Silenus

Fall 2005

Max Berger <max@berger.name>

November 30th, 2005

Overview

- Introduction
- Architecture
- Use Cases
- Silenus
- Byzantium
- Midas
- Interfaces
- Optimizers
- Conclusion

Introduction

- Problem Statement
- Possible users
- Objective / Approach

Problem Statement

Existing distributed file stores

- are difficult to set up
- require high maintanance
- have problems with server downtime
- don't handle topology changes
- don't provide privacy from administators
- don't scale for large nuber of nodes
- are incompatible with service-oriented architectures

Possible users

- Power User
- High performance computing lab
- Multi-student LAB
- Small Office / Workgroup
- Students rooming
- Family

Objective / Approach

Objective

- to create a new filestore based on the SORCER environment

Approach

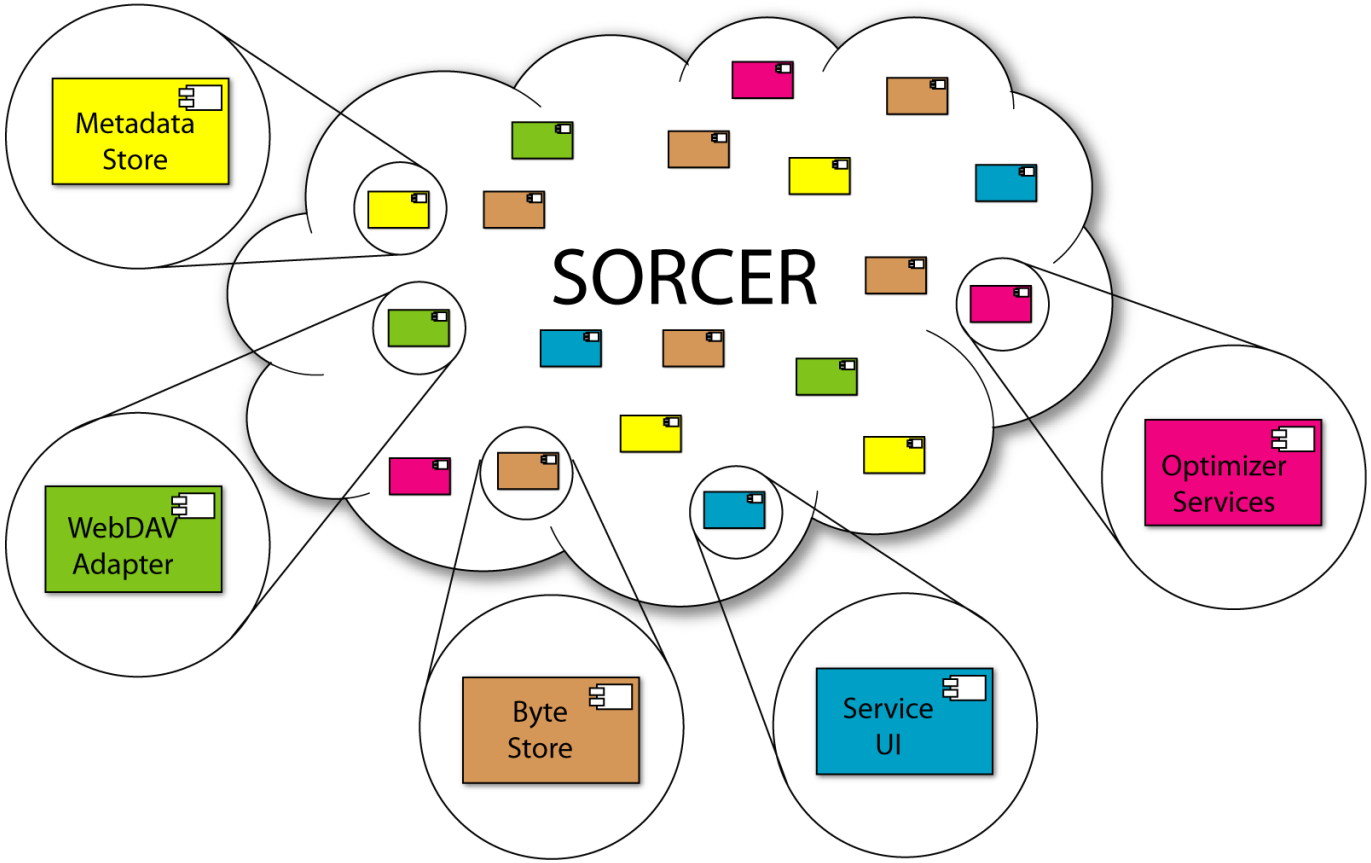
- Authentication for authorization
- Encryption for privacy
- Replication for availability
- Multisource download for performance

Architecture

- Services
- Architectural Overview
- User Interface Adapters
- Facade
- Internal Components

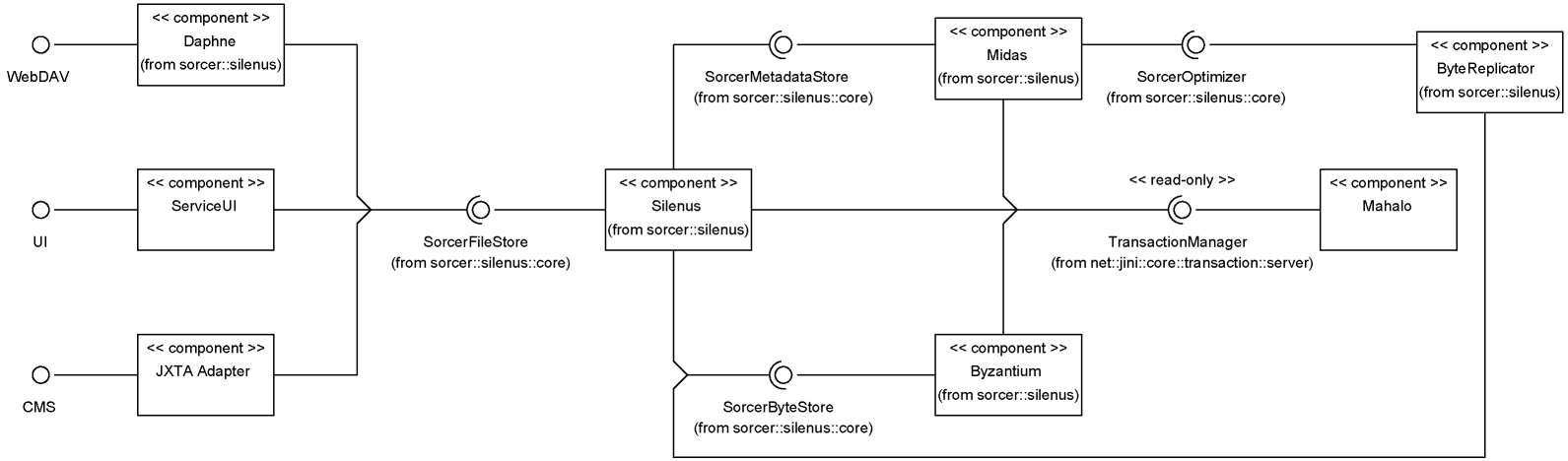
Services

Independent federated services that use SORCER for communication



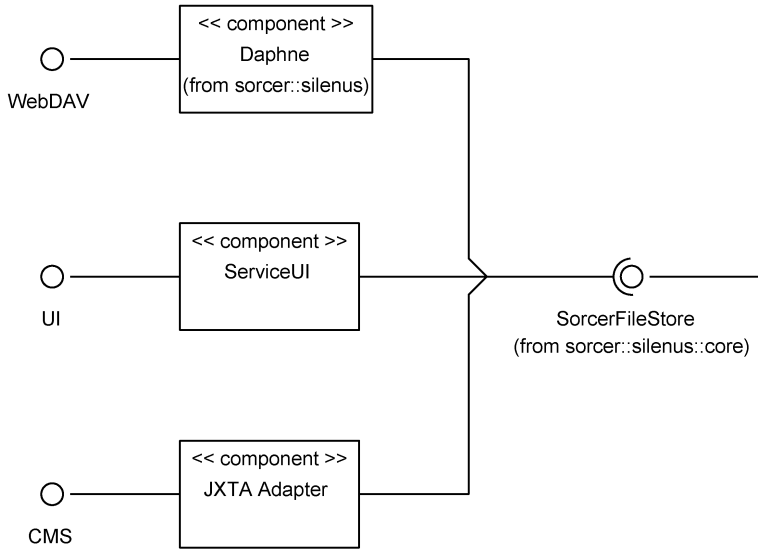
Architectural Overview

Even though all modules communicate via SORCER, not every module communicates with every module



