Online and Offline Operation of J2EE Enterprise Applications

Diploma Thesis in Computer Science
submitted by

Christian Stettler

from Dietikon, Switzerland
Register No. 99-710-048

Supervisor
Prof. Dr. Helmut Schauer
Software Engineering Group
Department of Information Technology
University of Zurich, Switzerland

Advisor
Dr. Thomas Ernst
Canoo Engineering AG
Basel, Switzerland

November 5, 2003

Abstract

Enterprise applications built on top of the Java 2 Enterprise Edition (J2EE) platform require a permanent connection between the application client and the J2EE application server. In scenarios where the enterprise application has to provide its services disconnected from the application server, because of missing connectivity or a server failure, the enterprise application has to support an offline operation mode.

This thesis discusses issues arising from operating an enterprise application in offline mode, such as data synchronization, offline security, and transitions between online and offline operation, and proposes architectural solutions that enable both online and offline operation for J2EE enterprise applications. In addition, this thesis provides a framework for the development of offline-capable J2EE enterprise applications including reference implementations for various design concepts discussed and solutions proposed. Both the conceptual and programmatic solutions are integrated in and demonstrated by a business-related 'proof of concept' showcase application.

Acknowledgement

Numerous people supported the formation of this diploma thesis, in different ways: First of all, I would like to thank Prof. Dr. Helmut Schauer and Dr. Bruno Schäffer for the proposition of the topic. Secondly, my appreciation goes to Dr. Thomas Ernst at Canoo Engineering AG, who guided me through the whole process of writing this thesis. In addition, I would like to thank the following people for valuable discussions, contribution and proof-reading (in alphabetic order): Norman Cohen, Daniel Grob, Volker Jach, Sibylle Peter, and especially Sandra Wendland.

Special thanks go to my parents Hans and Barbara and my brother Martin. It was their boundless support that made my studies at the university possible, and that finally brought it to success.

Dedicated to
Nora
with gratitude for her
patience, support and love.

Disclaimer

Java, J2EE, J2SE, EJB, JavaBeans and JSP are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. or other countries. JBoss and JBoss Group are registered trademarks and service marks of Marc Fleury under operation by The JBoss Group, LLC. All other products or services mentioned herein are the trademarks or service marks of their respective companies or organizations.

Contents

Section 1

Introduction

Architectures of today's enterprise applications have become increasingly server-centric. Application functionality which used to be implemented within a fat client application running on the client's machine now resides on a central server instance. The responsibilities of the client application are scaled down to presenting the user interface and acquisition of user inputs, making it a thin client. Thus, a permanent and reliable connection between the client and the server is required to operate an application that is based on this server-centric approach. However, not even the current network and Internet-dominated era offers ubiquitous connectivity. As a result, there is an increasing need for architectures that allow enterprise applications to operate in a disconnected offline mode.

This thesis analyzes the requirements and possibilities of offline-capable applications with respect to the Java 2 Enterprise Edition (J2EE), a Java-based realization of a server-centric application architecture approach. In addition, this work discusses the implications on the conceptual solution resulting from the different partitioning of application logic between client and server and highlights the key aspects of operating applications in both online and offline mode. A programmatic framework for the development of offline-capable J2EE enterprise applications is presented and verified using a prototypical demonstration application that implements a prevalent use case for offline operation. While most existing J2EE-oriented literature covers the online-specific aspects of J2EE enterprise applications, this thesis focuses on issues, concepts and solutions concerning offline operation of enterprise applications.

1.1 Structure

This section describes the objectives of the thesis and introduces the fundamental problems of providing offline operation support for J2EE enterprise applications. Section 2 introduces the J2EE platform and related Java-based technologies required for further understanding of the discussion. Section 3 portrays three enterprise application use cases, each highlighting specific characteristics of offline-capable enterprise applications.

While section 4 discusses detailed core issues related to offline operation and provides conceptual solutions for these issues, a programmatic framework supporting the development of offline-capable J2EE enterprise applications is proposed and explained in section 5. Section 6 proposes several architectural solutions that enable both online and offline operation for J2EE enterprise applications based on the framework mentioned. A demonstration application, implementing an enterprise application scenario with support for offline operation that builds on and integrates the concepts described in earlier sections, is presented in section 7. Section 8 summarizes and concludes the essential aspects and results of this work, while section 9 discusses unsolved issues, solution drawbacks and potential future developments.

Appendix A outlines the contents of the resource CD that accompanies the thesis with Java source code, JavaDoc documentation and additional electronic resources. Appendices B and C provide additional information on running and exploring the demonstration applications developed during the thesis work.

1.2 Objectives

The goal of this thesis is to examine the requirements, possibilities and consequences of developing offline-capable, J2EE-based enterprise applications. The arising issues will be addressed both conceptually and programmatically within J2EE context. Thereby, the aspects of different J2EE enterprise application architectures have to be taken into account in order to adapt the delivered solutions to the most relevant architectural variants. This thesis aims to provide a solid background on offline-related problems and to deliver concepts for the design and development of offline-capable enterprise applications with J2EE. Additionally, programmatic assistance for application developers in the form of a prototypic J2EE online-offline application framework that implements the conceptual solutions proposed and facilitates the development of offline-capable J2EE enterprise applications will be provided.

1.3 Fundamental Problems

Since J2EE enterprise applications are typically based on one or multiple servers providing most parts of the application logic[1] and hosting the application data[2], operating such applications disconnected from the server in offline operation[3] is not trivial. When running enterprise applications in offline operation, services and data residing on the server are not accessible due to the lack of connectivity to the server environment. This section provides an overview of the basic problems of operating an enterprise application in offline mode. These problems are discussed in detail in the course of this paper.

[1]Application logic is the part of an enterprise application that implements the business rules and processes the application-relevant data. Application logic is the active part of an enterprise application.

[2]Application data is the passive part of an enterprise application that represents business entities processed by the application logic. The terms 'business data' and 'business objects' are considered as being synonyms for application data.

[3]The terms 'offline operation' and 'online operation' are described in section 4.

1.3.1 Offline Availability of Application Logic

Application logic implemented in the server components of an enterprise application and accessed by connecting application clients[4] in online operation is not available to application clients that are running in offline operation. Thus, application logic required for offline operation has to be transferred to, installed on or simulated locally by the client machine hosting the application client.

1.3.2 Offline Availability of Application Data

In an enterprise application scenario, application data is centrally held in data stores on one or more servers. The amount of application data residing on the server is potentially large. Specific parts of this data need to be available on the client side when running the enterprise application in offline operation. Thus, this data has to be downloaded and stored locally on the client machine that runs the application client in offline operation.

1.3.3 Security of Offline Application Data

As enterprise applications often include sensitive application data, access to this data is typically restricted to users, roles or other privileges which are set and controlled by the server part of the application. In offline operation, where application data is stored locally on the client machine, similar access control mechanisms may be required to protect access to the locally stored data. Besides protecting data confidentiality, the issue of unauthorized modification of offline application data by a malicious user has to be tracked in order to avoid unwanted effects to the server-side data.

1.3.4 Data Synchronization

During offline operation, the local enterprise application on the client machine is completely disconnected from the server. Instances of the same application may be running on other client machines at the same time, either connected to or disconnected from the server. Data stored locally on the client machine running in offline mode may be modified or deleted or new data may be created. Potentially, the same data entity may be modified by multiple application clients concurrently, running either in online or offline operation, with potential effects on the server data store[5]. These modifications, which may be in conflict with others, have to be synchronized with the data on the server when an application client is switching back to online operation. During this process, conflicts need to be reconciled and updates have to be made available to other application clients connected.

[4]Unless specified further, an application client is considered as the part of an enterprise application which is presented to the user.

[5]The server data store is a persistent data store for application data objects that resides on a server machine. This data store is used for storing data objects when running the enterprise application in online mode. Server data store and central data store are considered to be synonyms.

Section 2

J2EE Technologies

This section intends to provide fundamental information on the Java 2 Enterprise Edition (J2EE) and its related technologies where required for further discussion of the topic. Additional information may be found in [Sha01], [Bod$^+$02] and [Sin$^+$02].

2.1 Introduction to J2EE

The Java 2 Enterprise Edition (J2EE) is a platform for developing distributed, server-centric enterprise applications. Sun Microsystems Inc. (Sun), the provider of J2EE, defines J2EE as

> 'a set of coordinated specifications and practices that together enable solutions for developing, deploying, and managing multi-tier server-centric applications. Building on the Java 2 Platform, Standard Edition (J2SE), J2EE adds the capabilities necessary to provide a complete, stable, secure, and fast Java platform to the enterprise level.' [Sun03a]

Under the auspices of the Java Community Process (JCP), Sun and other software industry leaders unified existing and upcoming enterprise-related standards and application programming interfaces (API) into the J2EE platform. This collection of specifications is implemented by Sun and other product vendors in the form of J2EE application servers. Vendor specific products completely satisfying the J2EE specification are referred to as J2EE-compliant application servers. Enterprise applications respecting the J2EE specification and relying on the J2EE technologies specified therein are considered to be portable across different J2EE-compliant application servers.

2.2 Enterprise Application

The term 'Enterprise Application' is used to describe business-related applications that consist of a server and a client part, also referred to as client-server application [Orf97]. As the client part of the enterprise application has to be connected to the application server using an appropriate communication infrastructure, enterprise applications are inherently considered to run in online operation. In contrast, an offline enterprise application is considered to be an enterprise application that is extended by an offline operation mode and therefore may either run in online or offline operation. According to [Fow02], an enterprise application generally has the following characteristics:

Multi-Tier Architecture An enterprise application is separated across multiple tiers, where each tier manages one or more specific aspects of the application. The architecture and design of an enterprise application is server-centric, therefore most of the tiers reside on a server machine. Each tier may be located on a separate physical machine or multiple tiers may be hosted on the same physical machine.

Multiple Users and Roles An enterprise application is accessed by multiple users that may act in different application user roles. Each role has a set of privileges associated that allows a user owning that role to perform certain operations based on application-defined security constraints. The behaviour and appearance of the application may differ depending on a particular user and the roles associated with it. Additionally, multiple users collaborate in order to fulfill a business process represented by the enterprise application.

Confidential Information An enterprise application manages and works on confidential information not intended to be accessible for the public, for example customer data, financial information or internal company knowledge. Disclosure of this information rapidly leads to high cost and loss of reputation.

Security Constraints An enterprise application typically has strong requirements concerning security. This may include the need for authentication, authorization and data integrity[1]. Access to enterprise applications is often not public but restricted to well-defined groups of users like employees, business partners or customers.

High Availability Requirements An enterprise application is often business-critical and therefore has to be continuously up and running. In addition, an enterprise application has to endure heavy loads of requests. Typically, already a short outage of the application results in significant financial and/or reputational loss.

Within this document, 'enterprise application' and 'business application' are considered to be synonyms. Also, when referring to 'application', an application having multiple or all of the characteristics of an enterprise application is meant.

2.3 J2EE Platform

The J2EE specification [Sha01] defines the architecture of the J2EE platform supporting distributed, multi-tiered enterprise applications. A distributed, multi-tiered enterprise

[1]Authentication, authorization and data integrity are further described in section 2.4.1.

application is an application whose application logic is divided into components according to function. The various application components are installed on different machines depending on the tier in the multi-tiered environment to which the application component belongs. [Bod+02]. Figure 2.1 shows the overview architecture diagram of the J2EE platform. A description of the participating components is provided in the following sections.

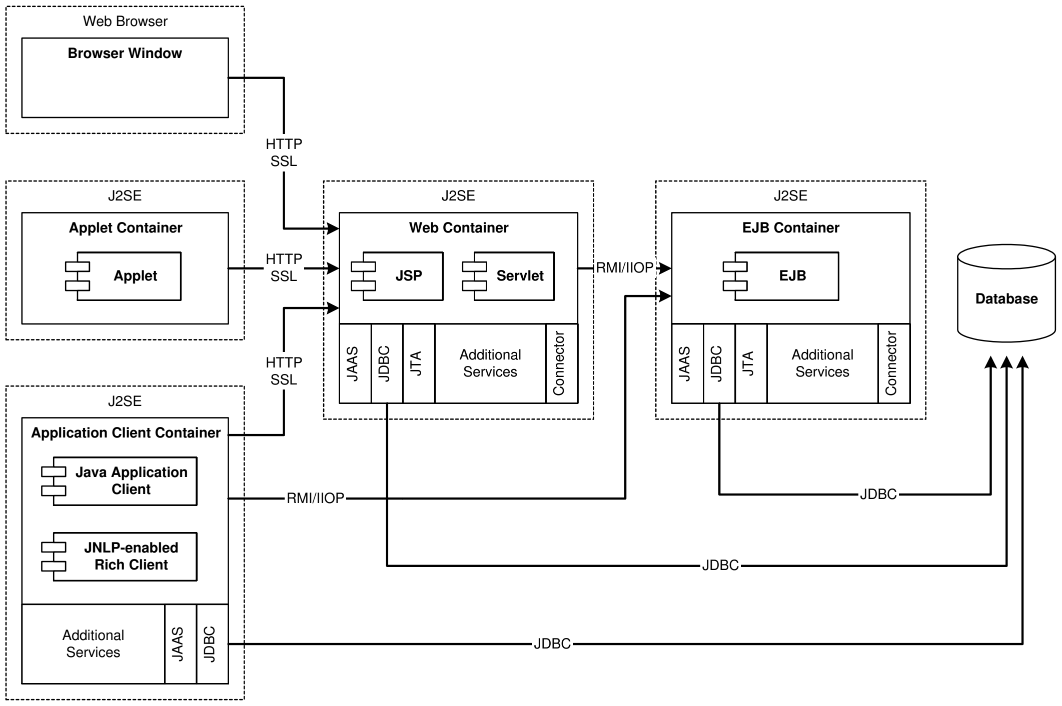


Figure 2.1: *Overview of the J2EE platform architecture diagram according to [Sha01].*

2.3.1 Tiers

A multi-tiered enterprise application may consist of a combination of the following tiers:

Client Tier The client tier contains components to present the graphical user interface of the enterprise application and to manage user interaction and event handling. The client tier resides on the client machine.

