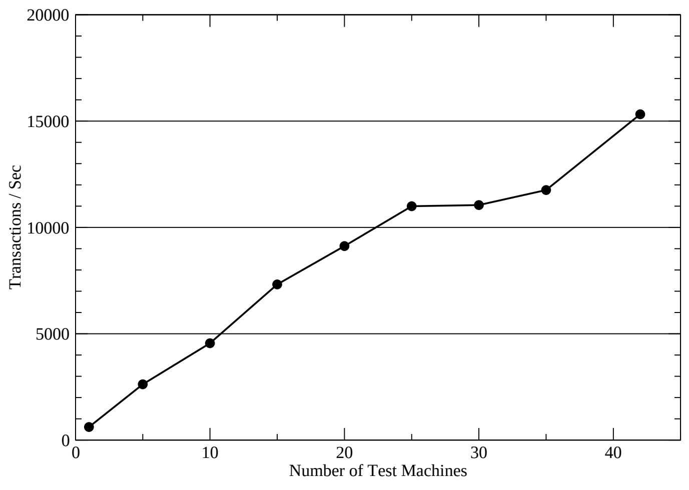


Figure 53: Benchmark: Random put, 1024 bytes of data

As shown in the benchmark result, ERT will scale pretty well as the number of nodes increases - although it did not scale entirely linear, this could be related to network stalling, as all participating machines share the same physical network connection.

It should be noted, that each transaction consists of two messages being sent: One request and one response, so the number of messages processed is twice the number as shown above.

Benchmarking was only done for "put" requests and not for the others like "get", "delete", as all requests have the same characteristics (all transactions are based on two messages being sent, see section 4.5), with the only difference that the destination node will either store, retrieve or delete data from its internal database. The "put" request is the most time consuming request anyway, as the underlying storage architecture (BERKELEYDB [65]) needs to allocate new disk-space and handles all insertions into its internal b-tree. Beside that, in current testing setup, all "put" requests have been executed in a random manner, to retrieve those values afterwards a lot of "get" requests would return an empty result, if no key-value pair for this particular "get" request is found.

5.2 Strengths / Weaknesses

During testing and developing, the following strengths and weaknesses were identified:

- **Strengths**
  - The complexity of adding new nodes to the network is very low, as there is not much configuration to be done. Basically, there are two types of nodes: The first one creates a new ERT network and needs to be started with a "first" flag. Then all other nodes can be added by supplying a "join" flag, and will then automatically try to join the existing network and assume responsibility for their own data range.
  - The overall performance of the system is good, although no intensive performance tuning has been done so far.
  - The system uses both CPUs efficiently, as the load of the system climbed almost to the maximum of 200% CPU utilization.
- **Weaknesses**
  - A client who wants to access the distributed data needs to open a network connection to all those nodes where the specific data is located. If a client needs to access all available data (e.g. by randomly accessing all of them), it potentially will open a network connection to all participating nodes on the network. This can cause problems if the network is very large (e.g. thousands of nodes).

- The new JAVA networking library (NIO), which is used in the NETTY2 plugin, as described in section 4.4.3.2, offers great performance, but it was added to the JAVA library only very recently, as of JAVA version 1.4. It seems not all vendors of JAVA have a robust implementation of it; e.g. the JAVA version included in Apple's MacOSX suffers from stability problems, which can cause the whole machine to crash. There is a workaround, but this decreases overall performance a bit. A bug report has been filed at Apple (Bug-ID 3866318), so this will hopefully be resolved later.
- Transferring large amounts of data, which can happen if the key-value pairs are very large in size, will decrease the overall performance of the system. As in a distributed system, the networking performance is usually the bottleneck. Sending large chunks of data will occupy the network, so other concurrent access to it will suffer. Also, sending large pieces of data will increase the memory footprint of the application, as all the data needs to be kept in the memory at some time.
- Migration of data under load is currently not very efficient and still has some bugs, which was an additional shortcoming during benchmarking.

6 Conclusion

This section will provide a short summary of the work and give an outlook.

6.1 Introduction

Current problems in data storage are the growing amount of data, the increasing level of access to those data and the need for finding the correct data. This leads to the fact that today's systems are built on distributed foundations and a trend towards more structured data. While there are some solutions for distributing data, most of them are done at a very high level.

This work will introduce a new low-level approach to distributing data - ERT (Extended Root Tree) - which is based on a distributed sorted key-value architecture. It can be used as a building block for higher-level applications as a storage tier.

6.2 Distributed Data Storage

In today's world, we see tendencies that indicate the amount of data is growing. More than 5 terabytes of data are produced every year. Moreover, companies, as well as private individuals, have data distributed to many places (e.g. company computer and personal laptop); so finding data is becoming increasingly difficult. A solution for finding the correct data is to structure it using several mechanisms. Depending on the use-case and application, low-level structures, such as key-value based algorithms, are used, or more high-level approaches, such as storing the data into databases, are chosen.

To discuss distributed data storage, we first need to define what characteristics a general distributed system has and then build a system, which should fulfill those requirements.

For a better understanding of how others have built distributed data storage systems, an overview of the large database vendors (Oracle, IBM, Microsoft) is given, and it shows that they all do the data distribution at a very high level. Further general research projects that focus on low-level distribution of data are discussed, and there we find two main directions: Smaller systems allow for sorted key-value access but most larger systems can only handle unsorted key-value access.

Regardless, for monolithic or distributed systems, they need to store data. As each machine only has direct access to its own hardware, the actual low-level storing is always done locally and a brief overview of such low-level storage is given. It shows that most systems implement a balanced tree (b-tree)-like structure for low-level storage.

Another problem in distributed systems is performance aspects: As distributed systems need communication between them, optimizing the network traffic is a key element of a successful system. Another one is asynchronous processing of requests to enable higher overall throughput of the system.

6.3 Overview of ERT

The main idea behind ERT is to split the data range into sub ranges and assign each computer on the network one or multiple of such sub ranges. The administration of the sub ranges (e.g. where to find a specific sub range) is done inside a distributed directory. The ERT approach does not favor a specific directory, so depending on the needs and size of the system, different such directory implementations can be used. For smaller systems, static directories could be used, for medium-sized systems, broadcast mechanisms and for very large systems, P2P (Peer-To-Peer)-like systems. The actual storing of key-value pairs on each node is done using a conventional balanced tree. Also, here ERT does not favor a specific implementation. Again, depending on the needs, different such balanced tree-like structures can be used.

6.4 Implementation of ERT

The prototype is being implemented in JAVA and basically consists of three main concepts: The service concept, which is responsible for decoupling the whole system into subcomponents, the plugin concept, which is responsible for abstracting the actual implementation from the services and the messaging concept, which is responsible for all communication between and inside all components.

6.5 Evaluation

Benchmarking showed that the system scales well as the number of nodes grow. There are bottlenecks, however, as networking bandwidth can limit scalability.

6.6 Outlook

The current prototype is suitable for distributed low-level data storage and is fast enough depending on the use case. Testing on how big the system can scale is still needed. The network interface could then become a bottleneck limiting further scalability. Compared to a monolithic system, the performance improvements in this distributed data structures only has an effect after an initial size is reached. Therefore, a system consisting of only two machines is slower than a monolithic system.

While a first prototype to explore the possibilities could be built, there is plenty of room for expansion:

- **Replication -** The replication feature is not fully implemented yet, but could be necessary for a fully functional service.
- **Large Data -** Before storing any data, the data itself needs to be held in the application memory for pre-processing. This can result in a huge memory footprint if the data is very large (e.g. big key-value pairs). An improvement could be to start pre-processing during receipt of messages, so the whole data flow then would look more like data streaming: While receiving data, the part already received is already pre-processed on-the-fly and stored to the data store.
- **More Testing -** The algorithms work well, but generally the implementation could be made more stable. In particular, JAVA exceptions are in most cases only caught but not really handled. There is room for improvement to optimize unexpected events (e.g. network connection loss, transmission errors). Currently all unrecognized messages are silently ignored.
- **GUI -** Administration and monitoring of the system could benefit from a graphical interface. Currently, the state of a component can only be checked by looking at the general log file.
- **Security Concerns -** The current version of ERT does not handle any security issues, so malicious systems could interfere with ERT and possibly disable the whole service.
- **Real World Use Case -** A real application that utilizes ERT should be built, to see if this low-level approach is useful.

List of Figures

References

[1] L. Allison. Tries, 2004. URL http://www.csse.monash.edu.au/~lloyd/tildeAlgDS/Tree/Trie/.

[2] Apple-Computer. Rendezvous, 2003. URL http://www.apple.com/macosx/features/rendezvous.

[3] Apple-Computer. Rendezvous Technology Brief, 2003. URL http://images.apple.com/macosx/pdf/Panther_Rendezvous_TB_10232003.pdf.

[4] Apple-Computer. Spotlight Technologies, 2004. URL http://www.apple.com/macosx/tiger/spotlighttech.html.

[5] Apple-Computer. Xsan for traditional IT services, 2004. URL http://www.apple.com/xsan/datacenter.html.

[6] ASN.1 Consortium. ASN.1 Consortium, 2004. URL http://www.asn1.org.

[7] J. Aspnes and G. Shah. Skip Graphs. In *Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 384–393, Baltimore, MD, USA, 12–14˜ 2003. URL http://www.cs.yale.edu/homes/shah/html/pubs/skip-graphs.html.

[8] H. Balakrishnan and Others. IRIS: Infrastructure for Resilient Internet Systems, 2004. URL http://iris.lcs.mit.edu.

[9] S. Brin and L. Page. The Anatomy of a Search Engine, 2004. URL http://www-db.stanford.edu/~backrub/google.html.

[10] R. Card, T. Ts'o, and S. Tweedie. Design and Implementation of the Second Extended Filesystem, 2004. URL http://web.mit.edu/tytso/www/linux/ext2intro.html.

[11] S. D. S. Center. Distributed Object Computation Testbed (DOCT), 2004. URL http://www.sdsc.edu/DOCT/Publications/f4-1/f4-1.html.

[12] CERN. CERN - European Organization for Nuclear Research, 2004. URL http://www.cern.ch.

[13] A. Crainiceanu, P. Linga, J. Gehrke, and J. Shanmugasundaram. Querying Peer-to-Peer Networks Using P-Trees, 2004. URL http://techreports.library.cornell.edu:8081/Dienst/UI/1.0/Display/cul.cis/TR2004-1926.

[14] A. Crainiceanu, P. Linga, A. Machanavajjhala, J. Gehrke, and J. Shanmugasundaram. P-Ring: An Index Structure for Peer-to-Peer Systems, 2004. URL http://techreports.library.cornell.edu:8081/Dienst/UI/1.0/Display/cul.cis/TR2004-1946.

[15] A. Crainiceanu, P. Linga, A. Machanavajjhala, J. Gehrke, and J. Shanmugasundaram. A Storage and Indexing Framework for P2P Systems. Technical report, Cornell University, 2004. URL http://www.cs.cornell.edu/johannes/papers/2004/www2004-indexingArchitecturePoster.pdf.

[16] A. Cumming. A Gentle Introduction to SQL, 2003. URL http://sqlzoo.net/.

[17] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica. Towards a Common API for Structured P2P Overlays. In *Proc. of IPTPS*, pages 33–44, Berkeley, CA, Feb 2003. URL http://www.cs.berkeley.edu/~ravenben/publications/abstracts/apis.html.

[18] DAMA. DAMA International - Data Facts You Can Use, 2004. URL http://www.dama.org/public/pages/index.cfm?pageid=461.

[19] Data & Object Factory. Design Patterns: Proxy, 2004. URL http://www.dofactory.com/Patterns/PatternProxy.aspx.

[20] Data & Object Factory. Design Patterns: Singleton, 2004. URL http://www.dofactory.com/Patterns/PatternSingleton.aspx.

[21] L. Dongman. Distributed Systems (ICE 601) - Part 1, 2004. URL http://cds.icu.ac.kr/course/ice601/2004/lecture/intro1.pdf.

[22] L. Dongman. Distributed Systems (ICE 601) - Part 2, 2004. URL http://cds.icu.ac.kr/course/ice601/2004/lecture/intro2.pdf.

[23] P. Druschel and A. Rowstron. Pastry - A substrate for peer-to-peer applications, 2004. URL http://research.microsoft.com/~antr/Pastry.

[24] J. Frankel. Gnutella File Sharing, 2004. URL http://www.gnutella.com.

[25] E. Fredkin. Trie memory. *Commun. ACM*, 3(9):490–499, 1960. ISSN 0001-0782.

[26] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison-Wesley Professional; 1st edition (January 15, 1995), 1995. ISBN 0201633612.

[27] Getty Research Institute. Introduction to Metadata, 2004. URL http://www.getty.edu/research/conducting_research/standards/intrometadata/.

[28] Google. Google Desktop Search, 2004. URL http://desktop.google.com/.

[29] Google. Google Print, 2004. URL http://print.google.com/.

[30] Google. Google Web Search Features - Cache, 2004. URL http://www.google.com/help/features.html#cached.

[31] Hewlett-Packard. Cluster Hardware Configuration, 2002. URL http://h30097.www3.hp.com/docs/base_doc/DOCUMENTATION/V51B_ACRO_DUX/ARHGWETE.PDF.

[32] M. Hitchens. Distributed Systems (ITEC 801) - Introduction, 2003. URL http://www.comp.mq.edu.au/units/itec801/lectures/01-intro-ovs.pdf.

[33] P. Howard. Database Performance IBM, Oracle and Microsoft an evaluation. Technical report, Bloor Research, 2003. URL ftp://ftp.software.ibm.com/software/data/pubs/papers/bloor0104.pdf.

[34] M. Huras. Matt Huras and Version 8 Enhancements for DB2 on Linux, UNIX, and Windows, 2002. URL http://www-106.ibm.com/developerworks/db2/library/techarticle/0207huras/0207huras.html.

[35] IETF. Zero Configuration Networking (Zeroconf), 1999. URL http://www.zeroconf.org.

[36] O. S. Initiative. Open Source Initiative OSI - The BSD License:Licensing, 2004. URL http://www.opensource.org/licenses/bsd-license.php.

[37] G. B. Jacqueline Bloemen. A technical comparison. Technical report, Fourth Millennium Technology Inc. USA, 2003. URL ftp://ftp.software.ibm.com/software/data/highlights/dbmscomparison.pdf.

[38] A. K. Jane Wright. Compaq TruCluster Server Unix-Based Clustering Solution. Technical report, Gartner Research, 2002. URL http://www.totaltec.com/Gartner-Cluster-Solutions.pdf.

[39] T. Johnson and A. Colbrook. A Distributed Data-Balanced Dictionary Based on the B-Link Tree. Technical Report MIT/LCS/TR-530, 1992. URL http://citeseer.ist.psu.edu/johnson92distributed.html.

[40] F. Kaashoek, R. Morris, F. Dabek, I. Stoica, E. Brunskill, D. Karger, R. Cox, and A. Muthitacharoen. The Chord Project, 2004. URL `http://www.pdos.lcs.mit.edu/chord`.

[41] B. Karp, S. Ratnasamy, S. Rhea, and S. Shenker. Spurring Adoption of DHTs with OpenHash, a Public DHT Service, 2004. URL `http://iptps04.cs.ucsd.edu/papers/karp-openhash.pdf`.

[42] J. Kubiatowicz. The OceanStore Project, 2004. URL `http://oceanstore.cs.berkeley.edu`.

[43] I. Kuz. Distributed Systems (COMP 9243) - Introduction, 2004. URL `http://www.cse.unsw.edu.au/~s9243/lectures/intro-slides.pdf`.

[44] I. Kuz. Distributed Systems (COMP 9243) - Notes, 2004. URL `http://www.cse.unsw.edu.au/~s9243/lectures/intro-notes.pdf`.

[45] J. Larmouth, D. Steedman, and J. E. White. ASN.1 Information Site, 2004. URL `http://asn1.elibel.tm.fr/en`.

[46] T. Lee. Netty 2 - Overview, 2004. URL `http://gleamynode.net/dev/projects/netty2/`.

[47] P. L. Lehman and s. Bing Yao. Efficient locking for concurrent operations on B-trees. *ACM Trans. Database Syst.*, 6(4):650–670, 1981. ISSN 0362-5915. URL `http://www.it.iitb.ac.in/~it603/resources/papers/lehman.pdf`.

[48] A. Lisitsa. Distributed System Requirements, 2004. URL `http://www.csc.liv.ac.uk/~alexei/COMP514/COMP514-distributed-systems-04.pdf`.

[49] Microsoft. The Boxwood Project, 2004. URL `http://research.microsoft.com/research/sv/Boxwood/`.

[50] Microsoft. Data Access and Storage Developer Center: Building WinFS Solutions, 2004. URL `http://msdn.microsoft.com/data/winfs/`.

[51] M. Miley. The Grid - bringing computing power to the masses. Oracle Corporation, 2003. URL `http://otn.oracle.com/tech/grid/collateral/Grid_Feature.pdf`.

[52] E. Miller, R. Swick, D. Brickley, B. McBride, J. Hendler, G. Schreiber, and D. Connolly. W3C Semantic Web, 2004. URL `http://www.w3.org/2001/sw/`.

[53] MIMI. Access time - (Computing): Definition, 2004. URL `http://en.mimi.hu/computing/access_time.html`.

[54] NAMESYS. Reiser File System, 2004. URL `http://www.namesys.com/`.

[55] S. Ogura and T. Miura. Paging B-Trees for Distributed Environment. *Workshop on Distributed Data and Structure (WDAS)*, 2002. URL `http://www.ieice.org/iss/de/DEWS/proc/2002/presentations/B2-6.pdf`.

[56] Oracle-Corporation. Tutorial: Grid Computing Demo, 2004. URL `http://otn.oracle.com/tech/grid/index.html`.

[57] V. Pande. Folding@Home Distributed Computing, 2004. URL `http://folding.stanford.edu/`.

[58] W. Pugh. Skip lists: a probabilistic alternative to balanced trees. *Commun. ACM*, 33(6):668–676, 1990. ISSN 0001-0782. URL `ftp://ftp.cs.umd.edu/pub/skipLists/skiplists.pdf`.

[59] S. Rhea. The Bamboo Distributed Hash Table, 2004. URL `http://bamboo-dht.org`.

[60] L. Richter. Verteilte Systeme - Backup Services, 1999. URL `http://www.ifi.unizh.ch/richter/Classes/VS_WS00/V-5-BackupServices.pdf`.

[61] L. Richter. Verteilte Systeme - Einführung, 1999. URL `http://www.ifi.unizh.ch/richter/Classes/VS_WS00/V-1-Einf.pdf`.

[62] L. Richter. Verteilte Systeme - Konzepte und Mechanismen, 1999. URL `http://www.ifi.unizh.ch/richter/Classes/VS_WS00/V-3-Konz&Mech.pdf`.

[63] Y. Sagiv. Concurrent operations on B-trees with overtaking. In *Proceedings of the fourth ACM SIGACT-SIGMOD symposium on Principles of database systems*, pages 28–37. ACM Press, 1985. ISBN 0-89791-153-9.

[64] C. Schmidt and M. Parashar. Flexible Information Discovery in Decentralized Distributed Systems. In *12th IEEE International Symposium on High Performance Distributed Computing*, 2003. URL `http://www.caip.rutgers.edu/~cristins/schmidtc_discovery.pdf`.

[65] Sleepycat-Software. Berkeley DB, 2004. URL `http://www.sleepycat.com/products/db.shtml`.

[66] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proceedings of the ACM SIGCOMM ’01 Conference*, San Diego, California, August 2001. URL `http://www.pdos.lcs.mit.edu/papers/chord:sigcomm01`.

[67] Sun Microsystems. Java Remote Method Invocation (Java RMI), 2004. URL `http://java.sun.com/products/jdk/rmi/`.

[68] Sun Microsystems. New I/O APIs, 2004. URL `http://java.sun.com/j2se/1.4/nio/`.

[69] A. S. Tanenbaum and M. van Steen. *Distributed systems : principles and paradigms*. Prentice Hall Pearson Education International, Upper Saddle River, NJ, international ed. edition, 2002. ISBN 0-13-1217860(pbk.-).

[70] UC Berkeley. How Much Information, 2004. URL `http://www.sims.berkeley.edu/research/projects/how-much-info-2003/execsum.htm`.

[71] A. van Hoff. JmDNS home page, 2004. URL `http://jmdns.sourceforge.net/`.

[72] M. Welsh, D. E. Culler, and E. A. Brewer. SEDA: An Architecture for Well-Conditioned, Scalable Internet Services. In *Symposium on Operating Systems Principles*, pages 230–243, 2001. URL `citeseer.ist.psu.edu/welsh01seda.html`.

[73] Wikipedia. AVL tree - Wikipedia, the free encyclopedia, 2004. URL `http://en.wikipedia.org/wiki/AVL_tree`.

[74] Wikipedia. B-tree - Wikipedia, the free encyclopedia, 2004. URL `http://en.wikipedia.org/wiki/B-tree`.

[75] Wikipedia. Backward compatibility - Wikipedia, the free encyclopedia, 2004. URL `http://en.wikipedia.org/wiki/Backward_compatibility`.

[76] Wikipedia. Binary search tree - Wikipedia, the free encyclopedia, 2004. URL `http://en.wikipedia.org/wiki/Binary_search_tree`.

[77] Wikipedia. Binary tree - Wikipedia, the free encyclopedia, 2004. URL `http://en.wikipedia.org/wiki/Binary_tree`.

[78] Wikipedia. Database - Wikipedia, the free encyclopedia, 2004. URL `http://en.wikipedia.org/wiki/Database`.

[79] Wikipedia. Forward compatibility - Wikipedia, the free encyclopedia, 2004. URL `http://en.wikipedia.org/wiki/Forward_compatibility`.

[80] Wikipedia. HFS Plus - Wikipedia, 2004. URL `http://en.wikipedia.org/wiki/HFS_Plus`.

[81] Wikipedia. Mainframe computer - Wikipedia, the free encyclopedia, 2004. URL `http://en.wikipedia.org/wiki/Mainframe`.

[82] Wikipedia. Metadata (computing) - Wikipedia, the free encyclopedia, 2004. URL http://en.wikipedia.org/wiki/Metadata_%28computing%29.

[83] Wikipedia. Network File System - Wikipedia, 2004. URL http://en.wikipedia.org/wiki/NFS.

[84] Wikipedia. NTFS - Wikipedia, 2004. URL http://en.wikipedia.org/wiki/NTFS.

[85] Wikipedia. Red-black tree - Wikipedia, the free encyclopedia, 2004. URL http://en.wikipedia.org/wiki/Red-black_tree.

[86] Wikipedia. Relational model - Wikipedia, the free encyclopedia, 2004. URL http://en.wikipedia.org/wiki/Relational_model.

[87] Wikipedia. Self-balancing binary search tree - Wikipedia, the free encyclopedia, 2004. URL http://en.wikipedia.org/wiki/Self-balancing_binary_search_tree.

[88] Wikipedia. Splay tree - Wikipedia, the free encyclopedia, 2004. URL http://en.wikipedia.org/wiki/Splay_tree.

[89] P. Yalagandula. Solving Range Queries in a Distributed System. Technical report, University of Texas at Austin, 2003. URL http://www.cs.utexas.edu/users/browne/cs395f2003/projects/YalagandalaReport.pdf.

[90] B. Y. Zhao, A. Joseph, and J. Kubiatowicz. Tapestry - Infrastructure for Fault-resilient, Decentralized Location and Routing, 2004. URL http://www.cs.berkeley.edu/~ravenben/tapestry.

A Storage Subsystems

Using a storage subsystem has several advantages over directly accessing the disk:

- More than one node can directly access data - otherwise, only one node can mount the physical hard-drive.
- Because of an HAL (Hardware Abstraction Layer), the actual implementation of a storage subsystem can abstract several hardware limitations normally found on direct disk-based access:
  - Redundancy: If one disk fails, another can take over (i.e. RAID)
  - Adding/Removing/Reallocating disk space on the fly
  - Enhanced Caching mechanisms
- Storage is in one place; this makes it easier to administrate.

A.1 TruCluster Shared Storage

Hewlett-Packard Corporation - TruCluster is the cluster offering from HP (former Compaq) and also contains a shared storage mechanism.

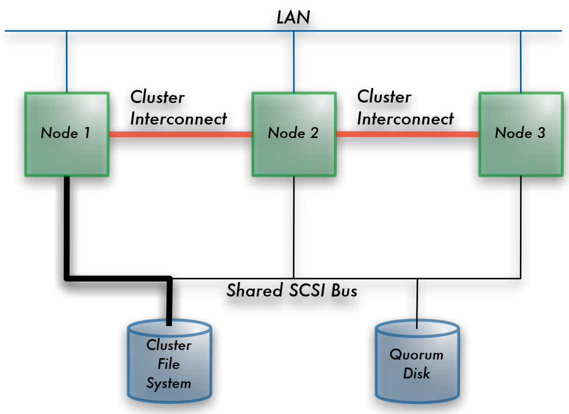


Figure 54: Shared Storage Overview (taken from [31])

Here one cluster member takes the lead and mounts the cluster-wide shared-storage disk. All other cluster members need to send all read/write requests to this leading member over the cluster-interconnect. If the leader is down, there is a sophisticated leader-elector process (not shown here) and the new leader then mounts the cluster-wide file system. This mechanism will not scale for a larger number of nodes. In general, truCluster is limited to 8 nodes in the current version.

A.2 Xsan

Apple Computer Corporation, Advanced Digital Information Corporation - As a newcomer to this market, they have a simple offering which helps consolidate storage to a common place. It uses technology from ADIC (Advanced Digital Information Corporation).

This approach also has limitations and will not scale indefinitely. The number of Storage Servers is limited (64 nodes) and also the concurrent access to the physical disks is limited. The overall performance could be higher than the truCluster approach, because it uses a dedicated storage access medium (Fibre Channel) used only for transmitting storage related data, where in truCluster, in addition to storage-related data, general cluster information data is also transmitted.

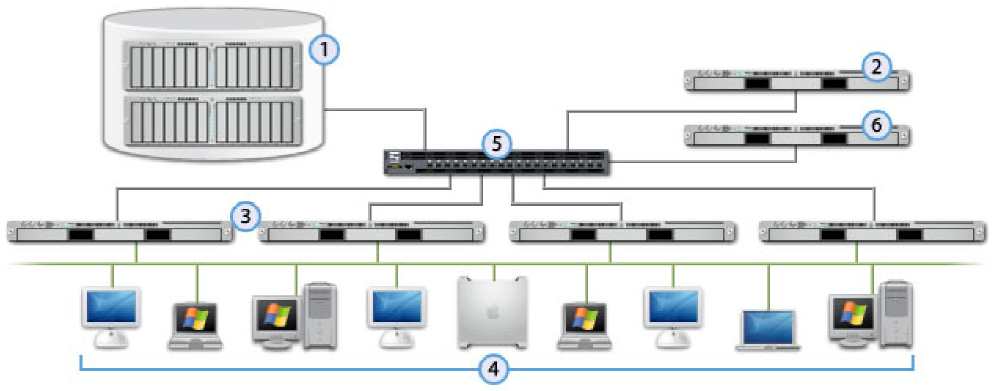


Figure 55: Xsan Overview (taken from [5])

1. **Storage Pool -** The actual disk drives.
2. **Metadata Controller -** Manages the Storage Servers (3) and directs them to the actual files in the Storage Pool.
3. **Storage Servers -** Are connected to the Storage Pool using Fibre Channel and use the Metadata Controller (2) for concurrent file access.
4. **Data Consumers -** Clients who want to access the data.
5. **Fibre Channel Switch -** Dedicated switch for Storage Pool (1).
6. **Backup Metadata Controller -** Provides high-availability insurance.

B Content of CD-ROM

The attached CD-ROM contains the following data:

- **Abstract -** A file containing the English abstract of this thesis.
- **Zusfsg -** A file containing the German abstract of this thesis.
- **code -** A folder containing the most recent version of the ERT software, including instructions on how to install and run the software.
- **benchmark -** A folder containing the scripts for automated testing, test protocol and a system profile of one of the test machines.
- **misc -** A folder containing additional files that have been produced during writing this thesis. This includes copies of PDF documents found during research and intermediate papers handed in during writing the thesis.
- **repository -** A folder containing the software repository for the software and the thesis paper itself. During the writing of the thesis, all intermediate steps were recorded into this repository. This allows retrieval of the thesis and source code at any given date. The repository is stored as a Subversion archive (see `http://subversion.tigris.org`) in two different formats. One as native format and the other as a dump file, to import the repository into another existing Subversion repository.
- **thesis -** A folder containing the final version of the thesis as PDF, LaTeX, LyX and plain text document.