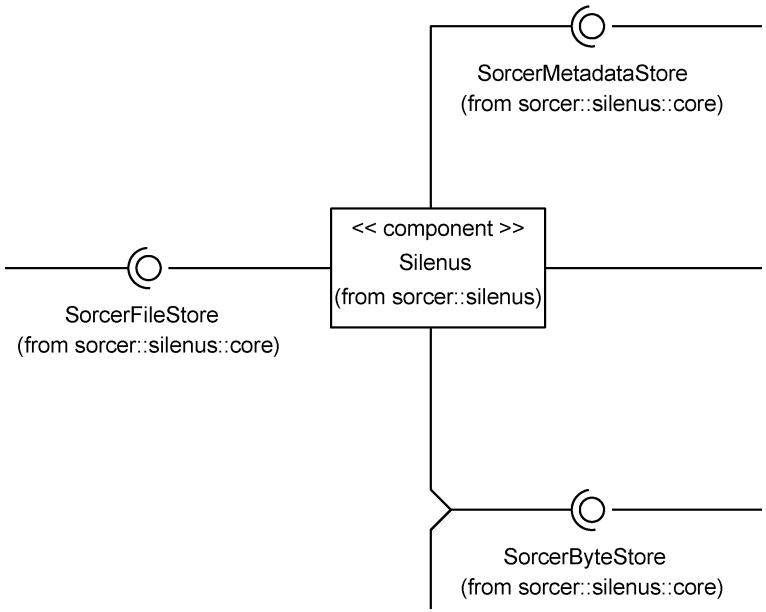
User Interface Adapters

Adapters provide user interfaces for existing file storage options

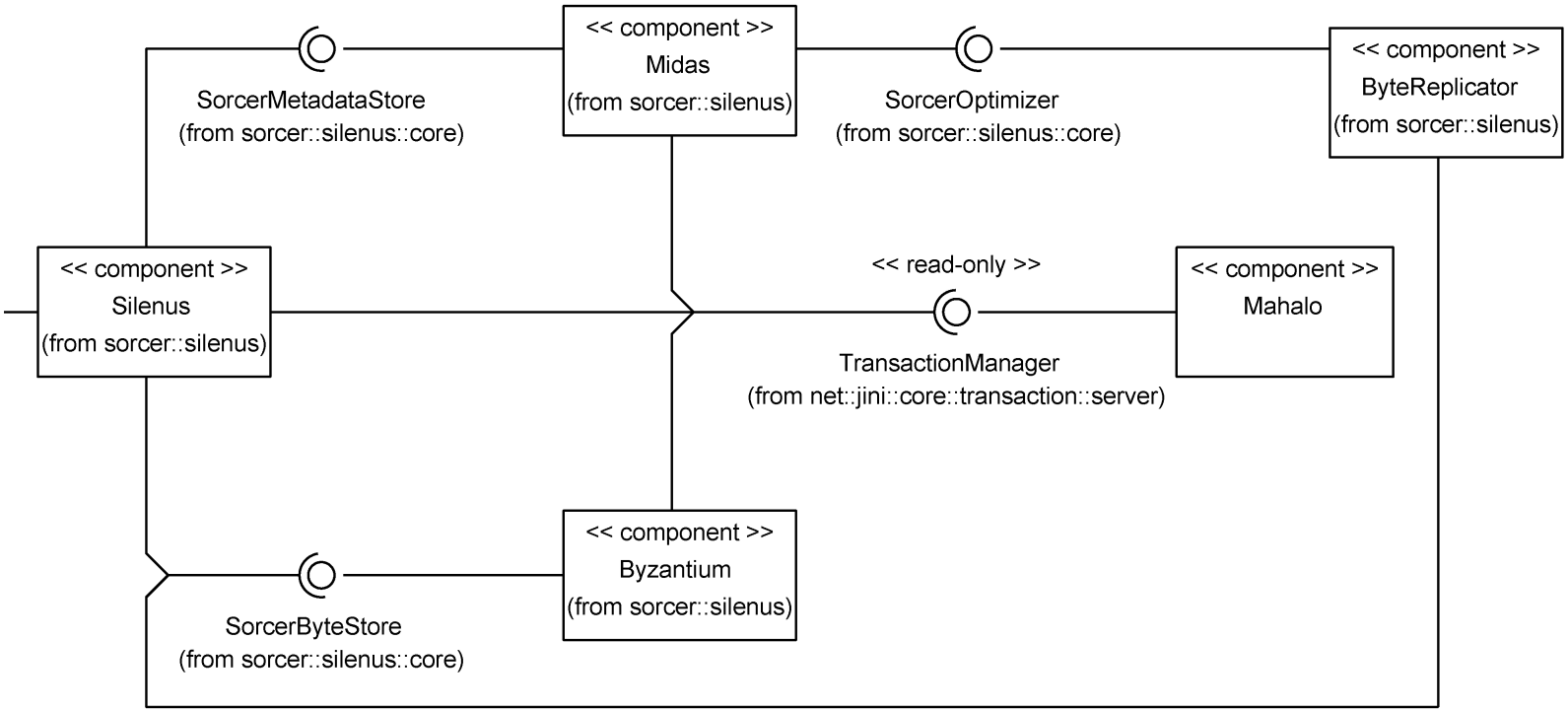


Facade

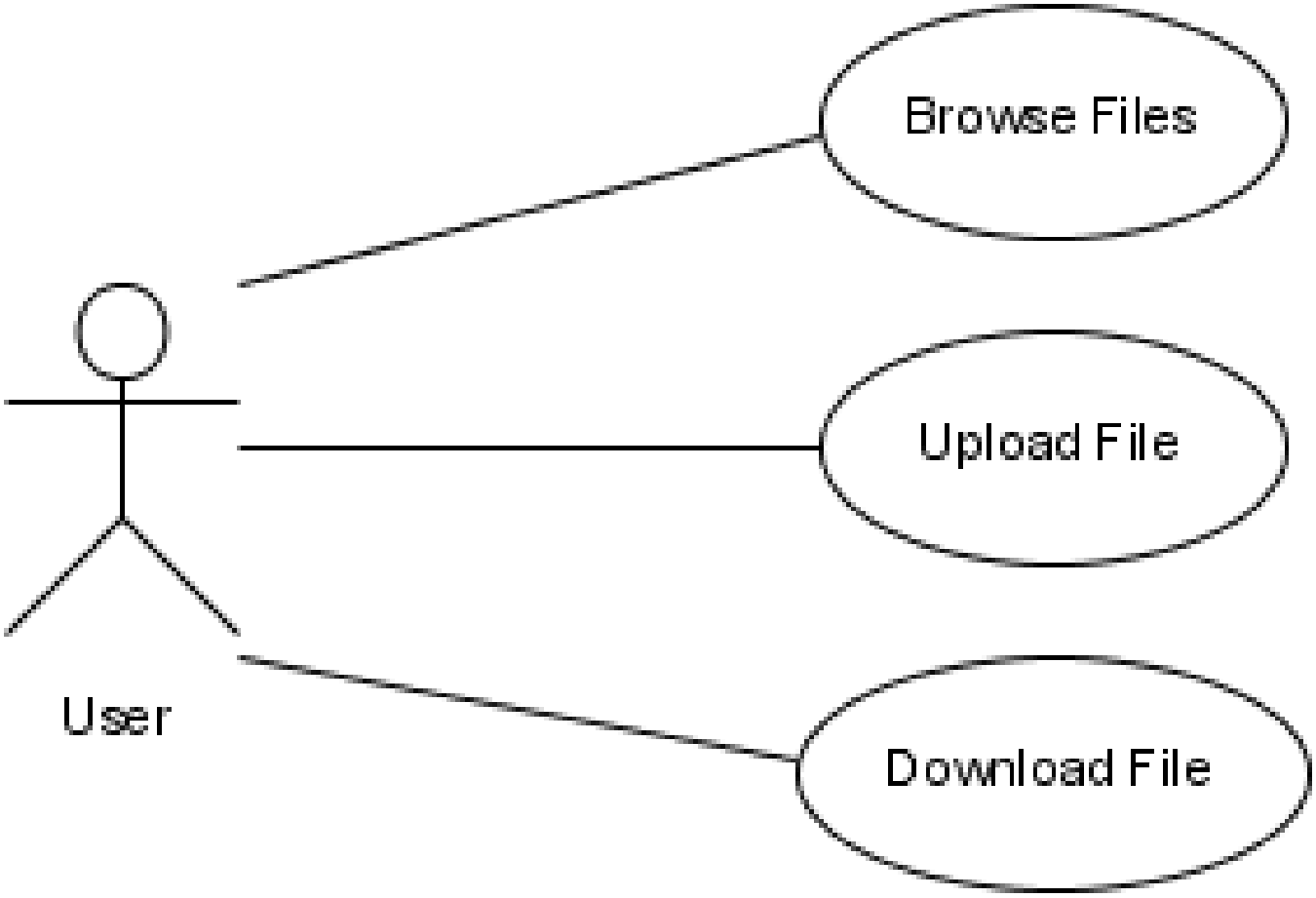
Provides "One face to customer"