Web Tier The web tier contains web components responsible to handle parts of the presentation logic and to provide central application services like user authentication. The web tier resides on a server machine running a J2EE-compliant application server.

Business Tier The business tier contains components implementing the business logic and representing business data. The business tier also resides on a server machine running a J2EE-compliant application server. Depending on performance requirements, the business tier may or may not reside on the same physical server machine as the web tier.

Enterprise Information System Tier (EIS Tier) The enterprise information system tier is responsible for integrating an enterprise application with existing legacy back-end systems and databases. This tier often resides on one or multiple server machines distinct from the one hosting the web and business tier.

Depending on the actual enterprise application architecture, several tiers may not be included. Figure 2.2 illustrates a possible combination of tiers and their typical allocation on physical machines, and highlights J2EE application components residing in each tier.

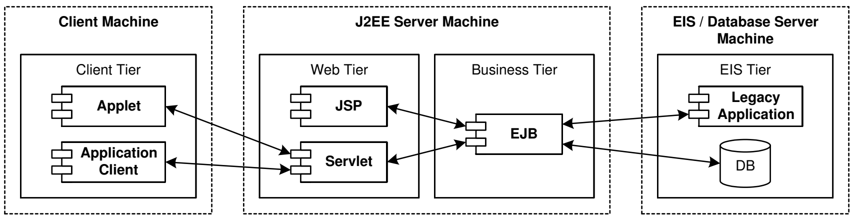


Figure 2.2: *Tiers of a multi-tiered enterprise application according to [Bod+02].*

Besides the concept of tiers, the J2EE platform architecture consists of multiple containers and application components. Inter alia, containers and application components are described in [Sin+02].

2.3.2 Containers

A container is a J2EE runtime environment, supporting a subset of the specified J2EE-related APIs for application components hosted within the container. Each container only hosts one or several specific application components and provides standard and container-specific J2EE services to contained application components. The following containers are part of J2EE:

Enterprise JavaBean Container An Enterprise JavaBean (EJB) container hosts EJB components and manages their execution and life cycle. An EJB container also provides additional services like transaction control, persistence management and security services to EJBs. An EJB container is server-side, running on a J2EE-enabled server machine.

Web Container A web container provides network services for request-response based interaction based on the HTTP(S) protocol. A web container manages the life cycle of contained web components and dispatches service requests from connecting clients to the requested web component. It also provides standard interfaces to context data like session state or request properties and additional security services like authentication and authorization. Web containers are server-side containers, running on a J2EE-enabled server machine.

Application Client Container An application client container manages the execution of Java application clients, a specific client component, and consists of a Java runtime environment. Application client containers are client-side containers running on the client machine.

Applet Container An applet container manages the execution of Java applets, a specific client component, and consists of a Java applet plug-in integrated into a web browser. Applet containers are client-side containers.

Besides the container-specific services, each container provides access to other J2EE services and communication APIs, including access to databases, custom transaction management, messaging services and legacy system connectors. A comprehensive list of services and APIs can be found in [Sha01] and its related specifications.

As enterprise applications does not need to use every container defined by the specification, a combination of client-side and server-side containers leads to the different architectures described in section 2.5.

2.3.3 Application Components

An application component is a single building block of an enterprise application that lives in a specific container. Each component may only exist in one specific container. The following application components are defined by the J2EE specification:

2.3.3.1 Client Components

Client components run on the client machine in a J2EE-capable runtime environment[2] that is either an application client container or an applet container.

Java Application Client A Java application client is a user interface program that may directly interact with an EJB component using RMI-IIOP[3] or a server vendor-specific protocol and has access to a number of J2EE services including JNDI[4] or JDBC[5]. Java application clients may also communicate with a web component using the HTTP(S) protocol or with a JDBC-aware database management system using JDBC.

[2] A J2EE-capable runtime environment may either be realized using the Java 2 Enterprise Edition environment or the Java 2 Standard Edition environment provided by Sun Microsystems Inc. extended by J2EE-specific libraries that implement a subset of the J2EE specification.

[3] RMI-IIOP is the default communication protocol for accessing EJB components within an EJB container. All J2EE-compliant application servers must support RMI-IIOP but may also offer vendor-specific communication protocols.

[4] Java Naming and Directory Interface (JNDI) is a standard extension to the Java platform, providing Java technology-enabled applications with a unified interface to multiple naming and directory services in the enterprise [Sun99].

[5] Java Database Connectivity (JDBC) provides programmatic access to relational data from the Java programming language [Ell$^+$01].

Applet An applet is a Java-based client component with, by default, restricted access to system resources on the client machine, such as the file system or the network communication infrastructure. Although it may get executed locally, an applet is often downloaded from the network and executed in a restricted runtime environment referred to as a sandbox. An applet is typically included in a web browser with a Java plug-in installed. Applets can communicate with web components on the server side using the HTTP(S) protocol. By signing the applet code with a digital certificate, the applet may leave its sandbox and may be granted access to system resources. A signed applet may also communicate with server-side components using RMI-IIOP or JDBC.

JNLP-enabled Rich Client A JNLP[6]-enabled rich client is a standalone client component based on a full-featured, rich graphical user interface that is made available through JNLP for download from a J2EE server and runs directly in a Java runtime environment on the client. Similar to an applet, a JNLP-featured rich client is per default executed in a restricted environment, but may get access to system resources by signing the client code using a digital certificate. Thereon, a JNLP-featured rich client may use different communication protocols besides HTTP(S).

Web browsers displaying a J2EE web application with a HTML front end are not considered to be a J2EE application component in the narrower sense, since no Java runtime environment is required on the client machine and no J2EE client component is involved. Nevertheless, browsers are widely used together with J2EE web applications on the client side.

2.3.3.2 Web Components

Web components run inside a web container and provide responses to client HTTP requests. Within the J2EE context, web components are typically used to manage the presentation logic, to generate a web browser-based user interface and to provide security related services for web-based applications. The J2EE platform includes specifications for the following web component technologies:

Servlet A Servlet is a Java-based web component that runs within a web container and generates dynamic content. Servlets may also implement parts of the business logic. Servlets are able to communication over HTTP(S) with a client component. [Cow01]

JavaServer Page A JavaServer Page (JSP) is a text-based web component to create dynamic content. In contrast to Servlets, the JSP technology provides a more natural approach to creating static content and allows the proper detachment of application logic from the presentation-related code. JSPs are built on the Servlet technology. [Pel53]

[6]The Java Network Launching Protocol (JNLP) is specified in [Sch01] and summarized in section 2.7.

2.3.3.3 Enterprise JavaBean Components

Enterprise JavaBeans, also referred to as EJBs or Enterprise Beans, are server-side components implementing application business logic or representing business data objects. EJBs run within an EJB container and thus provide, in cooperation with the container, additional services like transaction control, concurrency control, multi-user security, and persistence. The EJB specification [Dem$^+$01], part of the J2EE specification, describes three types of EJBs:

Session Bean A Session Bean typically contains business logic and is executed on behalf of a single client. Session Beans do not directly represent application data, but may access and manipulate such data. Session Beans exist in two variants: stateless and stateful. Stateless Session Beans are used for relatively short-lived operations where no session state is required. In contrast, stateful Session Beans are able to retain session state between subsequent calls of Session Bean methods.

Entity Bean An Entity Bean represents one or multiple business objects that are required to be held in a persistent manner. Persistence can be realized using the persistence mechanism of the EJB container (Container Managed Persistence, CMP) or using an object-specific mechanism implemented by the Entity Bean (Bean Managed Persistence, BMP). Entity Beans are shared between multiple users and are long-lived, surviving an EJB container crash.

Message-Driven Bean A Message-drive Bean is conceptually similar to a Session Bean, but is only executed upon receipt of a single client message. Message-driven Beans are stateless and are invoked asynchronously.

Enterprise Beans may offer their functionality to remote clients (either other EJB components, web components or client components) using remote interfaces. This technology allows the remote caller to use the same API as for local calls.

2.4 J2EE Security

The J2EE specification describes the J2EE security architecture in respect to the security requirements of enterprise applications. In its fullest details, this security architecture is specified and discussed in [Sha01]. This section limits itself to the vital aspects regarding security requirements for offline operation.

2.4.1 Security Concepts

Inter alia, [Sha01] mentions the following security concepts relevant in the context of enterprise applications:

Authentication is the process of verifying an identity claimed by or for a system entity [Shi00]. This entity may be a user, an internal application component or an external application. Authentication is achieved by requesting a credential that proves the identity of the entity. A credential may, for example, consist of a username and a password, a fingerprint or a digital certificate.

Authorization is the process of granting privileges to access particular protected resources [Shi00]. Authorization is typically based on authentication since, during authorization, an authenticated entity is associated with a set of privileges based on the identity provided that are later on compared to a set of required privileges in order to access a particular resource. The appropriate use of authorization within an enterprise application is crucial to ensure data confidentiality.

Data Integrity is the property that information has not been modified or destroyed in an unauthorized manner [Shi00]. Data integrity may be achieved by accompanying a data object with a checksum representing a snapshot of the current state of the data object. This checksum has to be fragile in a manner that changes in the state of a data object result in different checksum. An example is a digital signature that signs an object using a digital certificate. Data integrity becomes important when transferring data across networks and when reading data from a persistent store.

In enterprise applications, these security aspects are typically covered by the combination of infrastructure features, application-specific security implementations and organizational measures.

2.4.2 Authentication Model

Depending on the security level configured for a particular enterprise application and its components, a connecting client requesting access to a particular protected resource may be urged to authenticate itself with the resource or its enclosing container, respectively. A client may either be a human user, a J2EE component within the same or a different container or an external application. Authentication of a caller is a condition precedent to authentication. As J2EE containers only request authentication when it is first required, the concept of authentication is referred to as lazy authentication.

Since its specification version 1.3, J2EE includes support for the Java Authentication and Authorization Service (JAAS)[7] that, inter alia, specifies a common, flexible mechanism for authenticating and authorizing users. Based on JAAS, an enterprise application may individually configure a specific authentication procedure and the authentication data source using pluggable authentication modules.

2.4.3 Role-based Authorization Model

As the fundamental security model for authorization, J2EE specifies a role-based authorization model. A security role is a logical grouping of users indicating that the users in

[7]JAAS is further described in section 2.6 and in [Lai^{+}99].

that specific security role have a particular attribute in common. The assignment of users to specific security roles is done by an administrative instance, either a security system or an application administrator. A single user may participate in multiple security roles. Security services implemented by the web and EJB container use the concept of security roles in order to authorize a user requesting access to a specific resource. Each resource to be protected has one or multiple security roles specified that a requesting user must be assigned to in order to get access to this resource.

J2EE provides two mechanisms for the declaration of security requirements for resources:

Declarative Security The approach for declarative security allows the configuration of security requirements to access data objects or call methods in an artefact separated from the enterprise application code, referred to as deployment descriptor. Therein, the security requirements for each entity, for example an EJB or a web component, may be described using a J2EE-defined syntax. The respective container hosting the protected resource enforces the security constraints defined in the deployment descriptor.

Programmatic Security Security requirements may also be directly programmed into a particular entity to be protected. For this purpose, J2EE defines constructs to get information about the request initiator and the security roles assigned to it. Whereas hard-coded security requirements are not as easy to maintain as separately declared security requirements, programmatic security allows finer grained access control based on application-specific rules and object states.

These two approaches for declaring security requirements are not mutually exclusive but may be combined in order to achieve the level of security required for a particular enterprise application scenario.