C Source Code

File 1: base.Dispatcher

```java
/*
 * Created on Aug 13, 2004 11:44:01 AM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-26 15:28:00 +0200 (Tue, 26 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 138 $
 * Location:    $URL: Dispatcher.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.base;

import java.util.Hashtable;

import org.apache.log4j.Logger;

import ch.nix.ert.message.DataItem;
import ch.nix.ert.util.Queue;

import EDU.oswego.cs.dl.util.concurrent.LinkedQueue;

/**
 * TODO Edit this text for a better description
 *
 * This is a local dispatcher, he is responsible for all local requests. He does
 * not decide if the request is executed locally or remotely, he will ALWAYS
 * execute it remotly. It's the responsibility of each single service to relay
 * it to another node... <br>
 * If you don't know, if the request is for local or remote execution, you can
 * place it in Dispatcher, but he will forward it to KeyPartitionManager who
 * will then decide if executed locally or remotely (according their KeyGroupID)
 *
 * @author michi
 */
public class Dispatcher implements Runnable {

    // logger:
    static Logger logger = Logger.getLogger(Dispatcher.class);

    private LinkedQueue dispatcherQueue = new LinkedQueue();

    private Hashtable dispatcherServices = new Hashtable();

    private Hashtable serviceNames = new Hashtable();

    private Object queueLock = new Object();

    /**
     * A handle to the unique Dispatcher instance.
     */
    static private Dispatcher instance = null;

    private Dispatcher() {
    }

    static public synchronized Dispatcher getInstance() {
        if (null == instance) {
            instance = new Dispatcher();
            logger.info("Dispatcher started");
        }
        return instance;
    }

    /**
```

```
     * Only data which contain the "command" key are accepted, all other will be
     * silently ignored! TODO: perhpas we could also throw an exception?
     *
     * @param data
     */
    public void addWork(DataItem data) {
        synchronized (queueLock) {
            //check data, and only add it to queue, if it contains a operation
            // (like "put")
            if (data.getOperation() != null) {
                try {
                    //System.out.println("DispatcherWork: "+data);
                    dispatcherQueue.put(data);
                } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        }
    }

    /**
     * There is only ONE service allowed for each type... if an service is
     * already registered and a new one with the same keyword, subtype is added,
     * an exception is thrown TODO: implment this EXCEPTION
     *
     * @param command
     *            The events you are interested in (e.g. "put", "get",
     *            "delete","ping" <br>
     *            Some events have subtypes, subtypes can be: request, response
     *            (e.g. a "get" request can be of type "request", this means it
     *            has not been processed yet or of type "response" this means
     *            this is the response of the first request <br>
     *            <i>request </i>starting of a transaction <br>
     *            <i>response </i>ending of a transaction
     * @param queue
     *            The queue where incomming request are stored for this command.
     */
    public void registerService(String operation, Queue queue) {
        synchronized (dispatcherServices) {
            operation = operation.toLowerCase();
            dispatcherServices.put(operation, queue);

        }
    }

    public void registerQueue(Queue queue) {
        serviceNames.put(queue.getName(), queue);
    }

    public void unregisterService(String operation) {
        synchronized (dispatcherServices) {
            operation = operation.toLowerCase();
            dispatcherServices.remove(operation);
        }
    }

    /**
     * Start the dispatcher and ONLY ONE thread meight be inside it!
     */
    public void run() {
        // First set the name for this Thread, helps debugging :-)
        Thread.currentThread().setName("ERT.Dispatcher");
        TransactionMonitor transactionMonitor = new TransactionMonitor();
        DataItem dataToDispatch = null;
        String operation;
        String transactionType;
        String routingHint;
        int transactionSplit;
        while (true) {
            Queue operationQueue = null;
            // wait for something in dispatcherQueue
            try {
                dataToDispatch = (DataItem) dispatcherQueue.take();
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            // now dispatch request
            if (dataToDispatch != null) {

                // transactionType = dataToDispatch.getTransactionType();
                routingHint = dataToDispatch.getRouting();

                // reset the transactionSplit
                transactionSplit = 0;

                // now find the correct ServiceQueue where to put the value
                if (routingHint == null || routingHint.equals("local")) {
                    transactionType = dataToDispatch.getTransactionType();
                    if (transactionType != null
                            && transactionType.equals("end")) {
                        // we send the response right to the place where it
                        // originated
                        operationQueue = (Queue) serviceNames
                                .get(dataToDispatch.getTransactionServiceId());

                        // now check, if we this transaction is part to a
                        // splitted transaction
                        transactionSplit = dataToDispatch.getTransactionSplit();
                        if (transactionSplit > 0) {
                            // yes, it is a result from a splited transaction
                            // pass it on to the TransactionMonitor whoch holds
                            // references to all open splited transactions
                            dataToDispatch = transactionMonitor
                                    .put(dataToDispatch);
                            if (dataToDispatch == null) {
                                operationQueue = null;
                            }
                        }
                    } else {
                        // as I already checked at insertion time, this data has
                        // an operation defined!
                        operation = dataToDispatch.getOperation();
                        operationQueue = (Queue) dispatcherServices
                                .get(operation);
                    }
                } else {
                    // default means mostly the item is sent to the
                    // NodeCommunication layer to sending out to another
                    // node
                    operationQueue = (Queue) dispatcherServices.get("default");
                }

                if (operationQueue != null) {
                    try {
                        //System.out.println("Data2Dispatch: "+dataToDispatch+"
                        // is response? "+dataToDispatch.isRespose());
                        operationQueue.put(dataToDispatch);
//                        logger.debug("Work: " + dataToDispatch
//                                + " put on queue: " + operationQueue.getName());
                    } catch (InterruptedException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                    }
                } else if (transactionSplit == 0) {
                    logger.warn("No Queue found for dataItem: "
                            + dataToDispatch);
                }
            }
        }
    }
}
```

File 2: base.RootLogger

```
/*
 * Created on Sep 7, 2004 11:05:37 AM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-11-11 18:55:48 +0100 (Thu, 11 Nov 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 169 $
 * Location:    $URL: RootLogger.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.base;

import org.apache.log4j.BasicConfigurator;
import org.apache.log4j.DailyRollingFileAppender;
import org.apache.log4j.Level;
import org.apache.log4j.Logger;
```

```java
import org.apache.log4j.PatternLayout;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class RootLogger {

    static Logger logger = Logger.getRootLogger();

    /**
     *
     */
    public RootLogger() {
        // TTCCLayout layout = new TTCCLayout();
        PatternLayout layout = new PatternLayout(
                "%d{HH:mm:ss,SSS} [%15.15t] %-5p %30.30c.%-3L %x - %m%n");
        DailyRollingFileAppender appender = null;
        try {
            appender = new DailyRollingFileAppender(layout,
                    "/Users/tutor/ert/log/ert.log", "'.'yyyy-MM-dd");
        } catch (Exception e) {
        }

        logger.setLevel(Level.INFO);
        BasicConfigurator.configure(appender);
        logger.info("-------- ERT started --------");
    }
}
```

File 3: base.ServiceBase

```java
/*
 * Created on Sep 8, 2004 7:43:33 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-26 15:28:00 +0200 (Tue, 26 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 138 $
 * Location:    $URL: ServiceBase.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.base;

import org.apache.log4j.Logger;

import ch.nix.ert.message.DataItem;
import ch.nix.ert.message.Node;
import ch.nix.ert.util.Queue;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public abstract class ServiceBase implements ServiceInterface {

    // logger:
    static Logger logger = Logger.getLogger(ServiceBase.class);

    private Dispatcher dispatcher;

    private Queue serviceQueue;

    private int transactionId = 0;

    private Node currentNode;

    private String serviceName;

    private int splitSize = 0;

    private String storedId;

    private Node storedNodeId;

    private String storedServiceId;

    private String storedOperation;

    private String storedTransactionType;

    private String storedTransactionOperation;

    /**
     *
     */
    protected ServiceBase() {
        //this.getClass().getName().
        dispatcher = ServiceManager.getInstance().getDispatcher();
        currentNode = new Node();
        serviceName = this.getClass().getName();
        serviceQueue = new Queue(getQueueName());
        // register service Queue, so dispatcher knows where to send the
        // responses
        dispatcher.registerQueue(serviceQueue);
    }

    protected void registerService(String service) {
        dispatcher.registerService(service, serviceQueue);
    }

    public void addWork(DataItem data) {
        dispatcher.addWork(data);
    }

    //    public void transactionBeginStatefull(DataItem data) {
    //        data.setTransactionId(getTransactionId());
    //        transactionBeginGeneral(data);
    //    }
    //
    //    public void transactionBegin(DataItem data) {
    //        //data.setTransactionId(data.getOperation());
    //        transactionBeginGeneral(data);
    //    }

    public void transactionBegin(DataItem data) {
        data.setTransactionNodeId(getNodeId());
        data.setTransactionServiceId(getServiceId());
        data.setTransactionId(getTransactionId());
        data.setTransactionOperation(data.getOperation());
        data.setTransactionType("begin");
        dispatcher.addWork(data);
    }

    public void transactionContinue(DataItem data) {
        // only fill in data, if incoming request really was a transaction,
        // otherwise just send back the answer
        if (storedTransactionType != null) {
            data.setTransactionType("continue");
            data.setTransactionSplit(splitSize);
            data.setTransactionNodeId(storedNodeId);
            data.setTransactionServiceId(storedServiceId);
            data.setTransactionId(storedId);
            //data.setTransactionOperation(storedOperation);
        }
        dispatcher.addWork(data);
    }

    public void transactionSplit(DataItem data, int newSplitSize) {
        // only fill in data, if incoming request really was a transaction,
        // otherwise just send back the answer
        if (storedTransactionType != null) {
            data.setTransactionType("split");
            data.setTransactionSplit(splitSize+newSplitSize);
            data.setTransactionNodeId(storedNodeId);
            data.setTransactionServiceId(storedServiceId);
            data.setTransactionId(storedId);
            //data.setTransactionOperation(storedOperation);
        }
        dispatcher.addWork(data);
    }

    public void transactionEnd(DataItem data) {
        // only fill in data, if incoming request really was a transaction,
        // otherwise we don’t send back anything, as a transaction does not make
        // any sense...
        if (storedTransactionType != null) {
            data.setTransactionType("end");
            data.setTransactionSplit(splitSize);
            data.setTransactionNodeId(storedNodeId);
            data.setTransactionServiceId(storedServiceId);
            data.setTransactionId(storedId);
            data.setOperation(storedTransactionOperation);

            if (data.isTransactionLocal()) {
                data.setRouting("local");
            } else {
                data.setDirectDestination(data.getTransactionNodeId());
                data.setRouting("direct");
            }
            dispatcher.addWork(data);
        }
    }

    public Queue getServiceQueue() {
        return serviceQueue;
    }

    public DataItem getWork() {
        try {
            DataItem data = (DataItem) serviceQueue.take();
            //String transactionType = data.getTransactionType();
            //logger.warn("getWork: "+data);
            // if (transactionType != null && transactionType.equals("end")) {
            //logger.warn("if is executed");
            //    data.setOperation(data.getOperation() + "-end");
            //}
            storedOperation = data.getOperation();
            splitSize = data.getTransactionSplit();
            storedId = data.getTransactionId();
            storedServiceId = data.getTransactionServiceId();
            storedNodeId = data.getTransactionNodeId();
            storedTransactionType = data.getTransactionType();
            storedTransactionOperation = data.getTransactionOperation();

            return data;
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return null;
    }

    public DataItem getWork(long timeout) {
        try {
            return (DataItem) serviceQueue.poll(timeout);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return null;
    }

    //TODO do a nice regex search...
    private String getQueueName() {
        return this.getClass().getName();
        //System.out.println(this.getClass().getName().split("\\..*Service$"));
    }

    private Node getNodeId() {
        return currentNode;
    }

    private String getServiceId() {
        return this.getClass().getName();
    }

    private String getTransactionId() {
        return Integer.toString(transactionId++);
    }

}
```

File 4: base.ServiceInterface

```
/*
 * Created on Aug 13, 2004 3:03:03 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-26 15:28:00 +0200 (Tue, 26 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 138 $
```

```java
 * Location:    $URL: ServiceInterface.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.base;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public interface ServiceInterface extends Runnable {

    /**
     * Registers the Service at the Dispatcher
     */
    public void registerAtDispatcher();

    // public Queue getServiceQueue();

}
```

File 5: base.ServiceManager

```java
/*
 * Created on Aug 6, 2004 1:37:14 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-26 15:43:26 +0200 (Tue, 26 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 139 $
 * Location:    $URL: ServiceManager.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.base;

import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.util.ArrayList;

import org.apache.log4j.Logger;

/**
 * TODO Edit this text for a better description <br>
 *
 * Follows the Singelton-Design Pattern after Rodney Waldhoff.
 * http://radio.weblogs.com/0122027/stories/2003/10/20/implementingTheSingletonPatternInJava.html
 *
 * @author michi
 */
public class ServiceManager {

    //logger:
    static Logger logger = Logger.getLogger(ServiceManager.class);

    /**
     * A handle to the unique ServiceManager instance.
     */
    static private ServiceManager instance = null;

    static String[] serviceNameList = { "ch.nix.ert.main.MainService",
            "ch.nix.ert.callback.CallbackService",
            "ch.nix.ert.communication.NodeCommunicationService",
            "ch.nix.ert.datastore.DataStoreService",
            "ch.nix.ert.keypartition.KeyPartitionService",
            "ch.nix.ert.user.UserService" };

    static ArrayList serviceList = new ArrayList();

    static private Thread dispatcherThread;

    static private Dispatcher dispatcher;

    //private RootLogger ertLogger;

    /**
     * The constructor could be made private to prevent others from
     * instantiating this class. But this would also make it impossible to
     * create instances of ServiceManager subclasses.
     */
    private ServiceManager() {
        // TODO temporarely create logger here, afterwards it needs to be put
        // into startup class
        // ertLogger = new RootLogger();
    }

    /**
     * @return The unique instance of this class.
     */
    static public synchronized ServiceManager getInstance() {
        if (null == instance) {
            instance = new ServiceManager();
            dispatcher = Dispatcher.getInstance();
            dispatcherThread = new Thread(dispatcher);
            String serviceName;
            for (int i = 0; i < serviceNameList.length; i++) {
                serviceName = serviceNameList[i];
                ServiceInterface service;
                Thread serviceThread;
                Method method;

                try {
                    method = (Class.forName(serviceName).getMethod(
                            "getInstance", null));
                    service = (ServiceInterface) method.invoke(null, null);
                    service.registerAtDispatcher();
                    serviceThread = new Thread(service);
                    serviceList.add(serviceThread);
                    serviceThread.start();
                    logger.debug("Service loaded & started: " + serviceName);
                } catch (SecurityException e) {
                    logger.error("Error creating: " + serviceName + "\n" + e);
                } catch (NoSuchMethodException e) {
                    logger.error("Error creating: " + serviceName + "\n" + e);
                } catch (ClassNotFoundException e) {
                    logger.error("Error creating: " + serviceName + "\n" + e);
                } catch (IllegalArgumentException e) {
                    logger.error("Error creating: " + serviceName + "\n" + e);
                } catch (IllegalAccessException e) {
                    logger.error("Error creating: " + serviceName + "\n" + e);
                } catch (InvocationTargetException e) {
                    logger.error("Error creating: " + serviceName + "\n" + e);
                }

            }

            //and lastely:
            dispatcherThread.start();
            logger.info("ServiceManager started");
        }
        return instance;
    }

    public Dispatcher getDispatcher() {
        return dispatcher;
    }

}
```

File 6: base.TransactionMonitor

```java
/*
 * Created on Sep 24, 2004 4:14:16 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-26 15:43:26 +0200 (Tue, 26 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 139 $
 * Location:    $URL: TransactionMonitor.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.base;

import org.apache.log4j.Logger;

import ch.nix.ert.message.DataItem;
import ch.nix.ert.util.LinkedHashMap;

/**
 * Stores the temporary results for nested (or split) Transactions. Only if all
 * parts are arrived, this TransactionMonitor will join the results and return
 * the transaction
 * <p>
 *
 * In current implementation only boolean operations are allowed to be
 * split/joined (e.g. those who have ok/Error as result)
 *
 * @author michi
 */
public class TransactionMonitor {

    // logger:
    static Logger logger = Logger.getLogger(TransactionMonitor.class);

    // if more than 10000 open transactions, the oldest one will be discarded...
    private int bufferSize = 10000;

    private LinkedHashMap transactionIds;

    public TransactionMonitor() {
        transactionIds = new LinkedHashMap();
        transactionIds.setMaxSize(bufferSize);
    }

    /**
     * @param data
     * @return The final transaction DataItem, if we have now all parts
     *         together. It contains also an variable called result, which
     *         indicates, if operation was a success or not. If it was only a
     *         part of a transaction, it will return null
     */
    public DataItem put(DataItem dataItem) {
        // logger.error("Monitor: got it: "+dataItem);
        // first check the result type of this part, if it is already false, we
        // can already return this dataItem, as the whole transaction is wrong
        if (dataItem.getResult() == false) {
            removeId(dataItem.getTransactionId());
            return dataItem;
        }

        String transactionId = dataItem.getTransactionId();
        addOneToId(transactionId);

        int splitAmount = dataItem.getTransactionSplit();
        if (splitAmount <= getAmountForId(transactionId)) {
            // means we have all parts together and can return success :-)
            return dataItem;
        } else {
            return null;
        }

    }

    private int getAmountForId(String id) {
        return ((Integer) transactionIds.get(id)).intValue();
    }

    private void addOneToId(String id) {
        Integer value = (Integer) transactionIds.get(id);
        int amount;
        if (value == null) {
            amount = 0;
        } else {
            amount = value.intValue();
        }
        transactionIds.put(id, new Integer(amount + 1));
    }

    private void removeId(String id) {
        transactionIds.remove(id);
    }
}
```

File 7: callback.CallbackService

```java
/*
 * Created on Aug 18, 2004 12:55:22 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-26 15:28:00 +0200 (Tue, 26 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 138 $
 * Location:    $URL: CallbackService.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.callback;

import java.util.Calendar;
import java.util.TreeMap;

import org.apache.log4j.Logger;

import ch.nix.ert.base.ServiceBase;
import ch.nix.ert.message.DataItem;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class CallbackService extends ServiceBase {

    // logger:
    static Logger logger = Logger.getLogger(CallbackService.class);

    static private CallbackService instance = null;

    //private Queue serviceQueue = new Queue("CallbackQueue");

    //private Dispatcher dispatcher;

    private TreeMap timeSortedList = new TreeMap();

    private CallbackService() {
    }

    static public synchronized CallbackService getInstance() {
        if (null == instance) {
            instance = new CallbackService();
            //instance.dispatcher =
            // ServiceManager.getInstance().getDispatcher();
            logger.info("CallbackService started");
        }
        return instance;
    }

    /*
     * (non-Javadoc)
     *
     * @see ch.nix.ert.ServiceInterface#registerAtDispatcher()
     */
    public void registerAtDispatcher() {
        // TODO Auto-generated method stub
        registerService("calllater");
        registerService("callcontinous");
    }

    /*
     * (non-Javadoc)
     *
     * @see java.lang.Runnable#run()
     */
    public void run() {
        Thread.currentThread().setName("ERT.CallbackService");
        // get the first request
        long currentTime, wait;
        Long newestTime;
        DataItem dataItem = null;
        dataItem = getWork();
        // System.out.println("CallBackService got: "+dataItem);
        logger.debug("Register Callback: " + dataItem);
        this.putInSortedTimeList(dataItem);
        while (true) {
            currentTime = Calendar.getInstance().getTimeInMillis();
            newestTime = (Long) timeSortedList.firstKey();
            wait = newestTime.longValue();
            if (newestTime.longValue() < currentTime) {
                dataItem = (DataItem) timeSortedList.remove(newestTime);
                DataItem dataToExecute = dataItem.getNested();
                // fill in transaction-Data from souROunding dataItem:
                if (dataItem.getTransactionType() != null) {
                    // request was a transaction, so i Have to swap in all data
                    // from the main object into the nested one...
                    dataToExecute.setTransactionType("continue");
                    dataToExecute.setTransactionNodeId(dataItem
                            .getTransactionNodeId());
                    dataToExecute.setTransactionServiceId(dataItem
                            .getTransactionServiceId());
                    dataToExecute.setTransactionId(dataItem.getTransactionId());
                    dataToExecute.setTransactionOperation(dataToExecute
                            .getOperation());
                    transactionContinue(dataToExecute);
                } else {
                    addWork(dataToExecute);
                }
                logger.debug("Execute Callback: " + dataItem);
                //now check, if this is a continous service, so we have to put
                // it into queue again...
                if (dataItem.getOperation().equals("callcontinous")) {
                    this.putInSortedTimeList(dataItem);
                }
                if (timeSortedList.isEmpty()) {
                    wait = -1;
                } else {
                    wait = ((Long) timeSortedList.firstKey()).longValue();
                    //System.out.println("Set Waiting to: "+wait);
                }
            }
            if (wait == -1) {
                // means there is no element in the queue anymore, so I have
                // to wait indefinetly:
                dataItem = getWork();
                logger.debug("Register Callback: " + dataItem);
                // System.out.println("CallBackService got: "+dataItem);
            } else {
                dataItem = getWork(wait - currentTime);
                logger.debug("Register Callback or Timeout: " + dataItem);
                // System.out.println("CallBackService got: "+dataItem);
            }
            if (dataItem != null) {
                this.putInSortedTimeList(dataItem);
            }

        }
    }

    private void putInSortedTimeList(DataItem data) {
        long currentTime = Calendar.getInstance().getTimeInMillis();
        long delay = data.getDelay();
        long nextTime = currentTime + delay;
        //TODO: Performance enhancements todo...
        //As Java Maps/Sets do not allow duplicated keys, I have to check
        // first, if this key is already in queue... if so I have to postpone it
        // a bit... this is not very performant, but this class is anyway not
        // used that often in the application. If you rely on exact timing, a
        // external Priority-Queue class needs to be built, which supports
        // duplicated keys
        boolean duplicateFound = true;
        while (duplicateFound) {
            if (timeSortedList.containsKey(new Long(nextTime))) {
                nextTime++;
            } else {
                duplicateFound = false;
            }
        }
        timeSortedList.put(new Long(nextTime), data);
    }

}
```

File 8: communication.NodeCommunication

```
/*
 * Created on Aug 10, 2004 11:26:59 AM
```

```java
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-26 15:28:00 +0200 (Tue, 26 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 138 $
 * Location:    $URL: NodeCommunication.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.communication;

import org.apache.log4j.Logger;

import ch.nix.ert.message.DataItem;

/**
 * TODO Edit this text for a better description
 *
 * Follows the Singelton-Design Pattern after Rodney Waldhoff.
 * http://radio.weblogs.com/0122027/stories/2003/10/20/implementingTheSingletonPatternInJava.html
 *
 * @author michi
 */
public class NodeCommunication implements NodeCommunicationInterface {

    static Logger logger = Logger.getLogger(NodeCommunication.class);

    static private String plugin = "netty2";

    static private NodeCommunication instance = null;

    private NodeCommunicationInterface delegate = null;

    private NodeCommunication() {
    }

    static public synchronized NodeCommunication getInstance() {
        if (null == instance) {
            instance = new NodeCommunication();
            plugin = plugin.toLowerCase();
            try {
                instance.delegate = (NodeCommunicationInterface) (Class
                        .forName("ch.nix.ert.communication." + plugin
                                + ".NodeCommunication").newInstance());
                logger.info("NodeCommunication loaded plugin: "+plugin);
                return instance;
            } catch (InstantiationException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (IllegalAccessException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (ClassNotFoundException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            return null;
        }
        return instance;

    }

    /*
     * (non-Javadoc)
     *
     * @see ch.nix.ert.communication.NodeCommunicationInterface#send(ch.nix.ert.DataItem)
     */
    public void send(DataItem dataItem) {
        // TODO Auto-generated method stub
        delegate.send(dataItem);
    }
}
```

File 9: communication.NodeCommunicationInterface

```java
/*
 * Created on Aug 10, 2004 11:27:16 AM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-26 15:28:00 +0200 (Tue, 26 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 138 $
 * Location:    $URL: NodeCommunicationInterface.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.communication;

import ch.nix.ert.message.DataItem;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public interface NodeCommunicationInterface {

    public void send(DataItem data);

}
```

File 10: communication.NodeCommunicationService

```
/*
 * Created on Aug 17, 2004 11:51:59 AM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-26 15:28:00 +0200 (Tue, 26 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 138 $
 * Location:    $URL: NodeCommunicationService.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.communication;

import org.apache.log4j.Logger;

import ch.nix.ert.base.ServiceBase;
import ch.nix.ert.message.DataItem;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class NodeCommunicationService extends ServiceBase {

    // logger:
    static Logger logger = Logger.getLogger(NodeCommunicationService.class);

    static private NodeCommunicationService instance = null;

    private NodeCommunication nodeCommunication;

    private NodeDirectory nodeDirectory;

    private NodeCommunicationService() {
    }

    static public synchronized NodeCommunicationService getInstance() {
        if (null == instance) {
            instance = new NodeCommunicationService();
            instance.nodeCommunication = NodeCommunication.getInstance();
            instance.nodeDirectory = NodeDirectory.getInstance();
            logger.info("NodeCommunicationService started");
        }
        return instance;
    }

    /*
     * (non-Javadoc)
```

```java
     *
     * @see ch.nix.ert.ServiceInterface#registerAtDispatcher()
     */
    public void registerAtDispatcher() {
        registerService("send");
        registerService("publish");
        registerService("unpublish");

        registerService("default");
    }

    /*
     * (non-Javadoc)
     *
     * @see java.lang.Runnable#run()
     */
    public void run() {
        //      System.out.println("DataStoreService started");
        Thread.currentThread().setName("ERT.NodeCommunicationService");
        while (true) {
            DataItem dataItem = null;
            dataItem = getWork();
            // System.out.println("got request: " + dataItem);
            String operation = dataItem.getOperation();
            if (operation.equals("send")) {
                // now check if dataItem is nested. otherwise send the whole
                // datItem as it is.
                // This can happen, if we automatically send back the response
                // to a request, then it meight not have a nested dataItem, as
                // nobody has wrapped up the packet.
                DataItem dataToSend = dataItem.getNested();
                if (dataToSend == null) {
                    dataToSend = dataItem;
                }
                //System.out.println("Comm Service2send: " + dataToSend);
                nodeCommunication.send(dataToSend);
            } else if (operation.equals("publish")) {
                nodeDirectory.publish(dataItem);
            } else if (operation.equals("unpublish")) {
                nodeDirectory.unpublish(dataItem);
            } else {

                //TODO will be uncommented later!
                DataItem dataToSend = dataItem.getNested();
                if (dataToSend == null) {
                    dataToSend = dataItem;
                }
                // System.out.println("Comm Service: " + dataToSend);
                nodeCommunication.send(dataToSend);
            }
        }
    }
}
```

File 11: communication.NodeDirectory

```java
/*
 * Created on Aug 8, 2004 8:11:34 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-26 15:28:00 +0200 (Tue, 26 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 138 $
 * Location:    $URL: NodeDirectory.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.communication;

import org.apache.log4j.Logger;

import ch.nix.ert.message.DataItem;

/**
 * TODO Edit this text for a better description
 *
 * Follows the Singelton-Design Pattern after Rodney Waldhoff.
 * http://radio.weblogs.com/0122027/stories/2003/10/20/implementingTheSingletonPatternInJava.html
 *
 * In addition this class acts as a proxy to the actual implementation.
 *
 * @author michi
 */
public class NodeDirectory implements NodeDirectoryInterface {

    // logger:
    static Logger logger = Logger.getLogger(NodeDirectory.class);

    static private String plugin = "jmdns";

    static private NodeDirectory instance = null;

    private NodeDirectoryInterface delegate = null;

    private NodeDirectory() {
    }

    static public synchronized NodeDirectory getInstance() {
        if (null == instance) {
            instance = new NodeDirectory();
            plugin = plugin.toLowerCase();
            try {
                instance.delegate = (NodeDirectoryInterface) (Class
                        .forName("ch.nix.ert.communication." + plugin
                                + ".NodeDirectory").newInstance());
                return instance;
            } catch (InstantiationException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (IllegalAccessException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (ClassNotFoundException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            return null;
        }
        return instance;

    }

    /*
     * (non-Javadoc)
     *
     * @see ch.nix.ert.communication.NodeDirectoryInterface#publish(ch.nix.ert.KeyPartition)
     */
    public void publish(DataItem keyPartition) {
        logger.info("Publish: "+keyPartition);
        delegate.publish(keyPartition);
    }

    /*
     * (non-Javadoc)
     *
     * @see ch.nix.ert.communication.NodeDirectoryInterface#unpublish(ch.nix.ert.KeyPartition)
     */
    public void unpublish(DataItem keyPartition) {
        logger.info("Unpublish: "+keyPartition);
        delegate.unpublish(keyPartition);
    }

    /*
     * (non-Javadoc)
     *
     * @see ch.nix.ert.communication.NodeDirectoryInterface#findNode(ch.nix.ert.DataItem)
     */

    public DataItem findNode(DataItem data) {
        return delegate.findNode(data);
    }

//    public Node findNode(InternalKey key) {
//        DataItem myData = new DataItem();
//        myData.putValue("internalKey", key);
//        return this.findNode(myData);
//    }

}
```

File 12: communication.NodeDirectoryInterface

```
/*
 * Created on Aug 8, 2004 8:12:11 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-26 15:28:00 +0200 (Tue, 26 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 138 $
 * Location:    $URL: NodeDirectoryInterface.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.communication;

import ch.nix.ert.message.DataItem;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public interface NodeDirectoryInterface {

    public void publish(DataItem dataKeyPartition);
    public void unpublish(DataItem dataKeyPartition);
    public DataItem findNode(DataItem data);

}
```