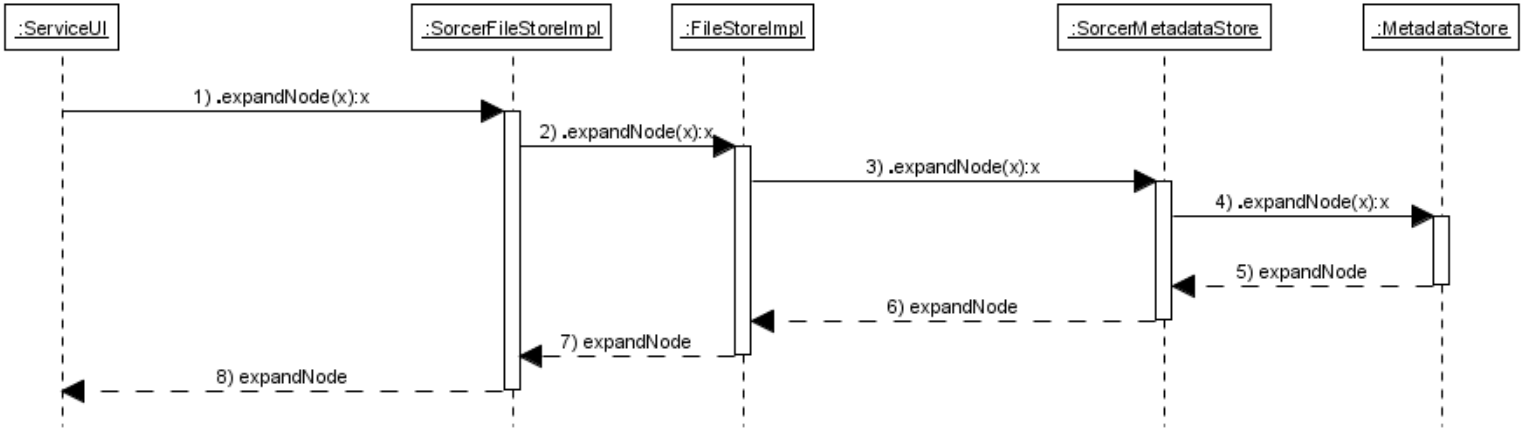
Internal Components



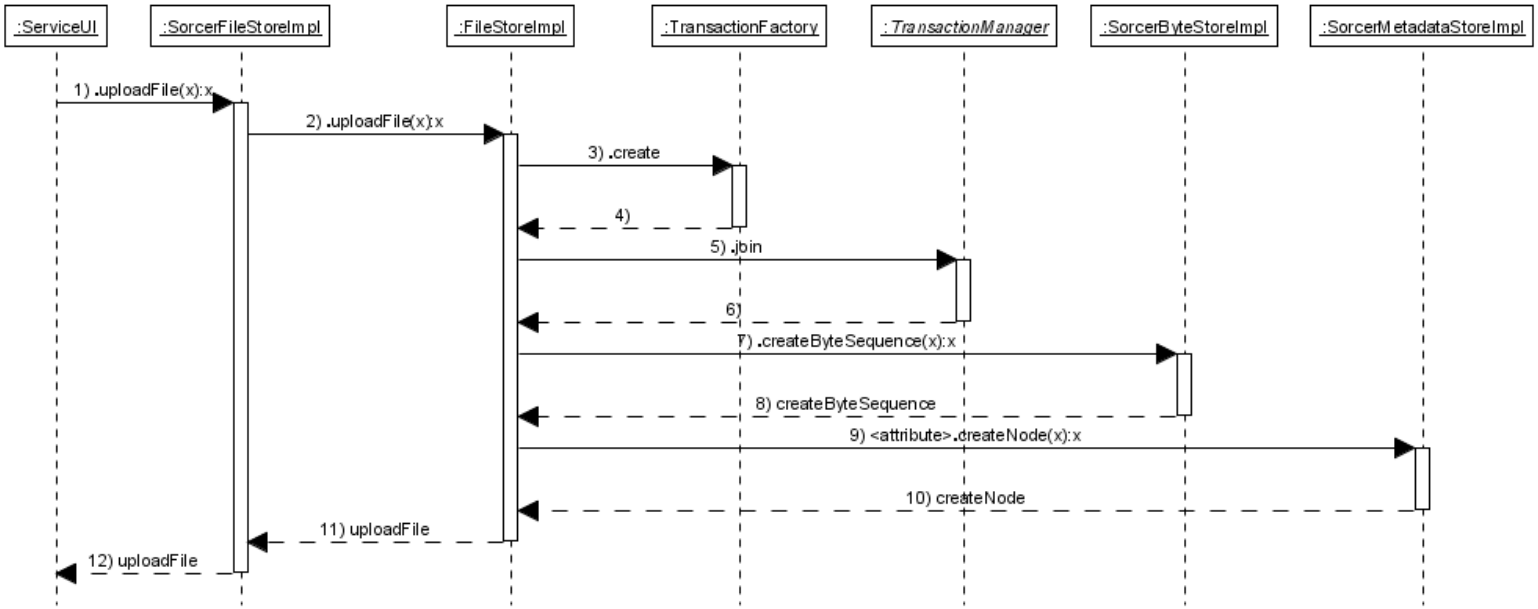
Use Cases



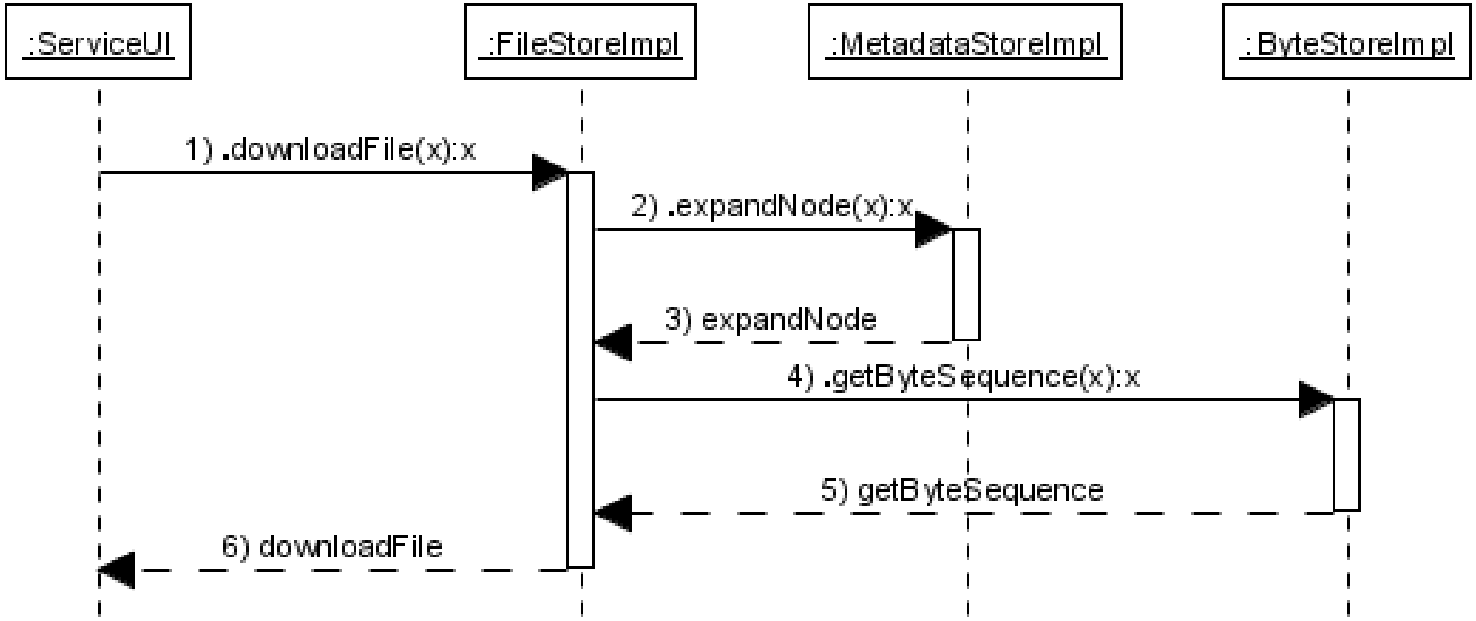
Browse Files



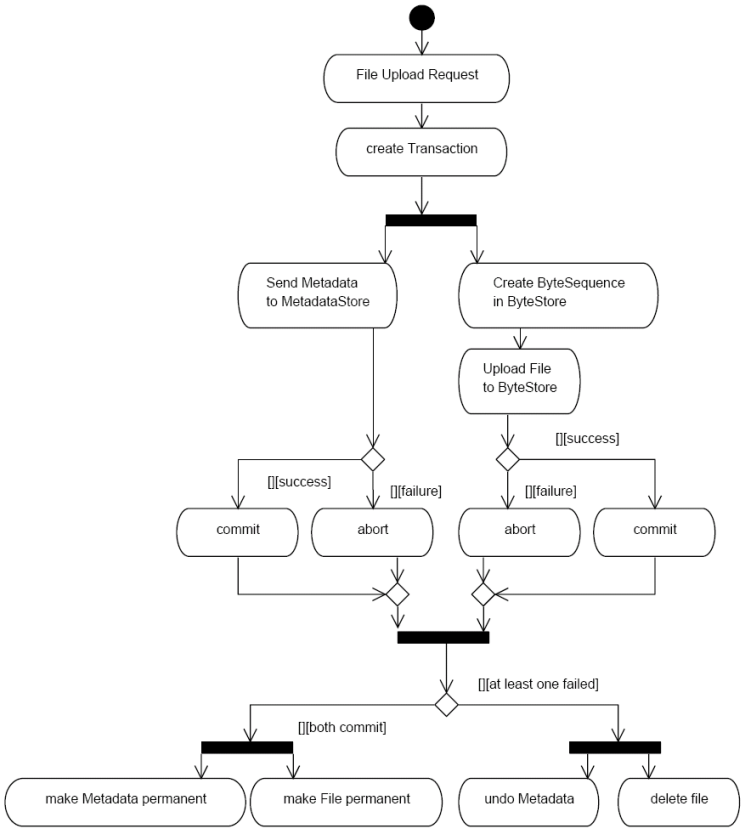
Upload File



