SILENUS

SORCER Integrated Local Enhanced New User Storage Proposal Presentation

Max Berger

Spring 2006

Agenda

- Introduction
- Existing Solutions
- Architectural Qualities
- System Usage Patterns
- Requirements
- Design
- Validation
- Conclusion

Introduction

- Speaker's Qualifications
- Eight Fallacies of Network Computing
- Eight Truth of Distributed Computing
- Problem Statement
- Goal

Speaker's Qualifications

- Max Berger is a graduate student in computer science. He is planning to graduate with a Ph.D. in December 2006.
- His Bachler's project topic was "RemoteFinder - A graphical interface for SCP"
- His Master's topic was "Integrated PIM data management with SyncML"
- His main research interest is distributed applications, with a focus on dynamic distributed applications.
- Has published 3 papers (2 related to SILENUS)
- His degree is currently funded through teaching introductory classes in computer science.

Eight Fallacies of Network Computing

Essentially everyone, when they first build a distributed application, makes the following eight assumptions. All prove to be false in the long run and all cause big trouble and painful learning experiences.

1. The network is reliable
2. Latency is zero
3. Bandwidth is infinite
4. The network is secure
5. Topology doesn't change
6. There is one administrator
7. Transport cost is zero
8. The network is homogeneous

Eight Truth of Distributed Computing

Based on these eight fallacies, I define eight truth of distributed computing: (no relation to RFC 1925)

1. The network can fail at any time
2. Network messages arrive in random order
3. The network is always too slow
4. Someone is always listening and trying to break in
5. Machines get added and removed at any time
6. Every system has its own configuration and administrator
7. Moving data costs money
8. There will be any possible OS / architecture out there. They all want to be part of the network!

Problem Statement

Existing file storage solutions

- Are difficult to set up
- Require high maintenance
- Have problems with server downtime
- Don't handle topology changes
- Don't provide confidentiality from administrators
- Don't scale for large number of nodes
- Are incompatible with service-oriented architectures

Goal

To develop a distributed file storage solutions that

- Is truly network centric
- Addresses the issues given in the problem statement

File system Core Features Table

Feature	NFS	CIFS	AFS	Coda	Glob	Avaki
Remote	Yes	Yes	Yes	Yes	Yes	Yes
Migratable (same)	>=4	Yes	Yes	Yes	Yes	Yes
Migratable (other)	>=4	No	Yes	Yes	Yes	Yes
Replicated	No	No	R / O	R / W	R / O	R / W
Self optimizing	No	No	No	No	Yes	Yes
Self managing	No	No	No	No	Yes	Yes
Easy install	Yes	Yes	No	No	No	No
Compat.	Yes	Yes	Yes	Yes	No	No

Architectural Qualities

In addition to the file system core features we define the following architectural qualities for distributed systems:

- Confidentiality
- Global Availability
- Disconnected Operation
- Manageability
- Scalability
- Reliability
- Modifiability
- Platform Independence

Confidentiality

Problem:

- Most existing systems check only credentials
- Administrators can fake credentials
- Data is often sent through a public network
- Anyone with access to the hardware has access to the data.

Confidentiality (cont.)

Provided by:

- None of the discussed Solutions

Approach:

- Store data encrypted.
- Data is decrypted on the latest node possible
- Encryption key is secured: Pass phrase, PIN, Smart card

Global Availability

Problem:

- Users switch computers more frequently
- Data should be available everywhere
- Use of smaller devices, such as mobile phones
- Not always possible to install drivers on used machine

Global Availability (cont.)

Provided by:

- CIFS if the client is Windows / Macintosh
- Part of Globus (GlobusFTP)
- None have support for mobile phones

Approach:

- Provide support for common Internet protocols (e.g. WebDAV)
- Provide client for mobile phones

Disconnected Operation

Problem:

- The network is not available everywhere (yet)
- Increasing use of mobile devices: cell phone, PDA, laptop.
- Network failures

Provided by:

- Coda

Approach:

- Expect disconnection
- Synchronization mechanism
- Support hoarding

Manageability

Problem:

- Manage a thousands of files?
- Across hundreds of machines
- Machine failure

Provided by:

- Globus, Avaki

Approach:

- Use federated services
- Autonomic computing

Scalability

Problem:

- Many users
- Many machines

Provided by:

- AFS, Coda, Globus, Avaki

Approach:

- Distribution
- From client-server to federated services

Reliability

Problem:

- Machine failure leads to data loss

Provided by:

- AFS, Globus (R/O)
- Coda, Avaki (R/W)

Approach:

- R/W replication

Modifiability

Problem:

- Every software has bugs
- New features may be desired

Provided by:

- None of the existing solutions

Approach:

- Dynamic code loading

Platform Independence

Problem:

- Multiple operating systems
- Multiple computer architectures

Provided by:

- None of the discussed solutions

Approach:

- Use a virtual machine
- That uses an open specification
- And is widely used (JVM)

System Usage Patterns

- Usage Roles
- File System Users

Usage Roles

- File system users
- Administrators
- Optimizers
- Provisioners
- Intergrid service providers
- Nomadic access on laptop
- Mobile access on smartphone and PDA

File System Users

- Power users
- High performance computing lab
- Multi student lab
- Small office / workgroup
- Students rooming
- Family

Requirements

Provide

- All the core file system features
- All the distributed system qualities

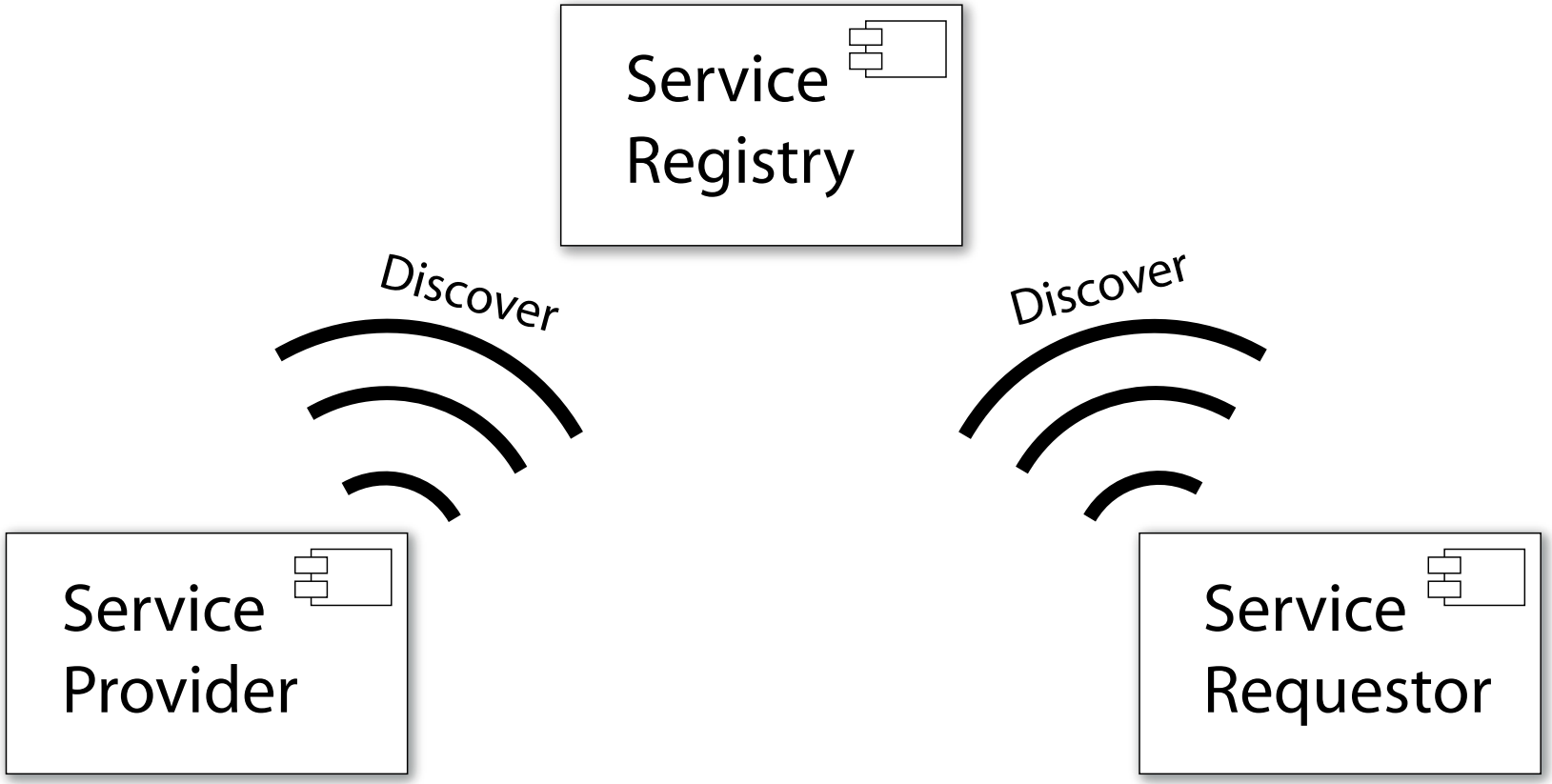
For:

- All the usage roles and their use cases

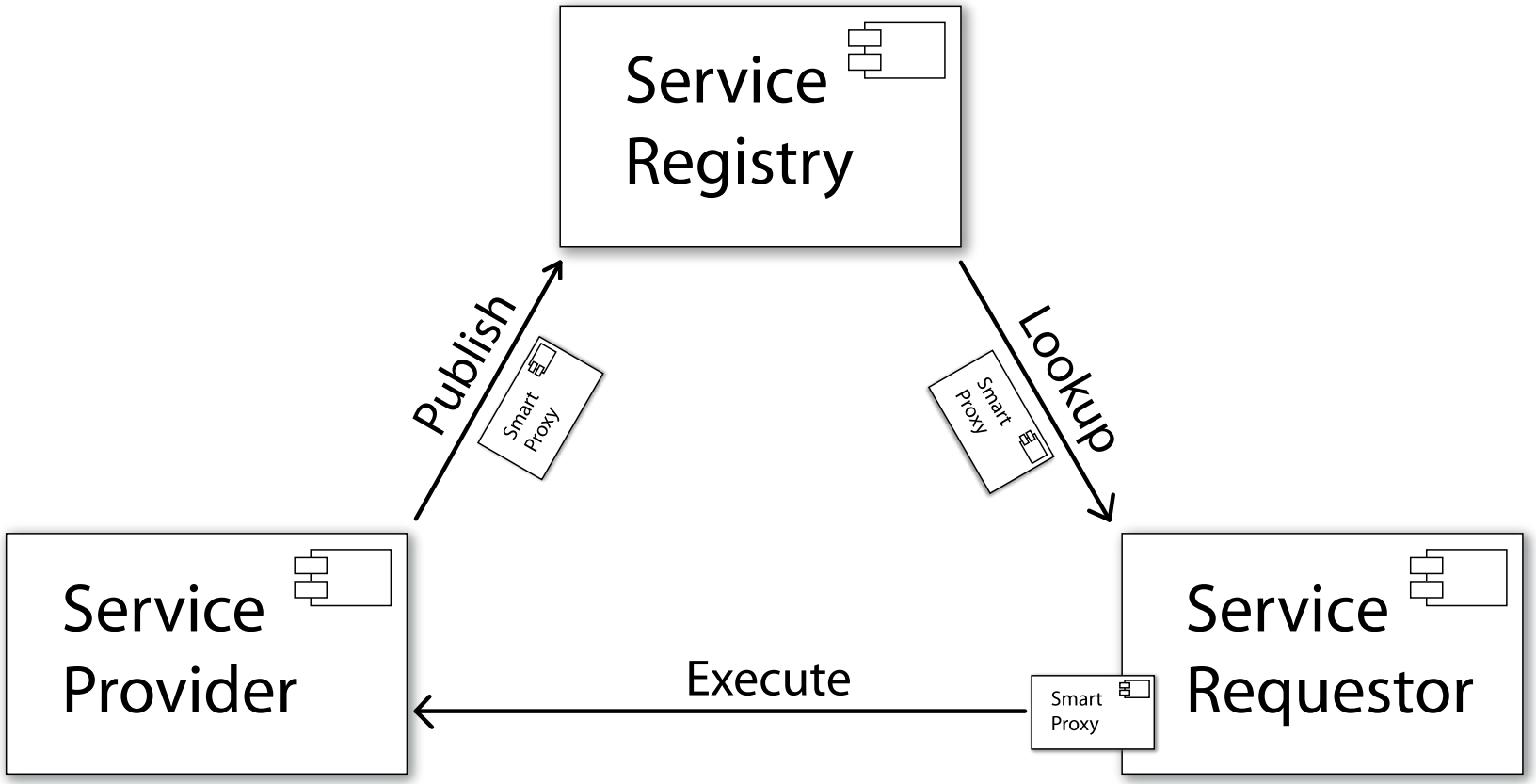
Design

- Service Object Oriented Architecture
- Service Tasks and Jobs
- System Architecture