File 13: communication.jmdns.NodeDirectory

```
/*
 * Created on Aug 4, 2004 3:29:26 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-11-11 19:33:00 +0100 (Thu, 11 Nov 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 171 $
 * Location:    $URL: NodeDirectory.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.communication.jmdns;

import java.io.IOException;
import java.util.Hashtable;
import java.util.SortedMap;

import javax.jmdns.JmDNS;
import javax.jmdns.ServiceInfo;

import org.apache.log4j.Logger;

import ch.nix.ert.communication.NodeDirectoryInterface;
import ch.nix.ert.message.DataItem;
import ch.nix.ert.message.InternalKey;
import ch.nix.ert.message.Node;

/**
 * TODO Edit this text for a better description
 *
 * This is a special implementation for zeroconf (aka Apple:Rendezvous,
 * Apple:OpenTalk) and published all KeyPartitions
 *
 * @author michi
 */
public class NodeDirectory implements NodeDirectoryInterface {

    // logger:
    static Logger logger = Logger.getLogger(NodeDirectory.class);

    private JmDNS jmdns;

    private Hashtable registeredServiceTable = new Hashtable();

    private NodeDirectoryListener nodeDirectoryListener = new NodeDirectoryListener();

    /**
     *
     */
    public NodeDirectory() {
        try {
            jmdns = new JmDNS();
            //now start Listener to all ExtendedRootTree nodes out there!
            jmdns.addServiceListener("_master._ert._tcp.local.",
                    nodeDirectoryListener);
        } catch (IOException e) {
            // TODO Auto-generated catch block
            logger.error("Error creating zeroconf service (JmDNS)" + e);
        }
    }

    // TODO change: the HashTable should contain a reference to the actual
    // KeyPartition, and should check if the information in this keyPartition
    // has changed: if so, it has to unpublish the old and publish the new one
    public void publish(DataItem data) {
        //KeyPartition keyPartition = data.getKeyPartition();
        InternalKey firstInternalKey = data.getFirstInternalKey();
        InternalKey lastInternalKey = data.getLastInternalKey();
        int keyPartitionId = data.getKeyPartitionId();

        Hashtable props = new Hashtable();
        //now check, if for some reason something is alreay registered at the
        // very same name! (To prevent the creation of [2]...)
        if (nodeDirectoryListener.getKeyPartitionList().containsKey(
                firstInternalKey)) {
            //there is already something out there with the name 2, I dont
            // publish this new information!
            //System.out.println("Already something published with this
            // name!");
            logger
                    .info("There is already something published with this KeyGroup-ID: "
                            + firstInternalKey
                            + ", I don't overwrite this informatin");
            return;
        }
        String publishedInfo = firstInternalKey.toString();
        if (registeredServiceTable.containsKey(new Integer(keyPartitionId))) {
            //This service is already published, now check, if KeyPartition has
            // changed!
            ServiceInfo oldServiceInfo = (ServiceInfo) registeredServiceTable
                    .get(new Integer(keyPartitionId));
            String oldPublishedInfo = oldServiceInfo.getName();
            logger.info("KeyPartition was already published, but has changed"
                    + " in meantime, I will update my Directory from: "
                    + oldPublishedInfo + " to: " + publishedInfo);
            //System.out.println("new: " + publishedInfo);
            //System.out.println("old: " + oldPublishedInfo);
            //System.out.println("Same?" +
            // publishedInfo.equals(oldPublishedInfo));
            if (publishedInfo.equals(oldPublishedInfo)) {
                // keyPartition is the same, nothing to do!
                //System.out.println("already published, nothing todo!");
                logger
                        .info("KeyPartition has changed, but is not relevant"
                                + " for NodeDirectory, so I don't re-publish anything now");
                return;
            } else {
                // keyPartitions have significantly changed, i need to update my
                // directory
                //System.out.println("unpublish/publish me: " + publishedInfo);
                logger
                        .info("KeyPartition has changed, and changes are relevant"
                                + " for NodeDirectory, I will now re-publish"
                                + " the updated information");
                jmdns.unregisterService(oldServiceInfo);
                registeredServiceTable.remove(new Integer(keyPartitionId));
            }
        }
        props.put("keygroup", publishedInfo);
        // TODO currently the port 1099 is hardcoded (RMI port), must be made
        // configurable.
        ServiceInfo newServiceInfo = new ServiceInfo(
                "_master._ert._tcp.local.", publishedInfo
                        + "._master._ert._tcp.local.", 1099, 0, 0, props);

        try {
```

```java
            //System.out.println("registering new service");
            logger.info("Register KeyPartition: (" + firstInternalKey + "-"
                    + lastInternalKey + ") with ID: " + keyPartitionId
                    + " (JmDNS ServiceInfo: "
                    + newServiceInfo.getNiceTextString() + ")");
            jmdns.registerService(newServiceInfo);
            registeredServiceTable.put(new Integer(keyPartitionId),
                    newServiceInfo);
        } catch (IOException e) {
            // TODO Auto-generated catch block
            System.out.println("ERROR registering zeroconf service (JmDNS) "
                    + e);
            logger
                    .error("Error registering KeyPartition in zeroconf service (JmDNS)");
        }
    }

    public void unpublish(DataItem data) {
        //KeyPartition keyPartition = data.getKeyPartition();
        InternalKey firstInternalKey = data.getFirstInternalKey();
        InternalKey lastInternalKey = data.getLastInternalKey();
        int keyPartitionId = data.getKeyPartitionId();

        ServiceInfo oldServiceInfo = (ServiceInfo) registeredServiceTable
                .get(new Integer(keyPartitionId));
        //System.out.println("KeyPartition: " + keyPartition);
        //System.out.println("ServiceHash: " + registeredServiceTable);
        if (oldServiceInfo != null) {
            logger.info("Unregister KeyPartition with ID: " + keyPartitionId
                    + " (JmDNS ServiceInfo: "
                    + oldServiceInfo.getNiceTextString() + ")");
            jmdns.unregisterService(oldServiceInfo);
        }
        registeredServiceTable.remove(new Integer(keyPartitionId));
    }

    public DataItem findNode(DataItem data) {
        if (data.getInternalKey() == null) {
            return null;
        }
        SortedMap headMap = nodeDirectoryListener.getKeyPartitionList()
                .headMap(data.getInternalKey().getSubsequentInternalKey());
        //if this beast is not empty, get the last key of it:
        Node node;
        if (!headMap.isEmpty()) {
            node = (Node) nodeDirectoryListener.getKeyPartitionList().get(
                    headMap.lastKey());
            DataItem returnData = new DataItem();
            returnData.setDirectDestination(node);
            return returnData;
        }
        // should not happen, as always all KeyPartitions should be available
        // (only in case a machine is down, network problem, this can happen
        return null;
    }
}
```

File 14: communication.jmdns.NodeDirectoryListener

```java
/*
 * Created on Aug 4, 2004 5:31:31 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-11-04 10:15:02 +0100 (Thu, 04 Nov 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 158 $
 * Location:    $URL: NodeDirectoryListener.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.communication.jmdns;

import java.util.TreeMap;

import javax.jmdns.JmDNS;
import javax.jmdns.ServiceInfo;
import javax.jmdns.ServiceListener;

import ch.nix.ert.message.InternalKey;
import ch.nix.ert.message.Node;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class NodeDirectoryListener implements ServiceListener {

    private TreeMap nodeDirectory = new TreeMap();

    public TreeMap getKeyPartitionList() {
        return nodeDirectory;
    }

    /*
     * (non-Javadoc)
     *
     * @see javax.jmdns.ServiceListener#addService(javax.jmdns.JmDNS,
     *      java.lang.String, java.lang.String)
     */
    public void addService(JmDNS jmdns, String type, String name) {
        String shortname;
        shortname = name.substring(0, name.length() - (type.length() + 1));
        //check if name is a number (e.g. not 1 [1])
        if (shortname.endsWith("]")) {
            //invalid name, discard it!
            //System.out.println("Wrong Name in addService: " + shortname);
            return;
        }
        InternalKey key = new InternalKey(shortname);
        Node value = new Node(jmdns.getServiceInfo(type, name).getAddress(),
                jmdns.getServiceInfo(type, name).getPort());
        nodeDirectory.put(key, value);
        //System.out.println("ADD: " + shortname);
    }

    /*
     * (non-Javadoc)
     *
     * @see javax.jmdns.ServiceListener#removeService(javax.jmdns.JmDNS,
     *      java.lang.String, java.lang.String)
     */
    public void removeService(JmDNS jmdns, String type, String name) {
        // TODO Auto-generated method stub
        if (name.endsWith("." + type)) {
            name = name.substring(0, name.length() - (type.length() + 1));
        }
        // check if name is a number (e.g. not 1 [1])
        if (name.endsWith("]")) {
            //invalid name, discard it!
            //System.out.println("Wrong Name in removeService: " + name);
            return;
        }
        InternalKey key = new InternalKey(name);
        nodeDirectory.remove(key);
        //System.out.println("REMOVE: " + name);
    }

    /*
     * (non-Javadoc)
     *
     * @see javax.jmdns.ServiceListener#resolveService(javax.jmdns.JmDNS,
     *      java.lang.String, java.lang.String, javax.jmdns.ServiceInfo)
     */
    public void resolveService(JmDNS jmdns, String type, String name,
            ServiceInfo info) {
        // TODO Auto-generated method stub
        System.out.println("RESOLVED: " + info);
    }

}
```

File 15: communication.netty2.ErtClassCatalog

```java
/*
 * Created on Oct 22, 2004 9:52:21 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-11-12 12:46:25 +0100 (Fri, 12 Nov 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 173 $
 * Location:    $URL: ErtClassCatalog.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.communication.netty2;

import java.io.ObjectStreamClass;
import java.util.Hashtable;

import org.apache.log4j.Logger;

import ch.nix.ert.message.DataItem;

import com.sleepycat.bind.serial.ClassCatalog;
import com.sleepycat.je.DatabaseException;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class ErtClassCatalog implements ClassCatalog {

    // logger:
    static Logger logger = Logger.getLogger(ErtClassCatalog.class);

    String[] classIds = { "ch.nix.ert.DataItem", "java.util.Hashtable",
            "java.lang.Boolean", "ch.nix.ert.Node",
            "java.net.InetSocketAddress", "java.net.InetAddress",
            "java.net.SocketAddress", "ch.nix.ert.InternalKey",
            "ch.nix.ert.UserKey" };

    /*
     * (non-Javadoc)
     *
     * @see com.sleepycat.bind.serial.ClassCatalog#close()
     */
    public void close() throws DatabaseException {
        // TODO Auto-generated method stub

    }

    /*
     * (non-Javadoc)
     *
     * @see com.sleepycat.bind.serial.ClassCatalog#getClassFormat(byte[])
     */
    public ObjectStreamClass getClassFormat(byte[] classID)
            throws DatabaseException, ClassNotFoundException {
        // System.out.println("getClassFormat:
        // "+ObjectStreamClass.lookup(DataItem.class));
        //System.out.println("Lookedup: " + classID[0]);
        String name = classIds[(int) classID[0]];
        return ObjectStreamClass.lookup(Class.forName(name));
    }

    /*
     * (non-Javadoc)
     *
     * @see com.sleepycat.bind.serial.ClassCatalog#getClassID(java.io.ObjectStreamClass)
     */
    public byte[] getClassID(ObjectStreamClass classDesc)
            throws DatabaseException, ClassNotFoundException {
        // TODO Auto-generated method stub
        // System.out.println("getClassID: "+classDesc);
        String name = classDesc.getName();
        for (int i = 0; i < classIds.length; i++) {
            if (((String) classIds[i]).equals(name)) {
                // System.out.println("calculated: " + i);
                byte[] result = new byte[1];
                result[0] = (byte) i;
                return result;
            }

        }
        logger.error("Undefined Class: " + name
                + ", please update ErtClassCatalog with the"
                + " new Class you introduced in DataItem");
        return new byte[0xff];
    }

}
```

File 16: communication.netty2.ErtMessage

```java
/*
 * Created on Oct 21, 2004 9:24:15 AM
 *
 * Project:      ExtendendRootTree
 * Last Change: $Date: 2004-10-28 22:38:52 +0200 (Thu, 28 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 144 $
 * Location:    $URL: ErtMessage.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.communication.netty2;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.nio.ByteBuffer;

import net.gleamynode.netty2.Message;
import net.gleamynode.netty2.MessageParseException;

import org.apache.log4j.Logger;

import ch.nix.ert.message.DataItem;
import ch.nix.ert.message.DataItemCoder;

import com.sleepycat.bind.serial.SerialInput;
import com.sleepycat.bind.serial.SerialOutput;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class ErtMessage implements Message {

    // logger:
    static Logger logger = Logger.getLogger(ErtMessage.class);

    public DataItem dataItem = null;

    private DataItemCoder dataItemCoder;

    public ErtMessage() {
        dataItemCoder = new DataItemCoder();
    }

    public DataItem getDataItem() {
        return dataItemCoder.getDataItem();
    }

    public void setDataItem(DataItem myData) {
        dataItem = myData;
        dataItemCoder.setDataItem(myData);
    }

    /*
     * (non-Javadoc)
     *
     * @see net.gleamynode.netty2.Message#read(java.nio.ByteBuffer)
     */
    public boolean read(ByteBuffer buf) throws MessageParseException {
        return dataItemCoder.decode(buf);
    }

    /*
     * (non-Javadoc)
     *
     * @see net.gleamynode.netty2.Message#write(java.nio.ByteBuffer)
     */
    public boolean write(ByteBuffer buf) {
        return dataItemCoder.encode(buf);
    }

}
```

File 17: communication.netty2.ErtMessageRecognizer

```
/*
 * Created on Oct 21, 2004 9:23:30 AM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-29 17:46:07 +0200 (Fri, 29 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 146 $
 * Location:    $URL: ErtMessageRecognizer.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.communication.netty2;

import java.nio.ByteBuffer;

import net.gleamynode.netty2.Message;
import net.gleamynode.netty2.MessageParseException;
import net.gleamynode.netty2.MessageRecognizer;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class ErtMessageRecognizer implements MessageRecognizer {

    /* (non-Javadoc)
     * @see net.gleamynode.netty2.MessageRecognizer#recognize(java.nio.ByteBuffer)
     */
    public  Message recognize(ByteBuffer buf) throws MessageParseException {
        // return null if we have not yet received the length of the message
        if (buf.remaining() < 5) {
            return null;
        }
        // we have the length so we get the message!
//        int length = buf.getInt();
//        if (length < 1) {
//            return null;
//        }
        return new ErtMessage();
    }

}
```

File 18: communication.netty2.NodeCommunication

```
/*
 * Created on Oct 21, 2004 8:30:54 AM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-11-09 01:55:31 +0100 (Tue, 09 Nov 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 165 $
 * Location:    $URL: NodeCommunication.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.communication.netty2;

import java.io.IOException;
import java.net.InetSocketAddress;

import org.apache.log4j.Logger;

import net.gleamynode.netty2.IoProcessor;
```

```java
import net.gleamynode.netty2.LowLatencyEventDispatcher;
import net.gleamynode.netty2.MessageRecognizer;
import net.gleamynode.netty2.OrderedEventDispatcher;
import net.gleamynode.netty2.Session;
import net.gleamynode.netty2.SessionServer;
import net.gleamynode.netty2.SimpleEventDispatcher;
import net.gleamynode.netty2.ThreadPooledEventDispatcher;
import ch.nix.ert.base.Dispatcher;
import ch.nix.ert.base.ServiceManager;
import ch.nix.ert.communication.NodeCommunicationInterface;
import ch.nix.ert.communication.NodeDirectory;
import ch.nix.ert.communication.rmi.NodeCommunicationRmiInterface;
import ch.nix.ert.message.DataItem;
import ch.nix.ert.message.Node;

/**
 * This is an implementation of Netty 2
 * (http://gleamynode.net/dev/projects/netty2)
 *
 * It is implemented on top of Java NIO and provides good network performance
 *
 * @author michi
 */
public class NodeCommunication implements NodeCommunicationInterface {

    // logger:
    static Logger logger = Logger.getLogger(NodeCommunication.class);

    private static final String HOSTNAME = "localhost";

    private static final int SERVER_PORT = 15123;

    private static final int CONNECT_TIMEOUT = 30;

    private static final int DISPATCHER_THREAD_POOL_SIZE = 1;

    private NodeCommunicationServerListener listener;

    NodeDirectory nodeDirectory = NodeDirectory.getInstance();

    private Dispatcher dispatcher;

    public NodeCommunication() {

        dispatcher = ServiceManager.getInstance().getDispatcher();

        // initialize I/O processor and event dispatcher
        IoProcessor ioProcessor = new IoProcessor();
        SimpleEventDispatcher eventDispatcher = new SimpleEventDispatcher();

        try {
            // start with the default number of I/O worker threads
            ioProcessor.start();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        // start with a few event dispatcher threads
        eventDispatcher.setThreadPoolSize(DISPATCHER_THREAD_POOL_SIZE);
        eventDispatcher.start();

        // prepare message recognizer
        ErtMessageRecognizer recognizer = new ErtMessageRecognizer();

        // prepare session event listener which will provide communication
        // workflow.
        listener = new NodeCommunicationServerListener();

        // prepare session server

        SessionServer server = new SessionServer();
        server.setIoProcessor(ioProcessor);
        server.setEventDispatcher(eventDispatcher);
        server.setMessageRecognizer(recognizer);

        server.addSessionListener(listener);
        server.setBindAddress(new InetSocketAddress(SERVER_PORT));

        // open the server port, accept connections, and start communication

        // logger.info("Listening on port " + SERVER_PORT);
        try {
            server.start();
        } catch (IOException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
    }

    /*
     * (non-Javadoc)
     *
     * @see ch.nix.ert.communication.NodeCommunicationInterface#send(ch.nix.ert.DataItem)
     */
    public void send(DataItem dataItem) {
        String ip = "";
        // now check where the dataItem has to be sent to
        // first check if we have a "directdelivery" packet, which does not need
        // to be routed
        Node node = dataItem.getDirectDestination();
        if (node == null) {
            // no indication that we have a direct delivery packet, so we have
            // to lookup the dataItem
            DataItem destination = nodeDirectory.findNode(dataItem);
            if (destination == null) {
                //                logger
                //                        .warn("Could not send (unknown destination): "
                //                                + dataItem);

                return;
            }
            Node destinationNode = destination.getDirectDestination();
            if (destinationNode.isLocalhost()) {
                // means we would send this dataItem to ourselfs, so we return
                // it here already!
                dataItem.setRouting("local");
                dispatcher.addWork(dataItem);
                return;
            }
            ip = destinationNode.getIp();

        } else {
            ip = node.getIp();
            // we don’t want to send over this information
            // TODO later we should write an cleanup method into DataItem which
            // automatically removes all unused parameters bevore sending it or
            // storing it...
            dataItem.removeDirectDestination();
        }

        //now add originator Address:
        dataItem.setDirectOriginator(new Node(SERVER_PORT));

        Session session = getSessionForAddress(ip);
        ErtMessage ertMessage = new ErtMessage();
        ertMessage.setDataItem(dataItem);
        //        logger.debug("for IP: " + ip + ", I took session: "
        //                + session.getSocketAddressString() + " " + session);
        //        session.setAttachment(this);
        //        boolean result = session.write(ertMessage);
        boolean result = false;
        int round = 0;
        while (!result && round < 2) {
            // sending was not successful, try again!
            session = getSessionForAddress(ip);
            result = session.write(ertMessage);
            round++;
        }
    }

    private Session getSessionForAddress(String ip) {
        Session cachedSession = null;
        cachedSession = SessionManager.getSessionForAddress(ip);
        //System.out.println(SessionManager.getSize());
        if (cachedSession == null) {
            logger
                    .info("In hashtable are "
                            + SessionManager.getSize()
                            + " connections stored, I need to create a new session for ip: "
                            + ip);
            IoProcessor clientIoProcessor = new IoProcessor();
            try {
                clientIoProcessor.start();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            SimpleEventDispatcher clientEventDispatcher = new SimpleEventDispatcher();
            clientEventDispatcher
```

```
                    .setThreadPoolSize(DISPATCHER_THREAD_POOL_SIZE);
            clientEventDispatcher.start();
            ErtMessageRecognizer clientRecognizer = new ErtMessageRecognizer();
            NodeCommunicationClientListener clientListener = new NodeCommunicationClientListener();

            cachedSession = new Session(clientIoProcessor,
                    new InetSocketAddress(ip, SERVER_PORT), clientRecognizer,
                    clientEventDispatcher);
            // set configuration
            cachedSession.getConfig().setConnectTimeout(CONNECT_TIMEOUT);
            // suscribe and start communication
            cachedSession.addSessionListener(clientListener);
            SessionManager.addAddressForSession(
                    ((InetSocketAddress) cachedSession.getSocketAddress())
                            .getAddress().getHostAddress(), cachedSession);
            //            logger.debug("Connecting to "
            //                    + ((InetSocketAddress) cachedSession.getSocketAddress())
            //                            .getAddress().getHostAddress());
            cachedSession.start();
            try {
                while (!cachedSession.isConnected()) {
                        Thread.sleep(500);
                }
            } catch (InterruptedException e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();
                logger.error("Interrupt: " + e1);
            }
        }
        return cachedSession;
    }

}
```

File 19: communication.netty2.NodeCommunicationClientListener

```
/*
 * Created on Oct 21, 2004 7:36:59 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-11-12 12:46:25 +0100 (Fri, 12 Nov 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 173 $
 * Location:    $URL: NodeCommunicationClientListener.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.communication.netty2;

import java.net.InetSocketAddress;

import net.gleamynode.netty2.Message;
import net.gleamynode.netty2.Session;
import net.gleamynode.netty2.SessionListener;

import org.apache.log4j.Logger;

import ch.nix.ert.base.Dispatcher;
import ch.nix.ert.base.ServiceManager;
import ch.nix.ert.message.DataItem;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class NodeCommunicationClientListener implements SessionListener {

    // logger:
    static Logger logger = Logger
            .getLogger(NodeCommunicationClientListener.class);

    private Dispatcher dispatcher;

    public NodeCommunicationClientListener() {
        dispatcher = ServiceManager.getInstance().getDispatcher();
    }

    /*
     * (non-Javadoc)
     *
     * @see net.gleamynode.netty2.SessionListener#
     * connectionEstablished(net.gleamynode.netty2.Session)
     */
    public void connectionEstablished(Session session) {
        logger.debug("netty2 connection established: " + session);

        // set idle time to 60 seconds
        session.getConfig().setIdleTime(60);
        SessionManager.addAddressForSession(((InetSocketAddress) session
                .getSocketAddress()).getAddress().getHostAddress(), session);
    }

    /*
     * (non-Javadoc)
     *
     * @see net.gleamynode.netty2.SessionListener#connectionClosed(net.gleamynode.netty2.Session)
     */
    public void connectionClosed(Session session) {
        logger.debug("netty2 connection closed: " + session);
        SessionManager.removeAddressForSession(((InetSocketAddress) session
                .getSocketAddress()).getAddress().getHostAddress());
    }

    /*
     * (non-Javadoc)
     *
     * @see net.gleamynode.netty2.SessionListener#messageReceived(net.gleamynode.netty2.Session,
     *      net.gleamynode.netty2.Message)
     */
    public void messageReceived(Session session, Message message) {
        DataItem dataItem = ((ErtMessage) message).getDataItem();
        // logger.debug("Recveived <- : " + dataItem);
        dataItem.setRouting("local");
        dispatcher.addWork(dataItem);
    }

    /*
     * (non-Javadoc)
     *
     * @see net.gleamynode.netty2.SessionListener#messageSent(net.gleamynode.netty2.Session,
     *      net.gleamynode.netty2.Message)
     */
    public void messageSent(Session session, Message message) {
        //        DataItem dataItem = ((ErtMessage) message).getDataItem();
        //        logger.debug("Sending -> "
        //                + ((InetSocketAddress) session.getSocketAddress()).getAddress()
        //                        .getHostAddress() + " : " + dataItem);

    }

    /*
     * (non-Javadoc)
     *
     * @see net.gleamynode.netty2.SessionListener#sessionIdle(net.gleamynode.netty2.Session)
     */
    public void sessionIdle(Session session) {
        //  logger.info("Disconnecting the idle session: " + session);

        SessionManager.removeAddressForSession(((InetSocketAddress) session
                .getSocketAddress()).getAddress().getHostAddress());
        session.close();
    }

    /*
     * (non-Javadoc)
     *
     * @see net.gleamynode.netty2.SessionListener#exceptionCaught(net.gleamynode.netty2.Session,
     *      java.lang.Throwable)
     */
    public void exceptionCaught(Session session, Throwable cause) {
        // cause.printStackTrace();
        logger.error("Unexpected exception: " + cause + "\n" + " in session: "
                + session);
        SessionManager.removeAddressForSession(((InetSocketAddress) session
                .getSocketAddress()).getAddress().getHostAddress());
        session.close();

    }

}
```

File 20: communication.netty2.NodeCommunicationServerListener

```java
/*
 * Created on Oct 21, 2004 10:20:43 AM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-11-12 12:46:25 +0100 (Fri, 12 Nov 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 173 $
 * Location:    $URL: NodeCommunicationServerListener.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.communication.netty2;

import java.net.ConnectException;
import java.net.InetSocketAddress;

import org.apache.log4j.Logger;

import com.sleepycat.util.RuntimeExceptionWrapper;

import ch.nix.ert.base.Dispatcher;
import ch.nix.ert.base.ServiceManager;
import ch.nix.ert.message.DataItem;
import net.gleamynode.netty2.Message;
import net.gleamynode.netty2.Session;
import net.gleamynode.netty2.SessionListener;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class NodeCommunicationServerListener implements SessionListener {

    // logger:
    static Logger logger = Logger
            .getLogger(NodeCommunicationServerListener.class);

    private Dispatcher dispatcher;

    public NodeCommunicationServerListener() {
        dispatcher = ServiceManager.getInstance().getDispatcher();
    }

    /*
     * (non-Javadoc)
     *
     * @see net.gleamynode.netty2.SessionListener#
     * connectionEstablished(net.gleamynode.netty2.Session)
     */
    public void connectionEstablished(Session session) {
        logger.debug("netty2 connection established: " + session);
        // set idle time to 60 seconds
        session.getConfig().setIdleTime(60);
        SessionManager.addAddressForSession(((InetSocketAddress) session
                .getSocketAddress()).getAddress().getHostAddress(), session);
    }

    /*
     * (non-Javadoc)
     *
     * @see net.gleamynode.netty2.SessionListener#
     * connectionClosed(net.gleamynode.netty2.Session)
     */
    public void connectionClosed(Session session) {
        logger.debug("netty2 connection closed: " + session);
        SessionManager.removeAddressForSession(((InetSocketAddress) session
                .getSocketAddress()).getAddress().getHostAddress());
    }

    /*
     * (non-Javadoc)
     *
     * @see net.gleamynode.netty2.SessionListener#
     * messageReceived(net.gleamynode.netty2.Session,
     *      net.gleamynode.netty2.Message)
     */
    public void messageReceived(Session session, Message message) {
        DataItem dataItem = ((ErtMessage) message).getDataItem();
        // logger.debug("Recveived <- : " + dataItem);
        dataItem.setRouting("local");
        dispatcher.addWork(dataItem);
    }

    /*
     * (non-Javadoc)
     *
     * @see net.gleamynode.netty2.SessionListener#
     * messageSent(net.gleamynode.netty2.Session,
     *      net.gleamynode.netty2.Message)
     */
    public void messageSent(Session session, Message message) {
        //        DataItem dataItem = ((ErtMessage) message).getDataItem();
        //        logger.debug("Sending -> "
        //                + ((InetSocketAddress) session.getSocketAddress()).getAddress()
        //                        .getHostAddress() + " : " + dataItem);

    }

    /*
     * (non-Javadoc)
     *
     * @see net.gleamynode.netty2.SessionListener#sessionIdle(net.gleamynode.netty2.Session)
     */
    public void sessionIdle(Session session) {
        logger.info("Disconnecting the idle session: " + session);
        SessionManager.removeAddressForSession(((InetSocketAddress) session
                .getSocketAddress()).getAddress().getHostAddress());
        session.close();
    }

    /*
     * (non-Javadoc)
     *
     * @see net.gleamynode.netty2.SessionListener#
     * exceptionCaught(net.gleamynode.netty2.Session,
     *      java.lang.Throwable)
     */
    public void exceptionCaught(Session session, Throwable cause) {
        cause.printStackTrace();
        logger.error("Unexpected exception: " + cause + "\n" + " in session: "
                + session);
        if (cause instanceof RuntimeExceptionWrapper) {
            Throwable causeDetails = ((RuntimeExceptionWrapper) cause)
                    .getDetail();
            logger.error("  Details: " + causeDetails);
            logger.error("  Message: " + causeDetails.getMessage());
        }

        if (cause instanceof ConnectException) {
            // failed to connect to the server, retry after a while.
            try {
                Thread.sleep(5000);
            } catch (InterruptedException e) {
            }
            session.start();
        } else {
            SessionManager.removeAddressForSession(((InetSocketAddress) session
                    .getSocketAddress()).getAddress().getHostAddress());
            session.close();
        }

    }

}
```

File 21: communication.netty2.SessionManager

```
/*
 * Created on Oct 21, 2004 1:39:20 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-26 15:28:00 +0200 (Tue, 26 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 138 $
 * Location:    $URL: SessionManager.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
```

```java
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.communication.netty2;

import java.util.Hashtable;

import ch.nix.ert.message.Node;

import net.gleamynode.netty2.Session;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class SessionManager {

    private static Hashtable sessionHash = new Hashtable();

    public static Session getSessionForAddress(String ip) {
        return (Session) sessionHash.get(ip);
    }

    public static void addAddressForSession(String ip, Session session) {
        sessionHash.put(ip, session);
    }

    public static void removeAddressForSession(String ip) {
        sessionHash.remove(ip);
    }

    public static int getSize() {
        return sessionHash.size();
    }


}
```

File 22: communication.rmi.NodeCommunication

```java
/*
 * Created on Aug 5, 2004 3:28:51 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-26 15:28:00 +0200 (Tue, 26 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 138 $
 * Location:    $URL: NodeCommunication.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.communication.rmi;

import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.ExportException;

import org.apache.log4j.Logger;

import ch.nix.ert.base.Dispatcher;
import ch.nix.ert.base.ServiceManager;
import ch.nix.ert.communication.NodeCommunicationInterface;
import ch.nix.ert.communication.NodeDirectory;
import ch.nix.ert.message.DataItem;
import ch.nix.ert.message.Node;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
// To do the RMI stub building, do:
// cd /Users/michi/Documents/eclipse/workspace/ExtendendRootTree/bin
// rmic ch.nix.ert.communication.rmi.NodeCommunicationRmi
public class NodeCommunication implements NodeCommunicationInterface {

    static Logger logger = Logger.getLogger(NodeCommunication.class);

    NodeCommunicationRmi communicationObject;

    NodeDirectory nodeDirectory = NodeDirectory.getInstance();

    private Dispatcher dispatcher;

    /**
     *
     */
    public NodeCommunication() {
        Registry reg;
        try {
            try {
                reg = LocateRegistry.createRegistry(1099); // registry
            } catch (ExportException e) {
                reg = LocateRegistry.getRegistry(1099);
            }
            dispatcher = ServiceManager.getInstance().getDispatcher();
            communicationObject = new NodeCommunicationRmi(this);
            communicationObject.startServer();
        } catch (RemoteException e) {
            // TODO Auto-generated catch block
            System.out.println("Generation of RMI failed" + e);
        }
    }

    public void send(DataItem dataItem) {
        String ip = "";
        // now check where the dataItem has to be sent to
        // first check if we have a "directdelivery" packet, which does not need
        // to be routed
        Node node = dataItem.getDirectDestination();
        if (node == null) {
            // no indication that we have a direct delivery packet, so we have
            // to lookup the dataItem
            DataItem destination = nodeDirectory.findNode(dataItem);
            if (destination == null) {
                logger.warn("Could not send (unknown destination): "+dataItem);
                return;
            }
            Node destinationNode = destination.getDirectDestination();
            if (destinationNode.isLocalhost()) {
                // means we would send this dataItem to ourselfs, so we return it here already!
                dataItem.setRouting("local");
                dispatcher.addWork(dataItem);
                return;
            }
            ip = destinationNode.getIp();

        } else {
            ip = node.getIp();
            // we don't want to send over this information
            // TODO later we should write an cleanup method into DataItem which
            // automatically removes all unused parameters bevore sending it or
            // storing it...
            dataItem.removeDirectDestination();
        }

        NodeCommunicationRmiInterface nodeCommunication;

        //now add originator Address:
        dataItem.setDirectOriginator(new Node(1099));

        try {
            nodeCommunication = (NodeCommunicationRmiInterface) Naming
                    .lookup("rmi://" + ip + "/NodeCommunicationRmi");
            // executes the remote "receive" method to acutally send this dataItem
            //System.out.println("Data sendingRMI ->: "+ dataItem);
            logger.debug("Sending -> "+ip+" : "+dataItem);
            nodeCommunication.receive(dataItem);
        } catch (MalformedURLException e) {
            logger.warn("Could not send: "+dataItem+"\n"+e);
        } catch (RemoteException e) {
            logger.warn("Could not send: "+dataItem+"\n"+e);
        } catch (NotBoundException e) {
            logger.warn("Could not send: "+dataItem+"\n"+e);
        }
    }
}
```