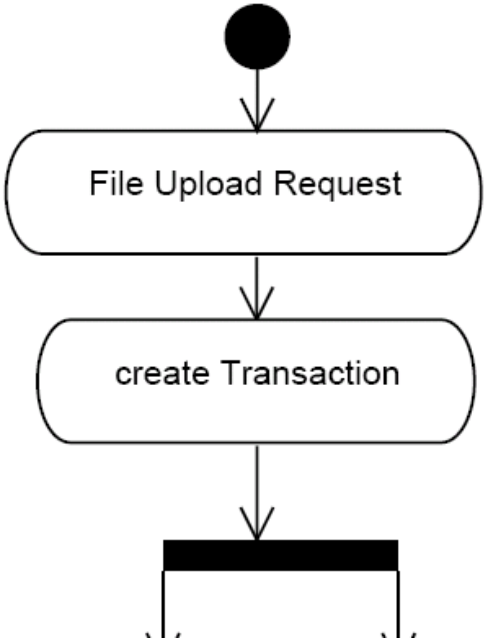
Download File



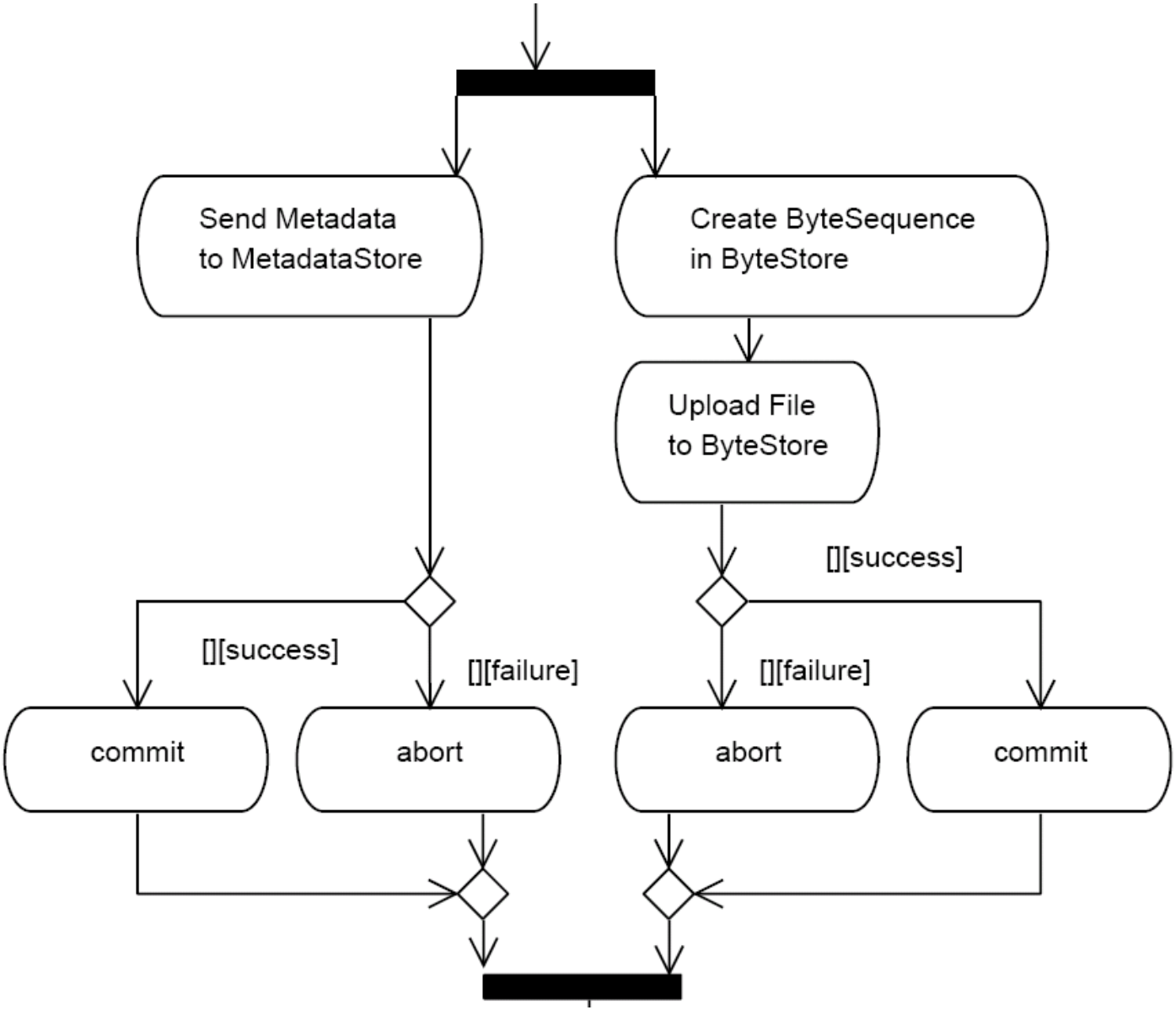
Transactions



Transactions (cont.)

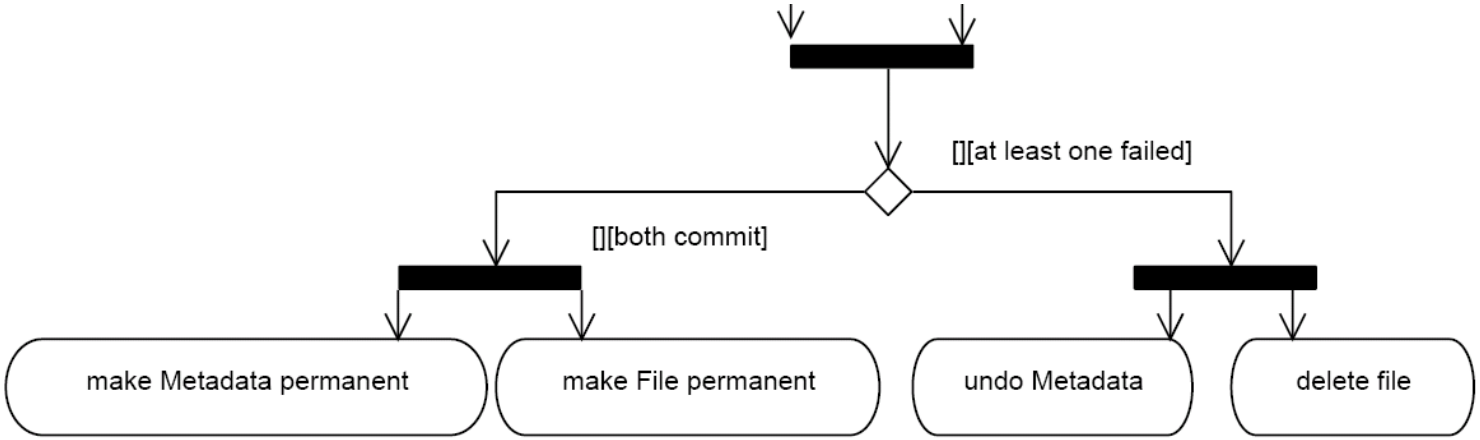


Transactions (cont.)



Use Cases - Transactions

Transactions (cont.)