2.5 J2EE Architectures

The various characteristics of enterprise applications related to functionality and design result in different architectural requirements. As a general platform for enterprise applications, J2EE does not offer a single standard architecture but a variety of components and containers able to integrate with each other. These components can be combined into several application architectures depending on the needs of a particular enterprise application.

2.5.1 Web-centric vs. EJB-centric Architectures

In most architectural scenarios, the server side may contain both a web container and an EJB container. Two different approaches exist regarding the allocation of business logic and data between the two containers [Sin+02]:

Web-centric Approach In a web-centric scenario, the web container is responsible for the complete application logic and data management. The web container handles content generation, presentation and user request handling. It also implements the core application logic, enforces business rules and manages business objects. The web container must also manage transactional requirements and connection pooling for persistent data access. A web-centric approach may be useful for small-sized applications with few transactional requirements but may easily lead to complex, monolithic applications. Whereas a strict web-centric architecture abandons an EJB container, a lesser web-centric approach may use EJB components for encapsulating persistent data objects as Entity Beans.

EJB-centric Approach In an EJB-centric application scenario, the core application logic and business object management are implemented in the EJB container. Business logic is implemented using Session Beans and persistent data is encapsulated in Entity Beans. Web components communicate with EJB components instead of directly accessing data resources in the database management system. Since the EJB container offers multiple container-specific services such as transaction control and security enforcement, the enterprise application itself is not required to attend to these points. An EJB-centric architecture also eases the reuse of application logic and data from multiple client components, for example a Java application client communicating directly with the EJB container and a web application client sending requests to the web container. Within an EJB-centric approach, a web container may be used to delegate HTTP(S) client requests to EJB components implementing the required business logic and thus integrates the HTTP(S) protocol with the EJB technology.

2.5.2 Architecture for Fat Client Applications

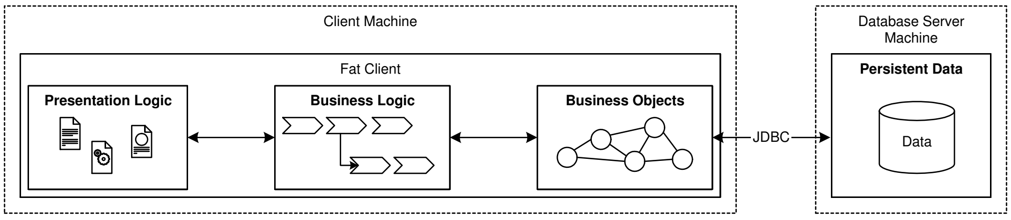


Figure 2.3: *Architecture overview for a fat client application.*

A fat client application consists of a heavyweight application client, completely handling presentation, event handling, application logic and business object management. The client component is either realized as a Java application client, a JNLP-enabled application client or, differing from the J2EE architecture blueprint[8], as an applet client. The server part of a fat client application typically consists of a JDBC-aware database

[8]The J2EE architecture blueprint does not explicitly mention JDBC and other service support for applet clients. Nevertheless, if an applet client is signed by a digital certificate, it may gain access to JDBC and other services.

management system, but does not offer a J2EE runtime environment. The fat client communicates with the data store on the server machine using the JDBC protocol and is responsible for reading persisted object states from and writing them to the database. This type of architecture is the most client-centric one, since the complete functionality except data persistence is situated on the client machine, but still represents a client-server based architecture.

Figure 2.3 highlights the building blocks of a fat client application and their allocation on the client and server machine. Although this enterprise application architecture does not rely on and does not benefit from the J2EE server-side components and containers, it represents a legitimate architecture variant within the J2EE architecture blueprint.

2.5.3 Architecture for Rich Client Applications

A rich client application is an enterprise application using either a Java application client, a JNLP-enabled rich client or an applet client as J2EE client component on the client machine. Figure 2.4 shows the schematic architecture of a rich client application.

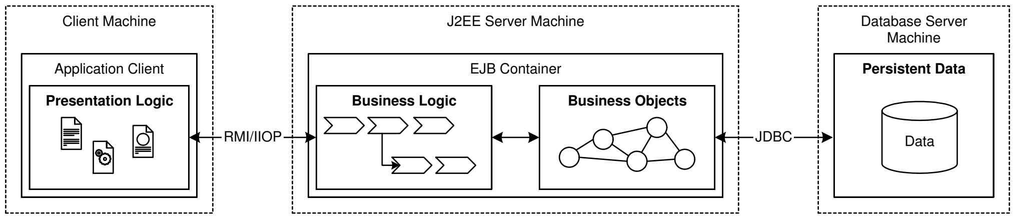


Figure 2.4: *Architecture for a rich client application.*

The client component contains the presentation logic used to display a rich graphical user interface and to manage user interaction on the client side. The server part of the enterprise application consists of the business logic and the business object management in the form of Enterprise JavaBean components. These EJB components are hosted within an EJB container on the server machine running a J2EE-compliant application server. The business logic part is typically implemented as a collection of Session Beans, either stateful or stateless, that works on the business objects implemented as Entity Beans. Message-driven Beans may be part of the business logic as well. The Entity Beans are made persistent using a database management system that may or may not reside on the same physical machine as the J2EE application server.

The application client communicates with the EJB components on the server side using RMI-IIOP or a vendor-specific communication protocol. If the application client has to use HTTP(S) as communication protocol, the server side of the enterprise application has to be extended with an appropriate Servlet component residing in a web container, that is able, at least, to listen to application client HTTP(S) requests and to propagate them to the corresponding Session Bean implementing the part of the business logic to be activated. Communication over HTTP(S) may, for example, be required because of firewall-related issues. Most firewall configurations allow communication using HTTP(S) between the client and the server machine. Introducing a web container may also be useful if the enterprise application has to support multiple client platforms and devices, such as notebooks, personal digital assistants or mobile phones, as the HTTP(S) protocol is often an established common denominator. The actual allocation of business logic between the web and EJB tier is application-specific and depends on whether a more web-centric or more EJB-centric approach is used.

For rich client architectures, authentication data is requested from the user by the application client, either on application startup or on-demand using lazy authentication. The authentication data gathered is typically transferred to the server in order to verify the credentials. Authorization occurs on the server side, at every access request to a protected resource.

2.5.4 Architecture for Web Applications

A web application consists mainly of server-side parts and uses a generic presentation engine, referred to as web browser, to display the graphical user interface on the client machine. The server part of the enterprise application contains all the presentation logic, the business logic implementing the business services, the business objects and their persistence system. As defined by J2EE, a web application architecture requires a web container, an EJB container and a database management system. The web container hosts multiple JSP and Servlet components that implement the presentation logic. Additionally, web components may be used to implement global services such as request filtering or authentication. Behind the web components, several EJB components implement the business logic and represent business objects. Figure 2.5 illustrates the architecture for web applications with the participating J2EE containers and components on the server side. A completely web-centric architecture for web applications does not include an EJB container and manages business logic and objects within the web container. Additionally, the database server machine may be replaced by an alternative mechanism for persisting business objects.

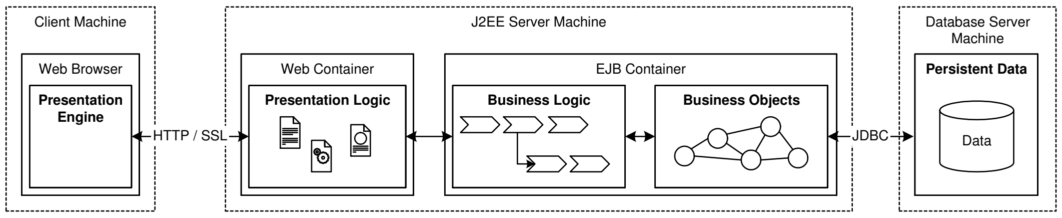


Figure 2.5: *Architecture for web applications.*

A web application uses a request-response-model for communication between the client and the server side. The web browser requests services on the server side using the HTTP(S) protocol and receives an appropriate response in form of HTML[9], XML[10] or

[9]The Hypertext Markup Language (HTML) is the standard technology for publishing hypertext on the World Wide Web [HWG03].

[10]The Extensible Markup Language (XML) is a markup language for structuring and exchanging arbitrary data [Bra+00].

in another markup language. The web browser parses the response and renders, based on the response information, the graphical user interface. The web browser also captures user inputs and interaction events and transfers them, by creating a new request, to the server machine.

2.6 Java Authentication and Authorization Service

The Java Authentication and Authorization Service (JAAS) described in [Lai+99] extends the standard Java 2 security architecture specified in [Gon98] by a pluggable authentication mechanism and subject-based authorization. JAAS is an integrated part of the J2EE specification since version 1.3 and must be supported by J2EE-compliant application servers. While the authorization extension provides the possibility to control access to protected resources based on the caller's identity and its privileges, the pluggable authentication mechanisms allows applications to remain independent from the underlying authentication technologies actually used to authenticate a specific user. Each application relies on a JAAS configuration file describing which login modules have to be called in order to authenticate a user. During the authentication process, the JAAS login context subsequently requests each login module specified to authenticate the user, for example based on username and password combinations or by reading the users fingerprint from a biometric device. The authentication method is completely encapsulated within the particular login module. The JAAS configuration also determines the rules for evaluating the overall success or failure of the authentication attempt based on the individual results of each login module. Each login module may associate the authenticating user, represented by the JAAS Subject class, with one or multiple principals. A principal is similar to a security role described in section 2.4.3. The associated set of principals may be used for further authorization. Figure 2.6 illustrates the core elements of JAAS participating in the processing of an authentication request.

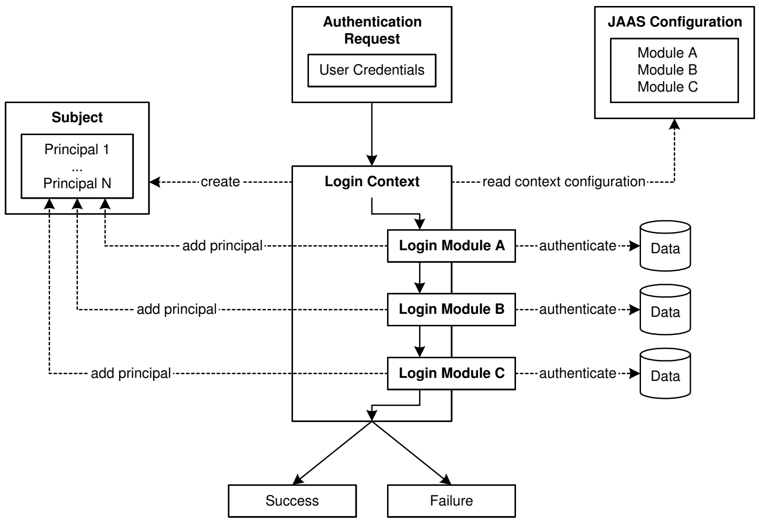


Figure 2.6: *Schematic processing of an authentication request using JAAS.*

2.7 Java Network and Launching Protocol

The Java Network and Launching Protocol (JNLP) specified in [Sch01] is a network deployment and launching protocol conceptually allowing the local execution of an application whose code base resides on the network. JNLP specifies a restricted execution environment for running applications downloaded from the network. A configuration file, referred to as JNLP file, contains the location of all application components packed as JAR[11] files and made available on a web server accessible through the HTTP(S) protocol. Before executing a JNLP-featured application, the JAR files specified in the JNLP file are downloaded and cached locally. Thereon, the application is started within the JNLP runtime environment. JNLP also supports the incremental download of updated application components and offers several services to access the surrounding environment, such as storing data and libraries locally or launching the web browser. JNLP specifies offline support for downloaded applications, meaning that already downloaded applications may get executed relying on the locally cached application components if no network connection to the location of the code base specified in the JNLP file is available. The JNLP-specified offline support is not equal to offline operation of an enterprise application as no application-specific, offline-related issues are respected.

[11]The Java Archive format is a platform-independent format that aggregates multiple files into one and that supports file compression. In addition, a JAR file can be digitally signed using a certificate in order to authenticate its origin.

Section 3

Use Cases for Offline Operation

This section shows a number of use cases where offline operation of an enterprise application may be reasonable. Each use case highlights different aspects of offline operation that are, besides others, discussed in the upcoming sections.

3.1 Company Information System

Consider an information system in a company's intranet containing internal information such as employee's guide, internal phone book or expense regulations. This information system is accessible for each employee logged into the intranet. Furthermore, employees are able to download information they regularly access onto their notebook for offline access when disconnected from the intranet, for example when travelling or for work at home. Since the amount of data in a company-wide information system can potentially be huge, an adequate mechanism for specifying which parts of the information system should be made available in offline operation is required.