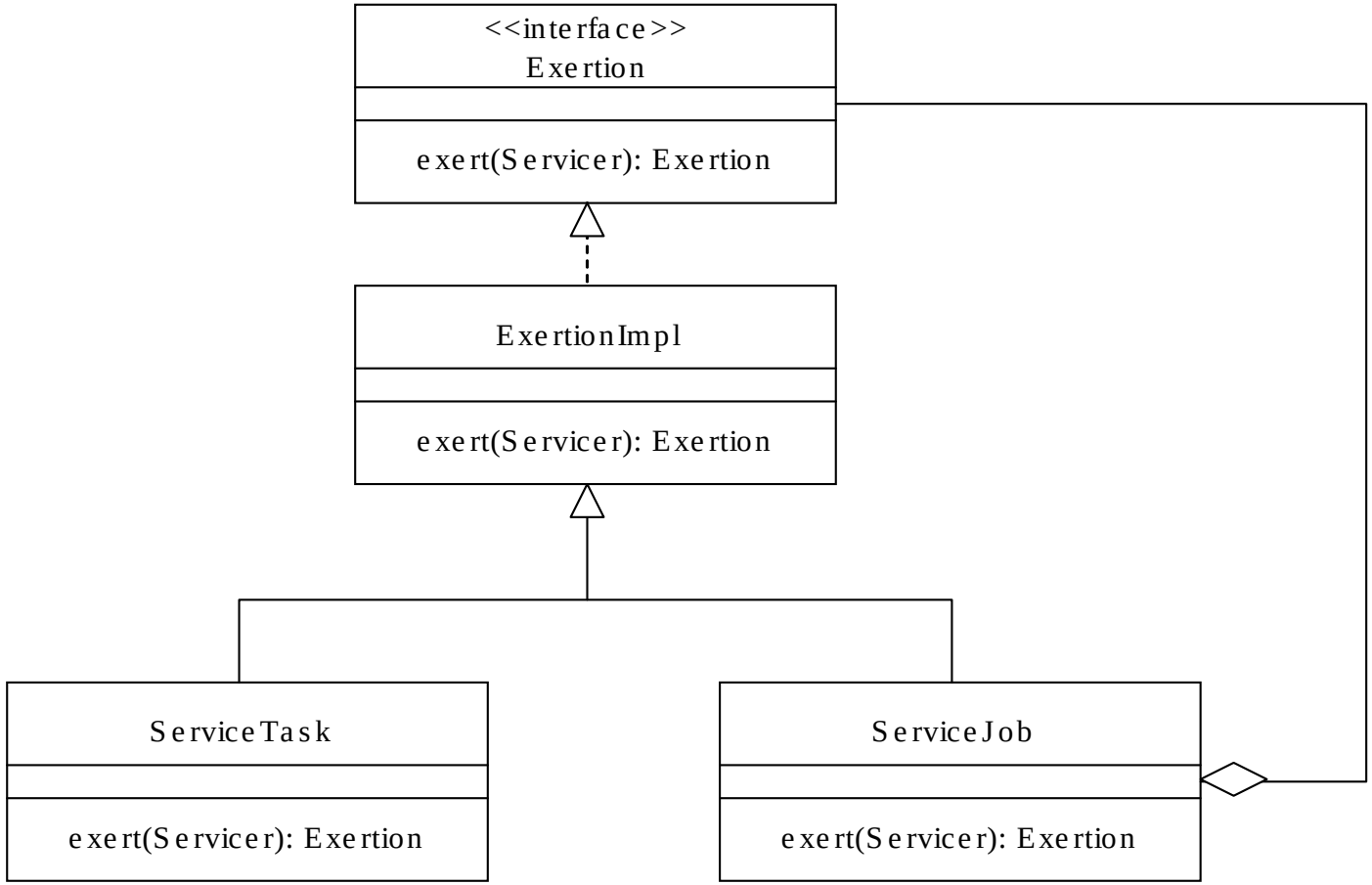
Service Object Oriented Architecture



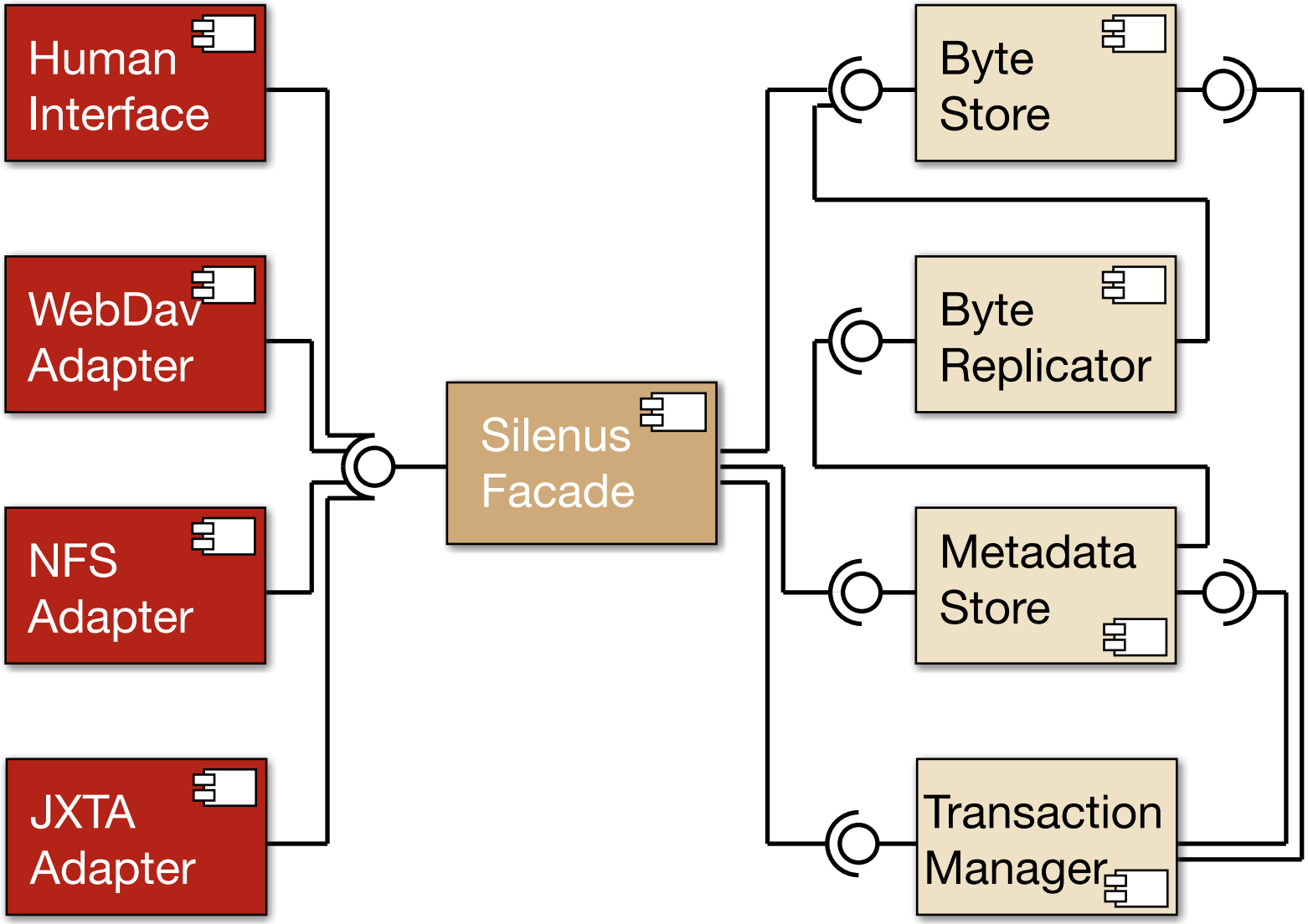
Service Object Oriented Architecture (cont.)



Service Tasks and Jobs



System Architecture



Validation

- Validation Requirements
- Conceptional Validation
- Operational Validation
- Data Validation

Validation Requirements

The following items need to be validated:

- All given file system core features
- All given distributed application qualities
- All given usage roles and their use cases

Conceptional Validation

Some items can be validated in the design.

Examples:

- Remote access
- Migrateable
- Self optimizing
- Confidentiality
- Modifiability
- etc.

Operational Validation

Some items have to be validated through experiments:

- Constant small tests during development (most features)
- Automatic unit tests during development
- Medium size integration tests in the SORCER lab (most qualities)
- Possible larger test with data from Sloane Sky Survey
- Possible larger test with HPCC (manageability, scalability)

Data Validation

- A file downloaded from the file system should have the same contents as a file uploaded to the file system
- The metadata schema should represent the same attributes for a file
- Validate all replicated files (content and metadata)

Conclusion

- Benefits
- Schedule

Benefits

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

Schedule

What	When
Background research	01/04 - 12/04
Initial proposal	06/04
Prototype design	06/04 - 12/04
Byte Store prototype	10/04 - 02/05
Metadata Store prototype	01/05 - 06/05
Service-Oriented redesign	07/05 - 12/05
Service-Oriented implementation	10/05 - 10/06
Proposal	03/06
Defense	10/06

Appendix A - File Systems

- NFS
- CIFS
- AFS
- Coda
- Globus
- Avaki
- Compatibility with Existing Software

NFS

- Network File System
- Widely used in Unix
- Based on Remote Procedure Call (RPC)
- Fails if network fails
- Administrators must trust each other
- Shares are identified by machine / path on the machine

CIFS

- Common Internet File System
- Formerly: Server Message Block (SMB)
- Standard in Windows
- User-based authentication
- Fails if network fails
- Shares are identified by machine / share name

AFS

- Andrew File System
- Developed at CMU, then IBM, now open source
- Need contact with AFS master node
- Provides read-only replicas of files
- Very difficult to set up (requires Kerberos server)

Coda

- Based on AFS
- Developed at CMU
- Provides read - write replicas
- Support for hoarding
- Sophisticated conflict resolution
- Difficult to set up

Globus

- Globus alliance: Technology for the "grid"
- Data-Grid solution
- Splits file metadata and file content
- Based on protocols
- Multi source download for performance
- Incompatible with existing software

Avaki

- Commercial solution
- Requires fast, reliable network
- Optimized for fast transfer rather than reliability
- Provides unification of data from different sources

Compatibility with Existing Software

Problem:

- Multiple operating systems: Windows, MacOS, Linux, Unix
- Every system would require a special driver

Provided by:

- CIFS, AFS, Coda

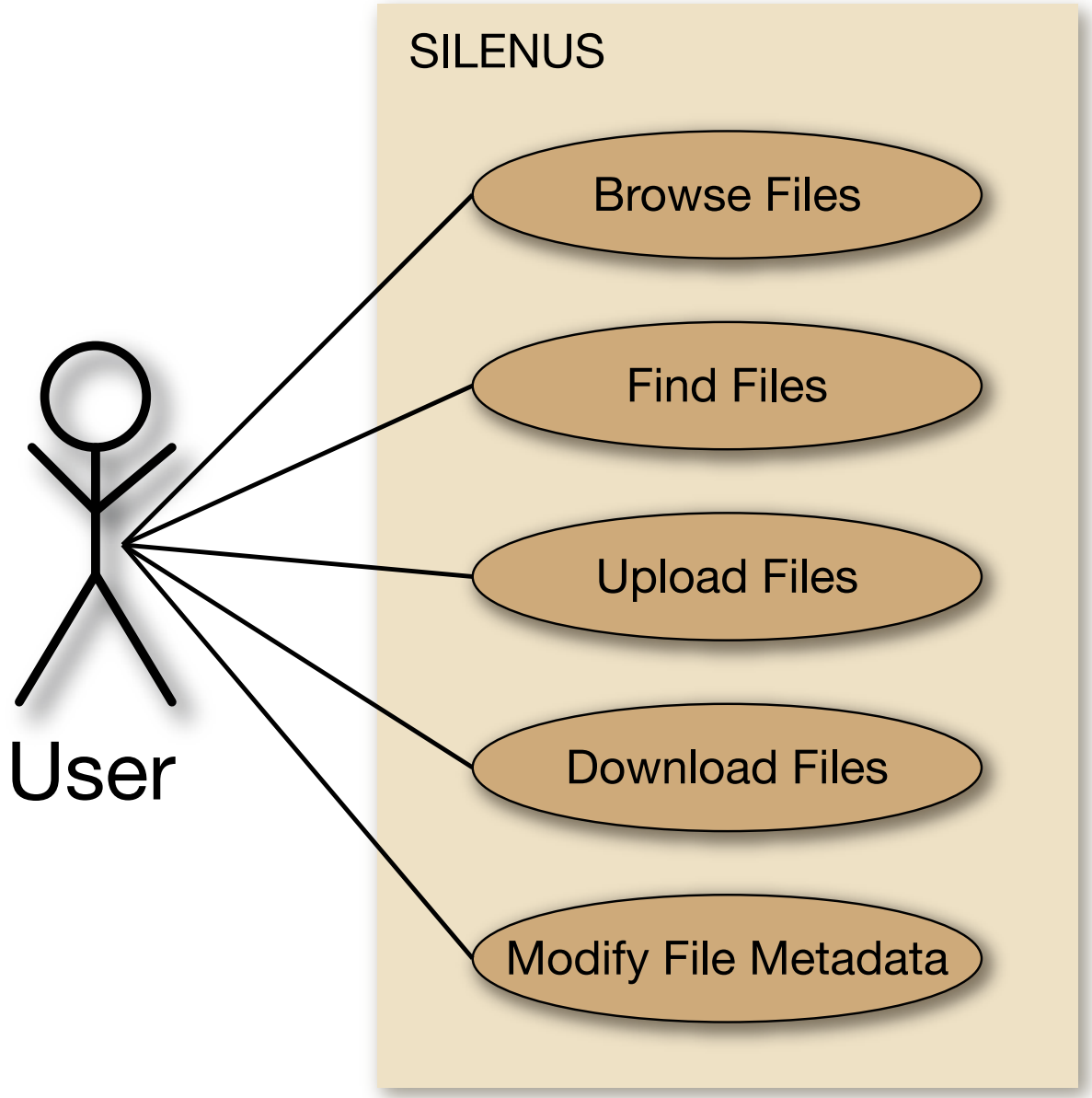
Approach:

- Use a driver that already exists
- Use open standards (WebDAV)

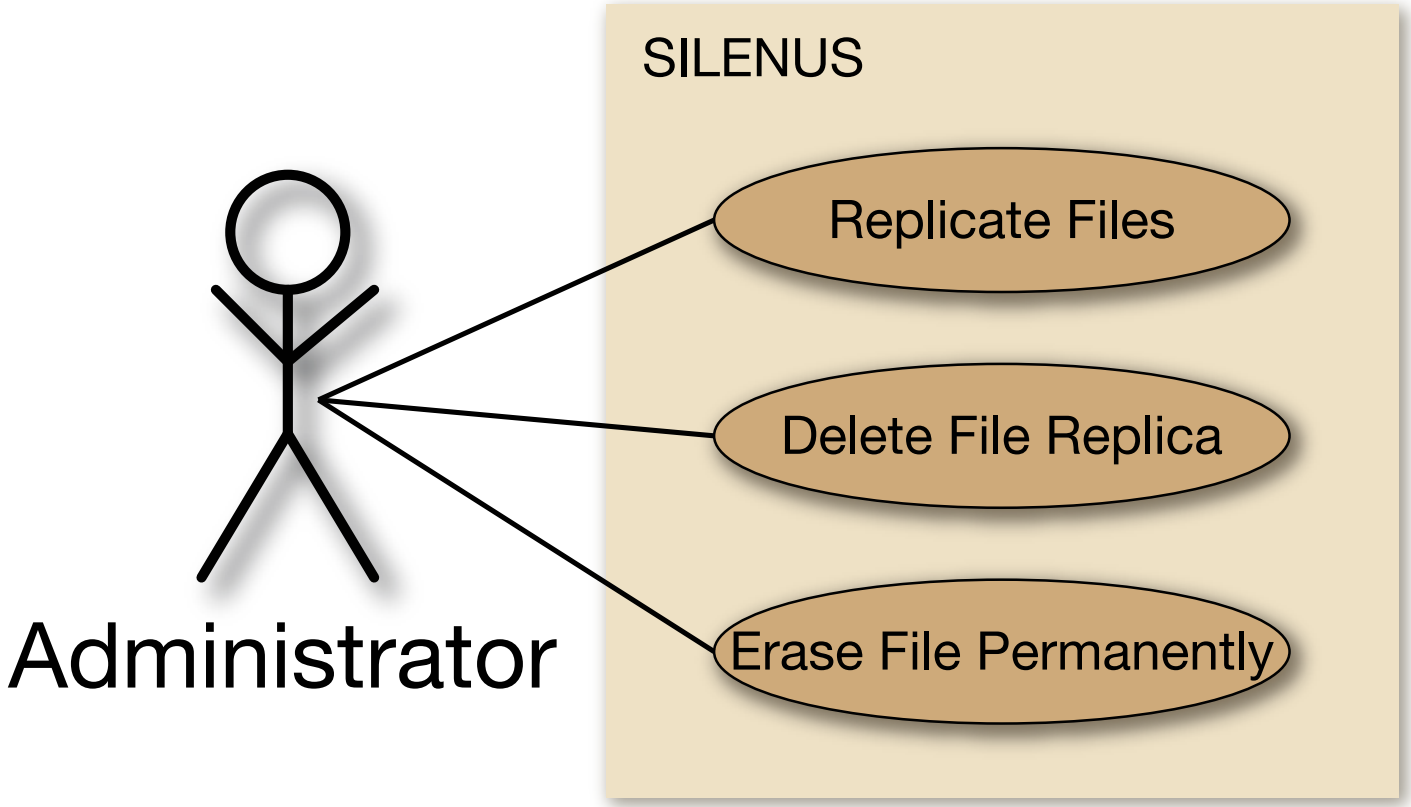
Appendix B - Use Cases

- User Use Cases
- Administrator Use Cases
- Use cases

User Use Cases



Administrator Use Cases



Use cases

- Add mobile access
- Add hoarding

Appendix C - SOA

- Service Oriented Computing
- Service Oriented Programming
- Peer-to-Peer Networking
- JXTA

Service Oriented Computing

- Uses SOA
- Need existing infrastructure

Example: Jini Network Technology

- Specification for SOC with Java
- Provides additional specification for attaching user interfaces to services
- Developed in community process
- Reference implementation by Sun

Service Oriented Programming

- Service Oriented Programming
- A SO program (job) consists of tasks
- Each task consists of a Service Method and a Service Context
- SORCER provides framework for SOP
- Currently job is a composite of elementary tasks and jobs
- Work is underway to expand the set of tasks to include conditional looping for full algorithmic logic.

Peer-to-Peer Networking

- Popularized by Napster
- Original: Each peer is equal
- Now: More lattice-type structure
- Examples: Napster, Gnutella, KaZaa, eDonkey, JXTA

JXTA

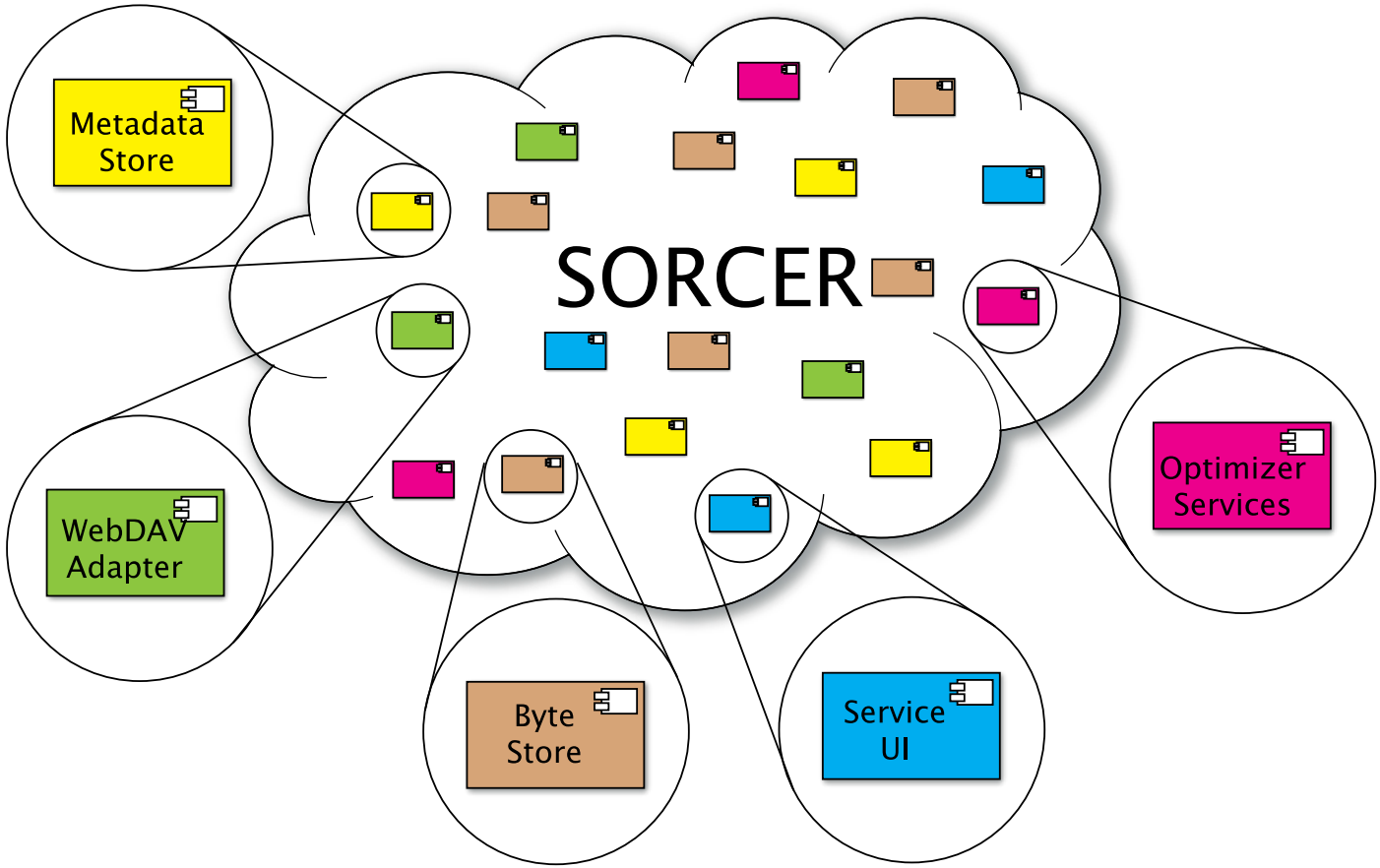
- Open specification for peer-to-peer architecture
- Developed in community process
- For all platforms
- Provides virtual channels for communication