Silenus

- Introduction

Introduction

- provide one interface for the user
- hides internal transactional semantics

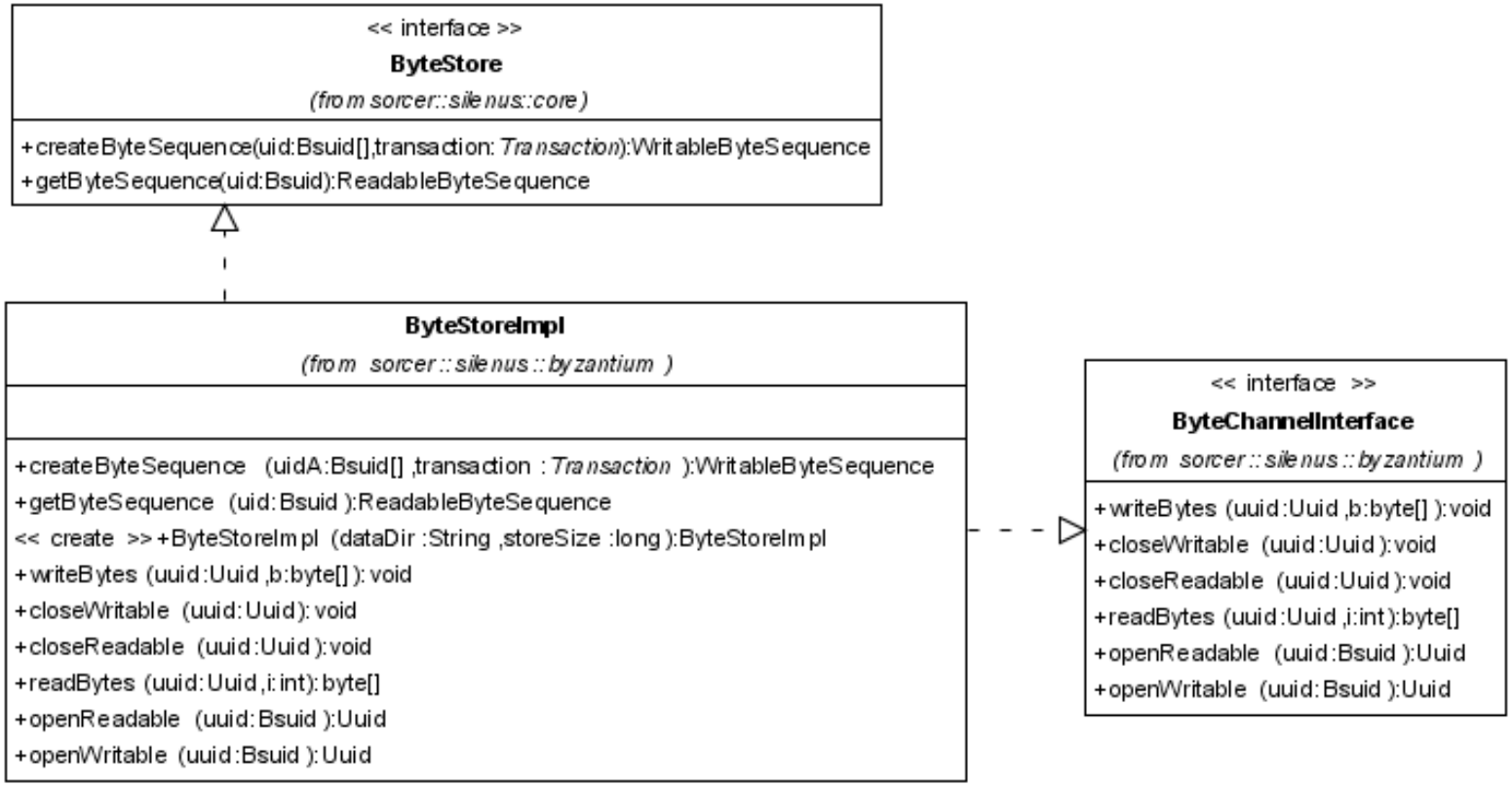
Byzantium

- Introduction
- Java Interface
- SORCER interface
- ByteSequences
- Reading Files
- Writing Files
- Upload File

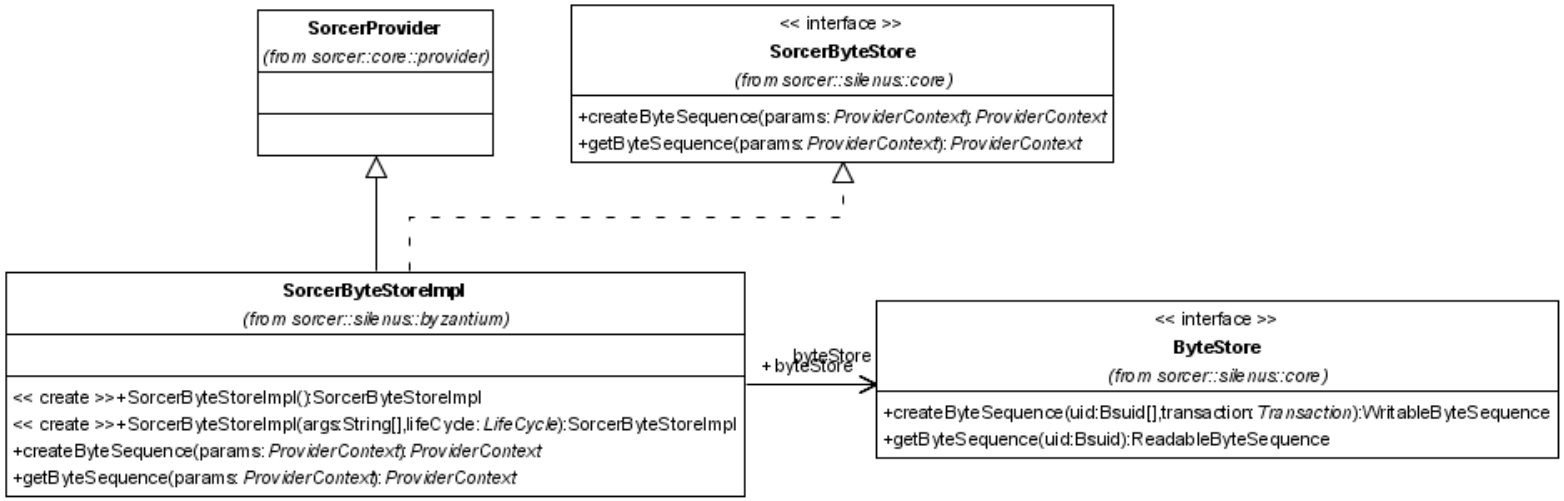
Introduction

- Byzantium is a ByteStore implementation
- Files are identified uniquily by ServiceID and Entry UUID
- Does nothing but file storage