Most employees have read-only access to data. This means that these employees are not allowed to add, modify or remove data that is part of the information system. Other employees in the role of content editors may be responsible for updating specific parts of the information system. Since the data stored locally on the employee's notebook may become out-of-date, a regularly refresh of this data is required. When an employee logs into the company's intranet, the locally stored version is compared to the information system server's content and modifications are synchronized between the two data stores. In this use case, conflicts will rarely occur under the presumption that for a specific content, only one content editor is defined at a time. As each element of data is associated with one specific author able to modify it, data is herein referred to as partitioned.

Offline-related aspects covered by this use case scenario are:

- local storage of selective application data
- refreshing locally stored data with updated data from the server storage
- transition to offline mode on demand

3.2 Expense Management

Companies often use an electronic tool to manage an employee's expenses like customer calls, seminars or business trips in order to refund the correct amount with the next wage payment. Basically, employees add new expense records to the expense management tool whereon their manager has to revise and approve the expense claim. Managers may also have access to summaries, reports and charts on the expenses of their subordinates. It may be interesting for travelling employees to be able to record expenses as they accrue during the journey, for example back in the hotel in the evening. Managers may want to use their time between customer meetings or during flights to revise and approve expense records created by their subordinates. At this time, a connection to the company's network may not be possible. Therefore such an application needs to be offline-capable and provide selective application features even when disconnected from the network.

While connected to the company's network, an employee or manager may decide to take expense records and employee data offline in order to work on that data when disconnected. During the offline work, existing data is modified and new expense records are expected to be created. On the next reconnect to the network, the modifications are propagated to the server's data store whereon potential conflicts are reconciled. Conflicts are possible since both an employee and its manager are able to modify the same expense record, for example if the employee increases the expense amount and the manager approves the previous stored expense record with the lower amount. Also, certain functionalities may not be useful in offline operation such as generating reports where a large amount of data is required.

Additionally to the aspects already mentioned, this scenario highlights the following aspects concerning offline operation of an enterprise application:

- synchronization of locally stored data with the application data on the central server
- potential synchronization conflicts resulting from concurrent modifications
- reduced set of application functionality in offline operation due to performance reasons

3.3 High Availability Solution for Front End Systems

Ticket agencies often use front end systems to sell tickets for stage performances, cinema shows and concerts. Customers may call the ticket agency and order several tickets for one of the events advertised by the agency. The ticket agency books, prints and ships the tickets to the requesting customer. Customers may pay their tickets by credit card or invoice. The front end system for ticket ordering relies on a central application server that persists events, ticket bookings, and customer addresses and manages credit card transactions. The application server must be dimensioned for high loads since newly advertised events may result in a run on tickets. A service interruption caused by a server failure or a communication problem between the server and the front end system potentially leads to a blackout of the ticket ordering system, to unpleased customers and financial loss.

Providing the front end system with an offline-capable application may bypass or ease such critical situations resulting from system failures. Each client machine that runs the front end system holds information on advertised events, parts of the customer data and a particular contingent of free tickets for each event in a local storage. If the server fails or is not available because of communication problems, the front end system uses the locally stored data to satisfy customer requests. As soon as the application server is available again, the transactions completed in offline operation are propagated to the server and reflected in the central data storage on the server machine. Since such service interruptions are not predictable in most instances, the front end system always has to be prepared for a transition to offline operation and thus, regularly has to synchronize modified data between the client and the server machine. In addition, the offline-capable application has to make sure that no ticket is sold twice and therefore, each client machine may store an exclusive set of tickets that are up for sale during offline operation. Due to the lack of coordination between multiple client machines in offline operation, a particular client may run out of tickets while others still may have free tickets for sale. Security constraints may prohibit to sell tickets by credit card in offline operation since the server is required to check the credit card validity. Application functionality that is less critical regarding calling customers, such as adding new events to the ticket ordering system or creating statistics on ticket sales, may be provided in online operation only.

Additional offline-related aspects resulting from this scenario are:

- transparent management of the operation mode
- switch to offline operation based on unexpected events and failures
- reduced set of application functionality in offline operation due to security and business constraints
- offline operation for a limited period of time during server problems

Section 4

Offline Operation

After specifying the term ’offline operation’, this section discusses the key aspects of operating an enterprise application both in online and offline mode.

Offline operation describes the operation of an enterprise application disconnected from one or more application servers running on other physical machines. This thesis assumes that an application in offline mode runs completely on one physical machine, referred to as the client machine, and without usage of the networking infrastructure outside the client machine. Additionally, this work presumes that an enterprise application capable of running in offline mode also supports an online operation mode. Online operation is the operation mode of an enterprise application where the application client is permanently connected to one or more central server machines. These servers are typically responsible for implementing the largest part of the application logic and data while the application client may manage presentation logic and user input control. The term ’offline mode’ describes the execution mode of an enterprise application that runs in offline operation. Equivalently, ’online mode’ describes the mode an enterprise application runs in during online operation.

4.1 Operation Sequence

Enterprise applications capable of offline operation typically do not run permanently in offline mode. In fact, an enterprise application runs largely in online mode when a connection to the application server and the application server itself is available, and switches to offline mode only if either the server or the connection is unavailable. This leads to the typical sequence of operation that contains phases of online operation, phases of offline operation and phases of switching between both modes, as shown in figure 4.1.

Consider an employee working in the office with the company-wide calendar application running in online mode, where modifications to appointments are immediately applied to the calendar data store on the server. Before leaving the office, the employee decides to synchronize parts of the calendar application and appointment data onto the notebook

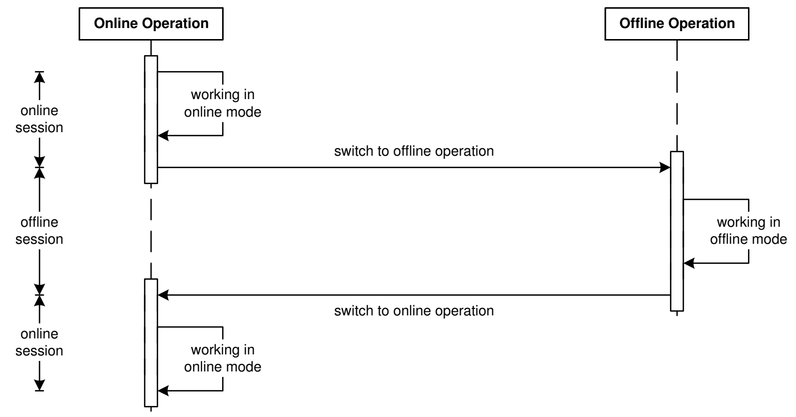


Figure 4.1: *A typical operation sequence for online and offline operation.*

in order to continue working at home where no connectivity to the company network is available. At home, working in offline operation, the employee creates new appointments, and moves and deletes others. The next morning, back at the office and connected to the network, the employee synchronizes the changes done to the local calendar on the notebook in offline operation with the calendar data on the application server before continuing working in online mode.

The term 'offline session' describes the time an enterprise application is running in offline operation. An offline session starts, when the enterprise application switches from online to offline operation or when it is initially started in offline operation, and ends when the enterprise application switches back to online operation the next time. During an offline session, the offline application client may be terminated and later restarted again. Similarly, 'online session' denotes the time an application runs in online operation.

4.2 Operation Mode Transition

An operation mode transition describes a switch from online operation to offline operation or vice versa. During this action, several tasks have to be accomplished in order to prepare the enterprise application for the new operation mode. These tasks depend on the direction of the transition.

During the operation mode transition, user interaction may be required in order to specify which data has to be downloaded for offline operation or to resolve conflicting data modifications. This is highly application-specific. It is possible to design an enterprise application where either no user interaction is required at all and the current operation mode is completely transparent to the user, or the follow a scenario where all decisions related to the operation mode transition are delegated to the user. Also, the point of time, at which the transition to a different operation mode is initiated, may differ from one scenario to another.

4.2.1 Transition to Offline Operation

When switching from online operation to offline operation, the following general tasks must at least be included in the procedure:

- download and install application functionality and potential infrastructure prerequisites for offline operation, as needed
- download the required application data to a local data store[1] on the client machine
- disconnect from the server and start the offline application client, either by putting the running application client into its offline mode or by starting a dedicated offline application client
- authenticate the user for offline operation, if required

Depending on the enterprise application design and architecture, several steps may be simplified since the offline application client may not differ from the online application client and therefore already resides on the client machine. But at least the source for application data has to be switched to a local data store since the server as host for central application data will not be available in offline operation. After changing to offline operation, the application needs to be able to run autarkic on the client machine.

4.2.2 Transition to Online Operation

When switching from offline operation back to online operation, following steps have do be performed:

- start the online application client, either by putting the running application client into its online mode or by starting a dedicated online application client, and reconnect to the server
- authenticate the user for online operation
- synchronize data modified or created in offline operation with the data on the server and reconcile potential conflicts resulting from concurrent data modifications

In turn, if the application client is identical for both online and offline operation and is not required to be terminated and restarted again during the operation mode transition, the sequence of switching back to online operation may happen more silently from the user's point of view. Generally, it may be useful to initiate a synchronization process when switching from offline to online operation in order to ensure data consistency between the local and server data store.

[1]A local data store is a persistent store for application data objects that resides on the client machine. The local data store contains application data objects required for offline operation.

4.3 Application Configuration

Besides the typical operation sequence described in 4.1, an enterprise application may run in different application configurations regarding its offline operation handling. Comparing two of the use cases described in section 3, use case 'Company Information System' and use case 'High Availability Solution for Front End Systems', the requirements for the offline operation are different: in the information system scenario, there is a predefined, predictable transition between the offline and the online mode that allows the system to undertake the necessary steps to prepare the offline operation on demand, whereas in the failover scenario, the system has to be prepared for an unforseen switch caused by a system or network failure. Within this thesis, the following two application configurations are distinguished:

4.3.1 'On-Demand' Configuration

The on-demand configuration describes a mode where an enterprise application changes to offline operation only on clearly specified actions typically triggered by the user. If one of these actions is triggered, the application undertakes the necessary steps to switch to offline operation. It is assumed that there is enough time to perform these tasks just after initiation of the operation mode switch and that the server machine is available for synchronization during the transition sequence. During the actual transition to the offline mode, the application may be blocked, disallowing the user to perform other tasks or the transition tasks may be performed transparently in the background. It is also possible to start a dedicated offline application client when switching to offline operation.

The on-demand configuration maximally divides the online operation from the offline operation since during online operation, no special offline-related considerations are required. As all offline- and transition-related tasks are performed at user request, no preparation concerning offline availability of application logic and data during online operation is necessary. On the other hand, the on-demand configuration does not allow the enterprise application to handle unexpected interruptions of the server-side application services or communication infrastructure.

4.3.2 'Be-Prepared' Configuration

The be-prepared configuration is used in a scenario where the transition between online and offline operation is triggered by an action of which both the user and the application are unaware or which is not predictable, for example an underlying system failure or communication problem. Therefore, the application has to be permanently in a state where a transition to offline operation is possible without major interruption to the service. In order to achieve this, an enterprise application has to take appropriate preparations during normal online operation. All data required for offline operation must be synchronized periodically with the local data store on the client machine. Additionally, the be-prepared configuration requires the application to perform these preparations transparently for the user, without significant impact on the normal operation. Enterprise applications running with the be-prepared configuration may not offer the possibility to switch to offline operation on demand since this configuration may be used to hide all operation mode specific issues from the application user. If the transition between online and offline operation has to be totally transparent to the user, the application client has to be reused for online and offline operation. This prohibits the start of a dedicated offline application client for offline operation.

The be-prepared configuration requires the online operation to be engaged in offline-related issues like offline data availability or synchronization in order to enable a transparent, unexpected transition to offline operation. This interlocks the online and offline operation and complicates a clear isolation of the two operation modes.

4.4 Local Data Management

Since an enterprise application running in offline operation has no access to the data store residing on the server machine, a data store on the client machine is required. This local data store contains all persistent application data accessed, created or modified by the enterprise application during offline operation. Besides data synchronization issues described in section 4.6, the following two aspects are important when storing data locally on a client machine.

4.4.1 Local Data Store Implementation

The implementation of the local data store defines how data is actually stored and held persistent on the client machine. In comparison to server environments where enterprise application data is almost solely made persistent in large database systems built for scalability and performance, the realization of local data stores on a client machine may be simplified. Since the amount of data stored locally should be minimized and is potentially small related to the amount of application data stored on the server and the number of concurrent accesses is potentially low, the local data store may be implemented based on files or using a lightweight database system. In specialized scenarios, an offline operation without local data persistence is conceivable, if all data is transiently held in memory while working in offline operation and not persisted until the application switches back to online operation. Although this prohibits the offline application client from being terminated and restarted during an offline session, a number of security-related issues with locally stored data may be avoided.

