Sieben Geisslein - Architecture

Niklas Mehner

October 3, 2007

Contents

1 Introduction

SiebenGeisslein is a transactional object store. Its features include near transparent persistence, ACID, multi-user support, crash-recovery and scalability.

1.1 Design Assumptions

While SiebenGeisslein shows many features of a DBMS, it is not aimed at replacing existing RDBMS or OODBMS, but rather at replacing initialization files, registry-databases and (if it proves to be scalable enough) the whole file system.

Obviously the way of using SiebenGeisslein is much different from the approaches described above.

While there are many different ways of using SiebenGeisslein, I am focusing on laying a good foundation for M"archenwald, a desktop environment, I am planning to base on SiebenGeisslein.

This modell assumes a central event dispatching loop:

```
while (true);
  Transaction t = createTransaction();
  try {
    Event e = getNextEvent();
    dispatchEvent(e);
    t.commit();
  } catch (Exception e) {
    t.abort();
  }
}
```

As you can imagine this approach requires transactions, that are very fast. On the other hand it is not important to know immediatly wether the transaction has failed (For example in a word processor an event might be a key-event. If the transaction fails, the user has eventually be informed, that this has happened, but not before the next keystroke is to be processed).

This observations leads to the use optimistic transactions (section 5.2). Changes are committed to the clients cache immediatly, but committing the changes on the server can be deferred.

A positive side effect of this is, that multiple transactions can be combined. And because of code locality many changed can be merged. In a situation where load on the server gets high and response time goes up, more work is done on the client and the server has to process less transactions (the same goes for network bandiwdth).

The architecture of SiebenGeisslein consists of four separate layers (Figure 1), which are described in the following sections.

2 Core Layer

The core layer (→ org.siebengeisslein.core.Core) is responsible for implementing a mapping from object IDs (OIDs) to core entries (→ org.siebengeisslein.core.CoreEntry). A core entry consists of a byte array containing the object data and a long array containing OIDs.

The core layer provides atomicity, consistency and durability. Isolation is provided by the server layer.

The core layer is also responsible for the garbage collection.

2.1 Core Tasks

Set Entry

Add Entry

Add Node

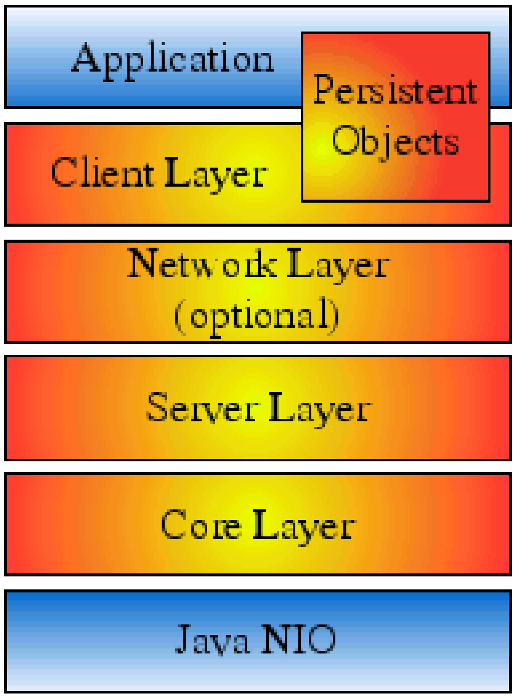

Figure 1: SiebenGeisslein Layers

2.2 BTree

2.3 DataStore

2.4 Garbage Collection

2.5 Recovering

When the Core performs an unplanned exit, it has to recover on the next startup.

Set Entry

Add Entry

Add Node

3 Server Layer

3.1 Transactions

3.2 Access Control

4 Network Layer

5 Client Layer

5.1 Transactions

5.2 Optimistic Transactions

5.3 Persistent Objects

5.3.1 Verification

5.3.2 Instrumentation

6 Applications

6.1 Overview

This section describes the application support in SiebenGeisslein.

6.2 Application

An application (→ org.siebengeisslein.application.Application) consists of code and data. The application is identified through a globally unique applicationId. To allow multiple parallel installations of the same application, there is also a localId, which identifies one local installation of the application.

When an object belonging to an application is persisted, the localId and the classId (an id identifying the class within the application) are used to identitfy the class of the object.

When the object is loaded back from persistent storage, the application is located and an application classloader (→ org.siebengeisslein.application.ApplicationClassLoader) instantiated to load the class of the object.

6.3 Classloading

6.4 Dependencies

Not yet supported.

7 Feder

7.1 Introduction

This section describes some aspects of the implementation of the Feder IDE.

7.2 Classloading