File 23: communication.rmi.NodeCommunicationRmi

```java
/*
 * Created on Aug 5, 2004 4:20:49 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-26 15:28:00 +0200 (Tue, 26 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 138 $
 * Location:    $URL: NodeCommunicationRmi.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.communication.rmi;

import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

import org.apache.log4j.Logger;

import ch.nix.ert.base.Dispatcher;
import ch.nix.ert.base.ServiceManager;
import ch.nix.ert.message.DataItem;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class NodeCommunicationRmi extends UnicastRemoteObject implements
        NodeCommunicationRmiInterface {

    static Logger logger = Logger.getLogger(NodeCommunicationRmi.class);

    NodeCommunication nodeCommunication;

    private Dispatcher dispatcher;

    /**
     * @throws RemoteException
     */
    public NodeCommunicationRmi(NodeCommunication myNodeCommunication)
            throws RemoteException {
        nodeCommunication = myNodeCommunication;
        //write the responses directly to the Dispatcher main queue:
        dispatcher = ServiceManager.getInstance().getDispatcher();
    }

    /*
     * (non-Javadoc)
     *
     * @see ch.nix.ert.NodeCommunicationRmiInterface#transmit(ch.nix.ert.DataItem)
     */
    public void receive(DataItem dataItem) throws RemoteException {
        // This will be executed, when remote site was sending us an object
        //System.out.println("Comm Received: "+dataItem);
        logger.debug("Recveived <- : "+dataItem);
        dataItem.setRouting("local");
        dispatcher.addWork(dataItem);
    }

    public void startServer() {

        try {
            Naming.rebind("NodeCommunicationRmi", NodeCommunicationRmi.this);
            //System.out.println ("NodeCommunication Server is ready.");
        } catch (MalformedURLException e) {
            // TODO Auto-generated catch block
            System.out.println("Error in NodeCommunicationRmi, Naming.bind: "
                    + e);
        } catch (RemoteException e) {
            // TODO Auto-generated catch block
            System.out.println("Error in NodeCommunicationRmi, Naming.bind: "
                    + e);
        }
    }
}
```

File 24: communication.rmi.NodeCommunicationRmiInterface

```java
/*
 * Created on Aug 5, 2004 4:21:16 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-26 15:28:00 +0200 (Tue, 26 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 138 $
 * Location:    $URL: NodeCommunicationRmiInterface.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.communication.rmi;

import java.rmi.Remote;
import java.rmi.RemoteException;

import ch.nix.ert.message.DataItem;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public interface NodeCommunicationRmiInterface extends Remote {

    public void receive(DataItem dataItem) throws RemoteException;

}
```

File 25: datastore.DataStore

```java
/*
 * Created on Aug 10, 2004 12:28:46 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-11-11 18:53:50 +0100 (Thu, 11 Nov 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 168 $
 * Location:    $URL: DataStore.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.datastore;

import ch.nix.ert.message.DataItem;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class DataStore implements DataStoreInterface {

    static private String plugin = "berkeleyje";

    private DataStoreInterface delegate = null;

    public DataStore(String instanceId) {
        try {
            delegate = (DataStoreInterface) (Class
                    .forName("ch.nix.ert.datastore." + plugin + ".DataStore")
                    .newInstance());
            delegate.setDatabaseId(instanceId);
        } catch (InstantiationException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            // TODO Auto-generated catch block
```

```
                e.printStackTrace();
            }
        }

        /*
         * (non-Javadoc)
         *
         * @see ch.nix.ert.datastore.DataStoreInterface#getDataItem(ch.nix.ert.DataItem)
         */
        public DataItem getDataItem(DataItem data) {
            // TODO Auto-generated method stub
            return delegate.getDataItem(data);
        }

        /*
         * (non-Javadoc)
         *
         * @see ch.nix.ert.datastore.DataStoreInterface#putDataItem(ch.nix.ert.DataItem)
         */
        public boolean putDataItem(DataItem data) {
            // TODO Auto-generated method stub
            return delegate.putDataItem(data);
        }

        /*
         * (non-Javadoc)
         *
         * @see ch.nix.ert.datastore.DataStoreInterface#deleteDataItem(ch.nix.ert.DataItem)
         */
        public boolean deleteDataItem(DataItem data) {
            // TODO Auto-generated method stub
            return delegate.deleteDataItem(data);
        }

        public int getDbSize() {
            return delegate.getDbSize();
        }

        /*
         * (non-Javadoc)
         *
         * @see ch.nix.ert.datastore.DataStoreInterface#getDatabaseId()
         */
        public String getDatabaseId() {
            // TODO Auto-generated method stub
            return delegate.getDatabaseId();
        }

        public void setDatabaseId(String id) {
            //do nothing as setting the Database id should not be allowed!
        }

        /*
         * (non-Javadoc)
         *
         * @see ch.nix.ert.datastore.DataStoreInterface#getNextDataItem(ch.nix.ert.DataItem,
         *      java.lang.String, int)
         */
        public DataItem getNextDataItem(DataItem data) {
            // TODO Auto-generated method stub
            return delegate.getNextDataItem(data);
        }

        /*
         * (non-Javadoc)
         *
         * @see ch.nix.ert.datastore.DataStoreInterface#getPreviousDataItem(ch.nix.ert.DataItem,
         *      java.lang.String, int)
         */
        public DataItem getPreviousDataItem(DataItem data) {
            // TODO Auto-generated method stub
            return delegate.getPreviousDataItem(data);
        }

}
```

File 26: datastore.DataStoreInterface

```
/*
 * Created on Aug 10, 2004 12:31:08 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-26 15:43:26 +0200 (Tue, 26 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 139 $
 * Location:    $URL: DataStoreInterface.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.datastore;

import ch.nix.ert.message.DataItem;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public interface DataStoreInterface {

    public DataItem getNextDataItem(DataItem data);

    public DataItem getPreviousDataItem(DataItem data);

    public DataItem getDataItem(DataItem data);

    public boolean putDataItem(DataItem data);

    public boolean deleteDataItem(DataItem data);

   // public void deleteDataItemRange(DataItem data);

    public int getDbSize();

    /**
     * If database is still closed, <code>databaseId</code> is null
     * <p>
     * If database is opened and running, <code>databaseId</code> contains an
     * unique Id (namespace like "ch.nix.accountDB")
     *
     * @return Returns the databaseId.
     */
    public String getDatabaseId();

    public void setDatabaseId(String databaseId);

}
```

File 27: datastore.DataStoreManager

```
/*
 * Created on Jul 28, 2004 4:59:10 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-26 15:43:26 +0200 (Tue, 26 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 139 $
 * Location:    $URL: DataStoreManager.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */

package ch.nix.ert.datastore;

import java.util.Hashtable;

import ch.nix.ert.datastore.DataStore;
import ch.nix.ert.message.DataItem;

/**
 *
 * TODO Edit this text for a better description Manages all DataStore objects.
 * Each DataStore object is responsible for his own namespace <br>
 *
 * Follows the Singelton-Design Pattern after Rodney Waldhoff.
 * http://radio.weblogs.com/0122027/stories/2003/10/20/implementingTheSingletonPatternInJava.html
 *
 * @author michi
 */
public class DataStoreManager {

    private Hashtable dataStores = new Hashtable();

    static private DataStoreManager instance = null;

    private DataStoreManager() {
    }

    /**
     * @return The unique instance of this class.
     */
    static public synchronized DataStoreManager getInstance() {
        if (null == instance) {
            instance = new DataStoreManager();
        }
        return instance;
    }

    public boolean put(DataItem data) {
        //check if namespace already exists
        String myNamespace = (String) data.getNamespace();
        if (dataStores.get(myNamespace) == null) {
            //namespace does no exist, I need to create a new DataStore first
            dataStores.put(myNamespace, new DataStore(myNamespace));
        }
        DataStore currentDataStore = (DataStore) dataStores.get(myNamespace);
        return currentDataStore.putDataItem(data);
    }

    public DataItem get(DataItem data) {
        DataStore currentDataStore = (DataStore) dataStores.get(data
                .getNamespace());
        if (currentDataStore == null) {
            return null;
        }
        return currentDataStore.getDataItem(data);
    }

    public boolean delete(DataItem data) {
        DataStore currentDataStore = (DataStore) dataStores.get(data
                .getNamespace());
        if (currentDataStore == null) {
            return false;
        }
        return currentDataStore.deleteDataItem(data);
    }

    public DataItem getNext(DataItem data) {
        DataStore currentDataStore = (DataStore) dataStores.get(data
                .getNamespace());
        if (currentDataStore == null) {
            return null;
        }
        return currentDataStore.getNextDataItem(data);
    }

    public DataItem getPrevious(DataItem data) {
        DataStore currentDataStore = (DataStore) dataStores.get(data
                .getNamespace());
        if (currentDataStore == null) {
            return null;
        }
        return currentDataStore.getPreviousDataItem(data);
    }

    public int getDbSize(DataItem data) {
        DataStore currentDataStore = (DataStore) dataStores.get(data
                .getNamespace());
        if (currentDataStore == null) {
            return 0;
        }
        return currentDataStore.getDbSize();
    }

}
```

File 28: datastore.DataStoreService

```
/*
```

```java
 * Created on Aug 13, 2004 3:02:42 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-28 22:38:52 +0200 (Thu, 28 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 144 $
 * Location:    $URL: DataStoreService.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.datastore;

import org.apache.log4j.Logger;

import ch.nix.ert.base.ServiceBase;
import ch.nix.ert.message.DataItem;
import ch.nix.ert.message.Node;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class DataStoreService extends ServiceBase {

    // logger:
    static Logger logger = Logger.getLogger(DataStoreService.class);

    static private DataStoreService instance = null;

    private DataStoreManager dataStoreManager;

    // private Queue serviceQueue = new Queue("DataStoreQueue");

    // private Dispatcher dispatcher;

    private DataStoreService() {
    }

    static public synchronized DataStoreService getInstance() {
        if (null == instance) {
            instance = new DataStoreService();
            // instance.dispatcher =
            // ServiceManager.getInstance().getDispatcher();
            instance.dataStoreManager = DataStoreManager.getInstance();
            logger.info("DataStoreService started");
        }
        return instance;
    }

    //    private Dispatcher getDispatcher() {
    //        return dispatcher;
    //    }

    /*
     * (non-Javadoc)
     *
     * @see ch.nix.ert.ServiceInterface#registerAtDispatcher()
     */
    public void registerAtDispatcher() {
        registerService("putdb");
        registerService("putdbnoack");
        registerService("getdb");
        registerService("getnextdb");
        registerService("getprevdb");
        registerService("deletedb");
        registerService("getsizedb");
    }

    /*
     * (non-Javadoc)
     *
     * @see java.lang.Runnable#run()
     */
    public void run() {
        //System.out.println("DataStoreService started");
        Thread.currentThread().setName("ERT.DataStoreService");
        while (true) {
            DataItem dataItem = null;

            dataItem = getWork();

            // System.out.println("got request: " + dataItem);
            String operation = dataItem.getOperation();
            if (operation.equals("getdb")) {
              //  logger.debug("getDB: " + dataItem);
                Node originator = dataItem.getDirectOriginator();
                DataItem response = dataStoreManager.get(dataItem);
                if (response == null) {
                    response = new DataItem();
                }
                response.setOperation("getdb");
                response.setDirectDestination(originator);
                transactionEnd(response);
              //  logger.debug("getDBResponse: " + response);
            } else if (operation.equals("getnextdb")) {
              //  logger.debug("getNextDB: " + dataItem);
                Node originator = dataItem.getDirectOriginator();
                DataItem response = dataStoreManager.getNext(dataItem);
                if (response == null) {
                    response = new DataItem();
                }
                response.setOperation("getnextdb");
                response.setDirectDestination(originator);
                transactionEnd(response);
              //  logger.debug("getNextDBResponse: " + response);
            } else if (operation.equals("getprevdb")) {
              //  logger.debug("getPrevDB: " + dataItem);
                Node originator = dataItem.getDirectOriginator();
                DataItem response = dataStoreManager.getPrevious(dataItem);
                if (response == null) {
                    response = new DataItem();
                }
                response.setOperation("getprevdb");
                response.setDirectDestination(originator);
                transactionEnd(response);
              //  logger.debug("getPrevDBResponse: " + response);
            } else if (operation.equals("putdb")) {
                // logger.debug("putDB: " + dataItem);
                boolean outcome = dataStoreManager.put(dataItem);
                DataItem response = new DataItem();
                response.setResult(outcome);
                transactionEnd(response);
                // logger.debug("putDBResponse"+outcome+": " + response);
            } else if (operation.equals("putdbnoack")) {
              //  logger.debug("putDB: " + dataItem);
                dataStoreManager.put(dataItem);
            }else if (operation.equals("deletedb")) {
              //  logger.debug("deleteDB: " + dataItem);
                boolean outcome = dataStoreManager.delete(dataItem);
                dataItem.setResult(outcome);
                transactionEnd(dataItem);
              //  logger.debug("deleteDBResponse: " + dataItem);
            } else if (operation.equals("getsizedb")) {
              //  logger.debug("getSizeDB: " + dataItem);
                int size = dataStoreManager.getDbSize(dataItem);
                DataItem response = new DataItem();
                response.setDbSize(size);
                response.setOperation("getsizedb");
                transactionEnd(response);
              //  logger.debug("getSizeDBResponse: " + response);
            } else {
                logger
                        .warn("DataItem dropped, because no processing was found for: "
                                + dataItem);
            }
        }
    }

}
```

File 29: datastore.berkeleyje.DataStore

```
/*
 * Created on Jul 27, 2004 11:49:54 AM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-11-12 12:46:25 +0100 (Fri, 12 Nov 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 173 $
 * Location:    $URL: DataStore.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
```

```java
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */

package ch.nix.ert.datastore.berkeleyje;

import ch.nix.ert.datastore.DataStoreInterface;
import ch.nix.ert.message.DataItem;
import ch.nix.ert.message.UserKey;

import com.sleepycat.bind.EntryBinding;
import com.sleepycat.bind.serial.SerialBinding;
import com.sleepycat.bind.serial.StoredClassCatalog;
import com.sleepycat.bind.tuple.TupleBinding;
import com.sleepycat.je.Cursor;
import com.sleepycat.je.DatabaseException;
import com.sleepycat.je.Database;
import com.sleepycat.je.DatabaseEntry;
import com.sleepycat.je.DatabaseStats;
import com.sleepycat.je.LockMode;
import com.sleepycat.je.OperationStatus;
import com.sleepycat.je.DatabaseConfig;
import com.sleepycat.je.Environment;
import com.sleepycat.je.EnvironmentConfig;
import com.sleepycat.je.StatsConfig;

import java.io.File;

import org.apache.log4j.Logger;

/**
 * @author michi
 *
 * TODO Edit this text for a better description Responsible for a namespace.
 * TODO Extend this class to allow range queries and browsing!
 * <p>
 * During first performance testing, this class should allow to store more than
 * 4000 objects per second (variance around 4000 - 8000) Each namespace holds
 * it's own database
 */
public class DataStore implements DataStoreInterface {

    // logger:
    static Logger logger = Logger.getLogger(DataStore.class);

    private String databaseId;

    private int syncCounter = 0;

    //TODO make configurable in a later version
    private String databasePath = "/Users/tutor/ert/db";

    private int requestsSync = 50;

    //private Hashtable cursorTable = new Hashtable();

    Environment myDbEnvironment;

    Database myDatabase;

    //EntryBinding myDataBinding;

    EntryBinding myKeyBinding;

    /**
     *
     */
    public DataStore() {
        super();
    }

    public void openDatabase() {
        try {
            // Open the environment. Create it if it does not alreadyexist.
            EnvironmentConfig envConfig = new EnvironmentConfig();
            envConfig.setAllowCreate(true);
            myDbEnvironment = new Environment(new File(databasePath), envConfig);

            // Open the database. Create it if it does not already exist.
            DatabaseConfig dbConfig = new DatabaseConfig();
            // set the Comparator Method:
            //              dbConfig.setBtreeComparator(Long.class);
            //              dbConfig.setDuplicateComparator(Long.class);
            dbConfig.setAllowCreate(true);
            dbConfig.setSortedDuplicates(false);
            myDatabase = myDbEnvironment.openDatabase(null, this
                    .getDatabaseId(), dbConfig);

            // Open the database that you use to store your class information.
            //myClassDb = myDbEnvironment.openDatabase(null, "classDb",
            // dbConfig);

            // Instantiate the class catalog
            //StoredClassCatalog classCatalog = new
            // StoredClassCatalog(myClassDb);

            // Create the binding
            //myDataBinding = new SerialBinding(classCatalog, DataItem.class);
            myKeyBinding = TupleBinding.getPrimitiveBinding(Long.class);
            logger.info("Berkeley Database opened");
        } catch (DatabaseException dbe) {
            // Exception handling goes here
            logger.error("Error opening DB: " + dbe);
        }
    }

    public void closeDatabase() {
        try {
            //close all open cursors:
            //            for (Enumeration e = cursorTable.elements();
            // e.hasMoreElements();) {
            //                ((Cursor) e.nextElement()).close();
            //            }

            if (myDatabase != null) {
                myDatabase.close();
            }
            //            if (myClassDb != null) {
            //                myClassDb.close();
            //            }
            if (myDbEnvironment != null) {
                myDbEnvironment.sync();
                myDbEnvironment.close();
            }
            logger.info("Berkeley Database closed");
        } catch (DatabaseException dbe) {
            // Exception handling goes here
            logger.error("Error closing DB: " + dbe);
        }

    }

    /*
     * (non-Javadoc)
     *
     * @see java.lang.Object#finalize()
     */
    protected void finalize() throws Throwable {
        try {
            closeDatabase(); // close open database
        } finally {
            super.finalize();
        }
    }

    public DataItem getNextDataItem(DataItem data) {

        int amount = data.getCursorSize();
        UserKey key = data.getUserKey();

        //check, if connection to database is already open
        if (myDatabase == null) {
            this.openDatabase();
        }
        try {
            // first create a cursor
            Cursor cursor = myDatabase.openCursor(null, null);

            // Create a pair of DatabaseEntry objects. theKey
            // is used to perform the search. theData is used
            // to store the data returned by the get() operation.
            DatabaseEntry theKey = new DatabaseEntry();
            DatabaseEntry theData = new DatabaseEntry();

            myKeyBinding.objectToEntry(new Long(key.getId()), theKey);

            if (cursor.getSearchKeyRange(theKey, theData, LockMode.DEFAULT)
                    == OperationStatus.SUCCESS) {
```

```
                // Recreate the data & key
                //                DataItem retrievedData = (DataItem) myDataBinding
                //                        .entryToObject(theData);
                DataItem retrievedData = new DataItem();
                retrievedData.setData(new String(theData.getData()));
                UserKey retrievedKey = (UserKey) new UserKey(
                        (Long) myKeyBinding.entryToObject(theKey));

                //add the namespace attribute to DataItem
                retrievedData.setNamespace(this.getDatabaseId());
                retrievedData.setUserKey(retrievedKey);

                // now check, that we retrieve the next key and not the value
                // for the current key
                if (key.equals(retrievedKey)) {
                    //means by coincidence the searching key was exactly a
                    // matching key in db, so we have to retrieve the next
                    // key...
                    OperationStatus retVal = cursor.getNext(theKey, theData,
                            LockMode.DEFAULT);
                    if (retVal == OperationStatus.SUCCESS) {
                        //                        retrievedData = (DataItem) myDataBinding
                        //                                .entryToObject(theData);
                        retrievedData.setData(new String(theData.getData()));
                        retrievedKey = (UserKey) new UserKey(
                                (Long) myKeyBinding.entryToObject(theKey));
                        // add the namespace attribute to DataItem
                        retrievedData.setNamespace(this.getDatabaseId());
                        retrievedData.setUserKey(retrievedKey);
                    } else {
                        return null;
                    }
                }

                // if user requested more than one dataitem, we have to use
                // the cursor functionality and put all remaining data
                // into "dataitems" field
                for (int i = 1; i < amount; i++) {
                    DataItem moreRetrievedData;
                    UserKey moreRetrievedKey;
                    OperationStatus retVal = cursor.getNext(theKey, theData,
                            LockMode.DEFAULT);
                    if (retVal == OperationStatus.SUCCESS) {
                        //                        moreRetrievedData = (DataItem) myDataBinding
                        //                                .entryToObject(theData);
                        moreRetrievedData = new DataItem();
                        moreRetrievedData
                                .setData(new String(theData.getData()));
                        moreRetrievedKey = (UserKey) new UserKey(
                                (Long) myKeyBinding.entryToObject(theKey));
                        // add the namespace attribute to DataItem
                        moreRetrievedData.setNamespace(this.getDatabaseId());
                        moreRetrievedData.setUserKey(moreRetrievedKey);
                        // add the additional dataItem into retreivedData
                        // from before:
                        retrievedData.addDataItems(moreRetrievedData);
                    } else {
                        // do nothing
                    }

                }
                cursor.close();
                return retrievedData;
            } else {
                //System.out.println("ERROR 1");
                return null;
            }
        } catch (Exception e) {
            // TODO Auto-generated catch block
            // Exception handling goes here
            logger.error("Get: Error hapened: " + e);
            return null;
        }
    }

    public DataItem getPreviousDataItem(DataItem data) {

        int amount = data.getCursorSize();
        UserKey key = data.getUserKey();

        //check, if connection to database is already open
        if (myDatabase == null) {
            this.openDatabase();
        }
        try {
```

```
        // first create a cursor
        Cursor cursor = myDatabase.openCursor(null, null);

        // Create a pair of DatabaseEntry objects. theKey
        // is used to perform the search. theData is used
        // to store the data returned by the get() operation.
        DatabaseEntry theKey = new DatabaseEntry();
        DatabaseEntry theData = new DatabaseEntry();

        myKeyBinding.objectToEntry(new Long(key.getId()), theKey);
        OperationStatus status = OperationStatus.SUCCESS;
        if (cursor.getSearchKeyRange(theKey, theData, LockMode.DEFAULT)
                != OperationStatus.SUCCESS) {
            // means we are at the end of the database, so now get the last
            // entry of the whole db
            status = cursor.getLast(theKey, theData, LockMode.DEFAULT);
        } else {
            // now we are one item to high, so we have the get the previous
            // one...
            status = cursor.getPrev(theKey, theData, LockMode.DEFAULT);
        }
        if (status == OperationStatus.SUCCESS) {
            // Recreate the data & key
            //                DataItem retrievedData = (DataItem) myDataBinding
            //                        .entryToObject(theData);
            DataItem retrievedData = new DataItem();
            retrievedData.setData(new String(theData.getData()));
            UserKey retrievedKey = (UserKey) new UserKey(
                    (Long) myKeyBinding.entryToObject(theKey));

            //add the namespace attribute to DataItem
            retrievedData.setNamespace(this.getDatabaseId());
            retrievedData.setUserKey(retrievedKey);

            // now check, that we retrieve the next key and not the value
            // for the current key
            if (key.equals(retrievedKey)) {
                //means by coincidence the searching key was exactly a
                // matching key in db, so we have to retrieve the next
                // key...
                OperationStatus retVal = cursor.getPrev(theKey, theData,
                        LockMode.DEFAULT);
                if (retVal == OperationStatus.SUCCESS) {
                    //                        retrievedData = (DataItem) myDataBinding
                    //                                .entryToObject(theData);
                    retrievedData.setData(new String(theData.getData()));
                    retrievedKey = (UserKey) new UserKey(
                            (Long) myKeyBinding.entryToObject(theKey));
                    // add the namespace attribute to DataItem
                    retrievedData.setNamespace(this.getDatabaseId());
                    retrievedData.setUserKey(retrievedKey);
                } else {
                    return null;
                }
            }

            // if user requested more than one dataitem, we have to use
            // the cursor functionality and put all remaining data
            // into "dataitems" field
            for (int i = 1; i < amount; i++) {
                DataItem moreRetrievedData;
                UserKey moreRetrievedKey;
                OperationStatus retVal = cursor.getPrev(theKey, theData,
                        LockMode.DEFAULT);
                if (retVal == OperationStatus.SUCCESS) {
                    moreRetrievedData = new DataItem();
                    //                        moreRetrievedData = (DataItem) myDataBinding
                    //                                .entryToObject(theData);
                    moreRetrievedData
                            .setData(new String(theData.getData()));
                    moreRetrievedKey = (UserKey) new UserKey(
                            (Long) myKeyBinding.entryToObject(theKey));
                    // add the namespace attribute to DataItem
                    moreRetrievedData.setNamespace(this.getDatabaseId());
                    moreRetrievedData.setUserKey(moreRetrievedKey);
                    // add the additional dataItem into retreivedData
                    // from before:
                    retrievedData.addDataItems(moreRetrievedData);
                } else {
                    // do nothing
                }

            }
            cursor.close();
            return retrievedData;
```

```java
            } else {
                // System.out.println("ERROR 1");
                return null;
            }
        } catch (Exception e) {
            // TODO Auto-generated catch block
            // Exception handling goes here
            logger.error("Get: Error hapened: " + e);
            return null;
        }
    }

    public DataItem getDataItem(DataItem data) {
        //extract key from data
        long key = data.getUserKey().getId();
        //check, if connection to database is already open
        if (myDatabase == null) {
            this.openDatabase();
        }
        try {
            // Create a pair of DatabaseEntry objects. theKey
            // is used to perform the search. theData is used
            // to store the data returned by the get() operation.
            DatabaseEntry theKey = new DatabaseEntry();
            DatabaseEntry theData = new DatabaseEntry();
            myKeyBinding.objectToEntry(new Long(key), theKey);

            // Perform the get.
            if (myDatabase.get(null, theKey, theData, LockMode.DEFAULT)
                    == OperationStatus.SUCCESS) {
                // Recreate the data.
                //                DataItem retrievedData = (DataItem) myDataBinding
                //                        .entryToObject(theData);
                DataItem retrievedData = new DataItem();
                retrievedData.setData(new String(theData.getData()));
                //add the namespace attribute to DataItem
                retrievedData.setNamespace(this.getDatabaseId());
                retrievedData.setUserKey(new UserKey(key));
                return retrievedData;
            } else {
                return null;
            }
        } catch (Exception e) {
            // Exception handling goes here
            logger.error("Get: Error hapened: " + e);
            return null;
        }
    }

    public boolean putDataItem(DataItem data) {
        //extract key from data
        long key = data.getUserKey().getId();
        //check, if connection to database is already open
        if (myDatabase == null) {
            this.openDatabase();
        }
        try {
            DatabaseEntry theKey = new DatabaseEntry();
            DatabaseEntry theData = new DatabaseEntry(data.getData().getBytes(
                    "UTF-8"));
            //TODO: remove the namespace attribute from data, because it would
            // need
            // unneccessary space on disk...
            //data.removeValue("namespace");
            //data.removeValue("userkey");

            // TODO: for this test, only store the real data, nothing else into
            // db! Can change in future
            DataItem storedDataItem = new DataItem();
            storedDataItem.setData(data.getData());

            myKeyBinding.objectToEntry(new Long(key), theKey);
            //            myDataBinding.objectToEntry(storedDataItem, theData);
            OperationStatus outcome = myDatabase.put(null, theKey, theData);
            sync();
            //TODO: revert namespace attribute so we don’t remove any data from
            // original DataItem
            //data.putValue("namespace", this.getDatabaseId());
            //data.setUserKey(new UserKey(key));
            if (outcome == OperationStatus.SUCCESS) {
                return true;
            } else {
                logger.error("Put: Error hapened: " + outcome);
                return false;
            }
        } catch (Exception e) {
            // Exception handling goes here
            logger.error("Put: Error hapened: " + e);
            return false;
        }
    }

    public boolean deleteDataItem(DataItem data) {
        //extract key from data
        long key = data.getUserKey().getId();
        //check, if connection to database is already open
        if (myDatabase == null) {
            this.openDatabase();
        }
        try {
            DatabaseEntry theKey = new DatabaseEntry();
            myKeyBinding.objectToEntry(new Long(key), theKey);
            // Perform the deletion. All records that use this key are deleted.
            OperationStatus outcome = myDatabase.delete(null, theKey);
            sync();
            if (outcome == OperationStatus.SUCCESS) {
                return true;
            } else {
                return false;
            }
        } catch (Exception e) {
            // Exception handling goes here
            logger.error("Delete: Error hapened: " + e);
            return false;
        }
    }

    public int getDbSize() {
        // TODO: As of update from je 1.5.1 -> 1.5.3 this does not work
        // anymore... new ideas needed
        //        StatsConfig statConfig = new StatsConfig();
        //        statConfig.setFast(false);
        //        if (myDatabase == null) {
        //            this.openDatabase();
        //        }
        //        try {
        //            DatabaseStats dbStat = myDatabase.getStats(statConfig);
        //            int size = dbStat.;
        //            logger.info("Current BerkeleyDB size: " + size);
        //            return size;
        //        } catch (DatabaseException e) {
        //            // TODO Auto-generated catch block
        //            logger.error("get DB Size error: " + e);
        //        }
        return -1;
    }

    private void sync() throws DatabaseException {
        if (syncCounter > requestsSync) {
            myDbEnvironment.sync();
            syncCounter = 0;
        }
        syncCounter++;
    }

    /**
     * If database is still closed, <code>databaseId</code> is null
     * <p>
     * If database is opened and running, <code>databaseId</code> contains an
     * unique Id (namespace like "ch.nix.accountDB")
     *
     * @return Returns the databaseId.
     */
    public String getDatabaseId() {
        return databaseId;
    }

    public void setDatabaseId(String id) {
        //TODO check, if id does not contain any reserved id
        databaseId = id;
    }

}
```