4.4.2 Security Aspects of Local Data Management

As described in section 2.2, data of enterprise applications is often confidential and requires special protection against unauthorized access and modification. This is especially critical when storing data locally on a client machine in order to enable offline operation. Data stored on the client machine is no longer controlled by a potentially well-secured server machine that ruthlessly enforces company- and application-specific security constraints. Not only may it be possible for users with corresponding system privileges to access or modify data outside of the application, the client machine may also get compromised by an internal or external attacker. The relevance of securing the local data store is increased by the fact that offline operation is typically used to run an application when the user is travelling around and thus, the client machine is mostly a portable device such as a notebook or personal digital assistant, that may get physically lost.

The security level claimed for a specific enterprise application may require the application data to be encrypted prior to be persisted in the local data store. In addition, assuring integrity of locally stored application data may require the use of digital signatures. Both approaches, encryption and digital signatures, require an appropriate certificate and key management infrastructure[2] and may lead to further issues related to distribution and storage of private keys. In other scenarios, the only acceptable solution may be to completely avoid persisting data on the client machine using a transient data store. Security of locally stored application data may depend on security measures that are beyond the scope of a particular offline-capable enterprise application, such as restricting access to the client machine for authorized users only. The level of security required significantly influences the complexity and measures of storing data locally and may restrict or even prohibit offline operation.

4.5 Offline Data

The actual amount of application data required to be available in offline mode is highly application-specific and may also be user-specific. Depending on the application functionality provided in offline operation, more or less application data has to be stored locally. One may also consider a scenario where a user is able to configure which data has to be accessible in offline operation, for example in a calendar application where each user may choose which appointments within which time interval have to be downloaded to and stored in the local data store. Additionally, a particular user may only have access to a subset of data stored in the server data store, based on the privileges associated with that user. Also, the amount of data within the server data store may be huge and may exceed the client machine's storage space. This leads to the insight that a local data store is, in general, not a simple mirror copy of the central server data store but includes only selective parts of it. An appropriate mechanism to specify collections of offline-available data is required. The following two approaches are possible:

Explicit Collection The data to be made available for offline operation may be explicitly enumerated in a list containing the unique identifiers of the specific data objects. This collection may be specified globally for all users or may be stored

[2] [Ken+03] describes a public key infrastructure (PKI) that supports certificates for encryption and digital signatures.

together with the users profile. A combination of both is possible. The explicit collection approach seems reasonable if a relatively small number of data objects are required to be available offline and if this collection is highly static. Explicit collections maintain an explicit reference to each object contained and thus are aware of their content.

Implicit Collection If the number of offline-required data objects is potentially large or the collection is highly dynamic, an implicit collection may be more adequate. Implicit collection denotes the description of objects to be stored offline by means of specific attributes or conditions. Each data object in the server data store having these attributes or fulfilling the conditions specified belongs to the implicit collection. The contents of a specific implicit collection depends on the current state of the server data store. Some data objects included in the collection at one moment of time may be excluded from the collection after they have been modified and thus, no longer have all required attributes or fulfill the conditions. Others may satisfy the conditions or get the required attributes after being modified. Implicit collections may not mandatorily maintain a reference to each object possessing an attribute or fulfilling the conditions, but may only define these restrictions. The contents of the explicit collection may be calculated lazily by checking the declared conditions against each object within the data store. This is similar to a filtering concept.

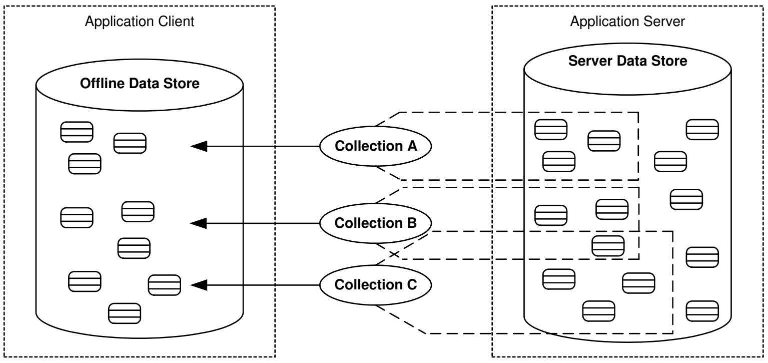


Figure 4.2: *Accumulation of data objects in collections.*

Figure 4.2 illustrates the concept of data collections specifying a subset of data stored on the server machine. Each collection 'Collection A', 'Collection B' and 'Collection C' contains multiple data objects from the server data store that have to be stored in the local data store on the client machine. The content of each collection may either be defined implicitly or explicitly. The content of two collections are not mutually exclusive.

4.6 Data Synchronization

In general, locally stored data may be modified by the enterprise application running in offline mode, even though in specific scenarios, locally stored data may be accessible read-only. During a single offline session, modifications of existing data objects and creation of new objects are only reflected on the local data store. Since only one application client instance accesses this local data store, conflicts are not expected to occur, whereas conflicts are possible when synchronizing the changes in the local data store with the central data store on the server. The process of harmonizing data residing on the client machine with the one stored on the server machine and merging data modifications into existing data is referred to as data synchronization or synchronization in short.

4.6.1 General Requirements

Regarding the usage of synchronization for reconciling application data between local data stores and the server data store, the following requirements may be postulated:

- The synchronization has to synchronize a local data store on a client machine with a central data store on the server machine. The central server data store is always one of the participating data stores in a synchronization process. There is no need for peer-to-peer synchronization between two local data stores on different client machines.
- The synchronization process must be able to detect conflicting modifications and must have a mechanism to reconcile these conflicts in order to get the two data stores in a consistent state.
- Potential conflicts that arise during the synchronization have to be resolved application-specific, and not based on a general reconciliation strategy. Each type of object may require an individual reconciliation strategy in order to maintain consistency and integrity.
- The synchronization process has to be efficient and optimized for short synchronization times and small data transfer amounts. Only required changes have to be exchanged between the two stores. This may inter alia be required to support the 'be-prepared'-configuration described in 4.3.2 where the synchronization process may be initiated frequently and in parallel to the normal application operation.
- The synchronization process has to allow the synchronization of only a subset of data objects, a specific data collection. This is referred to as selective synchronization.
- A failure occurring during the synchronization process should not corrupt the data consistency of one of the participating stores. A failed attempt of synchronization that is later restarted and completes successfully has to lead to the same state as if it had never failed.
- The synchronization mechanism has to be independent from a specific enterprise application architecture or communication protocol. It must be adaptable to a specific application scenario and to the communication infrastructure used.

- The synchronization of data must not conflict with a potentially existing application security policy[3]. Data synchronization between two stores should not allow a user to modify an object in a way that is prohibited by the application security policy.
- Creations of and modifications to business objects synchronized between two data stores have to be propagated to the corresponding synchronization partner in the exact same order as they occur on the local machine.

Depending on the specific enterprise application scenario, new requirements may arise as well as some of the requirements described above may become obsolete.

4.6.2 Basic Synchronization Process

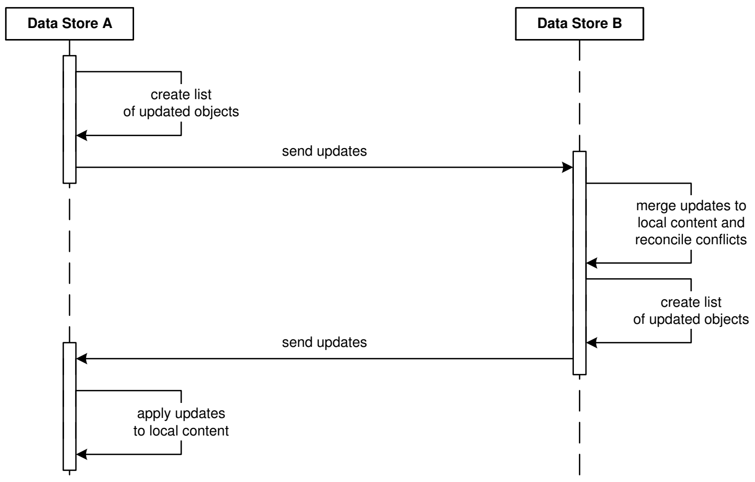

Data Store A
Data Store B
create list of updated objects
send updates
merge updates to local content and reconcile conflicts
create list of updated objects
send updates
apply updates to local content

Figure 4.3: *Protocol of a basic synchronization process.*

Among the wide range of existing synchronization protocols [Syn02], [Coh03], a sequence of process steps are commonly used. Figure 4.3 illustrates the protocol of basic synchronization process. During the synchronization of two data stores, store A and store B, store A sends all updates made to its content[4] to store B in order to merge these modifications with existing data and to apply them to the contents of store B. Conflicts are reconciled on the machine hosting store B. Thereon, store B sends all updates including the one resulting from the applied updates received from store A back to store A. From the point of view of store A, store B is referred to as replica data store and vice versa. A replica data store is the corresponding data store for a local data store on another machine, either a client or a server machine, with whom that local data store is synchronized. After a complete synchronization process, both store A and store B have the same state and are considered to be consistent or 'in sync'.

[3]A security policy is a set of rules and practices that specify or regulate how a system or organization provides security services to protect sensitive and critical system resources [Shi00]. An application security policy is a security policy valid for a specific application.

[4]The content of a data store is the sum of all currently contained data objects.

4.6.3 Object States

In order to be able to generate a list of created, modified or deleted objects that need to be synchronized with the synchronization replica, each data store has to keep track of data modifications and to maintain state information for each contained object. Independently from specific state tracking mechanisms, an object existing in a data store may be in one of the following states:

CLEAR The state 'CLEAR' describes an object that has not changed since the last synchronization with the replica data store.

ADDED The state 'ADDED' describes an object that has been newly added to the local data store and does not yet exist in the replica data store.

MODIFIED The state 'MODIFIED' describes an object that has been modified in the local data store since the last synchronization with the replica data store.

DELETED The state 'DELETED' describes an object that has been deleted in the local data store since the last synchronization with the replica data store.

Depending on the particular implementation of the tracking and synchronization mechanism, the state 'ADDED' may be reflected by the 'MODIFIED' state in order to reduce the number of distinct states and state combinations. Additionally, the specific mechanism used by a data store to track the state of each object may differ from one data store implementation to another. While one implementation may flag each object directly with the appropriate state whenever an action on an object is applied, another may use a history of actions applied to each object and derives the current state of a specific object from the update history. Also, the state information may be included directly in the data object or may be held externally using a separate mechanism.

4.6.4 Conflict Detection and Reconciliation

During the process of synchronizing two data stores, conflicts may occur if the same data object has been modified in both participating stores independently. Conflicts may be detected by comparison of the state information of the two objects. The resolution of a conflict is called reconciliation and is always performed on the data store receiving updates from another data store.

Figure 4.4 illustrates possible combinations of the states of two object versions originating from two different data stores that participate in a synchronization process. Figure 4.4 also describes the actions required on the replica data store depending on the object state combination during the phase of merging updates into the store content and highlights the state combinations resulting a conflict. Combinations shaded in gray may not occur under the presumption that the state of an objects is set to CLEAR after a successful synchronization with the equivalent version in the replica data store. Combinations

		Object State in Replica Data Store				
		CLEAR	ADDED	MODIFIED	DELETED	(not existing)
Object State in Local Data Store	CLEAR	ignored	(may not yet exist in local data store)	update in local data store	delete in local data store	(may not be CLEAR in local data store)
	ADDED	(may not yet exist in replica data store)	(same object may not be added in both stores)	(may not yet exist in replica data store)	(may not yet exist in replica data store)	add to replica data store
	MODIFIED	update in replica data store	(may not yet exist in local data store)	**CONFLICT** update in local data store / update in replica data store	**CONFLICT** delete in local data store / update in replica data store	add to replica data store
	DELETED	delete in replica data store	(may not yet exist in local data store)	**CONFLICT** update in local data store / delete in replica data store	ignored	delete in local data store
	(not existing)	(may not be clear in replica data store)	add to local data store	add to local data store	delete in replica data store	(impossible)

Figure 4.4: *Potential state combinations of two object versions and the actions required during the merging phase.*

marked with 'CONFLICT' are those resulting in a synchronization conflict. The actions denoted for conflicting combinations are the possible outcome of the reconciliation.

The state combinations possible in a specific scenario depend on the implementation of the synchronization process. If the state of a successful synchronized object is set to 'CLEAR' in both data stores, the combination 'MODIFIED' - 'ADDED', for example, may not occur since the object is first added in one data store, marked as 'ADDED' and later synchronized with the replica data store. After synchronization, both data stores set the state of their corresponding object version to 'CLEAR'. If the object is now modified in one data store, the state changes to 'MODIFIED', resulting in a combination of 'MODIFIED' - 'CLEAR' during the next synchronization. But if the actual object state is preserved after synchronization, which might be required in order to synchronize with another replica data store later on, some of the combinations shaded in gray in figure 4.4 become possible. In addition, depending on the mechanism used to uniquely identify data objects, conflicts may arise if a particular object is added on multiple clients independently and provided with the identical identifier. This scenario is often possible if the unique identifier is derived from a specific object property, such as the username for a user object.