Appendix D - Design

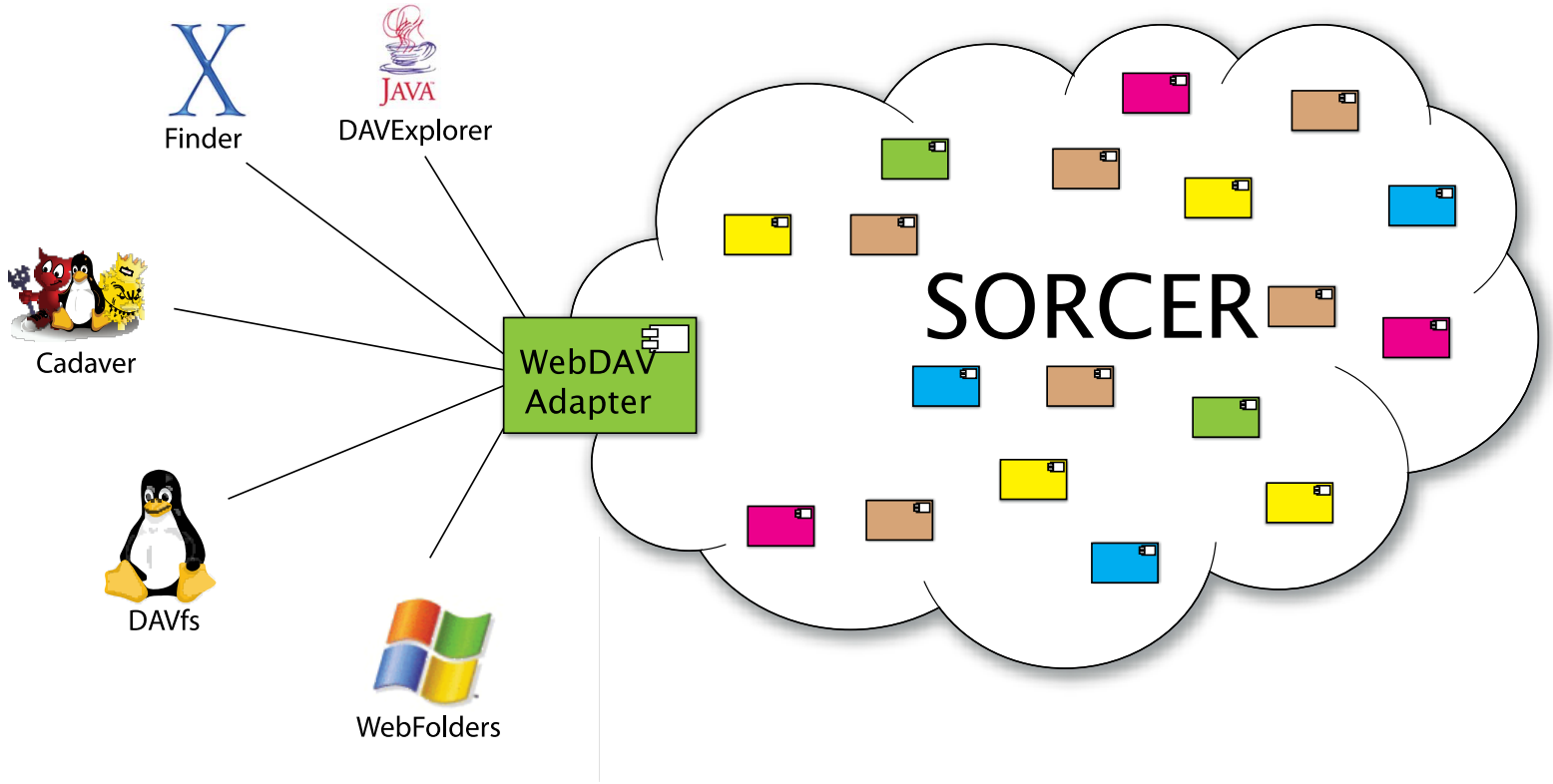
- Components
- WebDAV Adapter
- Service Context
- Browse Files
- File Upload
- Upload Transactions
- After Upload
- Download File
- Modify File Metadata