File 30: keypartition.KeyGroup

```
/*
 * Created on Aug 2, 2004 10:46:25 AM
```

```java
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-26 15:28:00 +0200 (Tue, 26 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 138 $
 * Location:    $URL: KeyGroup.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */

package ch.nix.ert.keypartition;

import ch.nix.ert.message.InternalKey;
import ch.nix.ert.message.Node;

/**
 * @author michi
 *
 * TODO Edit this text for a better description
 */
public class KeyGroup implements Comparable {

    private InternalKey keyGroupId = null;

    private Node uplinkNode, downlinkNode;

    /**
     * @param keyGroupId
     *            sets the unique ID of this KeyGroup.
     */
    public KeyGroup(InternalKey keyGroupId) {
        this.keyGroupId = keyGroupId;
    }

    /**
     * @return Returns the KeyGroup ID.
     */
    public InternalKey getKeyGroupId() {
        return keyGroupId;
    }

    /**
     * @return Returns the downlinkNode.
     */
    public Node getDownlinkNode() {
        return downlinkNode;
    }

    /**
     * @return Returns the uplinkNode.
     */
    public Node getUplinkNode() {
        return uplinkNode;
    }

    /* (non-Javadoc)
     * @see java.lang.Comparable#compareTo(java.lang.Object)
     */
    public int compareTo(Object o) {
        KeyGroup keyGroup = (KeyGroup) o;
        return getKeyGroupId().compareTo(keyGroup.getKeyGroupId());
    }

    public String toString() {
        return getKeyGroupId().toString();
    }
}
```

File 31: keypartition.KeyPartition

```java
/*
 * Created on Aug 2, 2004 12:28:43 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-11-12 12:46:25 +0100 (Fri, 12 Nov 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 173 $
 * Location:    $URL: KeyPartition.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.keypartition;

import org.apache.log4j.Logger;

import ch.nix.ert.base.Dispatcher;
import ch.nix.ert.base.ServiceManager;
import ch.nix.ert.message.DataItem;
import ch.nix.ert.message.InternalKey;
import ch.nix.ert.message.Node;
import ch.nix.ert.message.UserKey;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class KeyPartition {

    // logger:
    static Logger logger = Logger.getLogger(KeyPartition.class);

    static private int keyPartitionIdCounter = 0;

    private int keyPartitionId = 0;

    private Node uplinkNode, downlinkNode;

    private KeyGroup firstKeyGroup;

    private KeyGroup lastKeyGroup;

    private Dispatcher dispatcher = ServiceManager.getInstance()
            .getDispatcher();

    public KeyPartition() {
        keyPartitionId = keyPartitionIdCounter++;
    }

    public int getId() {
        return keyPartitionId;
    }

    /**
     * TODO During register process not all KeyGroups are necessary, only the
     * upper, lower one. We can remove all KeyGroups in between to enhance
     * performance, so we basically don’t need any TreeMap for this class, as we
     * only need to hold the upper, lower bound? It could be problematic, if we
     * want to decide, if a special KeyGroup is occupied? (E.g. to replicate it
     * to another node? If we only now the upper, lower KeyGroups, we can’t say
     * if a dedicated key-group is contained or if it is just empty) <br>
     * Also if we need to broadcast for all KeyGroups, that we are responsible
     * for it, could be a bit tricky, if we don’t know all key-groups
     *
     * @param newKeyGroup
     */
    public synchronized void register(KeyGroup newKeyGroup) {
        boolean hasChanged = add(newKeyGroup);
        if (hasChanged) {
            publish();
        }
    }

    public synchronized boolean add(KeyGroup newKeyGroup) {
        //keyGroupList.put(newKeyGroup.getKeyGroupId(), newKeyGroup);
        boolean hasChanged = false;

        if (firstKeyGroup == null
                || newKeyGroup.compareTo(getFirstKeyGroup()) < 0) {
            firstKeyGroup = newKeyGroup;
            hasChanged = true;
        }
        if (lastKeyGroup == null
                || newKeyGroup.compareTo(getLastKeyGroup()) > 0) {
            lastKeyGroup = newKeyGroup;
            hasChanged = true;
        }
        return hasChanged;
    }

    public void publish() {
        // first publish the information at the keypartitionManager
        KeyPartitionManager.getInstance().register(this);

        if (getFirstKeyGroup() != null && getLastKeyGroup() != null) {
            DataItem publish = new DataItem("publish");
            publish.setKeyPartitionId(keyPartitionId);
            publish.setFirstInternalKey(getFirstKeyGroup().getKeyGroupId());
            publish.setLastInternalKey(getLastKeyGroup().getKeyGroupId());
            dispatcher.addWork(publish);
        }
    }

    //    public synchronized void unregister(KeyGroup oldKeyGroup) {
    //        dispatcher.addWork(new DataItem("unpublish", this));
    //        keyGroupList.remove(oldKeyGroup.getKeyGroupId());
    //    }

    public synchronized void unpublish() {
        DataItem unpublish = new DataItem("unpublish");
        unpublish.setKeyPartitionId(keyPartitionId);
        unpublish.setFirstInternalKey(getFirstKeyGroup().getKeyGroupId());
        unpublish.setLastInternalKey(getLastKeyGroup().getKeyGroupId());
        dispatcher.addWork(unpublish);
    }

    public void addKeyPartition(KeyPartition keyPartition) {
        logger.info("Add: " + keyPartition + "(ID:" + keyPartition.getId()
                + ") to me: " + this + "(ID:" + this.getId() + ")");
        // I only add the first and last KeyGroup
        boolean hasChanged1 = false;
        boolean hasChanged2 = false;
        hasChanged1 = this.add(keyPartition.getFirstKeyGroup());
        hasChanged2 = this.add(keyPartition.getLastKeyGroup());
        if (hasChanged1 || hasChanged2) {
            publish();
        }
    }

    /**
     *
     *
     * @param keyPartition
     * @return the cutoff part of this KeyPartition, null if nothing could be
     *         cut off
     */
    public KeyPartition subtractKeyPartition(KeyPartition keyPartition) {
        logger.info("Subtract: " + keyPartition + "(ID:" + keyPartition.getId()
                + ") from me: " + this + "(ID:" + this.getId() + ")");
        KeyGroup cutoffFirst, cutoffLast;
        KeyPartition cutoff = new KeyPartition();
        KeyGroup firstKeyGroupOfSubstract = keyPartition.getFirstKeyGroup();
        KeyGroup lastKeyGroupOfSubstract = keyPartition.getLastKeyGroup();
        // 4 cases
        if (firstKeyGroupOfSubstract.compareTo(getFirstKeyGroup()) <= 0
                && isResponsible(lastKeyGroupOfSubstract)) {
            cutoffFirst = this.getFirstKeyGroup();
            cutoffLast = lastKeyGroupOfSubstract;

            cutoff.add(cutoffFirst);
            cutoff.add(cutoffLast);
            //register(new KeyGroup(lastKeyGroup.getKeyGroupId()
            //          .getSubsequentInternalKey()));
            this.firstKeyGroup = new KeyGroup(lastKeyGroupOfSubstract
                    .getKeyGroupId().getSubsequentInternalKey());
            this.publish();
            return cutoff;
        }

        if (isResponsible(firstKeyGroupOfSubstract)
                && lastKeyGroupOfSubstract.compareTo(getLastKeyGroup()) >= 0) {
            cutoffFirst = firstKeyGroupOfSubstract;
            cutoffLast = this.getLastKeyGroup();

            cutoff.add(cutoffFirst);
            cutoff.add(cutoffLast);
            //register(new KeyGroup(firstKeyGroup.getKeyGroupId()
            //         .getPreviousInternalKey()));
            this.lastKeyGroup = new KeyGroup(firstKeyGroupOfSubstract
                    .getKeyGroupId().getPreviousInternalKey());
            this.publish();
            return cutoff;
        }
        if (isResponsible(firstKeyGroupOfSubstract)
                && isResponsible(lastKeyGroupOfSubstract)) {
            // not supported now... we can’t remove from the MIDDLE of the
            // keygroup as this would imply a split!
            logger
                    .warn("Substracting this Keypartition would split it into 2 parts,"
                            + " this is not supported!");
            return null;
        }
        // this means both borders are outside of this KeyPartition, so we have
        // to get rid off the whole KeyPartition
        logger
                .info("The whole KeyPartition has been substracted, this is ok, but "
                        + "could also be a failure?");
        unpublish();
        cutoff.addKeyPartition(this);
        this.firstKeyGroup = null;
        this.lastKeyGroup = null;
        return cutoff;
    }

    //TODO currently returns false, if we could adopt it (adopting is
    // currently not supported)
    public boolean isResponsible(DataItem data) {
        InternalKey internalKey = data.getInternalKey();
        if (internalKey == null) {
            UserKey userKey = data.getUserKey();
            if (userKey == null) {
                // we have not found any data to see if this dataItem could fit
                // into our Keygroups, so we are not responsible for it
                return false;
            }
            internalKey = userKey.getInternalKey();
        }
        return isResponsible(internalKey);
    }

    public boolean isResponsible(KeyGroup keyGroup) {
        return isResponsible(keyGroup.getKeyGroupId());
    }

    public boolean isResponsible(InternalKey internalKey) {
        //      check if DataItem falls between my range of keyGroups
        //check if DataItem is bigger than my lowest value:
        if (internalKey.compareTo(getFirstKeyGroup().getKeyGroupId()) >= 0) {
            //check if DataItem is smaller than my highest value:
            if (internalKey.compareTo(getLastKeyGroup().getKeyGroupId()) <= 0) {
                return true;
            }
        }
        // DataItem does not fall into our range, now check if we could adopt
        // this new DataItem
        // This does not need to be done (perhaps) as we now ALWAYS extend the
        // KeyPartition to the highest possible value, so there is never a hole
        // in the KeyPartition. Example:
        // If node 1 has keyGroup 1,2,3
        // If node 2 has keyGroup 100,102,110
        // Then KeyPartition of node 1 is: [1, 99]
        // and KeyPartition of node 2 is: [100, 110]
        // So this "return false" is correct then and no need to further expand
        // the implementation of this method
        return false;
    }

    public KeyGroup getFirstKeyGroup() {
        return firstKeyGroup;
    }

    public KeyGroup getLastKeyGroup() {
        return lastKeyGroup;
    }

    protected void finalize() throws Throwable {
        logger.info("Finalized called for KeyPartition ID: " + this.getId());
        try {
            this.unpublish(); // close all open published infos on this
            // object!
        } finally {
            super.finalize();
        }
    }

    public String toString() {
        return "ID" + getId() + ":" + getFirstKeyGroup().toString() + "-"
                + getLastKeyGroup().toString();
    }
```

```

}
```

File 32: keypartition.KeyPartitionManager

```java
/*
 * Created on Aug 2, 2004 10:48:16 AM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-11-12 12:46:25 +0100 (Fri, 12 Nov 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 173 $
 * Location:    $URL: KeyPartitionManager.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */

package ch.nix.ert.keypartition;

import java.util.Iterator;
import java.util.SortedMap;
import java.util.TreeMap;

import org.apache.log4j.Logger;

import ch.nix.ert.message.DataItem;
import ch.nix.ert.message.InternalKey;
import ch.nix.ert.message.UserKey;

/**
 * TODO Has to check, if this node is responsible for the requested key (by
 * looking at all his key-partitions) and otherwise has to forward the request
 * to the corresponding node
 *
 * @author michi
 */
public class KeyPartitionManager {

    // logger:
    static Logger logger = Logger.getLogger(KeyPartitionManager.class);

    private TreeMap keyGroupList = new TreeMap();

    static private KeyPartitionManager instance = null;

    static boolean firstNode = false;

    private int dbSize = 0;

    private int putRequestsPerMinute = 0;

    private int putRequests = 0;

    private int getRequestsPerMinute = 0;

    private int getRequests = 0;

    private int deleteRequestsPerMinute = 0;

    private int deleteRequests = 0;

    protected boolean isJoining = false;

    private KeyPartitionService service;

    private MigrationServer migrationServer;

    private MigrationClient migrationClient;

    private KeyPartitionManager() {
    }

    static public synchronized KeyPartitionManager getInstance() {
        if (null == instance) {
            //System.out.println("Instanciation "+instance+" of
            // KeyPartitionManager: "+Thread.currentThread().getName());
            instance = new KeyPartitionManager();
            instance.service = KeyPartitionService.getInstance();
            instance.migrationServer = MigrationServer.getInstance();
            instance.migrationClient = MigrationClient.getInstance();
            //System.out.println("Instance set: "+instance+" of
            // KeyPartitionManager: "+Thread.currentThread().getName());
            //setting up instance
            DataItem dbSizeItem = new DataItem();
            dbSizeItem.setOperation("getsizedb");
            DataItem callBackItem = new DataItem();
            callBackItem.setOperation("callcontinous");
            callBackItem.setDelay(60000);
            callBackItem.setNested(dbSizeItem);

            DataItem statisticItem = new DataItem();
            statisticItem.setOperation("updatestatistic");
            DataItem callBackItem2 = new DataItem();
            callBackItem2.setOperation("callcontinous");
            callBackItem2.setDelay(60000);
            callBackItem2.setNested(statisticItem);

            //registering callback for database size
            instance.service.transactionBegin(callBackItem);
            //registering callback for statistic update
            instance.service.transactionBegin(callBackItem2);
            //System.out.println("Finished Instanciation "+instance+" of
            // KeyPartitionManager: "+Thread.currentThread().getName());
            logger.info("KeyPartitionManager initialized");
        }
        return instance;
    }

    public void register(KeyPartition newKeyPartition) {
        // first check, if we already have this "new" KeyPartition in our list:
        for (Iterator iter = keyGroupList.values().iterator(); iter.hasNext();) {
            KeyPartition keyPartition = (KeyPartition) iter.next();
            if (keyPartition.getId() == newKeyPartition.getId()) {
                iter.remove();
            }
        }
        keyGroupList.put(newKeyPartition.getFirstKeyGroup().getKeyGroupId(),
                newKeyPartition);
    }

    public void unregister(KeyPartition oldKeyPartition) {
        oldKeyPartition.unpublish();
        keyGroupList.remove(oldKeyPartition.getFirstKeyGroup().getKeyGroupId());
    }

    public KeyPartition responsible(DataItem dataItem) {
        InternalKey internalKey = dataItem.getInternalKey();
        if (internalKey == null) {
            UserKey userKey = dataItem.getUserKey();
            if (userKey == null) {
                return null;
            }
            return responsible(userKey.getInternalKey());
        } else {
            return responsible(internalKey);
        }
    }

    public KeyPartition responsible(InternalKey internalKey) {
        KeyPartition actualKeyPartition = null;
        SortedMap headMap;
        if (keyGroupList.isEmpty()) {
            // this means there is no KeyGroup there, so i return null
            return null;
        } else {
            headMap = keyGroupList.headMap(internalKey
                    .getSubsequentInternalKey());
            //if this beast is not empty, get the last key of it:
            if (!headMap.isEmpty()) {
                actualKeyPartition = (KeyPartition) keyGroupList.get(headMap
                        .lastKey());
            }
        }
        if (actualKeyPartition == null) {
            return null;
        }
        if (actualKeyPartition.isResponsible(internalKey)) {
            return actualKeyPartition;
        } else {
            return null;
        }
    }

```

```java
    /**
     *
     *
     * @param keyPartition
     * @return the cutoff part of this KeyPartition, null if nothing could be
     *         cut off
     */
    public KeyPartition substract(KeyPartition keyPartition) {
        // first find the correct keypartition in my list where I can cutoff
        // something
        InternalKey firstKey = keyPartition.getFirstKeyGroup().getKeyGroupId();
        InternalKey lastKey = keyPartition.getLastKeyGroup().getKeyGroupId();
        KeyPartition keyPartitionToCut = responsible(firstKey);
        if (keyPartitionToCut == null) {
            keyPartitionToCut = responsible(lastKey);
        }
        if (keyPartitionToCut == null) {
            // nothing to do...
            return null;
        } else {
            return keyPartitionToCut.subtractKeyPartition(keyPartition);
        }
    }

    public void add(KeyPartition keyPartition) {
        // I can only add a KeyPartition, if it will extend another one without
        // any hole in between, as otherwise we would have a splited
        // KeyPartition.

        if (keyGroupList.isEmpty()) {
            // this means we have the very first keyPartition here so we should
            // add it!
            logger.info("add: First KeyPartition to add: " + keyPartition);
            register(keyPartition);
            keyPartition.publish();
        } else {

            // as adding a new KeyPartition normally does not overlap with the
            // already existing KeyPartitions we have to add +1 to each edge for
            // searching the correct keypartition to add this new one
            InternalKey firstKey = keyPartition.getFirstKeyGroup()
                    .getKeyGroupId().getPreviousInternalKey();
            InternalKey lastKey = keyPartition.getLastKeyGroup()
                    .getKeyGroupId().getSubsequentInternalKey();
            KeyPartition keyPartitionToAdd = responsible(firstKey);
            if (keyPartitionToAdd == null) {
                keyPartitionToAdd = responsible(lastKey);
            }
            if (keyPartitionToAdd == null) {
                // nothing to do...
                logger
                        .info("Could not add keyPartition: "
                                + keyPartition
                                + " to my own: "
                                + this
                                + ", because they would result in a KeyPartition split...");
            } else {
                keyPartitionToAdd.addKeyPartition(keyPartition);
            }
        }
    }

    public KeyPartition getFirst() {
        if (keyGroupList.isEmpty()) {
            return null;
        }
        KeyPartition firstKeyPartition = (KeyPartition) keyGroupList
                .get(keyGroupList.firstKey());
        return firstKeyPartition;
    }

    public KeyPartition getLast() {
        if (keyGroupList.isEmpty()) {
            return null;
        }
        KeyPartition lastKeyPartition = (KeyPartition) keyGroupList
                .get(keyGroupList.lastKey());
        return lastKeyPartition;
    }

    public void get(DataItem data) {
        if (responsible(data) != null) {
            data.setOperation("getdb");
            service.transactionContinue(data);
            getRequests++;
        } else {
            logger
                    .info("get: I forward this request, as I'm not responsible for: "
                            + data);
            data.setRouting("routing");
            service.transactionContinue(data);
        }
    }

    public void put(DataItem data) {
        if (responsible(data) != null || migrationClient.isResponsible(data)) {
            // Yes this node is responsible for this data, store it!
            data.setOperation("putdb");
            if (migrationServer.isMigrating()
                    && migrationServer.isResponsible(data)) {
                // We are the server and are migrating data to somewhere, so I
                // have to send the put requests also there

                service.transactionSplit(data, 2);

                DataItem data2 = data.getClone();
                data2.setDirectDestination(migrationServer.getCurrentClient());
                data2.setRouting("direct");
                service.transactionSplit(data2, 2);

            } else {
                service.transactionContinue(data);
            }
            putRequests++;
            // System.out.println("Put Requests received sofar: " +
            // putRequests);
        } else {
            logger
                    .info("put: I forward this request, as I'm not responsible for: "
                            + data);
            data.setRouting("routing");
            service.transactionContinue(data);
        }
    }

    public void delete(DataItem data) {
        // TODO Perhaps check first if I'm responsible for this DataItem, and
        // redirect, if necessary?
        data.setOperation("deletedb");
        if (migrationServer.isMigrating()) {
            // We are the server and are migrating data to somewhere, so I
            // have to send the put requests also there

            service.transactionSplit(data, 2);

            DataItem data2 = data.getClone();
            data2.setDirectDestination(migrationServer.getCurrentClient());
            data2.setRouting("direct");
            service.transactionSplit(data2, 2);

        } else {
            service.transactionContinue(data);
        }
        deleteRequests++;
    }

    /**
     * This method should be only called for the first node in the whole
     * network. All other nodes should use the "join" method to actually join
     * the ert-network
     */
    public void isFirstNodeOnNetwork() {
        // check that this method is called ONLY once
        if (firstNode == false) {
            // I'm the first node, so I'm responsible for the whole
            logger.info("This node is the first node on the whole network!");
            KeyGroup firstKeyGroup = new KeyGroup(new InternalKey(
                    Long.MIN_VALUE));
            KeyGroup lastKeyGroup = new KeyGroup(
                    new InternalKey(Long.MAX_VALUE));
            KeyPartition firstKeyPartition = new KeyPartition();
            firstKeyPartition.add(firstKeyGroup);
            firstKeyPartition.add(lastKeyGroup);
            register(firstKeyPartition);
            firstKeyPartition.publish();
            firstNode = true;
        }
    }

    public void joinOnNetwork() {
        // we have to join on the network, so we have to migrate some data to
        // us!
        MigrationClient.getInstance().join();
    }

    /**
     * gives back the load of this current node: takes into account the amount
     * of items stored in db and later also CPU load, throughput of requests,
     * ...
     *
     * @return value between 0-1 (0 = idle, 1 = on system limit, >1 = heavily
     *         overloaded)
     */
    public double nodeLoad() {
        // this is hardcoded, so an amount of 100000 stored attributes are
        // looked as too much
        //        int maxStoredValues = 100000;
        //        float store = (float) dbSize / (float) maxStoredValues;
        //
        //        //hardcoded put value 500 requests per minute considered as very
        // high:
        //        int maxPutPerMinute = 500;
        //        float putdb = (float) putRequestsPerMinute / (float) maxPutPerMinute;
        //
        //        //hardcoded put value 500 requests per minute considered as very
        // high:
        //        int maxGetPerMinute = 500;
        //        float getdb = (float) getRequestsPerMinute / (float) maxGetPerMinute;
        //
        //        //hardcoded put value 500 requests per minute considered as very
        // high:
        //        int maxDeletePerMinute = 500;
        //        float deletedb = (float) deleteRequestsPerMinute
        //                / (float) maxDeletePerMinute;

        // now do a weighting of the values:
        // store: 0.4
        // put : 0.2
        // get : 0.3
        // delete: 0.1
        //        System.out.println("DB-Size:" + dbSize + " Puts/min:"
        //                + putRequestsPerMinute + " Gets/min:" + getRequestsPerMinute
        //                + " Dels/min:" + deleteRequestsPerMinute);
        //float load = (float) (store * 0.4 + putdb * 0.2 + getdb * 0.3 +
        // deletedb * 0.1);
        //TODO revert to line above
        //currently for simplicity only take into account the store:
        //float load = (float) store;

        // For simplicity I change nodeload to be dependant of the number of
        // KeyGroups: so the load is just the % of the whole key-range:
        long numberOfKeyGroups = 0;
        for (Iterator iter = keyGroupList.values().iterator(); iter.hasNext();) {
            KeyPartition keyPartition = (KeyPartition) iter.next();
            numberOfKeyGroups += keyPartition.getFirstKeyGroup()
                    .getKeyGroupId().amountBetween(
                            keyPartition.getLastKeyGroup().getKeyGroupId());
        }
        long maxPossibleKeyRange = InternalKey.getHigherLimit()
                - InternalKey.getLowerLimit();

        double load = (double) numberOfKeyGroups / (double) maxPossibleKeyRange;
        //load = (float) 0.4;
        return load;
    }

    /**
     * Calculates the KeyRange to migrate
     *
     * @param remoteLoad
     *            the load of the other node, which wants to have some data from
     *            this node
     * @param uplinkNode
     *            if true, the node who requests the migration is the
     *            uplinkNode, if false it is the downlonkNode
     * @return The DataItem with parameter "migrationCutPointLow" and
     *         "migrationCutPointHigh" of the keygroups to migrate
     */
    public DataItem calculateMigrationData(double remoteLoad, boolean uplinkNode) {
        double localLoad = nodeLoad();
        long storedDataItems = (long) (localLoad * 100000);

        // first check how many dataItems we potentionally could migrate:
        double difference = localLoad - remoteLoad;

```

```java
        if (storedDataItems == 0) {
            //means we have empty node, so we can easily share half of our keys
            // to the other node
            return null;
        }

        // 40% of the load is related to number of dataitems (see nodeload
        // method)
        long numberOfDataItems = (long) (difference * 100000 / 2);

        // now calculate the keygroups we could migrate
        // First calculate the amount of keygroups we currently hold
        long numberOfKeyGroups = 0;
        for (Iterator iter = keyGroupList.values().iterator(); iter.hasNext();) {
            KeyPartition keyPartition = (KeyPartition) iter.next();
            numberOfKeyGroups += keyPartition.getFirstKeyGroup()
                    .getKeyGroupId().amountBetween(
                            keyPartition.getLastKeyGroup().getKeyGroupId());
        }

        // now calculate the amount of keygroups to migrate:
        long keygroupsToMigrate = (long) (((double) numberOfKeyGroups /
                (double) storedDataItems) *
                (double) numberOfDataItems);

        // now calculate the correct keygroup where we should do the cut - we
        // migrate max one full keypartition, even if keygroupsToMigrate is
        // larger than the last keypartition
        DataItem response = new DataItem();
        if (uplinkNode) {
            // if the other node is the uplink node, we have to migrate data
            // from the higher end of the last keyPartition
            KeyPartition lastKeyPartition = (KeyPartition) keyGroupList
                    .get(keyGroupList.lastKey());

            InternalKey lastInternalKey = lastKeyPartition.getLastKeyGroup()
                    .getKeyGroupId();

            InternalKey migrationCutPoint = lastInternalKey
                    .getNthPreviousInternalKey(keygroupsToMigrate);
            // now check if this cutPoint is within the lastKeyPartition
            if (!lastKeyPartition.isResponsible(migrationCutPoint)) {
                //ops lastKeyPartition is not responsible for this key, so I
                // set
                // the migrationCutPoint to the first key!
                migrationCutPoint = lastKeyPartition.getFirstKeyGroup()
                        .getKeyGroupId();
            }
            response.setLastInternalKey(lastInternalKey);
            response.setFirstInternalKey(migrationCutPoint);
        } else {
            // if the other node is the downlink node, we have to migrate data
            // from the lower end of the first keyPartition
            KeyPartition firstKeyPartition = (KeyPartition) keyGroupList
                    .get(keyGroupList.firstKey());

            InternalKey firstInternalKey = firstKeyPartition.getFirstKeyGroup()
                    .getKeyGroupId();

            InternalKey migrationCutPoint = firstInternalKey
                    .getNthSubsequentInternalKey(keygroupsToMigrate - 1);
            // now check if this cutPoint is within the lastKeyPartition
            if (!firstKeyPartition.isResponsible(migrationCutPoint)) {
                //ops lastKeyPartition is not responsible for this key, so I
                // set
                // the migrationCutPoint to the first key!
                migrationCutPoint = firstKeyPartition.getLastKeyGroup()
                        .getKeyGroupId();
            }
            response.setFirstInternalKey(firstInternalKey);
            response.setLastInternalKey(migrationCutPoint);
        }
        return response;
    }

    protected void updatedbSize(int size) {
        //        System.out.println("Update DB size called! having done: " + size
        //                + " entries");
        dbSize = size;
    }

    protected void updateStatistic() {
        //        System.out.println("Update Statistic called! having done: "
        //                + putRequests + " putRequests");
        putRequestsPerMinute = putRequests;
```

```
            getRequestsPerMinute = getRequests;
            deleteRequestsPerMinute = deleteRequests;
            putRequests = 0;
            getRequests = 0;
            deleteRequests = 0;
        }

        public String toString() {
            String representation = "( ";
            for (Iterator iter = keyGroupList.values().iterator(); iter.hasNext();) {
                KeyPartition keyPartition = (KeyPartition) iter.next();
                representation = representation + keyPartition + " ";
            }
            return representation + ")";
        }

        //    public boolean isResponsible(DatItem data) {
        //
        //    }

    }
```