Java Interface



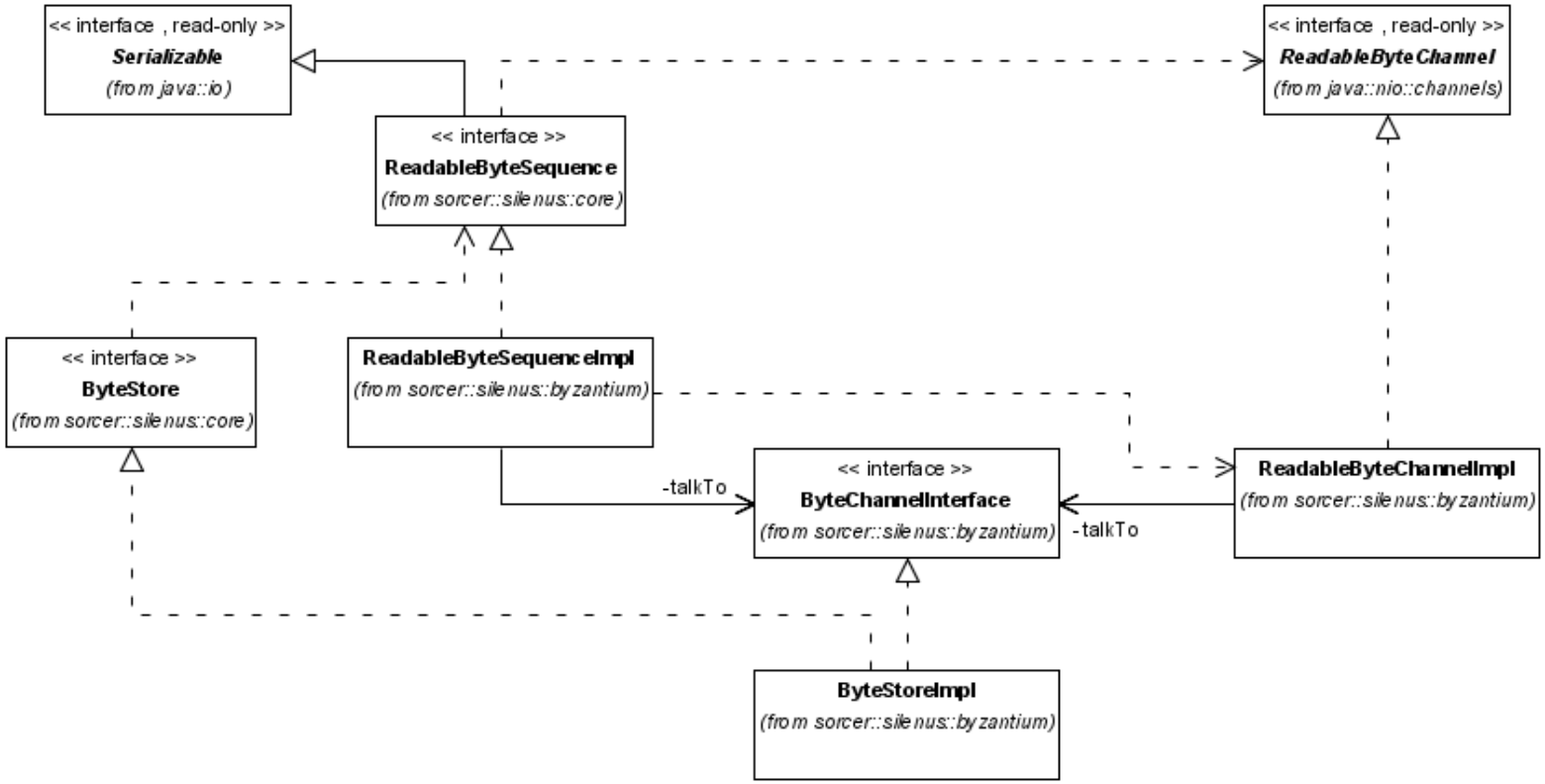
SORCER interface



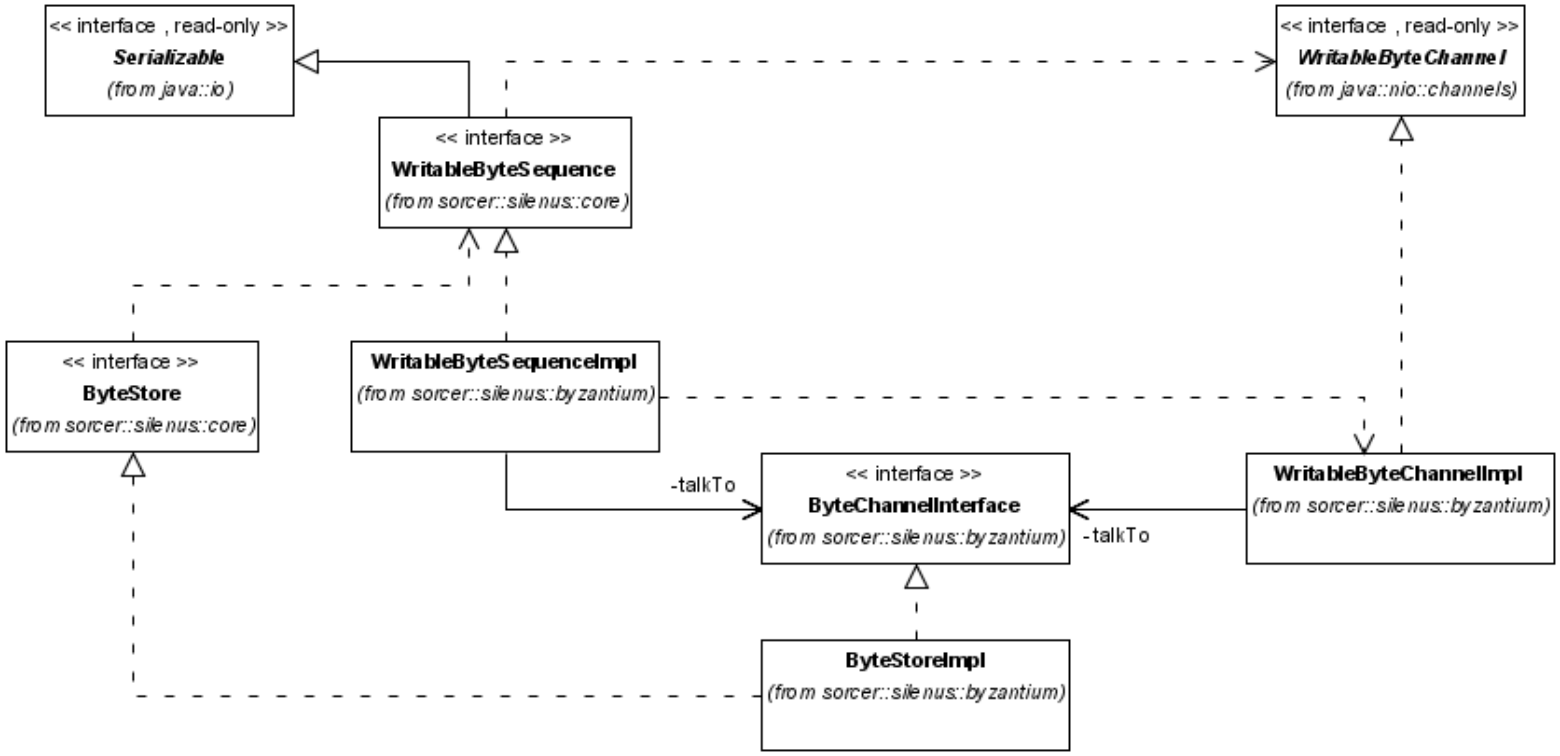
ByteSequences

- ReadableByteSequence and WritableByteSequence
- are Serializable
- are smart proxies
- know how to talk with the ByteStore
- are implemented using the ByteChannelInterface