Components

communicate via the SORCER infrastructure

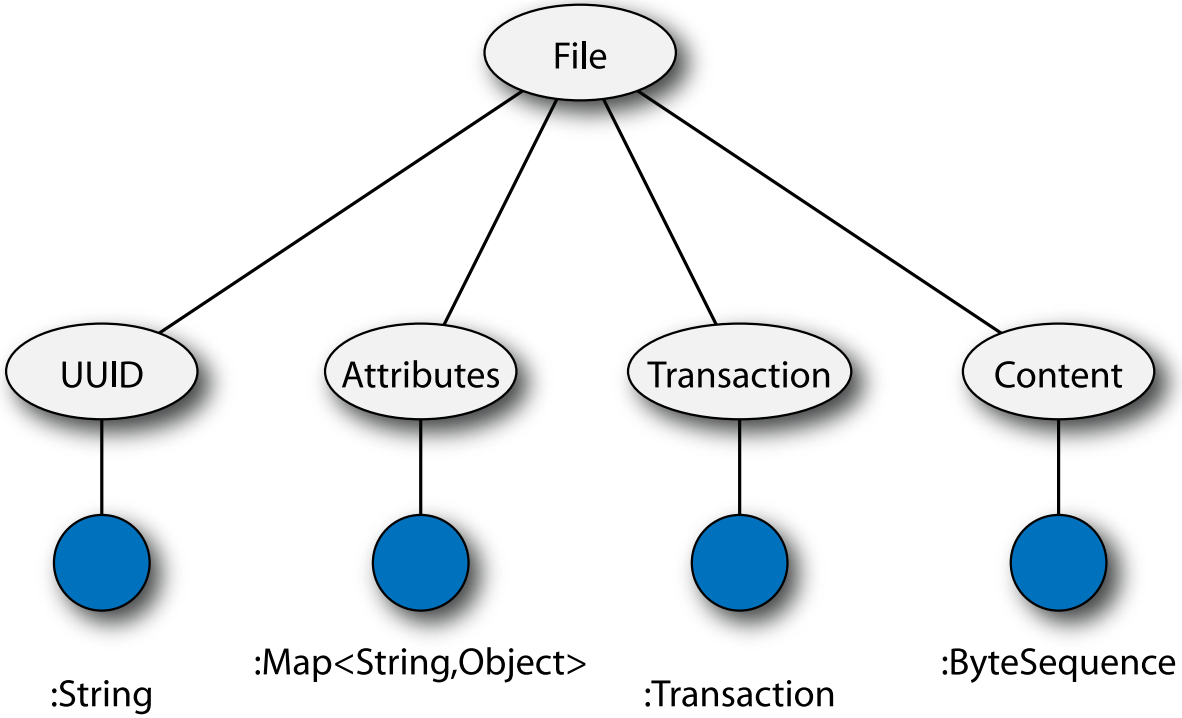


WebDAV Adapter

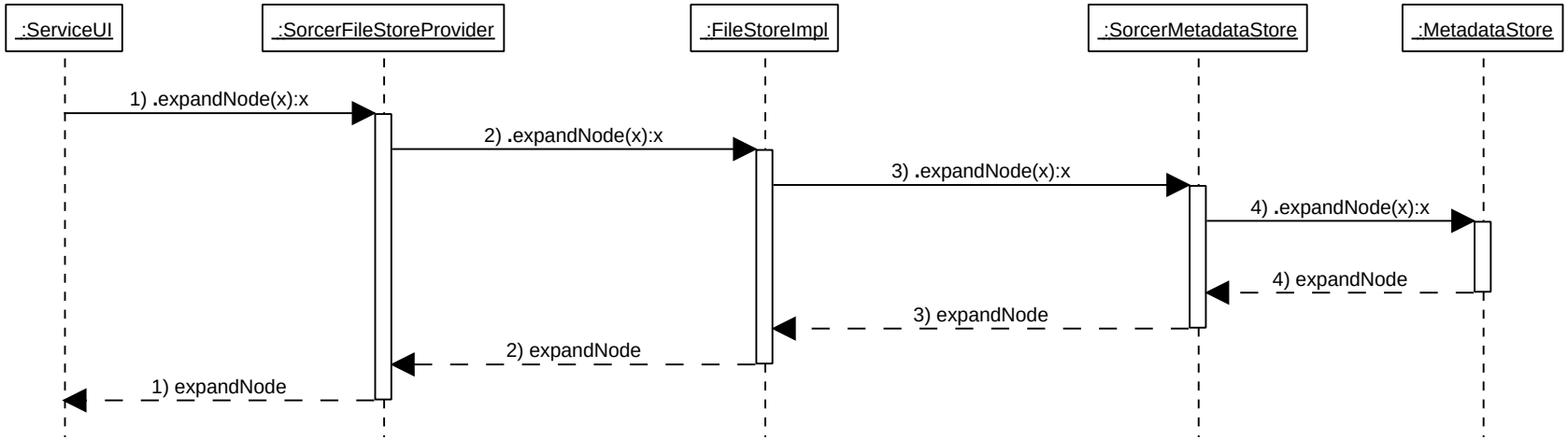


- Provides support for existing applications

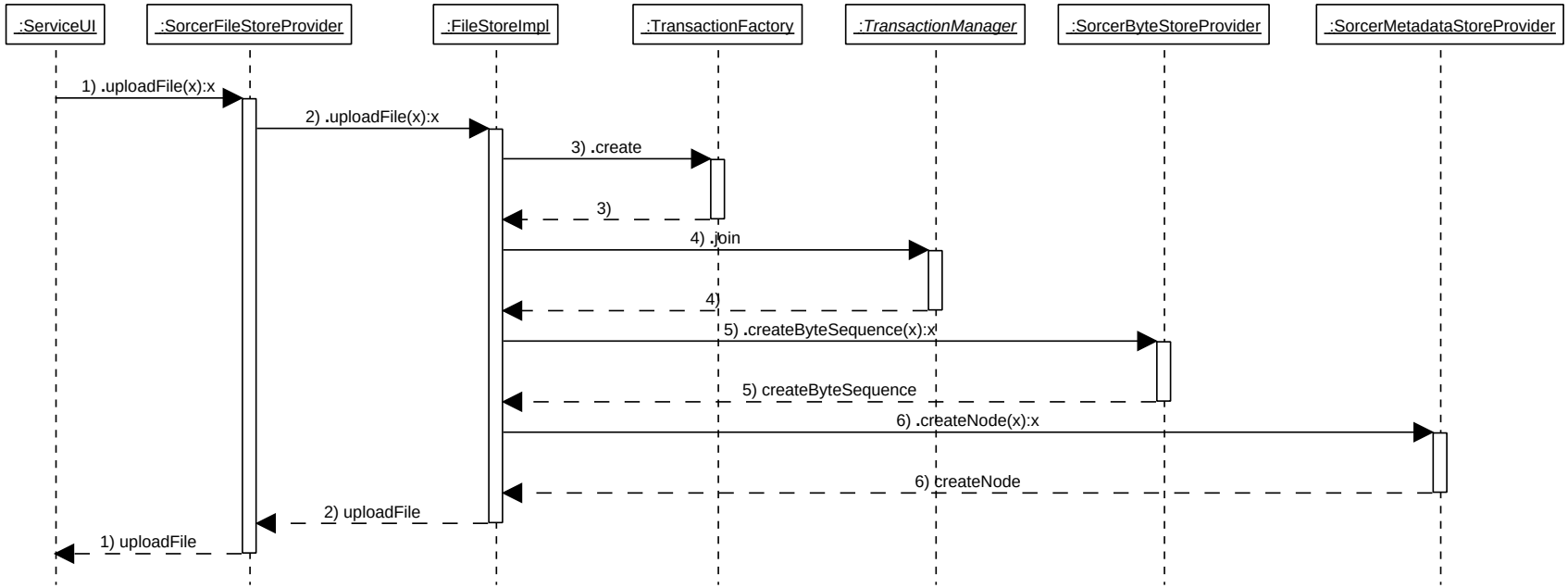
Service Context



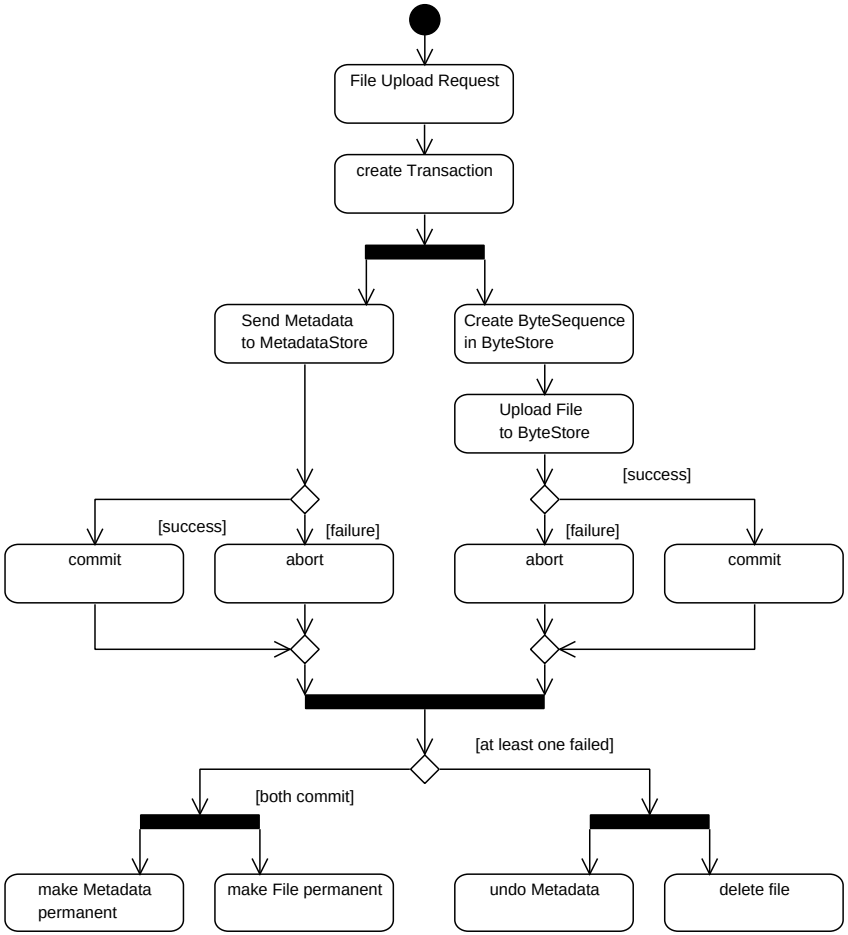
Browse Files



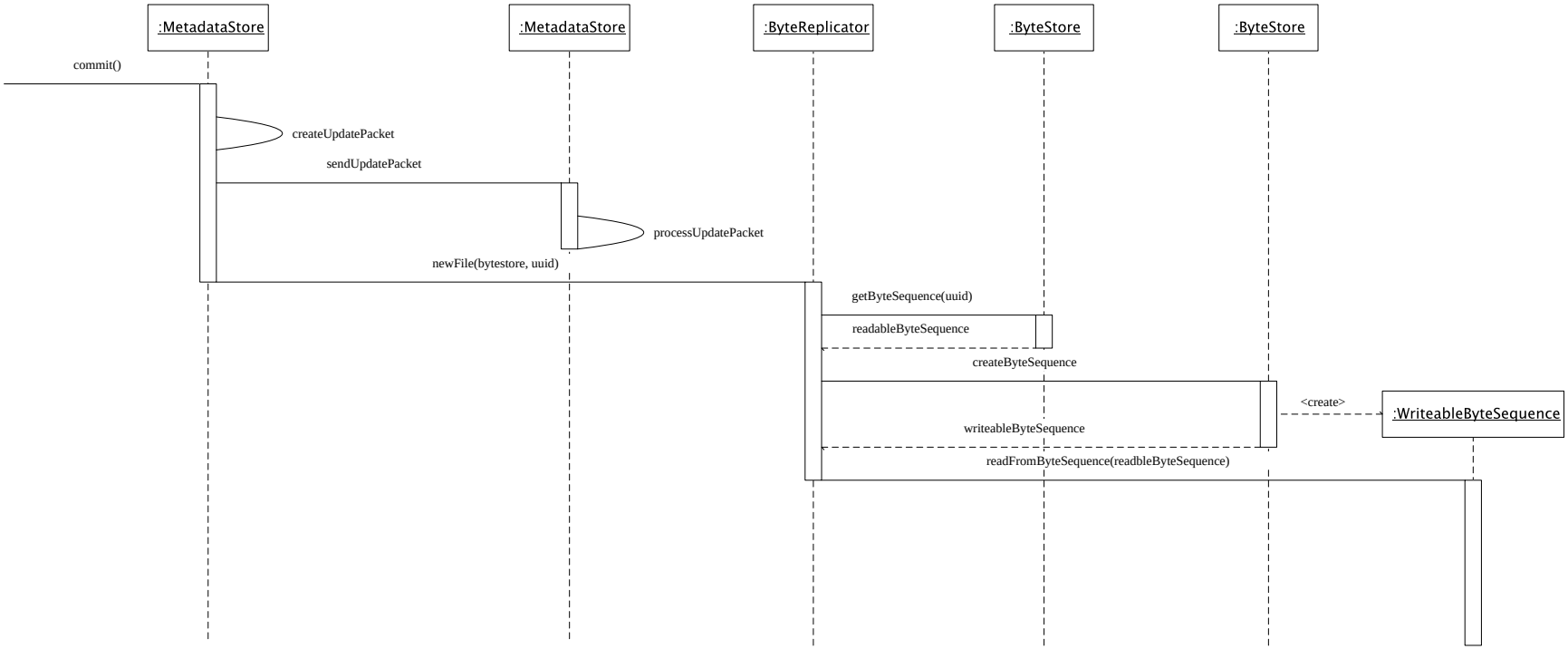
File Upload