Delegating the actual reconciliation process to the affected objects allows the use of specific reconciliation strategies depending on the object type and application-specific business rules. Since the actual object has the most information on the semantics of the conflict and the different object attributes, the reconciliation is expected to produce the most adequate results.

The object resulting after reconciliation depends on the strategy used to resolve conflicts. Besides others, the following strategies are conceivable:

Client Wins The object in the local data store on the client machine and sent to the server machine during synchronization dominates and is the result of conflict reconciliation.

Server Wins The object already residing in the server data store dominates and is the result of the reconciliation.

Last Changed Wins The posterior changed object is the result of the reconciliation. This strategy may require a synchronized timing instance for both data stores.

First Changed Wins The primarily changed object is the result of the reconciliation. This strategy may require a synchronized timing instance for both data stores.

4.6.5 'Out-Of-Collection' Problem

If the amount of data to be stored locally on the client machine is restricted using the concept of data collections, a problem herein referred to as 'out-of-collection' problem may occur. The concrete data collections representing the set of data to be synchronized with the replica data store typically depends on the privileges of a particular user and on application-specific configuration settings. Since the privileges associated with a particular user may be modified over time, the amount data available for offline operation also changes. This is reflected by a modified list of data collections. The 'out-of-collection' problem occurs if the privileges are constrained for a particular user. This potentially results in a smaller set of accessible objects. If the user has previously synchronized with the server, the client machine may store objects that are no longer included in one of the data collections. Such objects are no longer synchronized and may therefore become out-of-date. When the application switches to offline operation, these out-of-date objects are accessible for the application as they reside in the local data store and may be displayed to the user. Depending on the enterprise application functionality, such 'out-of-collection' data may get modified in offline operation. Although this is not a security issue as the user potentially already knows the contents of these objects from the time before the privileges were cut down, modifications to these objects may be refused by the server machine according to the application security policy and thus, may get lost. Objects no longer included in the synchronization process should be removed from the data store on the client machine in order to avoid confusion.

4.7 Application Logic for Offline Operation

Application logic denotes the part of an enterprise application that implements the core application functionality and processes the application data. Application logic is the active part of an enterprise application and may be further split up into presentation logic and business logic.

Presentation Logic is the part of the enterprise application that manages the presentation and event handling of the graphical user interface (GUI). The presentation logic is also responsible for gathering user input and calling the appropriate action in the business logic. The location of the presentation logic components highly depends on the application client technology used on the client machine. In traditional rich client architectures, the presentation logic resides on the client machine, whereas in web client architectures, the presentation logic is implemented on the server machine.

Business Logic is the part of the enterprise application that implements the actual business services and processes and manipulates application data. In architectures for enterprise applications, the business logic is mostly implemented on the server machine. Additionally, the business logic is considered to be independent from the application client technology used for realizing the presentation logic

The term 'application logic' is often only used for the business logic part.

In order to run an enterprise application in offline mode, all application logic required for offline operation has to reside on the client machine. The parts of the application logic that not yet reside on the client machine in online operation have to be transferred to the client machine before changing to offline operation. Depending on the underlying enterprise application architecture, this may affect the presentation logic and/or the business logic.

For offline operation, it may be appropriate to offer only a subset of the application functionality provided in online operation. Application features relying on an extensive amount of application data may be restricted to online operation only in order to avoid downloading this data to the local data store on the client machine. Security considerations may restrict several application features, where highly confidential application data is involved, from being supported in offline operation.

4.7.1 The Trivial Approach

Consider the following trivial approach to run an enterprise application in offline mode: install the complete enterprise application including the parts running on the server machine directly on the client machine and only regularly synchronize the persistent database using a vendor-specific replication system included in the database management system. This approach profits from the following advantages:

- The complete enterprise application functionality is available in offline operation.
- Only minor adaptations to the application code base are required in order to respect offline-specific requirements.
- No effort for an application-specific synchronization mechanism is required since most of the established database systems offer database replication mechanisms out of the box.

Although this approach may work for certain simple, specialized scenarios, it is expected to fail in general because of the following reasons:

- Each client machine is required to have all the server software installed, including a J2EE-compliant application server and a database server. The system resources of the client machine may not be powerful enough to run these servers with acceptable performance.
- Since an enterprise application may rely on other enterprise information systems running on different servers, these systems are also required to be installed locally on the client machine in order to have the complete application functionality available in offline operation.
- Several security issues may arise since not only is the complete application data held locally on the client machine, but also server configuration data, security policies and authentication and authorization information.

Instead, the selective reuse of application logic and data independent from the enclosing J2EE-compliant application server leads to a lightweight offline solution.

4.7.2 The Reuse Approach

For cost and complexity reasons, it may be useful to share the largest possible part of application logic between the online and offline operation mode. This requires the enterprise application design to be highly modular in order to reuse several application logic components both in the server environment for online operation and in the client environment for offline operation. Since the infrastructure available on a client machine may significantly differ from the one on the server machine, reusable application logic components have to abstract from any environment-specific technology feature. In online operation, large parts of the business logic reside on the server machine. In offline operation, parts of this business logic are reused within the client environment in order to provide the requested application functionality. In addition, the representation of business objects to be reused have to be identical both for online and offline operation. Since persistence of business data is expected to be environment-specific, the common representation of business objects have to be adapted to the environment-specific needs. The presentation logic may be reused in the form of a shared application client for both online and offline mode. Nonetheless, depending on the specific enterprise application scenario, the application functionality and thus the graphical representation of the application client may differ between online and offline operation.

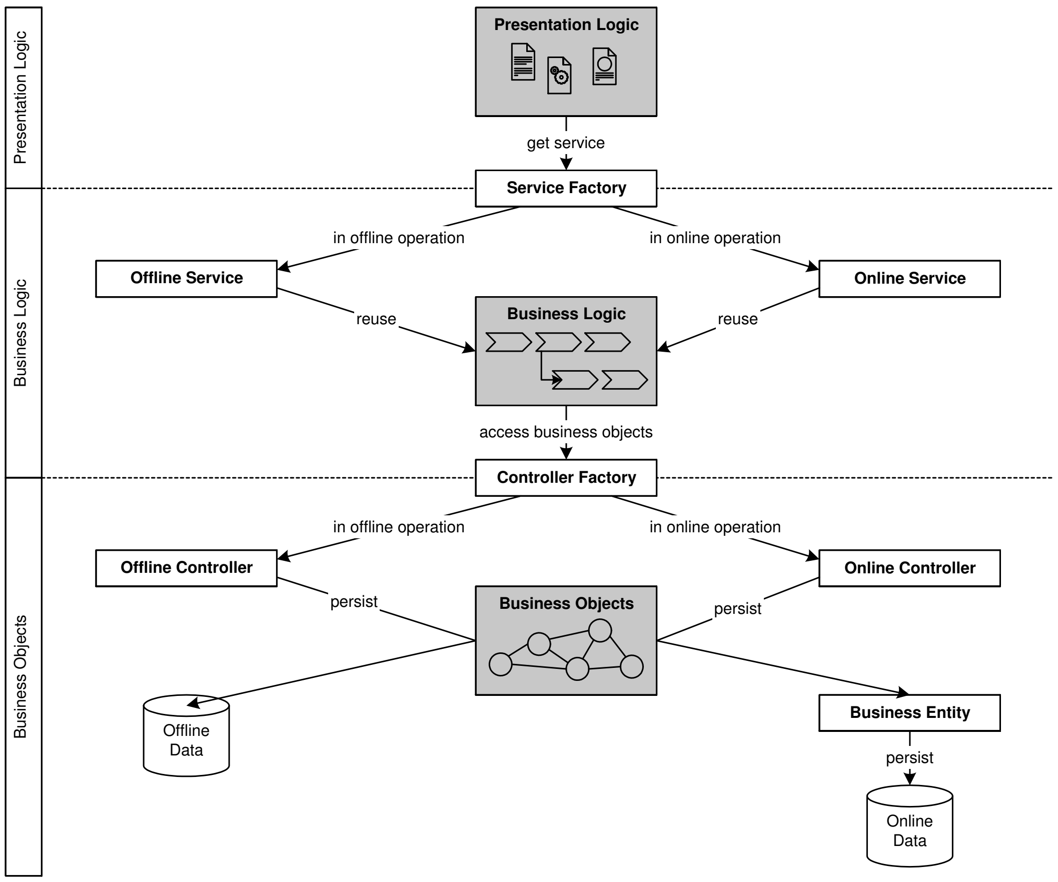


Figure 4.5: *Reuse of application logic between online and offline operation for an offline-capable architecture model for enterprise applications.*

The reuse approach results in the abandonment of J2EE technologies for offline operation, that are exclusively supported in the server environment, in order to exempt the client machine from running a J2EE-compliant application server. Rather than re-implementing the enterprise application for offline operation independently from any J2EE techniques like Session Bean support for application logic or the use of Entity Beans for persistent business objects, separation of the core application functionality from J2EE-specific technologies may be an appropriate approach for supporting both online and offline operation by the same code base. Figure 4.5 illustrates this approach in overview. The contained elements are discussed in the following sections.

4.7.2.1 Presentation Logic

The implementation of the presentation logic depends on the client component technology used. Using either a Java application client, a JNLP-featured rich client or an applet client, the presentation logic is implemented using a standard Java GUI library, for example the JFC/Swing library[5]. Additionally, the code for the presentation logic already resides and is executed on the client machine and may therefore be reused, at least partly, for offline operation. If using the client of a web application, a web browser, the presentation logic resides completely on the server machine, implemented either as JSP components or Servlet components. Since these components are required to run within a J2EE web container, a client machine running a web application in offline mode is also required to have a web container installed locally. Section 6.3 shows a solution for that issue.

4.7.2.2 Business Logic

Instead of implementing the application logic directly into a Session Bean, each application feature has to be implemented as a component only using standard Java programming language constructs that are supported both in the client and server environment. Thereon, each application feature component is encapsulated in either a service implementation based on a Session Bean for online operation or an implementation based on standard Java constructs supported on the client machine for offline operation. Several infrastructure services normally provided by the J2EE application server in online operation has to be implemented by the enterprise application for offline operation as needed, for example transaction or security services.

An abstract factory pattern[6] is applied to decide which service implementing the requested part of the application logic is to be used in a particular operation mode. Figure 4.6 illustrates the concept of a service factory that determines and creates the appropriate service implementation depending on the current operation mode. Using this pattern, the part of the business logic represented by the application feature component may be reused both for online and offline operation.

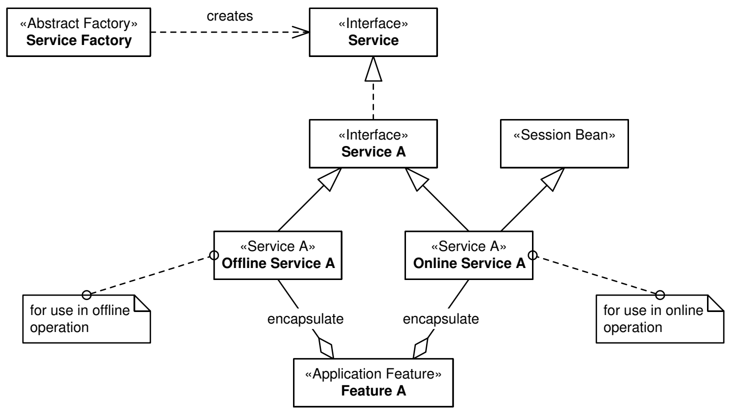


Figure 4.6: *Abstract factory pattern used to determine the appropriate service implementation for a particular operation mode.*

[5]The Java Foundation Classes Swing library supports development of cross-platform graphical user interfaces and graphical functionality for rich clients [Sun03c].
[6]The abstract factory pattern is described in [Gam$^+$95].

The modularization of the business logic in services is crucial both for the adaptation of a single application feature to a particular infrastructure and for the configuration of the application features supported in each operation mode. Specific services may be available in online operation but not in offline operation and vice versa. Additionally, the implementation of single services may differ from online operation to offline operation, for example by only supporting parts of the application feature encapsulated by the service.

4.7.2.3 Business Objects

While in an enterprise application that runs in online operation, business objects are typically implemented as Entity Beans, this technology is not supported on the client machine not running a J2EE-compliant application server. Similar to the approach for sharing business logic between online and offline operation, the business object implementations have to be separated from J2EE-specific technologies in order to reuse the same business object implementation for offline operation. Since business objects are typically shared between EJB components, web components and client components, they often support a representation referred to as value object[7]. A value object represents the state of a business object at one moment in time and may easily be exchanged between and used by different components. Value objects may either be immutable or mutable, depending on the business object represented. Traditionally, value objects are used to reduce chattiness between the calling and the called component, for example between the application client and a Session Bean. The caller retrieves a value object representing a business object with its complete state in one remote call instead of requesting each object field with a separate remote call. Herein, the concept of value objects is used to provide a business object representation independent from J2EE-specific technologies and to adapt the value object to technologies available in specific environments.