- provide the possiblity to create ByteChannels

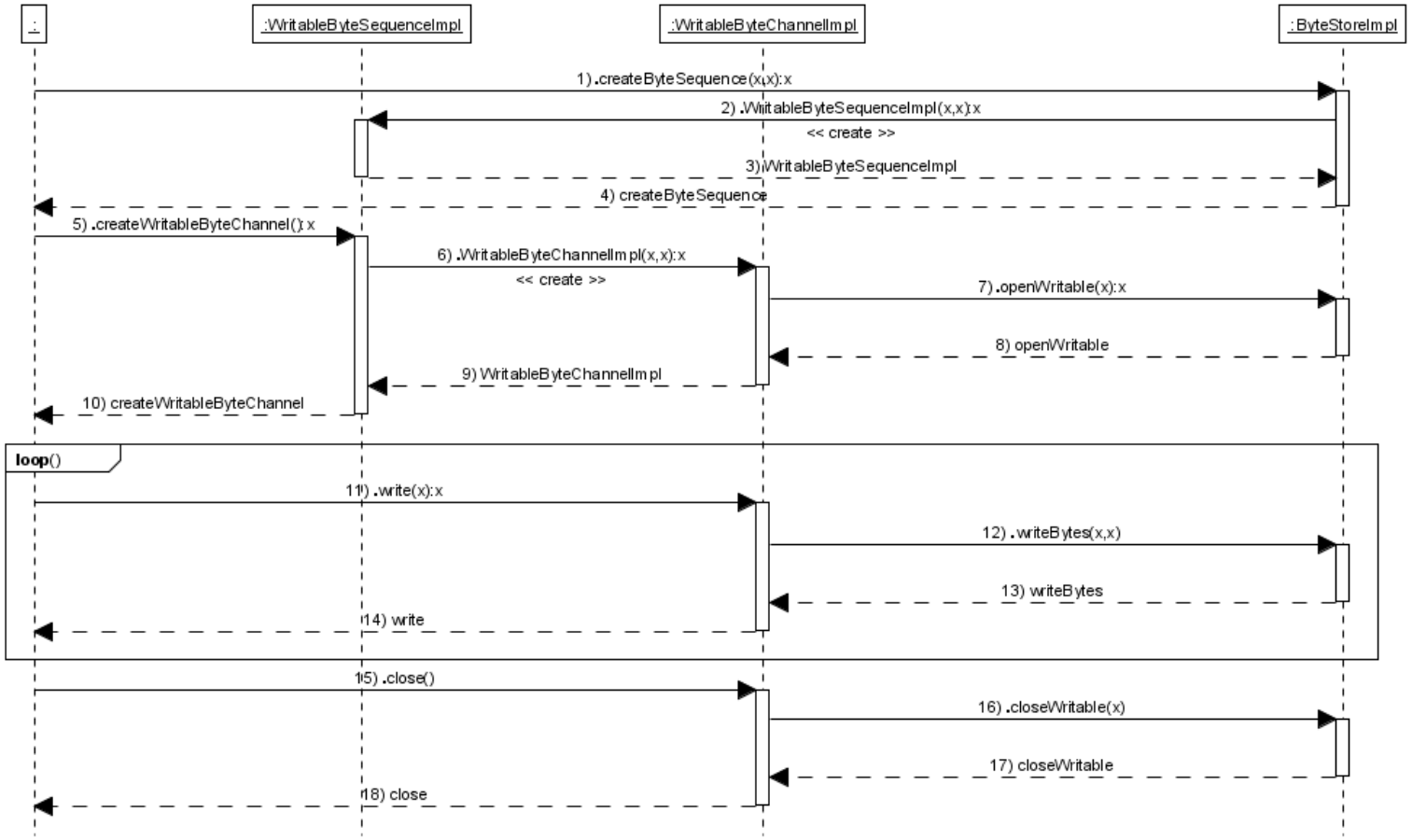
Reading Files



Writing Files



Upload File



Midas

- Introduction
- Solved issues
- Directory Structure
- Vector Clocks
- Internal Storage
- Security
- Versioning
- Conflict resolution
- Special meta information