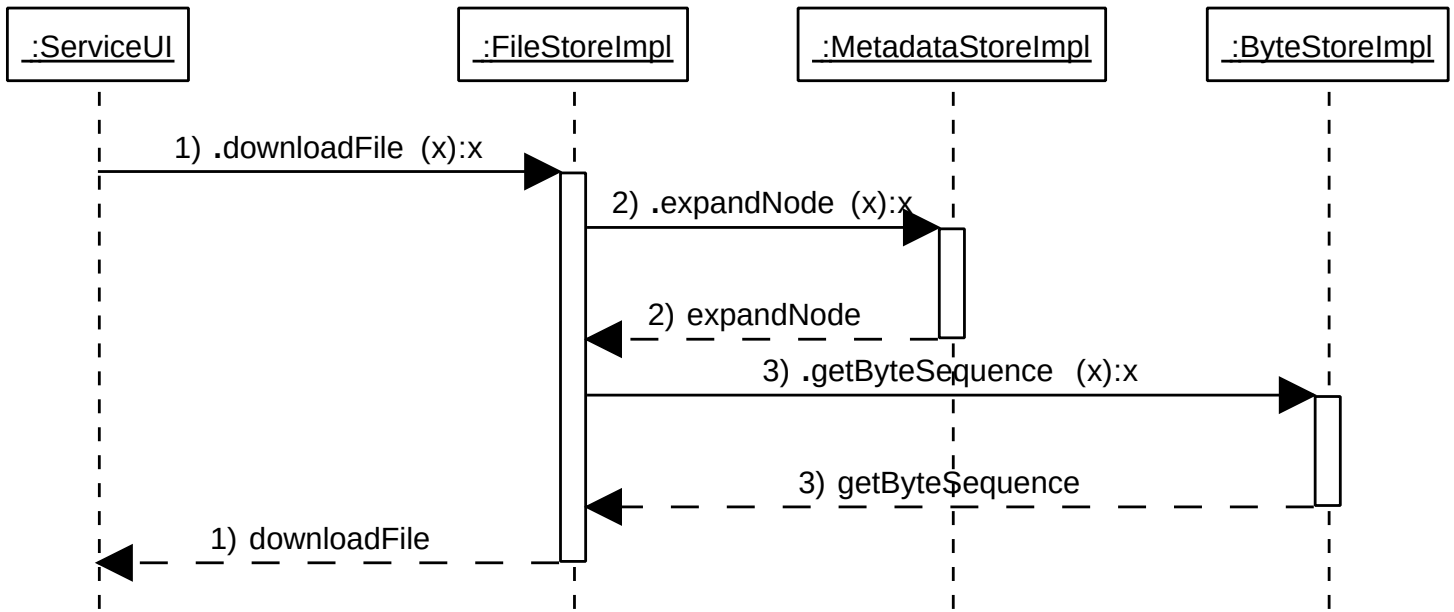
Upload Transactions



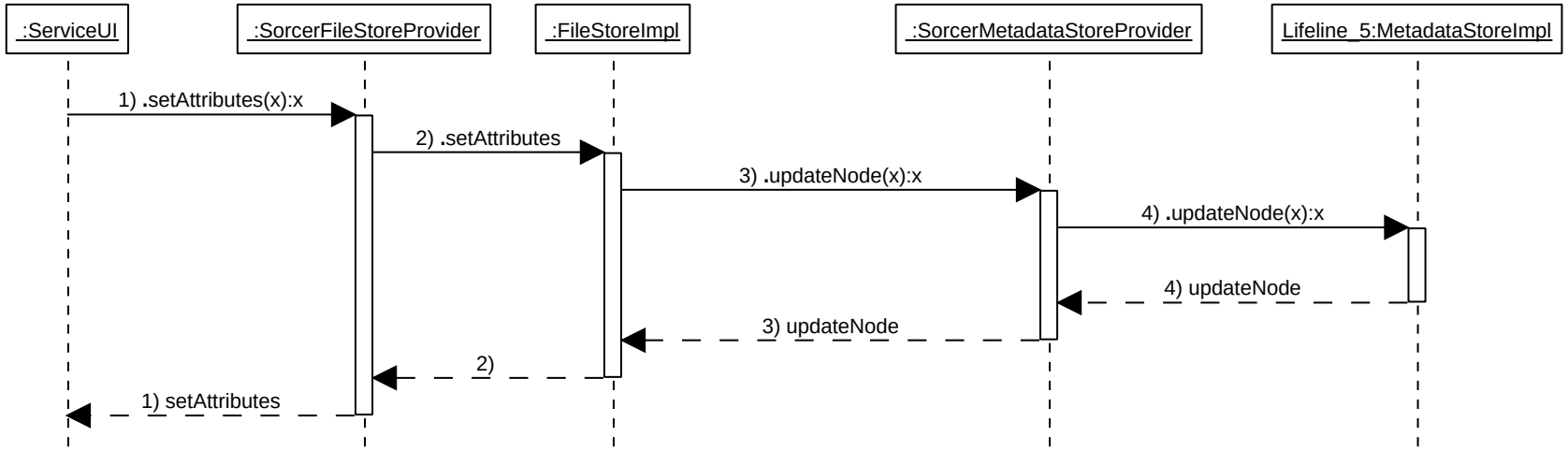
After Upload



Download File



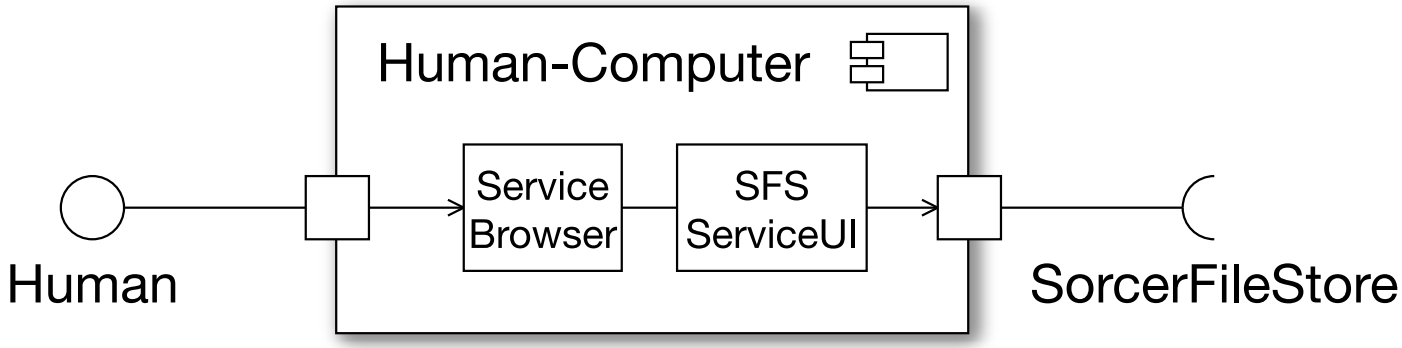
Modify File Metadata



Appendix E - Components

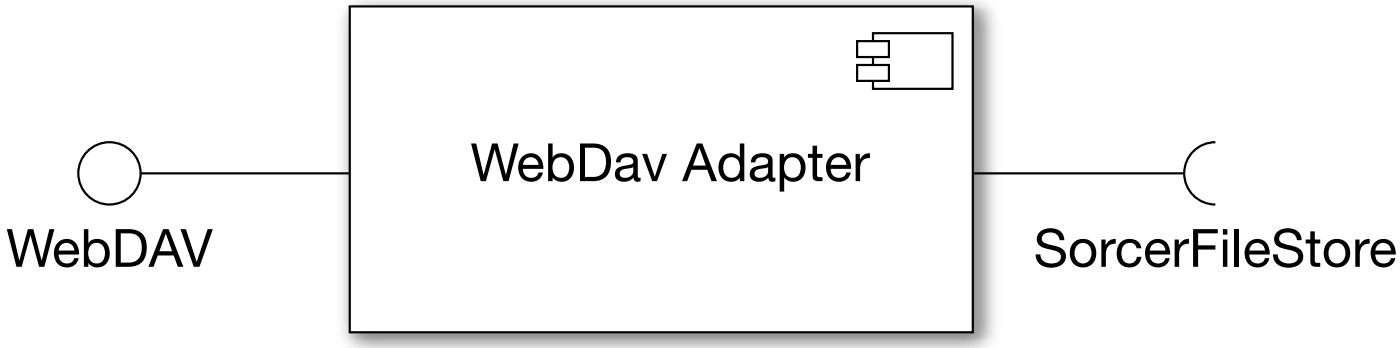
- Human-Computer Interface
- WebDAV Adapter
- SILENUS Facade
- Byte Store
- Metadata Store
- Optimizer Services