File 33: keypartition.KeyPartitionService

```
/*
 * Created on Aug 27, 2004 12:29:27 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-11-09 01:55:31 +0100 (Tue, 09 Nov 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 165 $
 * Location:    $URL: KeyPartitionService.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.keypartition;

import org.apache.log4j.Logger;

import ch.nix.ert.base.ServiceBase;
import ch.nix.ert.message.DataItem;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class KeyPartitionService extends ServiceBase {

    // logger:
    static Logger logger = Logger.getLogger(KeyPartitionService.class);

    static private KeyPartitionService instance = null;

    private KeyPartitionManager keyPartitionManager;

    private MigrationClient migrationClient;

    private MigrationServer migrationServer;

    // private TreeMap timeSortedList = new TreeMap();

    private KeyPartitionService() {
    }

    static public synchronized KeyPartitionService getInstance() {
        if (null == instance) {
            instance = new KeyPartitionService();
            instance.keyPartitionManager = KeyPartitionManager.getInstance();
            instance.migrationClient = MigrationClient.getInstance();
            instance.migrationServer = MigrationServer.getInstance();
            logger.info("KeyPartitionService started");
        }
        return instance;
    }

    /*
     * (non-Javadoc)
     *
     * @see ch.nix.ert.ServiceInterface#registerAtDispatcher()
     */
    public void registerAtDispatcher() {
        // TODO Auto-generated method stub
        registerService("get");
        registerService("put");
        registerService("delete");

        registerService("updatedbsize");
        registerService("updatestatistic");
        registerService("nodeload");

        registerService("firstnode");
        registerService("join");

        registerService("migrationprobe");
        //registerService("migrationservercancel");
        //registerService("migrationclientcancel");
        registerService("migrationstart");
        registerService("migrationend");
        registerService("getmigration");
        registerService("deletemigration");
    }

    /*
     * (non-Javadoc)
     *
     * @see java.lang.Runnable#run()
     */
    public void run() {
        //System.out.println("DataStoreService started");
        Thread.currentThread().setName("ERT.KeyPartitionService");
        while (true) {
            DataItem dataItem = null;
            dataItem = getWork();
            // logger.debug("Got Work: "+dataItem);
            String operation = dataItem.getOperationType();
            //System.out.println("Operation: "+operation+" equals? "+
            // (operation.equals("nodeload")));
            if (operation.equals("get")) {
                // logger.debug("get: " + dataItem);
                keyPartitionManager.get(dataItem);
            } else if (operation.equals("put")) {
                // logger.debug("put: " + dataItem);
                //System.out.println("put in keyservice...");
                keyPartitionManager.put(dataItem);
            } else if (operation.equals("delete")) {
                // logger.debug("delete: " + dataItem);
                keyPartitionManager.delete(dataItem);
            } else if (operation.equals("getsizedb-end")) {
                // logger.debug("getSizeDBResponse: " + dataItem);
                keyPartitionManager.updatedbSize(dataItem.getDbSize());
            } else if (operation.equals("updatestatistic")) {
                // logger.debug("udpadateStatistic: " + dataItem);
                //  System.out
                //  .println("KeyPartitionService:UPDATESTATISTIC called");
                keyPartitionManager.updateStatistic();
            } else if (operation.equals("firstnode")) {
                keyPartitionManager.isFirstNodeOnNetwork();
            } else if (operation.equals("join")) {
                keyPartitionManager.joinOnNetwork();
            } else if (operation.equals("nodeload")) {
                // logger.debug("nodeLoad: " + dataItem);
                //Node originator = dataItem.getDirectOriginator();

                double size = keyPartitionManager.nodeLoad();
                DataItem response = new DataItem();
                response.setNodeLoad(size);
                //response.setDirectDestination(originator);
                //response.setOperation("nodeloadresponse");
                //System.out.println("KeyPartitionService response: " +
                // response);
                transactionEnd(response);
                // logger.debug("nodeLoadResponse: " + response);
            } else if (operation.equals("nodeload-end")) {
                // logger.debug("nodeLoad-end: " + dataItem);
                if (keyPartitionManager.isJoining) {
                    migrationClient.joinOnNetwork(dataItem);
                } else {
                    migrationClient.nodeLoadResponse(dataItem);
                }
            } else if (operation.equals("migrationprobe")) {
                // logger.debug("migrationProbe: " + dataItem);
                migrationClient.probeServer();
```

```java
//            } else if (operation.equals("migrationservercancel")) {
//                logger.debug("migrationServerCancel: " + dataItem);
//                migrationServer.cancelMigration();
//            } else if (operation.equals("migrationclientcancel")) {
//                logger.debug("migrationClientCancel: " + dataItem);
//                migrationClient.cancelMigration();
            } else if (operation.equals("migrationstart")) {
                // logger.debug("migrationStart: " + dataItem);
                migrationServer.migrationStart(dataItem);
            } else if (operation.equals("migrationstart-end")) {
                // logger.debug("migrationStartResponse: " + dataItem);
                migrationClient.migrationStartResponse(dataItem);
            } else if (operation.equals("migrationend")) {
                // logger.debug("migrationEnd: " + dataItem);
                migrationServer.migrationEnd(dataItem);
            } else if (operation.equals("migrationend-end")) {
                // logger.debug("migrationEndResponse: " + dataItem);
                migrationClient.migrationEndResponse(dataItem);
            } else if (operation.equals("getmigration")) {
                // logger.debug("getMigration: " + dataItem);
                migrationServer.sendMigratedData(dataItem);
            } else if (operation.equals("getmigration-end")) {
                // logger.debug("getMigrationResponse: " + dataItem);
                migrationClient.receiveMigratedData(dataItem);
            } else {

                logger
                        .warn("DataItem dropped, because no processing was found for: "
                                + dataItem);
            }
        }

    }
}
```

File 34: keypartition.MigrationClient

```java
/*
 * Created on Sep 6, 2004 2:59:21 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-11-12 12:46:25 +0100 (Fri, 12 Nov 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 173 $
 * Location:    $URL: MigrationClient.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.keypartition;

import java.util.Calendar;

import org.apache.log4j.Logger;

import ch.nix.ert.message.DataItem;
import ch.nix.ert.message.InternalKey;
import ch.nix.ert.message.Node;
import ch.nix.ert.message.UserKey;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class MigrationClient {

    // logger:
    static Logger logger = Logger.getLogger(MigrationClient.class);

    static private MigrationClient instance = null;

    private boolean joinExecuted = false;

    private MigrationServer migrationServer = null;

    private KeyPartitionService service;

    private boolean isMigratingAsClient = false;

    private Node migrationNode;

    private KeyPartition migrationKeyPartition;

    private KeyPartitionManager keyPartitionManager;

    private int joinOnNetworkCount = 10;

    private Node highestLoadNode = new Node();

    private double highestLoad = 0;

    private long migrationTime = 0;

    // hardcoded set: timeout migration after 60 sec of inactivity
    private long migrationTimeout = 60000;

    private MigrationClient() {
    }

    static public synchronized MigrationClient getInstance() {
        if (null == instance) {
            instance = new MigrationClient();
            instance.service = KeyPartitionService.getInstance();
            instance.keyPartitionManager = KeyPartitionManager.getInstance();
            instance.migrationServer = MigrationServer.getInstance();
            // calling probeServer to get the migration started!
            //instance.probeServer(new Node("10.0.1.250", 1099));
            // now start the automatic probe: every 60 sec

            logger.info("MigrationClient initialised");
        }
        return instance;
    }

    public boolean isMigrating() {
        migrationTimeout();
        return isMigratingAsClient;
    }

    public void join() {
        if (!joinExecuted) {
            DataItem migrationProbe = new DataItem();
            migrationProbe.setOperation("migrationprobe");
            DataItem callBackItem = new DataItem();
            callBackItem.setOperation("callcontinous");
            callBackItem.setDelay(20000);
            callBackItem.setNested(migrationProbe);
            service.transactionBegin(callBackItem);
            joinExecuted = true;
        }
        joinOnNetwork(null);
    }

    public void joinOnNetwork(DataItem dataItem) {
        logger.info("Client: Probe a random Server for Joining Response: "
                + dataItem);
        if (joinOnNetworkCount > 0) {
            keyPartitionManager.isJoining = true;
        } else {
            keyPartitionManager.isJoining = false;
            initiateMigration(highestLoadNode);
        }
        if (dataItem != null) {
            if (dataItem.getNodeLoad() > highestLoad) {
                highestLoadNode = dataItem.getDirectOriginator();
                highestLoad = dataItem.getNodeLoad();
            }
        }
        if (joinOnNetworkCount > 0) {
            // now send to another random node:
            DataItem sendData = new DataItem("nodeload");
            sendData.setRouting("routing");
            sendData.setInternalKey(InternalKey.getRandom());
            service.transactionBegin(sendData);
            logger.info("Probe a random Server for Joining: " + sendData);
            joinOnNetworkCount-;
        }
    }

    // TODO must be: probeServer(Node node)
    public void probeServer() {

        logger.info("Client: Send Probe to Servers. My current KeyPartitions: "
                + keyPartitionManager);
        // first check for node-load of my Server
        KeyPartition firstKeyPartition = keyPartitionManager.getFirst();
        KeyPartition lastKeyPartition = keyPartitionManager.getLast();

        if (firstKeyPartition != null) {
            InternalKey downlink = firstKeyPartition.getFirstKeyGroup()
                    .getKeyGroupId().getPreviousInternalKey();
            if (keyPartitionManager.responsible(downlink) == null) {
                // avoid sending messages to myself, if I'm on the
                // lowest/highest edge of internal-Keys
                DataItem sendDataDownlink = new DataItem("nodeload");
                sendDataDownlink.setRouting("routing");
                sendDataDownlink.setInternalKey(downlink);
                logger.info("Client: Probe Server: " + sendDataDownlink);
                service.transactionBegin(sendDataDownlink);
            }
        }
        if (lastKeyPartition != null) {
            InternalKey uplink = lastKeyPartition.getLastKeyGroup()
                    .getKeyGroupId().getSubsequentInternalKey();
            if (keyPartitionManager.responsible(uplink) == null) {
                // avoid sending messages to myself, if I'm on the
                // lowest/highest edge of internal-Keys
                DataItem sendDataUpLink = new DataItem("nodeload");
                sendDataUpLink.setRouting("routing");
                sendDataUpLink.setInternalKey(uplink);
                logger.info("Client: Probe Server: " + sendDataUpLink);
                service.transactionBegin(sendDataUpLink);
            }
        }
        if (firstKeyPartition == null && lastKeyPartition == null) {
            // means we have not yet found a node to join, so we have to repeat
            // the node-join algorithm:
            //
            // first reset the join-related variables:
            joinOnNetworkCount = 10;
            highestLoad = 0;
            join();
        }
        //System.out.println("Data sent");
    }

    public void nodeLoadResponse(DataItem dataItem) {
        logger.info("Client: Probe Server Response: " + dataItem);
        if (!isMigrating()) {
            // we got an nodeLoadResponse, now check, if we can migrate...
            double localLoad = keyPartitionManager.nodeLoad();
            if (localLoad + 0.02 < dataItem.getNodeLoad() || (localLoad == 0)) {
                // the Server has at least a 5% higher load than we have, or the
                // other node is almost idle, so we can easily so we should
                // migrate
                initiateMigration(dataItem.getDirectOriginator());
                //instance.probeServer(new Node("192.168.0.111", 1099));
            }
        }
    }

    public void initiateMigration(Node node) {
        if (!migrationServer.isMigrating()) {
            migrationNode = node;
            DataItem sendData = new DataItem("migrationstart");
            sendData.setDirectDestination(migrationNode);
            KeyPartition firstKeyPartition = keyPartitionManager.getFirst();
            if (firstKeyPartition != null) {
                // this has to be done, so the Server Node finds out if this
                // node is
                // the uplink, or downlink node
                sendData.setInternalKey(firstKeyPartition.getFirstKeyGroup()
                        .getKeyGroupId());
            }
            sendData.setRouting("direct");
            sendData.setNodeLoad(keyPartitionManager.nodeLoad());
            logger.info("Client: Migration-Request Initiated: " + sendData);
            service.transactionBegin(sendData);
        }
    }

    public void migrationStartResponse(DataItem dataItem) {
        if (isMigratingAsClient == false && !migrationTimeout()) {
            logger.info("Client: Migration Start Response: " + dataItem);
            isMigratingAsClient = true;
            // first check, if we want to migrate this amount of data?
            // TODO: check - currently we always accept.

            // we first upload the the migration-data, and if successfully
            // finished will publish the new information!
            //
            // first we generate a temporary KeyPartition (which is not
            // published yet) and use it to fill up the data:
            migrationKeyPartition = new KeyPartition();
            // add the proposed data-Range of the data to migrate:
            migrationKeyPartition.add(new KeyGroup(dataItem
                    .getFirstInternalKey()));
            migrationKeyPartition.add(new KeyGroup(dataItem
                    .getLastInternalKey()));

            // set the DataItem to the highest value:
            DataItem getData = new DataItem("getmigration");
            getData.setUserKey(dataItem.getLastInternalKey().getLastUserKey()
                    .getSubsequentUserKey());
            getData.setRouting("direct");
            getData.setDirectDestination(migrationNode);
            service.transactionBegin(getData);
        }
    }

    private boolean migrationTimeout() {
        long currentTime = Calendar.getInstance().getTimeInMillis();
        if (currentTime > migrationTime + migrationTimeout
                && migrationTime != 0) {
            logger
                    .warn("Client: Migration has been canceled, because of an time-out "
                            + "during transfering of DataItems");
            isMigratingAsClient = false;
            migrationKeyPartition = null;
            migrationTime = 0;
            return true;
        }
        return false;
    }

    public void receiveMigratedData(DataItem dataItem) {
        if (isMigratingAsClient) {
            migrationTime = Calendar.getInstance().getTimeInMillis();
            // first check if migrated dataItem is within our
            // MigrationKeyPartition
            if (migrationKeyPartition.isResponsible(dataItem)) {
                // this dataItem is part of this Migration and we store it into
                // db
                UserKey currentUserKey = dataItem.getUserKey();
                dataItem.setOperation("putdbnoack");
                service.transactionBegin(dataItem);

                // now request the next dataItem
                DataItem getData = new DataItem("getmigration");
                getData.setUserKey(currentUserKey);
                getData.setDirectDestination(migrationNode);
                getData.setRouting("direct");
                service.transactionBegin(getData);
            } else {
                // means we are out of range -> means we have finished the
                // migration and could commit it
                logger.info("Client: Migration finished");
                DataItem sendData = new DataItem("migrationend");
                sendData.setDirectDestination(migrationNode);
                sendData.setRouting("direct");
                service.transactionBegin(sendData);
            }
        }
    }

    public boolean isResponsible(DataItem dataItem) {
        if (migrationKeyPartition == null) {
            return false;
        }
        return migrationKeyPartition.isResponsible(dataItem);
    }

    public void migrationEndResponse(DataItem dataItem) {
        if (isMigratingAsClient) {
            logger.info("Client: Migration End Response: " + dataItem);
            keyPartitionManager.add(migrationKeyPartition);
            migrationKeyPartition = null;
            migrationNode = null;
            isMigratingAsClient = false;
            migrationTime = 0;
        }
    }

```

```
}
```

File 35: keypartition.MigrationServer

```
/*
 * Created on Sep 6, 2004 2:59:39 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-11-12 12:46:25 +0100 (Fri, 12 Nov 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 173 $
 * Location:    $URL: MigrationServer.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.keypartition;

import java.util.Calendar;

import org.apache.log4j.Logger;

import ch.nix.ert.message.DataItem;
import ch.nix.ert.message.InternalKey;
import ch.nix.ert.message.Node;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class MigrationServer {

    // logger:
    static Logger logger = Logger.getLogger(MigrationServer.class);

    static private MigrationServer instance = null;

    private MigrationClient migrationClient = null;

    private Node currentClient = null;

    private KeyPartitionService service;

    private boolean requestMigration = false;

    private boolean isMigratingAsServer = false;

    private KeyPartitionManager keyPartitionManager;

    private KeyPartition migrationKeyPartition;

    private long requestMigrationTime = 0;

    private long migrationTime = 0;

    // hardcoded set: timeout migration after 60 sec of inactivity
    private long requestMigrationTimeout = 60000;

    private long migrationTimeout = 60000;

    private MigrationServer() {
    }

    static public synchronized MigrationServer getInstance() {
        if (null == instance) {
            instance = new MigrationServer();
            instance.service = KeyPartitionService.getInstance();
            instance.keyPartitionManager = KeyPartitionManager.getInstance();
            instance.migrationClient = MigrationClient.getInstance();
            logger.info("MigrationServer initialised");
        }
        return instance;
    }

    public boolean isMigrating() {
        requestMigrationTimeout();
        migrationTimeout();
        return requestMigration;
    }

    public Node getCurrentClient() {
        return currentClient;
    }

    private boolean requestMigrationTimeout() {
        long currentTime = Calendar.getInstance().getTimeInMillis();
        if (currentTime > requestMigrationTime + requestMigrationTimeout
                && requestMigrationTime != 0) {
            if (requestMigration && !isMigratingAsServer) {
                logger
                        .info("Server: Migration has been canceled, because of an time-out "
                                + "while waiting for the first DataItem to be transfered");
                requestMigration = false;
                isMigratingAsServer = false;
                migrationKeyPartition = null;
                currentClient = null;
                requestMigrationTime = 0;
                migrationTime = 0;
                return true;
            }
        }
        return false;
    }

    private boolean migrationTimeout() {
        long currentTime = Calendar.getInstance().getTimeInMillis();
        if (currentTime > migrationTime + migrationTimeout
                && migrationTime != 0) {
            logger
                    .info("Server: Migration has been canceled, because of an time-out "
                            + "during transfering of DataItems");
            requestMigration = false;
            isMigratingAsServer = false;
            migrationKeyPartition = null;
            currentClient = null;
            requestMigrationTime = 0;
            migrationTime = 0;
            return true;
        }
        return false;
    }

    public void migrationStart(DataItem dataItem) {
        if (requestMigration == false && !migrationClient.isMigrating()
                && !requestMigrationTimeout() && !migrationTimeout()) {
            requestMigrationTime = Calendar.getInstance().getTimeInMillis();
            logger.info("Server: Migration Start: " + dataItem);
            // now check which data we can can migrate
            //
            // first check if the node who requests the data is uplink or
            // downlink node
            InternalKey remoteInternalKey = dataItem.getInternalKey();
            boolean uplink = true;
            if (remoteInternalKey == null) {
                // no InternalKey set, so remote does not yet know so we have to
                // decide :-)
                uplink = true;
            } else if (dataItem.getInternalKey().compareTo(
                    keyPartitionManager.getFirst().getFirstKeyGroup()
                            .getKeyGroupId()) < 0) {
                uplink = false;
            }

            DataItem remoteData = keyPartitionManager.calculateMigrationData(
                    dataItem.getNodeLoad(), uplink);
            dataItem.setLastInternalKey(remoteData.getLastInternalKey());
            dataItem.setFirstInternalKey(remoteData.getFirstInternalKey());

            migrationKeyPartition = new KeyPartition();
            // add the proposed data-Range of the data to migrate:
            migrationKeyPartition.add(new KeyGroup(dataItem
                    .getFirstInternalKey()));
            migrationKeyPartition.add(new KeyGroup(dataItem
                    .getLastInternalKey()));
            dataItem.removeNodeLoad();
            service.transactionEnd(dataItem);
            currentClient = dataItem.getTransactionNodeId();
            requestMigration = true;
        }
    }

    //    public void cancelMigration() {
    //        if (requestMigration && !isMigratingAsServer) {
```

```
//            logger
//                    .info("Server: Migration has been canceled, because of an time-out");
//            requestMigration = false;
//            isMigratingAsServer = false;
//            migrationKeyPartition = null;
//        }
//    }

    public void sendMigratedData(DataItem dataItem) {
        if (requestMigration) {
            migrationTime = Calendar.getInstance().getTimeInMillis();
            isMigratingAsServer = true;
            //logger.info("sendMigratedData: " + dataItem);
            //TODO: as getprev will ALWAYS omit the first key, we have to
            // check, that for first execution we actually do a normal
            // get-Operation...
            // Or because we currently let the whole migration finish untill we
            // change the Responsibilities of a KeyRange, we can always fetch
            // backwards...

            // forward this request to DataStore layer, so he actually returns
            // the data to the Migration-Client
            dataItem.setOperation("getprevdb");
            service.transactionContinue(dataItem);
        }
    }

    public boolean isResponsible(DataItem dataItem) {
        if (migrationKeyPartition == null) {
            return false;
        }
        return migrationKeyPartition.isResponsible(dataItem);
    }

    public void migrationEnd(DataItem dataItem) {
        if (requestMigration) {
            logger.info("Server: Migration End: " + dataItem);
            // now change my KeyPartition
            keyPartitionManager.substract(migrationKeyPartition);

            // now delete all my obsolete data
            // TODO... currently no data is deleted, perhaps we can do a general
            // "cleanup" Service which will delete unused data from db during
            // low CPU time...

            service.transactionEnd(dataItem);
            migrationKeyPartition = null;
            requestMigration = false;
            isMigratingAsServer = false;
            requestMigrationTime = 0;
            migrationTime = 0;
        }
    }

}
```

File 36: main.MainService

```
/*
 * Created on Oct 20, 2004 4:06:01 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-11-12 12:46:25 +0100 (Fri, 12 Nov 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 173 $
 * Location:    $URL: MainService.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.main;

import java.awt.Frame;
import java.util.ArrayList;
import java.util.Arrays;

import org.apache.log4j.Logger;

import ch.nix.ert.base.RootLogger;
```

```java
import ch.nix.ert.base.ServiceBase;
import ch.nix.ert.base.ServiceManager;
import ch.nix.ert.message.DataItem;

/**
 * MainService class which is executed at startup. <br>
 * It will initialize the whole system and starts it up
 *
 * @author michi
 */
public class MainService extends ServiceBase {

    // logger:
    static Logger logger = Logger.getLogger(MainService.class);

    static private MainService instance = null;

    static ServiceManager serviceManager;

    static private RootLogger ertLogger;

    static public synchronized MainService getInstance() {
        if (null == instance) {
            instance = new MainService();
            logger.info("MainService started");
        }
        return instance;
    }

    public static void main(String[] args) {
        // Disable IPv6, as it is not used at the moment
        System.setProperty("java.net.preferIPv6Addresses", "false");
        System.setProperty("java.net.preferIPv4Stack", "true");
        // A possible workaround for an apple bug in kqueue, so I now use the
        // older select
        System.setProperty("java.nio.preferSelect", "true");

        ertLogger = new RootLogger();
        serviceManager = ServiceManager.getInstance();

        MainService mainService = MainService.getInstance();
        ArrayList parameters = new ArrayList(Arrays.asList(args));
        boolean success = false;

        if (!parameters.isEmpty() && parameters.contains("-v")) {
            String version = "$AntReplaceVersion$";
            String date = "$AntReplaceDate$";
            System.out.println();
            System.out.println("ert version:");
            System.out.println("   Revision: " + version);
            System.out.println("       Date: " + date);
            System.out.println();
            System.exit(0);
        }

        if (!parameters.isEmpty() && parameters.contains("-c")) {
            // this means a command has been provided, now get the command
            if (parameters.size() > 1) {
                String command = (String) parameters.get(parameters
                        .indexOf("-c") + 1);
                command = command.toLowerCase();
                // only commands allowed are: "join" "first"
                if (command.equals("join")) {
                    logger.info("'-c join' command supplied at startup");
                    mainService.addWork(new DataItem("join"));
                    success = true;
                }
                if (command.equals("first")) {
                    logger.info("'-c first' command supplied at startup");
                    mainService.addWork(new DataItem("firstnode"));
                    success = true;
                }
                if (command.equals("writebench")) {
                    logger.info("'-c writebench' command supplied at startup");
                    String sendStatistics = (String) parameters.get(parameters
                            .indexOf("-s") + 1);
                    String dataSize = (String) parameters.get(parameters
                            .indexOf("-d") + 1);
                    if (sendStatistics != null) {
                        DataItem data = new DataItem("writebench");
                        data.setData(sendStatistics);
                        data.setDbSize(Integer.parseInt(dataSize));
                        mainService.addWork(data);
                        success = true;
                    }
                }
                if (command.equals("readbench")) {
                    logger.info("'-c readbench' command supplied at startup");
                    String sendStatistics = (String) parameters.get(parameters
                            .indexOf("-s") + 1);
                    if (sendStatistics != null) {
                        DataItem data = new DataItem("readbench");
                        data.setData(sendStatistics);
                        mainService.addWork(data);
                        success = true;
                    }
                }
                if (command.equals("deletebench")) {
                    logger.info("'-c deletebench' command supplied at startup");
                    String sendStatistics = (String) parameters.get(parameters
                            .indexOf("-s") + 1);
                    if (sendStatistics != null) {
                        DataItem data = new DataItem("deletebench");
                        data.setData(sendStatistics);
                        mainService.addWork(data);
                        success = true;
                    }
                }
                if (command.equals("bulk")) {
                    logger.info("'-c bulk' command supplied at startup");
                    mainService.addWork(new DataItem("bulk"));
                    success = true;
                }
                if (command.equals("collector")) {
                    logger.info("'-c collector' command supplied at startup");
                    String amount = (String) parameters.get(parameters
                            .indexOf("-a") + 1);
                    if (amount != null) {
                        DataItem data = new DataItem("collector");
                        data.setCursorSize(Integer.parseInt(amount));
                        mainService.addWork(data);
                        success = true;
                    }
                }
                if (command.equals("fillup")) {
                    logger.info("'-c fillup' command supplied at startup");
                    mainService.addWork(new DataItem("fillup"));
                    success = true;
                }
            }
        }

        if (success == true) {
            if (!parameters.isEmpty() && parameters.contains("-b")) {
                // this means we should run in the background with no GUI
                // starting
                // up
                logger.info("'-b' command supplied at startup."
                        + " Run in background, no GUI starting up");
                success = true;
            } else {
                Frame frame = new Frame();
                SplashScreen splashScreen = new SplashScreen("logo.jpg", frame);
            }
        } else {
            logger.error("Wrong or no command parameters supplied at startup");
            System.out.println();
            System.out.println("ert usage:");
            System.out.println();
            System.out.println("     -c [join|first|user]   - command:");
            System.out.println("        join"
                    + "                  join an existing network");
            System.out.println("        first"
                    + "                 be the first node in a new network");
            System.out.println("        user"
                    + "                  startup in user mode only,"
                    + " no storage on this node");
            System.out.println("     -b" + "                     "
                    + "- runs the server in background (no GUI)");
            System.out.println("     -v" + "                     "
                    + "- prints out the application version");
            System.out.println();
            System.out.println("example: java -jar ert.jar -c join -b");
            System.out.println();
            System.exit(1);
        }
    }

    /*
     * (non-Javadoc)
     *
     * @see ch.nix.ert.ServiceInterface#registerAtDispatcher()
     */
    public void registerAtDispatcher() {
        // TODO Auto-generated method stub
        registerService("quit");
    }

    /*
     * (non-Javadoc)
     *
     * @see java.lang.Runnable#run()
     */
    public void run() {
        // TODO Auto-generated method stub
        Thread.currentThread().setName("ERT.MainService");
        while (true) {
            DataItem dataItem = null;
            dataItem = getWork();
            // logger.debug("Got Work: "+dataItem);
            String operation = dataItem.getOperationType();
            if (operation.equals("quit")) {
                logger.info("quit request received, shutting down ERT");
                System.exit(0);
            }
        }
    }

}
```

File 37: main.SplashScreen

```
/*
 * Created on Oct 20, 2004 5:12:51 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-26 15:28:00 +0200 (Tue, 26 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 138 $
 * Location:    $URL: SplashScreen.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul written at
 * the University of Zurich, Switzerland Department of Information Technology
 * (www.ifi.unizh.ch) For copyright information please contact kussmaul@nix.ch
 *
 */

package ch.nix.ert.main;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Frame;
import java.awt.Toolkit;

import javax.swing.ImageIcon;
import javax.swing.JLabel;
import javax.swing.JWindow;
import javax.swing.SwingConstants;

/**
 * Example code taken from: <br>
 * http://www.javaworld.com/javaworld/javatips/jw-javatip104.html
 *
 * @author michi
 */
class SplashScreen extends JWindow {

    public SplashScreen(String filename, Frame f) {
        super(f);
        setBackground(Color.WHITE);

        JLabel pictureLabel = new JLabel(new ImageIcon(SplashScreen.class.getResource(filename)));
        JLabel versionLabel = new JLabel(" Version: $AntReplaceVersion$");
        versionLabel.setFont(new Font("SansSerif",Font.PLAIN, 10));
        versionLabel.setHorizontalAlignment(SwingConstants.LEFT);
        getContentPane().add(pictureLabel, BorderLayout.CENTER);
        getContentPane().add(versionLabel, BorderLayout.NORTH);
        pack();
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        Dimension labelSize = pictureLabel.getPreferredSize();
```

```java
        setLocation(screenSize.width / 2 - (labelSize.width / 2),
                screenSize.height / 2 - (labelSize.height / 2));
        setVisible(true);
        screenSize = null;
        labelSize = null;
    }
}
```