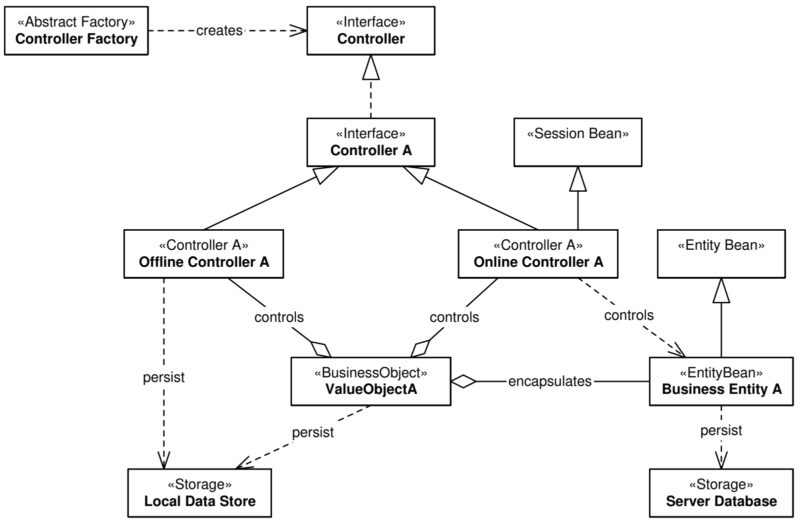


Figure 4.7: *Value objects representing business objects and controllers managing persistence of business objects.*

[7]The design pattern 'Value Object' is further described in [Alu01].

Figure 4.7 illustrates the concept of value objects representing business objects and controllers managing the persistence of business objects. For online operation, the server part of an enterprise application typically encapsulates value objects within Entity Beans, offering methods to get a value object from the current state of the Entity Bean and to set the state of the Entity Bean to the state of the value object. Again, an abstract factory pattern may be used to determine the current mechanism for persisting business objects represented by value objects. For each type of business object, an appropriate controller instance is defined that is aware of persistently adding, modifying and deleting business objects of this type. In online operation, controllers are realized as Session Beans and work on business object instances implemented as Entity Beans. The persistence of Entity Beans is managed according to the persistence mechanism specified by J2EE and implemented by the EJB container. In offline operation, business objects are made persistent directly by the appropriate controller component. A particular controller instance encapsulates the knowledge to persist a specific type of business object. Each type may have its own controller instance or multiple types may be managed by a common controller instance. The controller factory component is used to determine and to retrieve the appropriate controller instance for a particular type of business object.

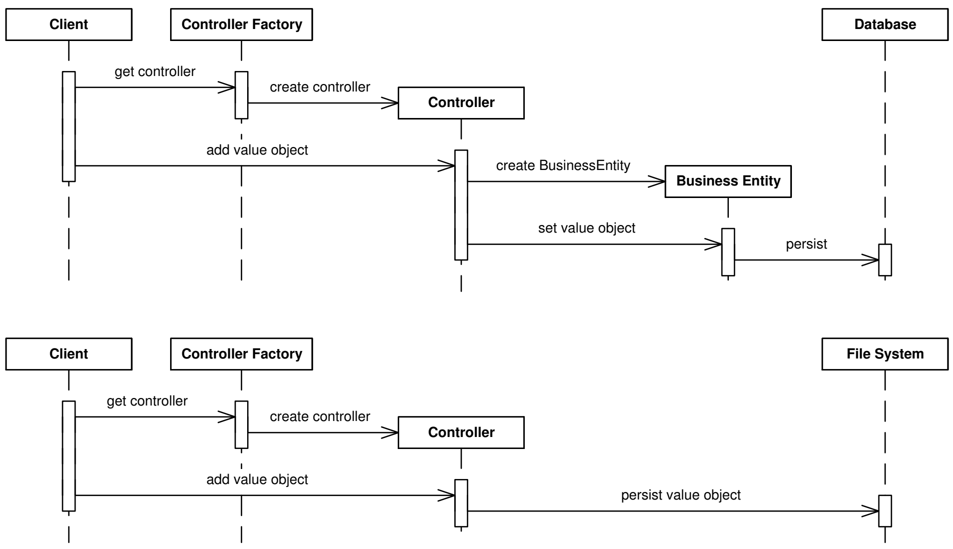


Figure 4.8: *Sequence of persisting business objects in online and offline operation.*

Figure 4.8 compares the sequence of persisting a business object represented by a value object when running an enterprise application either in online or in offline operation. In both operation modes, the controller instance retrieved using the controller factory component is called to persist a particular new business object represented as value object. In online operation (top), the controller instance creates a new business entity object, which is in fact an Entity Bean instance, and sets the state of the newly created business entity object to the state of the value object. The business entity object itself is responsible for persisting its state to the database. In offline operation (bottom), the controller instance stores the value object persistently, for example, to the local file system.

4.8 Security

As enterprise applications often work on and contain confidential company and customer data, security is a core issue. Appropriate measures have to be taken in order to protect data from unauthorized access and illegal modification. Whereas security is a general issue within the context of enterprise applications rather than an offline-specific one, security in offline operation requires additional considerations.

4.8.1 Security for Offline Operation

In online operation, security measures such as authentication and authorization take place preferably on the server side of the enterprise application since the server environment may be considered as more concise, better protected and supervised. Additionally, the server environment typically offers more sophisticated features for implementing security-related functionality and is often closely coupled with security-specific information system such as a user account database or a security policy system.

In offline operation, the relevance of security changes. The environment on the client machine has to be considered as fundamentally insecure since the client machine is not under the same control and surveillance as a server machine. It may not always be predictable which entities have which privileges on the client machine and may therefore gain access to locally stored data that is protected. This may be a potential risk both for confidentiality and data integrity since protected data may be read and/or modified. Generally, two approaches for dealing with the insecure client environment are conceivable:

The 'try-to-secure' approach undertakes a wide range of security-related measures to increase the client-side security to a similar level as on the server environment. Measures may include restrictive privileges for system users logging into the client machine, data encryption and digital signatures for persisted data and a mechanism for offline authentication and authorization[8] within enterprise applications. The 'try-to-secure' approach aims to create a trusted environment that spans both the client and server machine. Because of the countless number of security-related influencing factors and potential risks and attackers, reaching an appropriate security level may result in an exorbitant effort. This approach seems only feasible if the client machine and its operating system are well-secured regarding to system user authentication, file system protection and system privileges. The enterprise application itself must not be expected to be able to secure the client machine without support from the underlying system. In addition, it is not recommended to shift security mechanisms from the central server to the decentralized client machines.

The 'accept-as-insecure' approach leaves the client environment in the potential insecure state but controls and secures the transition points between the client and the server environment. Applied to an enterprise application running in offline operation, this means that no security-related actions are undertaken during offline

[8] Offline authentication and authorization are further described in section 4.8.2.

operation, but the underlying application security policy is enforced when accessing data stored on the server machine and during synchronization. All modifications made during offline operation that do not comply with the application security policy are discarded. Depending on the enterprise application design, this approach may allow the user to perform actions during offline operation that are not allowed in the online mode and therefore are discarded during synchronization. This may lead to a displeasing user experience.

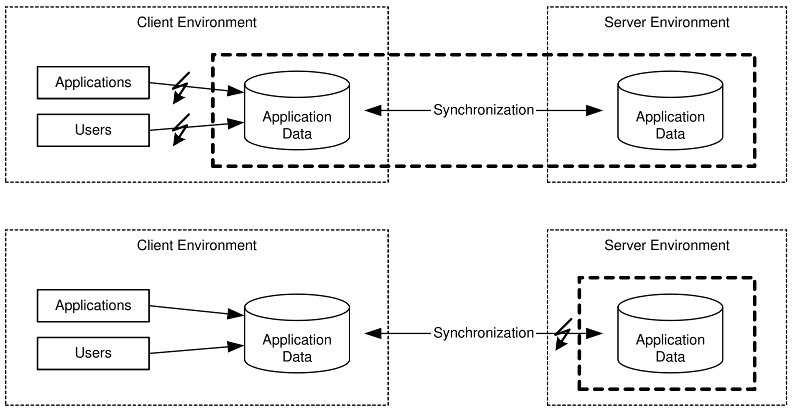


Figure 4.9: *Two security approaches for offline operation.*

Figure 4.9 illustrates the two approaches described. The bold dotted line surrounds areas where the application security policy is enforced. Arrows mark potential situations where the application security policy may be violated. The 'try-to-secure' approach (top) secures the access to the locally stored data on the client environment. The 'accept-as-insecure' approach (bottom) enforces the application security policy during synchronization.

The two approaches described are not mutually exclusive but may be combined in order to optimize the security level, costs and complexity and the user experience during offline operation. By combining the second approach with offline authentication and basic authorization in offline mode, a misleading user experience may be avoided in advance since the application functionality and appearance may be customized based on the privileges associated with the authenticated user. In addition, by relying on operating system security mechanisms such as system user authentication or file system privileges, the 'accept-as-insecure' approach may protect sensitive application data from being read by unauthorized users.

The following guidelines may be proposed concerning security for offline operation:

- Only store the minimum amount of data required for providing the desired offline application functionality within the local data store on the client machine. Without special protection such as encryption, data stored on the client machine must be considered as being readable for every user able to login to the client machine and may even be arbitrarily modified by privileged system users.

- Restrict the application functionality that is provided in offline operation to the minimum. Each additional application feature provided tends to require additional data to be stored in the local data store and that is potentially exposed to unauthorized access. Additionally, avoid providing security-critical features such as user privileges administration.

- Perform offline user authentication if possible. Offline authentication allows the application to know which user is currently using the application and may therefore adapt the appearance and behaviour accordingly.

- Enforce the application security policy not later than on the transition from offline operation back to online operation. If adequate, enforce parts of the security constraints also during offline operation.

- If especially critical data modifications such as user privilege manipulations are required to be possible in offline operation, ask for a confirmation before synchronizing these modifications with the server data store. A malicious attacker may try to modify data stored locally on a client machine without the knowledge of the application user. If the application user is associated with extended privileges for the particular offline-capable enterprise application, the modifications appearing to be made by the legitimated application user are synchronized with and applied to the server data store the next time the application user runs the enterprise application in online operation.

4.8.2 Offline Authentication and Authorization

Consider a scenario for an enterprise application where authentication and authorization is required in offline operation, for example to restrict access to an enterprise application running in offline mode. In online operation, the verification of the credentials provided for authentication typically takes place on the server side with the reference credentials in an account database and the assignment of privileges to the authenticated user is based on security-related information residing on the server side. In offline operation, the verification of user-provided credentials has to be performed on the client machine, where the reference credentials and privileges to be assigned to the user after authentication must also be stored locally on the client machine. Figure 4.10 shows the difference between the authentication process in online and offline operation. When running an enterprise application in online mode (Figure 4.10, top), the user credentials are gathered by the application client on the client machine and are sent to the server machine in order to verify them against an authentication database. In offline operation (Figure 4.10, bottom), the credentials gathered by the application client are checked with locally stored authentication data objects.

When storing authentication and authorization data locally on the client and relying on offline authentication and authorization, the following points have to be taken into account:

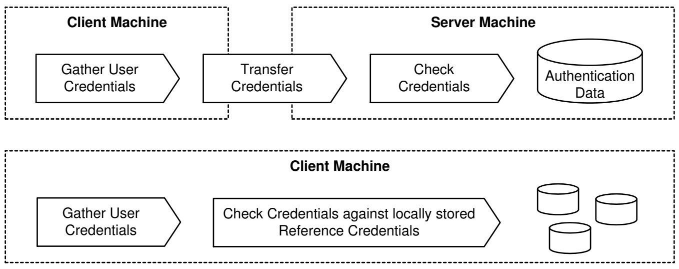


Figure 4.10: *Authentication process for online and offline operation.*

- Only credentials for specific users that actually need to authenticate in offline operation on that particular client machine should be stored locally. Ideally, each client machine is only used by one application user and not shared with other users, thus there is no need to locally store other credentials than the ones for that particular user.

- Confidential information such as passwords should not be stored in plain text, but only as one-way secure hash values. Private and public keys used for encryption and digital signatures should reside in protected key stores.

- Authentication and authorization data stored locally on the client machine represent the state at one specific moment in time and may not always reflect the current version residing on the server. Several privileges may have been added to or removed from the users privilege collection on the server side in the meantime. Thus, locally stored credentials must also be updated regularly with the ones residing in the server data store.