Human-Computer Interface



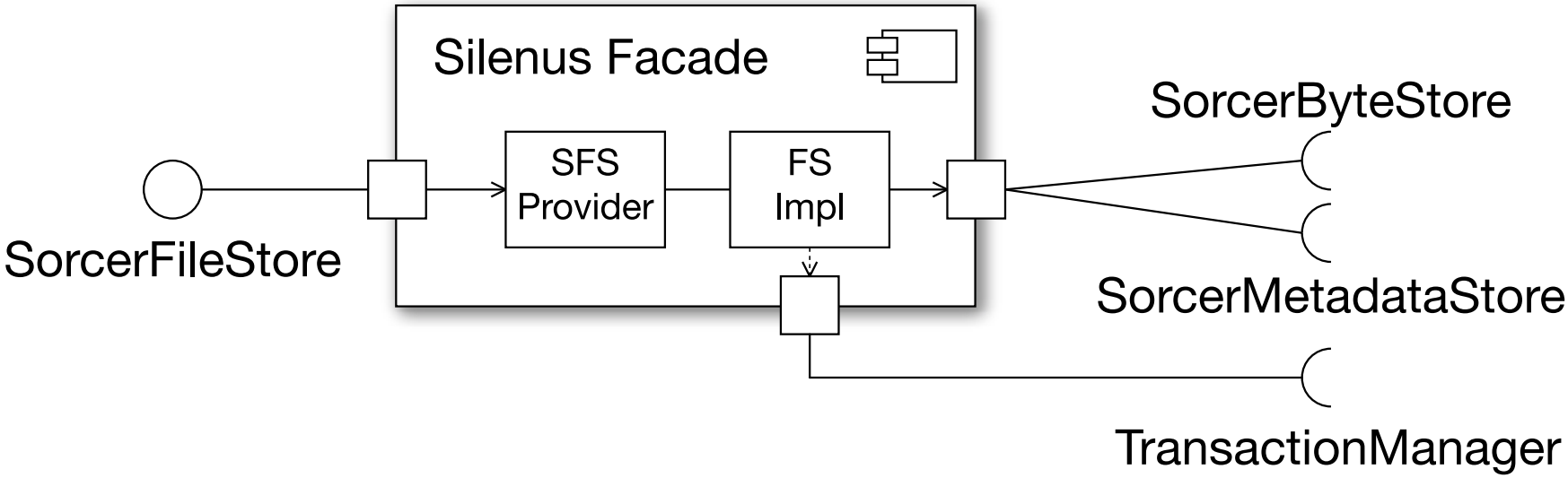
- Zero install
- Need only Service Browser (like Web Browser)
- Dynamically downloads interface
- Provides interface for all functions provided by the File Store

WebDAV Adapter



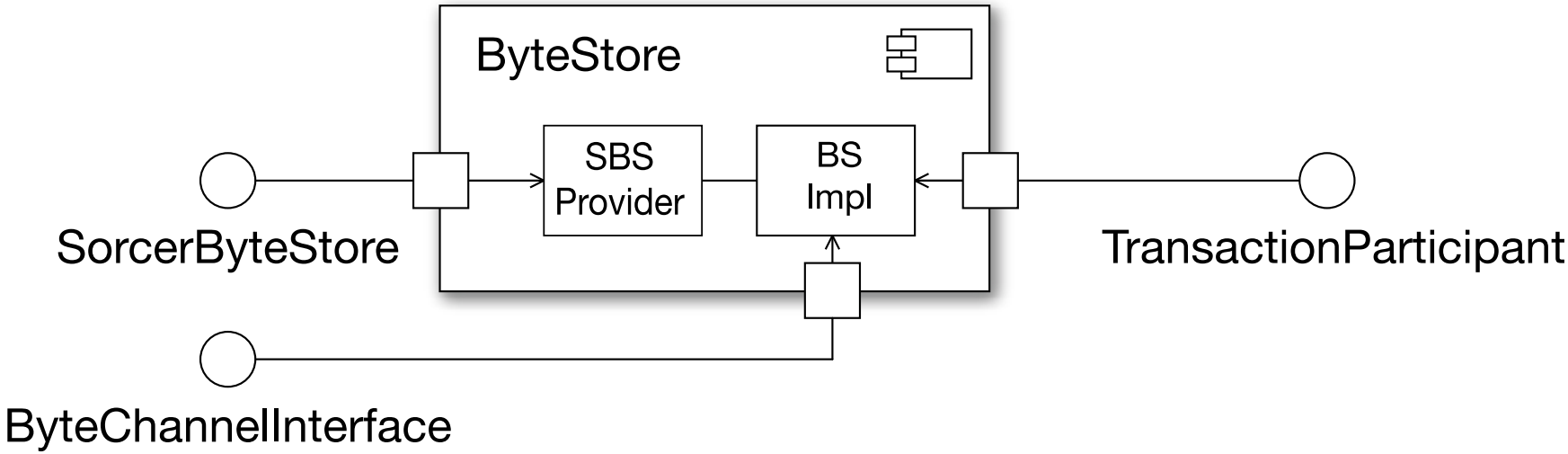
- Provides support for all OS's that support WebDAV (Linux, OS X, Windows)
- Work by Fajin Wang

SILENUS Facade



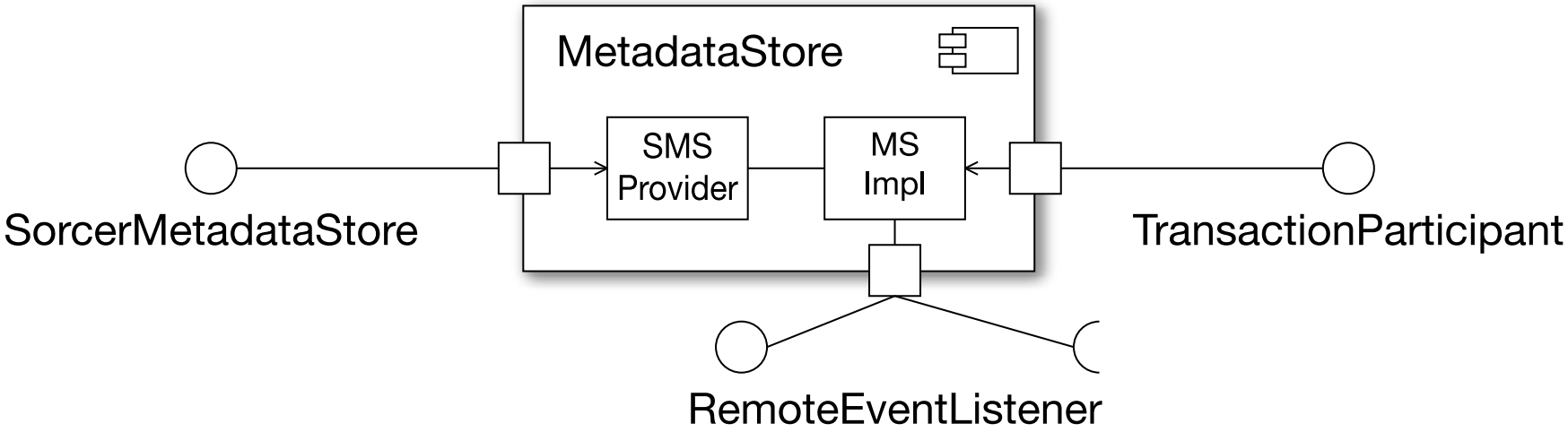
- Common entry point
- Provide one interface for the user
- Hides internal transactional semantics
- Splits up / combines files into file contents and file metadata

Byte Store



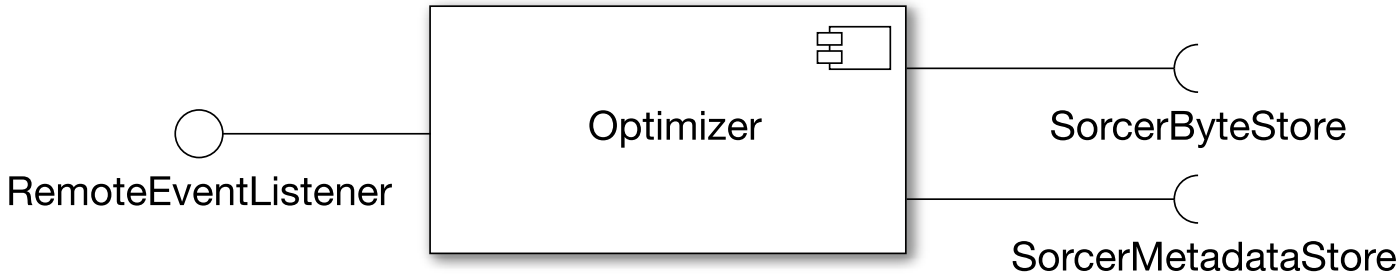
- Stores the actual file data
- Files are identified uniquely by service ID and entry ID
- Does nothing but file storage
- Analogy: hard drive
- Different byte stores contain different content

Metadata Store



- Stores the file metadata
- Creates hierarchical structure
- Stores any kind of metadata: file name, size, modified date, icon, security information, etc.
- Analogy: file system
- Uses embedded database
- Synchronizes with other metadata stores to have the same information available at all times

Optimizer Services



- There can be many different optimizer services
- Example service: ByteReplicator
- Gets triggered after a file is uploaded
- Replicates that file onto another machine
- Can also be triggered when byte stores become unavailable
- Can detect full byte stores and provision new services (may use Rio)