File 38: message.DataItem

```java
/*
 * Created on Oct 28, 2004 10:28:43 AM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-11-09 01:55:31 +0100 (Tue, 09 Nov 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 165 $
 * Location:    $URL: DataItem.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.message;

import java.io.Serializable;
import java.util.ArrayList;

import ch.nix.ert.message.operations.OperationFactory;
import ch.nix.ert.message.operations.OperationInterface;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class DataItem implements Cloneable, Serializable {

    protected Object[] operations = new Object[25];

    public Object[] getInternalArray() {
        return operations;
    }

    private void putValue(int opCode, Object value) {
//        if (operations[opCode] == null) {
//            operations[opCode] = OperationFactory.getOperation(opCode);
//        }
        operations[opCode] = value;
    }

    private Object getValue(int opCode) {
//        if (operations[opCode] == null) {
//            return null;
//        }
        return operations[opCode];
    }

    private void removeValue(int opCode) {
        operations[opCode] = null;
    }

    private synchronized boolean existValue(int opCode) {
        if (operations[opCode] == null) {
            return false;
        } else {
            return true;
        }
    }

    public Object clone() {
        try {
            Object dataItemCloned = super.clone();
            ((DataItem) dataItemCloned).operations = (Object[]) operations.clone();
            return dataItemCloned;
        } catch (CloneNotSupportedException e) {
            return null;
        }
    }

    public DataItem getClone() {
        return (DataItem) clone();
    }

    public DataItem() {
    }

    // helper Constructors, for common wrapping of Objects into a DataItem
    public DataItem(String operation) {
        this.setOperation(operation);
    }

    //    public DataItem(KeyPartition keyPartition) {
    //        //TODO always make a deep copy of this object first, otherwise the
    //        // keypartition does not allow concurrent access from multiple
    //        // threads....
    //        this.setKeyPartition(keyPartition);
    //    }
    //
    //    public DataItem(String operation, KeyPartition keyPartition) {
    //        //TODO always make a deep copy of this object first, otherwise the
    //        // keypartition does not allow concurrent access from multiple
    //        // threads....
    //        this.setOperation(operation);
    //        this.setKeyPartition(keyPartition);
    //    }

    public DataItem(InternalKey internalKey) {
        this.setInternalKey(internalKey);
    }

    public DataItem(DataItem nestedItem) {
        this.setNested(nestedItem);
    }

    public DataItem(String operation, DataItem nestedItem) {
        this.setOperation(operation);
        this.setNested(nestedItem);
    }

    public DataItem(Node destinationNode) {
        this.setDirectDestination(destinationNode);
    }

    //for the most common keys accessors are supplied

    public UserKey getUserKey() {
        return (UserKey) this.getValue(OperationInterface.USERKEY);
    }

    public void setUserKey(UserKey key) {
        this.putValue(OperationInterface.USERKEY, key);
    }

    public void removeUserKey() {
        this.removeValue(OperationInterface.USERKEY);
    }

    // Destination InternalKey (where the DataItem needs to be processed)
    public InternalKey getInternalKey() {
        InternalKey internalKey = (InternalKey) this
                .getValue(OperationInterface.INTERNALKEY);
        if (internalKey == null) {
            UserKey userkey = this.getUserKey();
            if (userkey == null) {
                return null;
            }
            internalKey = userkey.getInternalKey();
        }
        return internalKey;
    }

    public void setInternalKey(InternalKey iKey) {
        this.putValue(OperationInterface.INTERNALKEY, iKey);
    }

    public void removeInternalKey() {
        this.removeValue(OperationInterface.INTERNALKEY);
    }

    public String getNamespace() {
        String namespace = (String) this.getValue(OperationInterface.NAMESPACE);
        if (namespace == null) {
            namespace = "default";
        }
        return namespace;
    }

    public void setNamespace(String namespace) {
        this.putValue(OperationInterface.NAMESPACE, namespace);
    }

    public void removeNamespace() {
        this.removeValue(OperationInterface.NAMESPACE);
    }

    public String getData() {
        return (String) this.getValue(OperationInterface.DATA);
    }

    public void setData(String data) {
        this.putValue(OperationInterface.DATA, data);
    }

    public void removeData() {
        this.removeValue(OperationInterface.DATA);
    }

    public ArrayList getDataItems() {
        return (ArrayList) this.getValue(OperationInterface.DATAITEMS);
    }

    public void addDataItems(DataItem data) {
        if (this.getValue(OperationInterface.DATAITEMS) == null) {
            //lazy initialisation of ArrayList
            this.putValue(OperationInterface.DATAITEMS, new ArrayList());
        }
        ((ArrayList) this.getValue(OperationInterface.DATAITEMS)).add(data);
    }

    public void removeDataItems() {
        this.removeValue(OperationInterface.DATAITEMS);
    }

    public String getTransactionId() {
        return (String) this.getValue(OperationInterface.TRANSACTIONID);
    }

    public void setTransactionId(String transactionId) {
        if (transactionId != null) {
            this.putValue(OperationInterface.TRANSACTIONID, transactionId);
        }
    }

    public void removeTransactionId() {
        this.removeValue(OperationInterface.TRANSACTIONID);
    }

    public String getTransactionOperation() {
        return (String) this.getValue(OperationInterface.TRANSACTIONOPERATION);
    }

    public void setTransactionOperation(String operation) {
        if (operation != null) {
            this.putValue(OperationInterface.TRANSACTIONOPERATION, operation);
        }
    }

    public void removeTransactionOperation() {
        this.removeValue(OperationInterface.TRANSACTIONOPERATION);
    }

    public Node getTransactionNodeId() {
        return (Node) this.getValue(OperationInterface.TRANSACTIONNODEID);
    }

    public void setTransactionNodeId(Node node) {
        if (node != null) {
            this.putValue(OperationInterface.TRANSACTIONNODEID, node);
        }
    }

    public void removeTransactionNodeId() {
        this.removeValue(OperationInterface.TRANSACTIONNODEID);
    }

    public String getTransactionServiceId() {
        return (String) this.getValue(OperationInterface.TRANSACTIONSERVICEID);
    }

    public void setTransactionServiceId(String serviceId) {
        if (serviceId != null) {
            this.putValue(OperationInterface.TRANSACTIONSERVICEID, serviceId);
        }
    }

    public void removeTransactionServiceId() {
        this.removeValue(OperationInterface.TRANSACTIONSERVICEID);
    }

    public String getTransactionType() {
        String type = (String) this
                .getValue(OperationInterface.TRANSACTIONTYPE);
        return type;
    }

    public void setTransactionType(String type) {
        this.putValue(OperationInterface.TRANSACTIONTYPE, type);
    }

    public void removeTransactionType() {
        this.removeValue(OperationInterface.TRANSACTIONTYPE);
    }

    public void setTransactionSplit(int splitSize) {
        if (splitSize == 0) {
            removeTransactionSplit();
        } else {
            this.putValue(OperationInterface.TRANSACTIONSPLIT, new Integer(
                    splitSize));
        }
    }

    public int getTransactionSplit() {
        Integer value = (Integer) this
                .getValue(OperationInterface.TRANSACTIONSPLIT);
        if (value == null) {
            return 0;
        } else {
            return value.intValue();
        }
    }

    public void removeTransactionSplit() {
        this.removeValue(OperationInterface.TRANSACTIONSPLIT);
    }

    public boolean isTransactionLocal() {
        Node node = getTransactionNodeId();
        if (node == null) {
            return true;
        } else {
            return getTransactionNodeId().isLocalhost();
        }
    }

    public String getRouting() {
        //values are: local, routing, direct
        String routing = (String) this.getValue(OperationInterface.ROUTING);
        return routing;
    }

    public void setRouting(String routingHint) {
        this.putValue(OperationInterface.ROUTING, routingHint);
    }

    public void removeRounting() {
        this.removeValue(OperationInterface.ROUTING);
    }

    public String getOperationType() {
        String type = this.getTransactionType();
        String operation = this.getOperation();
        if (type != null && type.equals("end")) {
            return operation + "-" + type;
        }
        return operation;
    }

    public String getOperation() {
        return ((String) this.getValue(OperationInterface.OPERATION));
    }

    public void setOperation(String operation) {
        this.putValue(OperationInterface.OPERATION, operation);
```

```java
    }

    public void removeOperation() {
        this.removeValue(OperationInterface.OPERATION);
    }

    public Node getDirectDestination() {
        return (Node) this.getValue(OperationInterface.DIRECTDESTINATION);
    }

    public void setDirectDestination(Node node) {
        this.putValue(OperationInterface.DIRECTDESTINATION, node);
    }

    public void removeDirectDestination() {
        this.removeValue(OperationInterface.DIRECTDESTINATION);
    }

    public Node getDirectOriginator() {
        return (Node) this.getValue(OperationInterface.DIRECTORIGINATOR);
    }

    public void setDirectOriginator(Node node) {
        this.putValue(OperationInterface.DIRECTORIGINATOR, node);
    }

    public void removeDirectOriginator() {
        this.removeValue(OperationInterface.DIRECTORIGINATOR);
    }

    public void setServiceOriginator(String service) {
        this.putValue(OperationInterface.SERVICEORIGINATOR, service);
    }

    public String getServiceOriginator() {
        return (String) this.getValue(OperationInterface.SERVICEORIGINATOR);
    }

    public void removeServiceOriginator() {
        this.removeValue(OperationInterface.SERVICEORIGINATOR);
    }

    public DataItem getNested() {
        return (DataItem) this.getValue(OperationInterface.NESTED);
    }

    public void setNested(DataItem data) {
        this.putValue(OperationInterface.NESTED, data);
    }

    public void removeNested() {
        this.removeValue(OperationInterface.NESTED);
    }

    //    // TODO currently only used for NodeDirectory to publish, unpublish
    // data...
    //    // will later perhaps removed to simplify the complex DataItem Object...
    //    public KeyPartition getKeyPartition() {
    //        return (KeyPartition) this.getValue("keypartition");
    //    }
    //
    //    public void setKeyPartition(KeyPartition keyPartition) {
    //        this.putValue("keypartition", keyPartition);
    //    }
    //
    //    public void removeKeyPartition() {
    //        this.removeValue("keypartition");
    //    }

    //START: Statistic related stuff
    public void setDbSize(int size) {
        this.putValue(OperationInterface.DBSIZE, new Integer(size));
    }

    public int getDbSize() {
        return ((Integer) this.getValue(OperationInterface.DBSIZE)).intValue();
    }

    public void removeDbSize() {
        this.removeValue(OperationInterface.DBSIZE);
    }

    public void setNodeLoad(double size) {
        this.putValue(OperationInterface.NODELOAD, new Double(size));
    }

    public double getNodeLoad() {
        return ((Double) this.getValue(OperationInterface.NODELOAD))
                .doubleValue();
    }

    public void removeNodeLoad() {
        this.removeValue(OperationInterface.NODELOAD);
    }

    //END: Statistic related stuff

    //START: Callback Service related stuff:
    public void setDelay(long delay) {
        this.putValue(OperationInterface.DELAY, new Long(delay));
    }

    public long getDelay() {
        return ((Long) this.getValue(OperationInterface.DELAY)).longValue();
    }

    public void removeDelay() {
        this.removeValue(OperationInterface.DELAY);
    }

    //END: Callback related stuff

    public void setCursorSize(int size) {
        this.putValue(OperationInterface.CURSORSIZE, new Integer(size));
    }

    public int getCursorSize() {
        if (this.existValue(OperationInterface.CURSORSIZE)) {
            return ((Integer) this.getValue(OperationInterface.CURSORSIZE))
                    .intValue();
        } else {
            return 1;
        }
    }

    public void removeCursorSize() {
        this.removeValue(OperationInterface.CURSORSIZE);
    }

    public InternalKey getLastInternalKey() {
        return (InternalKey) this.getValue(OperationInterface.LASTINTERNALKEY);
    }

    public void setLastInternalKey(InternalKey iKey) {
        this.putValue(OperationInterface.LASTINTERNALKEY, iKey);
    }

    public void removeLastInternalKey() {
        this.removeValue(OperationInterface.LASTINTERNALKEY);
    }

    public InternalKey getFirstInternalKey() {
        return (InternalKey) this.getValue(OperationInterface.FIRSTINTERNALKEY);
    }

    public void setFirstInternalKey(InternalKey iKey) {
        this.putValue(OperationInterface.FIRSTINTERNALKEY, iKey);
    }

    public void removeFirstInternalKey() {
        this.removeValue(OperationInterface.FIRSTINTERNALKEY);
    }

    public void setKeyPartitionId(int id) {
        this.putValue(OperationInterface.KEYPARTITIONID, new Integer(id));
    }

    public int getKeyPartitionId() {
        if (this.existValue(OperationInterface.KEYPARTITIONID)) {
            return ((Integer) this.getValue(OperationInterface.KEYPARTITIONID))
                    .intValue();
        } else {
            return -1;
        }
    }

    public void removeKeyPartitionId() {
        this.removeValue(OperationInterface.KEYPARTITIONID);
    }

    public void setResult(boolean outcome) {
        this.putValue(OperationInterface.RESULT, Boolean.valueOf(outcome));
    }

    public boolean getResult() {
        Boolean value = (Boolean) this.getValue(OperationInterface.RESULT);
        if (value == null) {
            return true;
        } else {
            return value.booleanValue();
        }
    }

    public void removeResult() {
        this.removeValue(OperationInterface.RESULT);
    }

    public String toString() {
        String output = "";
        for (int i = 0; i < operations.length; i++) {
            Object array_element = operations[i];
            if (array_element != null) {
                output += OperationInterface.OPERATIONNAMES[i] + ":"
                        + array_element + " ";
            }
        }
        ;
        return "[ " + output + "]";
    }
}
```

File 39: message.DataItemCoder

```
/*
 * Created on Oct 26, 2004 4:14:09 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-11-12 12:46:25 +0100 (Fri, 12 Nov 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 173 $
 * Location:    $URL: DataItemCoder.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.message;

import java.nio.ByteBuffer;
import java.util.Enumeration;
import java.util.Hashtable;
import java.util.Iterator;

import org.apache.log4j.Logger;

import ch.nix.ert.communication.netty2.NodeCommunicationClientListener;
import ch.nix.ert.message.operations.OperationFactory;
import ch.nix.ert.message.operations.OperationInterface;

/**
 * Codes the DataItem in ASN.1-kind way (key-length-value)
 *
 * @author michi
 */
public class DataItemCoder {

    //    // logger:
    //    static Logger logger = Logger.getLogger(DataItemCoder.class);

    private Object[] operations;

    private int currentArrayElement = 0;

    private byte[] bytePart = null;

    private byte operationCode;

    private boolean operationCodeRead = false;

    private int offsetPart;
```

```java

    private boolean startNewPart = true;

    final static private byte endMarker = (byte) 0xff;

    private DataItem receivedDataItem = null;

    public DataItemCoder() {
        receivedDataItem = new DataItem();
        operations = receivedDataItem.getInternalArray();
    }

    public void setDataItem(DataItem data) {
        operations = data.getInternalArray();
    }

    public DataItem getDataItem() {
        return receivedDataItem;
    }

    public synchronized boolean encode(ByteBuffer buf) {
        int bufferSize = buf.remaining();
        // first write all the data which could net yet be put on the buffer
        if (bytePart != null) {
            int stillToSend = bytePart.length - offsetPart;
            if (bufferSize >= stillToSend) {
                buf.put(bytePart, offsetPart, stillToSend);
                bufferSize -= stillToSend;
                offsetPart = 0;
            } else {
                buf.put(bytePart, offsetPart, bufferSize);
                offsetPart += bufferSize;
                return false;
            }
        }
        while (true) {
            for (int opCode = currentArrayElement; opCode < operations.length; opCode++) {
                Object element = operations[opCode];
                if (element != null) {
                    OperationInterface operationCoder = OperationFactory
                            .getOperation(opCode);
                    bytePart = operationCoder.encode(element);
                    int length = bytePart.length;
                    if (bufferSize > 4) {
                        buf.put((byte) opCode).putInt(length);
                        bufferSize -= 5;
                    }
                    currentArrayElement = opCode + 1;
                    if (bufferSize >= length) {
                        buf.put(bytePart, 0, length);
                        bufferSize -= length;
                    } else {
                        buf.put(bytePart, 0, bufferSize);
                        offsetPart = bufferSize;
                        return false;
                    }
                }
            }
            if (bufferSize > 0) {
                buf.put(endMarker);
                //                byte[] buffer = buf.array();
                //                for (int i = 0; i < buffer.length; i++) {
                //                  System.out.println("buffer: "+buffer[i]);
                //              }
                // logger.debug("Wrote endmarker " + operations[6].getValue());
                return true;
            } else {
                return false;
            }
        }
    }

    public synchronized boolean decode(ByteBuffer buf) {
        int bufferSize = buf.remaining();
        //System.out.println("decoder: "+buf);
        while (true) {
            // for (int opCode = currentArrayElement; opCode <
            // operations.length; opCode++) {
            if (startNewPart) {
                if (bufferSize >= 1 && !operationCodeRead) {
                    operationCode = buf.get();
                    operationCodeRead = true;
                    bufferSize -= 1;
                    if (operationCode == (byte) 0xff) {
                        // this is the marker, that we have reached the end of
                        // this dataItem
                        return true;
                    }
                }
                if (bufferSize > 4) {
                    int sizeOfPart = buf.getInt();
                    bytePart = new byte[sizeOfPart];
                    offsetPart = 0;
                    startNewPart = false;
                    bufferSize -= 4;
                } else {
                    return false;
                }
            }
            int stillToReceive = bytePart.length - offsetPart;
            if (bufferSize >= stillToReceive) {
                buf.get(bytePart, offsetPart, stillToReceive);
                OperationInterface newOperation = OperationFactory
                        .getOperation(operationCode);
                operations[operationCode] = newOperation.decode(bytePart);
                bufferSize -= stillToReceive;
                operationCodeRead = false;
                startNewPart = true;
            } else {
                buf.get(bytePart, offsetPart, bufferSize);
                offsetPart += bufferSize;
                return false;
            }

        }
    }

}
```

File 40: message.InternalKey

```java
/*
 * Created on Jul 27, 2004 4:14:37 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-11-01 11:26:55 +0100 (Mon, 01 Nov 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 151 $
 * Location:    $URL: InternalKey.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */

package ch.nix.ert.message;

import java.io.Serializable;

/**
 *
 * TODO Edit this text for a better description
 * <p>
 * TODO Has to check what the ranges of the keys are (e.g. 0-1000 or
 * 0-100000000000) and should reject invalid values!
 *
 *
 * @author michi
 */
public class InternalKey implements Serializable, Comparable {

    private long id;

    static private long lowerLimit = 0;

    static private long higherLimit = 1000000;

    /**
     *
     */
    public InternalKey(long uniqueId) {
        uniqueId = adjustKey(uniqueId);
        this.id = uniqueId;
    }

    public InternalKey(String uniqueId) {
        this.id = adjustKey(Long.decode(uniqueId).longValue());
    }

    public InternalKey getSubsequentInternalKey() {
        return new InternalKey(this.id + 1);
    }

    public InternalKey getNthSubsequentInternalKey(long amount) {
        return new InternalKey(this.id + amount);
    }

    public InternalKey getPreviousInternalKey() {
        return new InternalKey(this.id - 1);
    }

    public InternalKey getNthPreviousInternalKey(long amount) {
        return new InternalKey(this.id - amount);
    }

    /**
     * @return Returns the uniqueID.
     */
    public long getId() {
        return id;
    }

    static public long getHigherLimit() {
        return higherLimit;
    }

    static public long getLowerLimit() {
        return lowerLimit;
    }

    static public InternalKey getRandom() {
        long difference = getHigherLimit() - getLowerLimit();
        //Random generator = new Random();
        long random = (long) (Math.random() * (double) difference);
        return new InternalKey(random + getLowerLimit());
    }

    public UserKey getFirstUserKey() {
        long internalKeyDiff = getHigherLimit() - getLowerLimit();
        long userKeyDiff = UserKey.getHigherLimit()
                - UserKey.getLowerLimit();
        float scalingFactor = (float) userKeyDiff / (float) internalKeyDiff;
        //System.out.println("scalingFactor: "+scalingFactor);
        long calculatedUserKey = (long) ((float) getId() * scalingFactor);
        //System.out.println("calculatedUserKey: "+calculatedUserKey);
        return new UserKey(calculatedUserKey
                + UserKey.getLowerLimit());
    }

    public UserKey getLastUserKey() {
        long internalKeyDiff = getHigherLimit() - getLowerLimit();
        long userKeyDiff = UserKey.getHigherLimit()
                - UserKey.getLowerLimit();
        float scalingFactor = (float) userKeyDiff / (float) internalKeyDiff;
        //System.out.println("scalingFactor: "+scalingFactor);
        long calculatedUserKey = (long) ((float) (getId()+1) * scalingFactor);
        //System.out.println("calculatedUserKey: "+calculatedUserKey);
        return new UserKey(calculatedUserKey
                + UserKey.getLowerLimit() -1 );

    }

    /**
     * @param uniqueID
     *            The uniqueID to set.
     */
    public void setId(long uniqueId) {
        uniqueId = adjustKey(uniqueId);
        this.id = uniqueId;
    }

    /*
     * (non-Javadoc)
     *
     * @see java.lang.Comparable#compareTo(java.lang.Object)
     *
     * Note: This class has a natural ordering that is inconsistent with equals
     */
    public int compareTo(Object o) {
        InternalKey internalKey = (InternalKey) o;
        if (internalKey.getId() > this.getId()) {
            return -1;
        } else if (internalKey.getId() < this.getId()) {
            return 1;
        } else {
            return 0;
        }
    }

    public long amountBetween(InternalKey internalKey) {
        return Math.abs(this.getId() - internalKey.getId());
    }

    private long adjustKey(long uniqueId) {
        if (uniqueId > higherLimit) {
            return higherLimit;
        }
        if (uniqueId < lowerLimit) {
            return lowerLimit;
        }
        return uniqueId;
    }

    public String toString() {
        return String.valueOf(this.getId());
    }
}
```

File 41: message.Node

```java
/*
 * Created on Aug 2, 2004 11:06:57 AM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-28 22:38:52 +0200 (Thu, 28 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 144 $
 * Location:    $URL: Node.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */

package ch.nix.ert.message;

import java.io.Serializable;
import java.net.InetAddress;
import java.net.InetSocketAddress;
import java.net.UnknownHostException;

/**
 * TODO Edit this text for a better description
 *
 * Represents the nodes (in this implementation IP/Port)
 *
 * @author michi
 */
public class Node implements Serializable {

    InetSocketAddress nodeAddress;

    /**
     *
     */
    public Node(String ip, int port) {
        nodeAddress = new InetSocketAddress(ip, port);
    }

    public Node(InetAddress ip, int port) {
        nodeAddress = new InetSocketAddress(ip, port);
    }

    /**
     * Constructs a node with localhost as ip, and a  port as supplied
     */
    public Node(int port) {
        try {
            nodeAddress = new InetSocketAddress(InetAddress.getLocalHost(),port);
        } catch (UnknownHostException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    /**
     * Constructs a node with localhost as ip, and a standard port as port... (15000)
     */
    public Node() {
        try {
            nodeAddress = new InetSocketAddress(InetAddress.getLocalHost(),15000);
        } catch (UnknownHostException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    /**
     * @return true: Node is on localhost
     *         <p>
     *         false: Node is on a remote host
     */
    public boolean isLocalhost() {
        InetAddress local;
        try {
            local = InetAddress.getLocalHost();
            return local.equals(nodeAddress.getAddress());
        } catch (UnknownHostException e) {
            // TODO Auto-generated catch block
            System.out.println("Error in Node isLocalhost: " + e);
        }
        return false;
    }

//    public boolean isAvailable() {
//        // compose a "ping" packet!
//        DataItem ping = new DataItem();
//        ping.setDirectRecipient(this);
//        ping.setOperation("ping");
//        DataItem response = ServiceManager.getInstance().getMessageHandler().sendAndWait(ping);
//        if (response == null) {
//            return false;
//        } else {
//            return true;
//        }
//
//    }

    public InetSocketAddress getInetSocketAddress() {
        return nodeAddress;
    }

    public String getIp() {
        return nodeAddress.getAddress().getHostAddress();
    }

    public int getPort() {
        return nodeAddress.getPort();
    }

    public String toString() {
        return nodeAddress.getAddress().getHostAddress() + ":"
                + nodeAddress.getPort();
    }

}
```

File 42: message.UserKey

```java
/*
 * Created on Sep 14, 2004 12:40:18 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-11-01 11:26:55 +0100 (Mon, 01 Nov 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 151 $
 * Location:    $URL: UserKey.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.message;

import java.io.Serializable;
import java.util.Comparator;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class UserKey implements Serializable, Comparable, Comparator {

    private long id;

    static private long lowerLimit = 0;

    static private long higherLimit = 100000000;

    public UserKey(long uniqueId) {
        this.id = uniqueId;
    }

    public UserKey(Long uniqueId) {
        this.id = uniqueId.longValue();
    }

    public UserKey(String uniqueId) {
        this.id = Long.decode(uniqueId).longValue();
    }

    public UserKey getSubsequentUserKey() {
        return new UserKey(this.id + 1);
    }

    public UserKey getPreviousInternalKey() {
        return new UserKey(this.id - 1);
    }

    static public long getHigherLimit() {
        return higherLimit;
    }

    static public long getLowerLimit() {
        return lowerLimit;
    }

    public InternalKey getInternalKey() {
        long internalKeyDiff = InternalKey.getHigherLimit()
                - InternalKey.getLowerLimit();
        long userKeyDiff = getHigherLimit() - getLowerLimit();
        float scalingFactor = (float) userKeyDiff / (float) internalKeyDiff;
        long calculatedInternalKey = (long) ((float) getId() / scalingFactor);
        return new InternalKey(calculatedInternalKey
                + InternalKey.getLowerLimit());
    }

    /**
     * @return Returns the uniqueID.
     */
    public long getId() {
        return id;
    }

    /*
     * (non-Javadoc)
     *
     * @see java.lang.Comparable#compareTo(java.lang.Object)
     */
    public int compareTo(Object o) {
        InternalKey internalKey = (InternalKey) o;
        if (internalKey.getId() > this.getId()) {
            return -1;
        } else if (internalKey.getId() < this.getId()) {
            return 1;
        } else {
            return 0;
        }
    }

    public String toString() {
        return String.valueOf(this.getId());
    }

    public boolean equals(Object o) {
        try {
            UserKey userKey = (UserKey) o;
            if (userKey.getId() == getId()) {
                return true;
            } else {
                return false;
            }
        } catch (ClassCastException e) {
            return false;
        }
    }

    public int hashCode() {
        throw new UnsupportedOperationException();
    }

    /*
     * (non-Javadoc)
     *
     * @see java.util.Comparator#compare(java.lang.Object, java.lang.Object)
     */
    public int compare(Object o1, Object o2) {
        UserKey userKey1 = (UserKey) o1;
        UserKey userKey2 = (UserKey) o2;
        return userKey1.compareTo(userKey2);
    }

}
```

File 43: message.operations.CursorSize

```
/*
 * Created on Oct 28, 2004 3:25:48 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-31 12:28:25 +0100 (Sun, 31 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 150 $
 * Location:    $URL: CursorSize.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.message.operations;

import netbyte.NetByte;
import netbyte.NetByteDecodingException;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class CursorSize implements OperationInterface {

    public byte[] encode(Object cursorSize) {
        return NetByte.encodeInt(((Integer) cursorSize).intValue());
    }

    public Object decode(byte[] buffer) {
        try {
            return new Integer(NetByte.decodeInt(buffer));
        } catch (NetByteDecodingException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            return null;
        }
    }

}
```

File 44: message.operations.Data

```
/*
 * Created on Oct 28, 2004 3:18:41 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-31 12:28:25 +0100 (Sun, 31 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 150 $
 * Location:    $URL: Data.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.message.operations;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class Data implements OperationInterface {

    public byte[] encode(Object data) {
        return ((String) data).getBytes();
    }

    public Object decode(byte[] buffer) {
        return new String(buffer);
    }

}
```

File 45: message.operations.DataItems

```
/*
 * Created on Oct 28, 2004 3:19:08 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-31 12:28:25 +0100 (Sun, 31 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 150 $
 * Location:    $URL: DataItems.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.message.operations;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class DataItems implements OperationInterface {

    /*
     * (non-Javadoc)
     *
     * @see ch.nix.ert.message.operations.OperationInterface#encode()
     */
    public byte[] encode(Object DataItems) {
        // TODO Auto-generated method stub
        return null;
    }

    /*
     * (non-Javadoc)
     *
     * @see ch.nix.ert.message.operations.OperationInterface#decode(byte[])
     */
    public Object decode(byte[] binary) {
        // TODO Auto-generated method stub
        return null;
    }

}
```

File 46: message.operations.DbSize

```java
/*
 * Created on Oct 28, 2004 3:24:49 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-31 12:28:25 +0100 (Sun, 31 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 150 $
 * Location:    $URL: DbSize.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.message.operations;

import netbyte.NetByte;
import netbyte.NetByteDecodingException;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class DbSize implements OperationInterface {

    public byte[] encode(Object dbSize) {
        return NetByte.encodeInt(((Integer) dbSize).intValue());
    }

    public Object decode(byte[] buffer) {
        try {
            return new Integer(NetByte.decodeInt(buffer));
        } catch (NetByteDecodingException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            return null;
        }
    }
}
```

File 47: message.operations.Delay

```java
/*
 * Created on Oct 28, 2004 3:25:30 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-31 12:28:25 +0100 (Sun, 31 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 150 $
 * Location:    $URL: Delay.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.message.operations;

import netbyte.NetByte;
import netbyte.NetByteDecodingException;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class Delay implements OperationInterface {

    public byte[] encode(Object delay) {
        return NetByte.encodeLong(((Long) delay).intValue());
    }

    public Object decode(byte[] buffer) {
        try {
            return new Long(NetByte.decodeLong(buffer));
        } catch (NetByteDecodingException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            return null;
        }
    }
}
```

File 48: message.operations.DirectDestination

```
/*
 * Created on Oct 28, 2004 3:23:04 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-31 12:28:25 +0100 (Sun, 31 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 150 $
 * Location:    $URL: DirectDestination.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.message.operations;

import java.io.Serializable;
import java.net.InetAddress;
import java.net.InetSocketAddress;
import java.net.UnknownHostException;

import netbyte.NetByte;
import netbyte.NetByteDecodingException;
import ch.nix.ert.message.Node;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class DirectDestination implements OperationInterface {

    public byte[] encode(Object value) {
        Node destinationNode = (Node) value;
        // will store it as:
        //   ip
        //   port
        byte[] port = NetByte.encodeInt(destinationNode.getPort());
        byte[] address = destinationNode.getInetSocketAddress().getAddress()
                .getAddress();
        byte[] returnValue = new byte[address.length + 4];
        System.arraycopy(address, 0, returnValue, 0, address.length);
        System.arraycopy(port, 0, returnValue, 4, 4);
        //        for (int i = 0; i < returnValue.length; i++) {
        //            System.out.println("ENCODE " + i + ": " + returnValue[i]);
        //        }
        return returnValue;
    }

    public Object decode(byte[] buffer) {
        //        for (int i = 0; i < buffer.length; i++) {
        //            System.out.println("DECODE " + i + ": " + buffer[i]);
        //        }
        try {
            byte[] byteIp = new byte[4];
            System.arraycopy(buffer, 0, byteIp, 0, 4);
            InetAddress ip = InetAddress.getByAddress(byteIp);
            int port = NetByte.decodeInt(buffer, 4);
            //String ip = new String(buffer, 4, buffer.length - 4);
            return new Node(ip, port);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            System.out.println("ERROR in here");
            e.printStackTrace();
            return null;
        }
    }
}
```

File 49: message.operations.DirectOriginator

```java
/*
 * Created on Oct 28, 2004 3:23:30 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-11-11 18:55:48 +0100 (Thu, 11 Nov 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 169 $
 * Location:    $URL: DirectOriginator.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.message.operations;

import java.io.Serializable;
import java.net.InetAddress;

import netbyte.NetByte;
import ch.nix.ert.message.Node;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class DirectOriginator implements OperationInterface {

    public byte[] encode(Object value) {
        Node originatingNode = (Node) value;
        // will store it as:
        //   ip
        //   port
        byte[] port = NetByte.encodeInt(originatingNode.getPort());
        byte[] address = originatingNode.getInetSocketAddress().getAddress()
                .getAddress();
        byte[] returnValue = new byte[address.length + 4];
        System.arraycopy(address, 0, returnValue, 0, address.length);
        System.arraycopy(port, 0, returnValue, 4, 4);
        return returnValue;
    }

    public Object decode(byte[] buffer) {
        try {
            byte[] byteIp = new byte[4];
            System.arraycopy(buffer, 0, byteIp, 0, 4);
            InetAddress ip = InetAddress.getByAddress(byteIp);
            int port = NetByte.decodeInt(buffer, 4);
            //String ip = new String(buffer, 4, buffer.length - 4);
            return new Node(ip, port);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            System.out.println("ERROR in here");
            e.printStackTrace();
            return null;
        }
    }
}
```