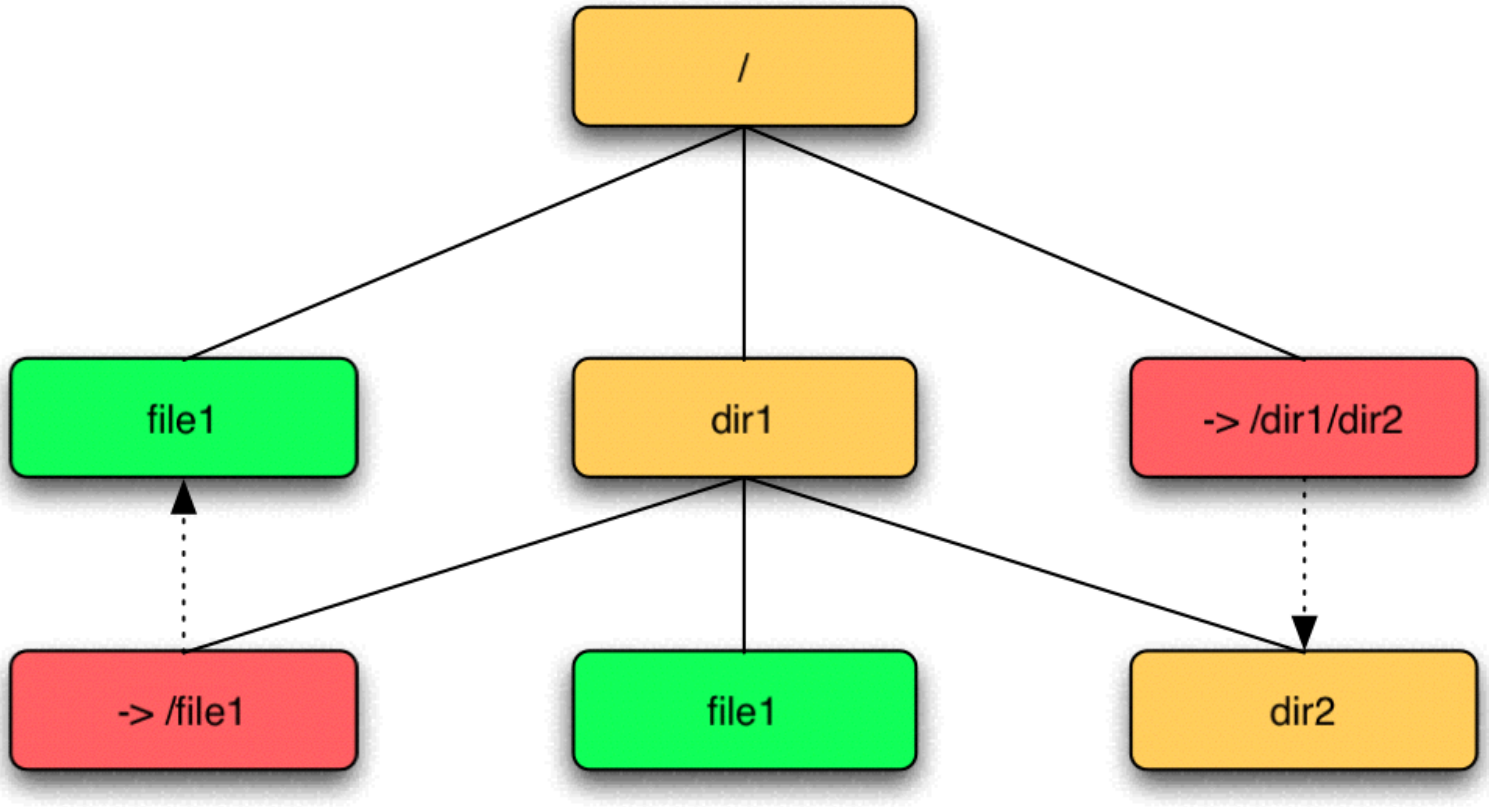
Introduction

- Midas - Meta Information Database And Storage
- Midas is a MetaStore Implementation
- Based on Vivek's ReplicaProvider
- Stores meta information for files saved in bytestore
- Meta Information is stored in embedded database

Solved issues

- Meta Information is consistent across MetaStores
- Support for all kinds of meta information: File name, size, modified date, icon, security information, etc.
- Two parts: Service and Descriptors
- Provide support for file movement / hoarding

Directory Structure



Vector Clocks

- Each event increases the logical clock of that particular system by one
- Each system keeps a vector of all logical clocks it knows
- Update events include the vector clock
- Allows total odering of events, needed for synchronization
- Two clocks per node: One for internal events, another one for external events.

Internal Storage

Stores:

- Metainformation for the Metadata Store (such as clocks, database version)
- Meta Info table

Stored in internally using the Data Acess Objects pattern

- can use any database as backend (currently uses McKoi)

Security

- All files are stored encypted
- Decryption keys are file objects in special directory
- Decryption is done at the last node possible (in the proxy)
- Users secret key can be in FileStore or on local computer

Versioning

- Concurrency Problem is solved by versioning
- Version includes MetaStore Identifier
- Old Version can be stored for backup

Conflict resolution

Virtual duplication:

If an item A is changed on two MetaStores, it will be available 3 times:

- as A.1
- as A.2
- as A linking to either A.1 or A.2 (depending on which system)

Special meta information

- Minimum Number of copies in network
- Number of old versions to keep
- Special meta information template items

Interfaces

- Service UI
- WebDAV

Service UI

- Provides a Jini ServiceUI compatible interface for all functions provided by the MetaStore
- Zero install: started from service browser (e.g. IncaX)

WebDAV

- Daphne is the WebDAV Adapter for the SILENUS filestore
- WebDAV is well documented and supported in Win32, Mac OS X, Linux
- Each OS has different quirks that need to be addressed

Optimizers

- Introduction
- Solved issues
- Open issues

Introduction

- The optimizer services will make this FileStore intelligent by doing automatic data movement, replication, indexing, ...

Solved issues

Optimizer is not a single service, but is a set of federated services:

- MetaInformationCompleter: Will read files and add missing meta information: Mime-Type, artist/title for mp3s, codec and play length for movies, ...
- Replicator: Reads the MetaInformation "Minimum copies" and ensures that there are that many copies in the network
- SpaceSaver: Saves space by reducing number of replicas in the network. Uses "Last Used" meta information.

Optimizers - Solved issues

Open issues

- No cool names yet
- Preventive Mover: Should detect that a certain file/directory is used from 9-5 at work and from 6-9 at home. Should then automagically move file from work to home between 5-6
- Other intelligence that I can't think of

Conclusion

- Benefits
- Schedule

Benefits

- compehensive security
- completely self-managed
- fully transparent for the user
- adding disk space is easy
- compatible with legacy applications via WebDAV interface

Schedule

Background research	01/04 - 12/04
Initial proposal	06/04
System design	06/04 - 12/04
Byzantium	10/04 - 02/04
Midas	01/05 - 06/05
User Interface	03/05 - 11/06
Service Oriented Redesign	07/05 - 12/05
Implementation	01/06 - 11/06
Defense	11/06

Conclusion - Schedule