File 50: message.operations.FirstInternalKey

```java
/*
 * Created on Oct 28, 2004 3:26:26 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-31 12:28:25 +0100 (Sun, 31 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 150 $
 * Location:    $URL: FirstInternalKey.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.message.operations;
```

```java

import netbyte.NetByte;
import netbyte.NetByteDecodingException;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class FirstInternalKey implements OperationInterface {

    public byte[] encode(Object iKey) {
        return NetByte.encodeLong(((ch.nix.ert.message.InternalKey) iKey)
                .getId());
    }

    public Object decode(byte[] buffer) {
        try {
            return new ch.nix.ert.message.InternalKey(NetByte
                    .decodeLong(buffer));
        } catch (NetByteDecodingException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            return null;
        }
    }
}
```

File 51: message.operations.InternalKey

```java
/*
 * Created on Oct 28, 2004 3:18:18 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-31 12:28:25 +0100 (Sun, 31 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 150 $
 * Location:    $URL: InternalKey.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.message.operations;

import netbyte.NetByte;
import netbyte.NetByteDecodingException;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class InternalKey implements OperationInterface {

    public byte[] encode(Object iKey) {
        return NetByte.encodeLong(((ch.nix.ert.message.InternalKey) iKey).getId());
    }

    public Object decode(byte[] buffer) {
        try {
            return new ch.nix.ert.message.InternalKey(NetByte.decodeLong(buffer));
        } catch (NetByteDecodingException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            return null;
        }
    }
}
```

File 52: message.operations.KeyPartitionId

```java
/*
 * Created on Oct 28, 2004 3:26:58 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-31 12:28:25 +0100 (Sun, 31 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 150 $
 * Location:    $URL: KeyPartitionId.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.message.operations;

import netbyte.NetByte;
import netbyte.NetByteDecodingException;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class KeyPartitionId implements OperationInterface {

    public byte[] encode(Object partId) {
        return NetByte.encodeInt(((Integer) partId).intValue());
    }

    public Object decode(byte[] buffer) {
        try {
            return new Integer(NetByte.decodeInt(buffer));
        } catch (NetByteDecodingException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            return null;
        }
    }
}
```

File 53: message.operations.LastInternalKey

```java
/*
 * Created on Oct 28, 2004 3:26:08 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-31 12:28:25 +0100 (Sun, 31 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 150 $
 * Location:    $URL: LastInternalKey.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.message.operations;

import netbyte.NetByte;
import netbyte.NetByteDecodingException;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class LastInternalKey implements OperationInterface {

    public byte[] encode(Object iKey) {
        return NetByte.encodeLong(((ch.nix.ert.message.InternalKey) iKey)
                .getId());
    }

    public Object decode(byte[] buffer) {
        try {
            return new ch.nix.ert.message.InternalKey(NetByte
                    .decodeLong(buffer));
        } catch (NetByteDecodingException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            return null;
        }
    }
}
```

File 54: message.operations.Namespace

```
/*
 * Created on Oct 28, 2004 2:58:08 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-31 12:28:25 +0100 (Sun, 31 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 150 $
 * Location:    $URL: Namespace.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.message.operations;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class Namespace implements OperationInterface {

    public byte[] encode(Object namespace) {
        return ((String) namespace).getBytes();
    }

    public Object decode(byte[] buffer) {
        return new String(buffer);
    }

}
```

File 55: message.operations.Nested

```
/*
 * Created on Oct 28, 2004 3:24:26 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-31 12:28:25 +0100 (Sun, 31 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 150 $
 * Location:    $URL: Nested.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.message.operations;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class Nested implements OperationInterface {

    public byte[] encode(Object nested) {
        //TODO not supported yet...
        return null;
    }

    public Object decode(byte[] buffer) {
        return null;
    }
}
```

File 56: message.operations.NodeLoad

```
/*
 * Created on Oct 28, 2004 3:25:10 PM
```

```java
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-11-09 01:55:31 +0100 (Tue, 09 Nov 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 165 $
 * Location:    $URL: NodeLoad.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.message.operations;

import netbyte.NetByte;
import netbyte.NetByteDecodingException;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class NodeLoad implements OperationInterface {

    public byte[] encode(Object load) {
        return NetByte.encodeDouble(((Double) load).doubleValue());
    }

    public Object decode(byte[] buffer) {
        try {
            return new Double(NetByte.decodeDouble(buffer));
        } catch (NetByteDecodingException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            return null;
        }
    }
}
```

File 57: message.operations.Operation

```java
/*
 * Created on Oct 26, 2004 5:18:41 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-31 12:28:25 +0100 (Sun, 31 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 150 $
 * Location:    $URL: Operation.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.message.operations;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class Operation implements OperationInterface {

    public byte[] encode(Object operationCode) {
        return ((String) operationCode).getBytes();
    }

    public Object decode(byte[] buffer) {
        return  new String(buffer);
    }

    //    public boolean encode(Object operation, ByteBuffer buf) {
    //        String code = (String) operation;
    //        byte[] raw = ((String) operation).getBytes();
    //        int length = raw.length;
    //        if (buf.remaining() < length) {
    //            return false;
    //        }
    //        buf.put(OPERATION).putInt(length).put(raw);
    //        return true;
    //    }
    //
    //    public void decode (DataItem data, ByteBuffer buf) {
    //        int length = buf.getInt();
    //        byte[] raw = new byte[length];
    //        String opCode = new String(buf.get(raw).array());
    //        data.putValue("operation", opCode);
    //    }

    /*
     * (non-Javadoc)
     *
     * @see ch.nix.ert.message.operations.OperationInterface#getName()
     */

}
```

File 58: message.operations.OperationFactory

```
/*
 * Created on Oct 28, 2004 12:21:26 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-28 17:44:43 +0200 (Thu, 28 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 143 $
 * Location:    $URL: OperationFactory.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.message.operations;

import org.apache.log4j.Logger;

import ch.nix.ert.keypartition.MigrationClient;

import com.sleepycat.je.tree.DIN;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class OperationFactory {

    // logger:
    static Logger logger = Logger.getLogger(OperationFactory.class);

    public static OperationInterface getOperation(int opCode) {
        switch (opCode) {
        case OperationInterface.OPERATION:
            return new Operation();
        case OperationInterface.USERKEY:
            return new UserKey();
        case OperationInterface.INTERNALKEY:
            return new InternalKey();
        case OperationInterface.NAMESPACE:
            return new Namespace();
        case OperationInterface.DATA:
            return new Data();
        case OperationInterface.DATAITEMS:
            return new DataItems();
        case OperationInterface.TRANSACTIONID:
            return new TransactionId();
        case OperationInterface.TRANSACTIONOPERATION:
            return new TransactionOperation();
        case OperationInterface.TRANSACTIONNODEID:
            return new TransactionNodeId();
        case OperationInterface.TRANSACTIONSERVICEID:
            return new TransactionServiceId();
        case OperationInterface.TRANSACTIONTYPE:
            return new TransactionType();
        case OperationInterface.TRANSACTIONSPLIT:
            return new TransactionSplit();
        case OperationInterface.ROUTING:
            return new Routing();
        case OperationInterface.DIRECTDESTINATION:
```

```java
            return new DirectDestination();
        case OperationInterface.DIRECTORIGINATOR:
            return new DirectOriginator();
        case OperationInterface.SERVICEORIGINATOR:
            return new ServiceOriginator();
        case OperationInterface.NESTED:
            return new Nested();
        case OperationInterface.DBSIZE:
            return new DbSize();
        case OperationInterface.NODELOAD:
            return new NodeLoad();
        case OperationInterface.DELAY:
            return new Delay();
        case OperationInterface.CURSORSIZE:
            return new CursorSize();
        case OperationInterface.LASTINTERNALKEY:
            return new LastInternalKey();
        case OperationInterface.FIRSTINTERNALKEY:
            return new FirstInternalKey();
        case OperationInterface.KEYPARTITIONID:
            return new KeyPartitionId();
        case OperationInterface.RESULT:
            return new Result();
        default:
            logger.error("Can't handle unknown Message Attribute with ID"+opCode);
            return null;
        }
    }

}
```

File 59: message.operations.OperationInterface

```java
/*
 * Created on Oct 26, 2004 5:22:53 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-11-02 01:36:34 +0100 (Tue, 02 Nov 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 154 $
 * Location:    $URL: OperationInterface.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.message.operations;

import java.io.Serializable;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public interface OperationInterface extends Serializable {

    // list of all keys for DataItem
    public static final int OPERATION = 0;

    public static final int USERKEY = 1;

    public static final int INTERNALKEY = 2;

    public static final int NAMESPACE = 3;

    public static final int DATA = 4;

    public static final int DATAITEMS = 5;

    public static final int TRANSACTIONID = 6;

    public static final int TRANSACTIONOPERATION = 7;

    public static final int TRANSACTIONNODEID = 8;

    public static final int TRANSACTIONSERVICEID = 9;

    public static final int TRANSACTIONTYPE = 10;

    public static final int TRANSACTIONSPLIT = 11;

    public static final int ROUTING = 12;

    public static final int DIRECTDESTINATION = 13;

    public static final int DIRECTORIGINATOR = 14;

    public static final int SERVICEORIGINATOR = 15;

    public static final int NESTED = 16;

    public static final int DBSIZE = 17;

    public static final int NODELOAD = 18;

    public static final int DELAY = 19;

    public static final int CURSORSIZE = 20;

    public static final int FIRSTINTERNALKEY = 21;

    public static final int LASTINTERNALKEY = 22;

    public static final int KEYPARTITIONID = 23;

    public static final int RESULT = 24;

    public static final String[] OPERATIONNAMES = { "Operation", "UserKey",
            "InternalKey", "Namespace", "Data", "DataItems", "TransactionId",
            "TransactionOperation", "TransactionNodeId",
            "TransactionServiceId", "TransactionType", "TransactionSplit",
            "Routing", "DirectDestination", "DirectOriginator",
            "ServiceOriginator", "Nested", "DbSize", "NodeLoad", "Delay",
            "CursorSize", "FirstInternalKey", "LastInternalKey", "KeyPartitionId", "Result" };

    public byte[] encode(Object value);

    public Object decode(byte[] binary);

    // public boolean encode(Object value, ByteBuffer buf);

    // public void decode(DataItem data, ByteBuffer buf);

    //    public Object getValue();
    //
    //    public void setValue(Object value);
    //
    //    public String getName();

}
```

File 60: message.operations.Result

```java
/*
 * Created on Oct 28, 2004 3:27:17 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-31 12:28:25 +0100 (Sun, 31 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 150 $
 * Location:    $URL: Result.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.message.operations;


/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class Result implements OperationInterface {

    public byte[] encode(Object result) {
        return new byte[(byte) ( ((Boolean) result).booleanValue() ? 1 : 0)];
    }

    public Object decode(byte[] buffer) {
        return Boolean.valueOf(buffer[0] == 0);
    }
}
```

File 61: message.operations.Routing

```
/*
 * Created on Oct 28, 2004 3:22:37 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-31 12:28:25 +0100 (Sun, 31 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 150 $
 * Location:    $URL: Routing.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.message.operations;

/**
 * Values for routing are: <br>
 * local: menans this dataItem has a local destination <br>
 * routing: means NodeDirectory needs to resolve destination <br>
 * direct: means IP is already know, where to send this dataItem <br>
 *
 * @author michi
 */
public class Routing implements OperationInterface {

    public byte[] encode(Object routing) {
        return ((String) routing).getBytes();
    }

    public Object decode(byte[] buffer) {
        return new String(buffer);
    }

}
```

File 62: message.operations.ServiceOriginator

```
/*
 * Created on Oct 28, 2004 3:23:59 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-31 12:28:25 +0100 (Sun, 31 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 150 $
 * Location:    $URL: ServiceOriginator.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.message.operations;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class ServiceOriginator implements OperationInterface {

    public byte[] encode(Object serviceOriginator) {
        return ((String) serviceOriginator).getBytes();
    }

    public Object decode(byte[] buffer) {
        return new String(buffer);
    }

}
```

File 63: message.operations.TransactionId

```java
/*
 * Created on Oct 28, 2004 3:19:39 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-31 12:28:25 +0100 (Sun, 31 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 150 $
 * Location:    $URL: TransactionId.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.message.operations;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class TransactionId implements OperationInterface {

    public byte[] encode(Object transactionId) {
        return ((String) transactionId).getBytes();
    }

    public Object decode(byte[] buffer) {
        return new String(buffer);
    }
}
```

File 64: message.operations.TransactionNodeId

```java
/*
 * Created on Oct 28, 2004 3:20:25 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-31 12:28:25 +0100 (Sun, 31 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 150 $
 * Location:    $URL: TransactionNodeId.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.message.operations;

import java.net.InetAddress;

import netbyte.NetByte;
import ch.nix.ert.message.Node;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class TransactionNodeId implements OperationInterface {

    public byte[] encode(Object value) {
        Node transactionNodeId = (Node) value;
        // will store it as:
        //   ip
        //   port
        byte[] port = NetByte.encodeInt(transactionNodeId.getPort());
        byte[] address = transactionNodeId.getInetSocketAddress().getAddress()
                .getAddress();
        byte[] returnValue = new byte[address.length + 4];
        System.arraycopy(address, 0, returnValue, 0, address.length);
        System.arraycopy(port, 0, returnValue, 4, 4);
        return returnValue;
    }

    public Object decode(byte[] buffer) {
        try {
            byte[] byteIp = new byte[4];
            System.arraycopy(buffer, 0, byteIp, 0, 4);
            InetAddress ip = InetAddress.getByAddress(byteIp);
            int port = NetByte.decodeInt(buffer, 4);
            //String ip = new String(buffer, 4, buffer.length - 4);
            return new Node(ip, port);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            System.out.println("ERROR in here");
            e.printStackTrace();
            return null;
        }
    }

}
```

File 65: message.operations.TransactionOperation

```
/*
 * Created on Oct 28, 2004 3:20:04 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-31 12:28:25 +0100 (Sun, 31 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 150 $
 * Location:    $URL: TransactionOperation.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.message.operations;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class TransactionOperation implements OperationInterface {

    public byte[] encode(Object transactionOp) {
        return ((String) transactionOp).getBytes();
    }

    public Object decode(byte[] buffer) {
        return new String(buffer);
    }

}
```

File 66: message.operations.TransactionServiceId

```
/*
 * Created on Oct 28, 2004 3:20:49 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-31 12:28:25 +0100 (Sun, 31 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 150 $
 * Location:    $URL: TransactionServiceId.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.message.operations;

/**
 * TODO Edit this text for a better description
 *
```

```java
 * @author michi
 */
public class TransactionServiceId implements OperationInterface {

    public byte[] encode(Object transactionServiceId) {
        return ((String) transactionServiceId).getBytes();
    }

    public Object decode(byte[] buffer) {
        return new String(buffer);
    }
}
```

File 67: message.operations.TransactionSplit

```java
/*
 * Created on Oct 28, 2004 3:21:51 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-31 12:28:25 +0100 (Sun, 31 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 150 $
 * Location:    $URL: TransactionSplit.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.message.operations;

import netbyte.NetByte;
import netbyte.NetByteDecodingException;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class TransactionSplit implements OperationInterface {

    public byte[] encode(Object transactionSplit) {
        return NetByte.encodeInt(((Integer) transactionSplit).intValue());
    }

    public Object decode(byte[] buffer) {
        try {
            return new Integer(NetByte.decodeInt(buffer));
        } catch (NetByteDecodingException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            return null;
        }
    }

}
```

File 68: message.operations.TransactionType

```java
/*
 * Created on Oct 28, 2004 3:21:15 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-31 12:28:25 +0100 (Sun, 31 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 150 $
 * Location:    $URL: TransactionType.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.message.operations;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class TransactionType implements OperationInterface {

    public byte[] encode(Object transactionType) {
        return ((String) transactionType).getBytes();
    }

    public Object decode(byte[] buffer) {
        return new String(buffer);
    }

}
```

File 69: message.operations.UserKey

```java
/*
 * Created on Oct 28, 2004 3:17:54 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-10-31 12:28:25 +0100 (Sun, 31 Oct 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 150 $
 * Location:    $URL: UserKey.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.message.operations;

import netbyte.NetByte;
import netbyte.NetByteDecodingException;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class UserKey implements OperationInterface {

    public byte[] encode(Object uKey) {
        return NetByte.encodeLong(((ch.nix.ert.message.UserKey) uKey).getId());
    }

    public Object decode(byte[] buffer) {
        try {
            return new ch.nix.ert.message.UserKey(NetByte.decodeLong(buffer));
        } catch (NetByteDecodingException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            return null;
        }
    }

}
```

File 70: user.UserService

```java
/*
 * Created on Oct 5, 2004 11:18:43 AM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-11-07 22:22:16 +0100 (Sun, 07 Nov 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 164 $
 * Location:    $URL: UserService.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.user;

```

```java
import java.util.Date;
import java.util.StringTokenizer;

import org.apache.log4j.Logger;

import ch.nix.ert.base.ServiceBase;
import ch.nix.ert.message.DataItem;
import ch.nix.ert.message.InternalKey;
import ch.nix.ert.message.Node;
import ch.nix.ert.message.UserKey;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class UserService extends ServiceBase {

    //  logger:
    static Logger logger = Logger.getLogger(UserService.class);

    static private UserService instance = null;

    static int reqSec = 0;

    static int throughput = 0;

    static int collectorAmount = 0;

    static int collected = 0;

    static String[][] statistics;

    private UserService() {
    }

    static public synchronized UserService getInstance() {
        if (null == instance) {
            instance = new UserService();
            logger.info("UserService started");
        }
        return instance;
    }

    /*
     * (non-Javadoc)
     *
     * @see ch.nix.ert.ServiceInterface#registerAtDispatcher()
     */
    public void registerAtDispatcher() {
        // nothing to register, as this UserService should not listen to any
        // keywords, he should only retrieve transaction results
        registerService("fillup");
        registerService("bulk");
        registerService("writebench");
        registerService("readbench");
        registerService("deletebench");
        registerService("collect");
        registerService("collector");
    }

    /*
     * (non-Javadoc)
     *
     * @see java.lang.Runnable#run()
     */
    public void run() {
        Thread.currentThread().setName("ERT.UserService");
        long startMillis = 0;
        long startMillis2 = 0;
        int amount = 2000;
        int counter = amount;
        boolean bulkMode = false;
        while (true) {
            DataItem dataItem = null;
            dataItem = getWork();
            // logger.debug("Got Work: "+dataItem);
            String operation = dataItem.getOperationType();

            if (operation.equals("collector")) {
                // I’m a collector, so I will retrieve the amount of machines, i
                // have to collect...
                collectorAmount = dataItem.getCursorSize();
                statistics = new String[collectorAmount][10];
            } else if (operation.equals("collect")) {
                String statisticData = dataItem.getData();
                StringTokenizer st = new StringTokenizer(statisticData);
                int i = 0;
                while (st.hasMoreTokens()) {
                    statistics[collected][i] = st.nextToken();
                    i++;
                }
                collected++;
                if (collected >= collectorAmount) {
                    // now generate statistics
                    for (int j = 0; j < 10; j++) {
                        String timeslice = "";
                        double timesliceSum = 0;
                        for (int k = 0; k < collectorAmount; k++) {
                            String number = statistics[k][j];
                            int len = number.length();
                            if (len < 4) {
                                for (int index = 0; index < (4 - len); index++) {
                                    number = " " + number;
                                }
                            }
                            timeslice += number + " ";
                            timesliceSum += Double.parseDouble(number);
                        }
                        // timesliceSum = timesliceSum / (double) collectorAmount;
                        String sum = "" + Math.round(timesliceSum);
                        int len = sum.length();
                        if (len < 9) {
                            for (int index = 0; index < (9 - len); index++) {
                                sum = " " + sum;
                            }
                        }
                        logger.error("Timeslice: " + sum + " - " + timeslice);
                    }
                    collected = 0;
                }
            } else if (operation.equals("bulk")) {
                bulkMode = true;
                try {
                    Thread.sleep(10000);
                } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
                logger.info("Starting benchmark");
                Date start = new Date();
                startMillis = start.getTime();
                for (int i = 0; i < amount; i = i + 1) {
                    long random = (long) (Math.random() * (double) 100000000);
                    put(random, "Fillup " + random);
                    //System.out.println(random);
                }
                Date stop = new Date();
                long stopMillis = stop.getTime();
                logger
                        .info("Sending "
                                + amount
                                + " entries took: "
                                + (stopMillis - startMillis)
                                + "ms ("
                                + ((double) amount
                                        / (double) (stopMillis - startMillis) * (double) 1000)
                                + " requests/sec)");

            } else if (operation.equals("put-end") && bulkMode) {
                // logger.info("Got: "+dataItem);
                if (counter == amount) {
                    Date start2 = new Date();
                    startMillis2 = start2.getTime();
                }
                counter-;
                //                  if (counter%1 == 0) {
                //                      logger.info("Still missing responses: "+counter);
                //                  }
                if (counter < 1) {
                    Date stop = new Date();
                    long stopMillis = stop.getTime();
                    logger
                            .info("Receiving "
                                    + amount
                                    + " entries took: "
                                    + (stopMillis - startMillis2)
                                    + "ms ("
                                    + ((double) amount
                                            / (double) (stopMillis - startMillis2) * (double) 1000)
                                    + " requests/sec)");
                    logger
                            .info("Overall Storing "
                                    + amount
                                    + " entries took: "
                                    + (stopMillis - startMillis)
                                    + "ms ("
                                    + ((double) amount
                                            / (double) (stopMillis - startMillis) * (double) 1000)
                                    + " requests/sec)");
                    // restart
                    counter = amount;
                    addWork(new DataItem("bulk"));
                }
            }

            else if (operation.equals("writebench")) {
                final Node collectorNode = new Node(dataItem.getData(), 0);
                final int dataSize = dataItem.getDbSize();
                try {
                    Thread.sleep(10000);
                } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
                Thread t1 = new Thread("ThreadT1") {
                    long startMillis;
                    public void run() {

                        int round = 1;
                        StringBuffer buffer = new StringBuffer(60);
                        String data = "";
                        for (int i = 0; i < dataSize; i++) {
                            data += "d";
                        }
                        while (true) {
                            Date start = new Date();
                            startMillis = start.getTime();
                            for (int i = 0; i < throughput + 10; i = i + 1) {
                                long random = (long) (Math.random() * (double) 100000000);
                                put(random, data);
                            }
                            Date stop = new Date();
                            long stopMillis = stop.getTime();
                            long timeToWait = 1000 - (stopMillis - startMillis) % 1000;
                            try {
                                Thread.sleep(timeToWait);
                            } catch (InterruptedException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                            }
                            logger.info("Received requests in last second: "
                                    + reqSec);
                            buffer.append(" "+reqSec);
                            if (round >= 10) {
                                // send statistics to collector:
                                DataItem collectorData = new DataItem("collect");
                                collectorData.setDirectDestination(collectorNode);
                                collectorData.setRouting("direct");
                                collectorData.setData(buffer.toString());
                                addWork(collectorData);
                                round = 0;
                                buffer = new StringBuffer(60);
                            }
                            round++;
                            throughput = reqSec;
                            reqSec = 0;
                        }
                    }
                };
                logger.info("Starting benchmark");
                t1.start();
            } else if (operation.equals("put-end")) {
                reqSec++;
            } else if (operation.equals("fillup")) {
                try {
                    Thread.sleep(10000);
                } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
                Thread t1 = new Thread("ThreadT1") {
                    long startMillis;

                    public void run() {
```

```java
                        logger.info("Starting filling DB");
                        for (int i = 0; i < 1000000; i = i + 1) {
                            put(i, "Fillup " + i);
                            if (i % 1000 == 0) {
                                logger.info("Filling DB: " + i
                                        + " entries in db");
                            }
                            try {
                                Thread.sleep(10);
                            } catch (InterruptedException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                            }
                        }
                        logger.info("Stopped filling DB: 1’000’000 entries");
                    }
                };
                t1.start();
            } else if (operation.equals("deletebench")) {
                final Node collectorNode = new Node(dataItem.getData(), 0);
                try {
                    Thread.sleep(10000);
                } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
                Thread t1 = new Thread("ThreadT1") {
                    long startMillis;

                    public void run() {
                        int round = 1;
                        StringBuffer buffer = new StringBuffer(60);
                        while (true) {
                            Date start = new Date();
                            startMillis = start.getTime();
                            for (int i = 0; i < throughput + 10; i = i + 1) {
                                long random = (long) (Math.random() * (double) 100000000);
                                delete(random);
                            }
                            Date stop = new Date();
                            long stopMillis = stop.getTime();
                            long timeToWait = 1000 - (stopMillis - startMillis) % 1000;
                            try {
                                Thread.sleep(timeToWait);
                            } catch (InterruptedException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                            }
                            logger.info("Received requests in last second: "
                                    + reqSec);
                            buffer.append(" "+reqSec);
                            if (round >= 10) {
                                // send statistics to collector:
                                DataItem collectorData = new DataItem("collect");
                                collectorData.setDirectDestination(collectorNode);
                                collectorData.setRouting("direct");
                                collectorData.setData(buffer.toString());
                                addWork(collectorData);
                                round = 0;
                                buffer = new StringBuffer(60);
                            }
                            round++;
                            throughput = reqSec;
                            reqSec = 0;
                        }
                    }
                };
                logger.info("Starting benchmark");
                t1.start();
            } else if (operation.equals("delete-end")) {
                reqSec++;
            } else if (operation.equals("readbench")) {
                final Node collectorNode = new Node(dataItem.getData(), 0);
                try {
                    Thread.sleep(10000);
                } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
                Thread t1 = new Thread("ThreadT1") {
                    long startMillis;

                    public void run() {
                        int round = 1;
                        StringBuffer buffer = new StringBuffer(60);
```

```
                            while (true) {
                                Date start = new Date();
                                startMillis = start.getTime();
                                for (int i = 0; i < throughput + 10; i = i + 1) {
                                    long random = (long) (Math.random() * (double) 100000000);
                                    get(random);
                                }
                                Date stop = new Date();
                                long stopMillis = stop.getTime();
                                long timeToWait = 1000 - (stopMillis - startMillis) % 1000;
                                try {
                                    Thread.sleep(timeToWait);
                                } catch (InterruptedException e) {
                                    // TODO Auto-generated catch block
                                    e.printStackTrace();
                                }
                                logger.info("Received requests in last second: "
                                        + reqSec);
                                buffer.append(" "+reqSec);
                                if (round >= 10) {
                                    // send statistics to collector:
                                    DataItem collectorData = new DataItem("collect");
                                    collectorData.setDirectDestination(collectorNode);
                                    collectorData.setRouting("direct");
                                    collectorData.setData(buffer.toString());
                                    addWork(collectorData);
                                    round = 0;
                                    buffer = new StringBuffer(60);
                                }
                                round++;
                                throughput = reqSec;
                                reqSec = 0;
                            }
                        }
                    };
                    logger.info("Starting benchmark");
                    t1.start();
                } else if (operation.equals("get-end")) {
                    reqSec++;
                }

            }

        }

        public void sendRaw(DataItem dataItem) {
            transactionBegin(dataItem);
        }

        public synchronized void put(long key, String data) {
            UserKey userKey = new UserKey(key);
            InternalKey internalKey = userKey.getInternalKey();

            DataItem putData = new DataItem();
            putData.setUserKey(userKey);
            putData.setInternalKey(internalKey);
            putData.setData(data);
            putData.setOperation("put");
            putData.setRouting("routing");
            transactionBegin(putData);
        }

        public synchronized boolean putSync(long key, String data) {
            put(key, data);

            // now wait for a response...
            DataItem result = getWork();
            return result.getResult();
        }

        public synchronized void get(long key) {
            UserKey userKey = new UserKey(key);
            InternalKey internalKey = userKey.getInternalKey();

            DataItem getData = new DataItem();
            getData.setUserKey(userKey);
            getData.setInternalKey(internalKey);
            getData.setOperation("get");
            getData.setRouting("routing");
            transactionBegin(getData);
        }

        public synchronized Object getSync(long key) {
            get(key);

```

```java
        // now wait for a response...
        DataItem result = getWork();
        return result.getData();
    }

    public synchronized void delete(long key) {
        UserKey userKey = new UserKey(key);
        InternalKey internalKey = userKey.getInternalKey();

        DataItem deleteData = new DataItem();
        deleteData.setUserKey(userKey);
        deleteData.setInternalKey(internalKey);

        deleteData.setOperation("delete");
        deleteData.setRouting("routing");
        transactionBegin(deleteData);
    }

    public synchronized boolean deleteSync(long key) {
        delete(key);

        // now wait for a response...
        DataItem result = getWork();
        return result.getResult();
    }

}
```

File 71: util.LinkedHashMap

```java
/*
 * Created on Sep 24, 2004 5:33:57 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-09-24 17:58:24 +0200 (Fri, 24 Sep 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 97 $
 * Location:    $URL: LinkedHashMap.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.util;

import java.util.Map;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class LinkedHashMap extends java.util.LinkedHashMap {

    public void setMaxSize(int size) {
        MAX_ENTRIES = size;
    }

    private int MAX_ENTRIES = 10000;

    protected boolean removeEldestEntry(Map.Entry eldest) {
       return size() > MAX_ENTRIES;
    }

}
```

File 72: util.Queue

```java
/*
 * Created on Sep 8, 2004 5:08:21 PM
 *
 * Project:     ExtendendRootTree
 * Last Change: $Date: 2004-09-08 17:26:50 +0200 (Wed, 08 Sep 2004) $
 * Changed by:  $Author: michi $
 * Revision:    $Rev: 79 $
 * Location:    $URL: Queue.java $
 *
 * This source-code is part of the diploma thesis of Michael Kussmaul
 * written at the University of Zurich, Switzerland
 * Department of Information Technology (www.ifi.unizh.ch)
 * For copyright information please contact kussmaul@nix.ch
 *
 */
package ch.nix.ert.util;

import EDU.oswego.cs.dl.util.concurrent.LinkedQueue;

/**
 * TODO Edit this text for a better description
 *
 * @author michi
 */
public class Queue extends LinkedQueue {

    private String myName = "noname";

    private Object lockObject = new Object();

    /**
     *
     */
    public Queue() {
        super();
    }

    public Queue(String name) {
        super();
        myName = name;
    }

    public void setName(String name) {
        synchronized (lockObject) {
            myName = name;
        }
    }

    public synchronized String getName() {
        synchronized (lockObject) {
            return myName;
        }
    }

}
```