- When relying on credentials stored locally on the client machine for offline operation, verifying the integrity of the credentials before authenticating a user may be important in order to avoid and/or detect malicious modification of privileges assigned to that particular user.

- Never rely only and completely on offline authentication or authorization. Use offline authentication and authorization in offline mode only. Rely on standard online authentication and authorization mechanism in online operation, when switching back from offline to online operation and during synchronization. Nonetheless, offline authentication and authorization may be useful to customize the appearance and behaviour of the enterprise application during offline operation.

4.8.3 Offline Authentication Approach

As described in section 4.8.2, performing authentication in offline operation requires special consideration regarding malicious modification of authentication and authorization data stored locally on the client machine. This section describes an approach for offline authentication satisfying the requirements listed above.

The approach is based on the Java Authentication and Authorization Service (JAAS) framework and assumes that authentication is performed using a distinct username for each user, accompanied by a secret password. After successful authentication, the user owns a number of privileges based on security roles associated with the user, that are used by the enterprise application in order to authorize access to protected resources. As base for offline authentication, username, password and the list of security roles have to be stored locally on the client machine. Whereas the password is critical regarding confidentiality, the username and especially the list of security roles are critical regarding integrity. By guessing other security roles currently not owning and adding them to the list of associated security roles, a malicious user may get unauthorized access to protected resources[9].

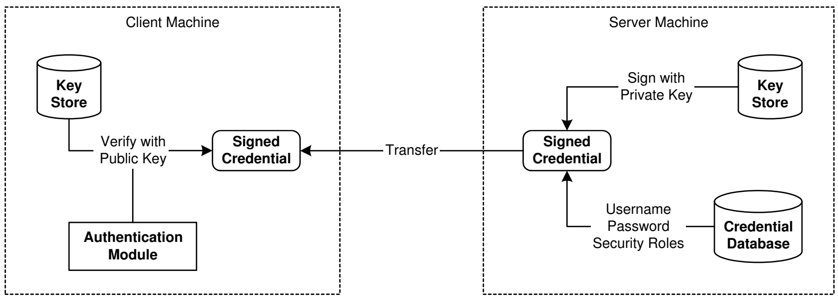


Figure 4.11: *Offline authentication approach with digitally signed credential object.*

Figure 4.11 illustrates the offline authentication approach. When switching to offline operation, the server part of the enterprise application creates a credential object containing the username, the password and the list of currently associated security roles and digitally signs this object using the enterprise application's private key. Thereon, the server sends the signed credential object to the client machine where it is persisted for future use. The application client has the corresponding public key, that is associated with the private key used for creating the digital signature, stored in the local key store and may therefore verify the digital signature of the signed credential object.

Figure 4.12 illustrates the sequence of offline authentication based on the offline authentication approach proposed. Whenever the client attempts to authenticate locally in offline operation, the authentication service component is called. This service reads the JAAS login configuration, creates a login context and requests the offline login module specified in the JAAS configuration file to authenticate the credentials provided by the authenticating user. Since the offline login module performs the actual offline authentication, it first look for an existing signed credential object and verifies the integrity of the signed credential object by checking its digital signature using the enterprise application's public key from the key store. If the credentials received from the user matches the ones stored in the signed credential, the offline login module adds the security roles retrieved from the signed credential object to the JAAS Subject. The user is now authenticated and owns

[9]By respecting the security guidelines mentioned in section 4.8.1, this is rather a convenience issue than a security issue since all actions and their effects performed in offline operation have to be checked based on the server-controlled privileges when switching back to online operation.

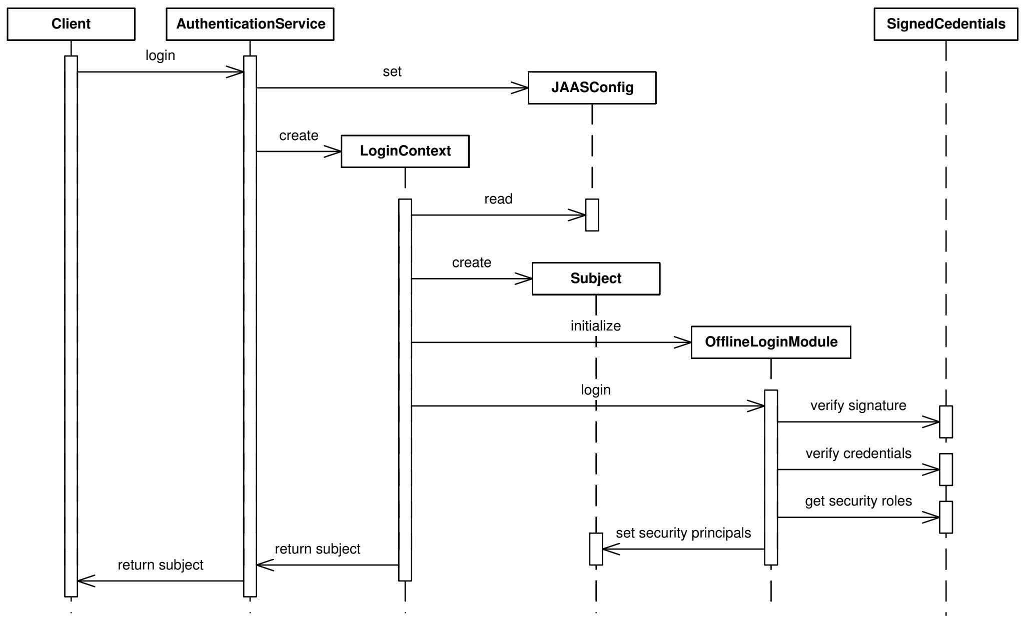
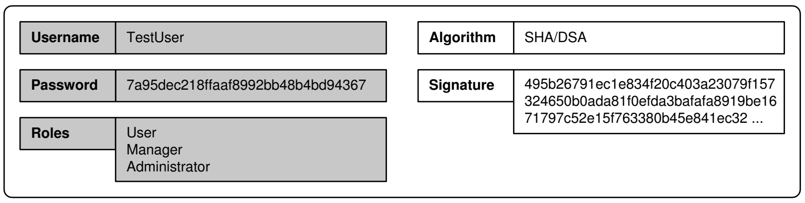


Figure 4.12: *Login sequence during offline authentication.*

the privileges listed in the credential object and may use them for further authorization. The object representing the currently authenticated user may be retained transiently in a static security context if other instances are required to have access to the authenticated user object, for example to query for a specific security role association.

In order to protect the secret password from being read by unauthorized parties, the credential object should contain only a secure one-way hash of the actual password and not the password itself in plain text. Upon authentication, the identical hash function is applied to the password provided by the user and checked with the hashed password in the credential object. Figure 4.13 shows the state of a signed credential object, consisting of the username, the one-way-hashed password, a list of security roles and the digital signature. The gray shaded area is reflected by the digital signature.



Username	TestUser
Password	7a95dec218ffaaf8992bb48b4bd94367
Roles	User Manager Administrator
Algorithm	SHA/DSA
Signature	495b26791ec1e834f20c403a23079f157 324650b0ada81f0efda3bafafa8919be16 71797c52e15f763380b45e841ec32 ...

Figure 4.13: *A signed credential object.*

This approach relies on the fact that the signature of the signed credential object may only be verified using the corresponding public key associated with the private key that was used to generate the signature, resulting from the concept of asymmetric encryption systems. Therefore, it must be assured that the public key may not be replaced on the client machine. Otherwise, an attacker may modify the credential object, re-generate the digital signature using an alternative private key and force the application client to use the replaced public key to verify the signature.

Whereas this approach facilitates offline authorization because the enterprise application may trustfully rely on the security roles associated with the authenticated user in offline mode, it prohibits the user from changing the password in offline operation. The locally stored credentials may only be modified with knowledge of the private key used for signing the credentials object, since changes to the object state requires the creation of an updated digital signature. Otherwise, the integrity of the credential object is broken. The application client is not in possession of the enterprise application's private key since the private key resides in the key store on the server machine and is not intended to be distributed.

4.8.4 Authentication and Operation Mode Transition

Special consideration is required regarding authentication when switching between online and offline operation. In online operation, the user is authenticated with the server using an appropriate authentication scheme, such as JAAS. Typically with lazy authentication used by J2EE, the user is asked for authentication when attempting to access a protected resource. The client component must be able to satisfy this request by either presenting an input dialog to the user or by propagating previously fetched credentials to the server.

When changing to offline operation, the user authenticated for online operation may either be propagated to the enterprise application running in offline mode or users may be urged to re-authenticate with the local authentication service. Depending on the authentication mechanism used, propagation may be implemented by transferring the authenticated user object to the client machine and setting this object in the static security context, if required. Re-authentication may either be solved by presenting the input dialog asking for the user credentials again or by silently authenticating with the same credentials used for server-side authentication. This requires the user credentials to be stored transiently in the application client context.

When switching back to online operation, propagation of the authenticated user back to the server is not recommended, since, depending on the offline authentication approach used, the authenticated user may not be absolutely trustful. A malicious user may have found a way to gain additional security roles associated with it. Propagating the potentially compromised user object to the server may result in a security harm. Therefore, only re-authentication using the online authentication service guarantees unbroken security privileges in online operation. The user credentials used for offline authentication, username and password for example, may be reused to silently authenticate the user.

4.8.5 Synchronization Security

As a solution for synchronization should be application-independent and reusable for multiple enterprise applications, the synchronization itself may not provide any application-specific, security-related mechanisms. It is in the responsibility of the enterprise application to enforce the application security policy during synchronization. Thus, synchronization has to be performed within the same security context used for normal application operation. This may be achieved by running the actual synchronization process, or at least the phase of sending updates from the client to the server machine, with the privileges of the particular user currently logged in. If the synchronization process uses the same data access mechanisms as the enterprise application itself when creating, modifying or deleting objects, it may reuse the security constraints defined within the particular data access instances. This ensures that the synchronization process does not perform unallowed modifications.

However, the synchronization mechanism may be required to perform modifications on objects on the client machine exceeding the user's privileges. For example, a standard user may not be allowed to modify a specific object state, but an administrator may. On the server side, an administrator updates this particular field. During synchronization, the locally stored object has to be updated with the modified object state from the server. This is not a priori possible when running the synchronization process under the user's privileges. Depending on the chosen approach regarding offline authorization described in section 4.8.1, two solutions are conceivable:

Solution for the 'try-to-secure' Approach By following the 'try-to-secure' approach specified in section 4.8.1, the client machine has to perform authentication and authorization in offline mode. This requires the synchronization process to provide adequate credentials when updating a protected object on the client machine. Updates that require extended privileges in order to be applied to the local data store on the client machine can only originate from modifications to the data store on the server machine and thus, are always sent from the server to the client side, not vice versa. Applying updates received from the server machine to the local data store on the client machine may be performed using the privileges of a dedicated synchronization user that owns extended privileges. Running all synchronization phases under the synchronization user's privileges is not recommended since this may allow a malicious user to unauthorized modify locally stored objects and to synchronize them to the server machine.

Solution for the 'accept-as-insecure' Approach Alternatively, if using the 'accept-as-insecure' approach, no authorization is performed on the client machine in offline operation and therefore, arbitrary modifications on locally stored objects are potentially possible by the user currently logged in. This also enables the synchronization process, which runs with the privileges of the current user, to arbitrarily update objects within the local data store on the client machine where the user effectively has no privileges regarding the application security policy enforced on the server machine. In this scenario, it is not necessary to execute the synchronization process with different privileges than the ones of the user currently logged in. Data modifications are not checked with the application security policy until they are propagated and applied to the server data store during synchronization.

Section 5

A Framework for Offline J2EE Applications

This section discusses a programmatic framework that enables the development of offline-capable J2EE enterprise applications. This framework is aimed to conceal offline-specific issues such as the internals of data modification tracking, data synchronization and management of transitions between online and offline operation mode from the application developer by providing reasonable standard implementations for these tasks. Nevertheless, these standard solutions have to be flexible and customizable to meet the requirements of a wide range of offline enterprise application scenarios. The prototypic framework described here is, for simplicity, referred to as J2OO framework, a J2EE online-offline framework.

5.1 Responsibilities

The J2OO framework accepts the following responsibilities regarding the offline-capability of enterprise applications:

Data Modification Tracking The J2OO framework provides a mechanism to track additions, modifications and deletions of application data objects. Each of these actions are logged by the framework and assigned with a distinct version indicating a specific state of the affected business object.

Data Synchronization Based on the mechanism for data modification tracking, the J2OO framework implements a data synchronization concept that is able to synchronize and reconcile modified business objects between a local data store on the client machine and the central data store on the server machine. The solution provided respects the requirements listed in section 4.6.1.

Transition between Operation Modes The J2OO framework provides components that manage the transitions between different operation modes and perform the required steps during the transition. Specific parts of the transition procedure are configurable for a concrete enterprise application scenario. In addition, a watchdog component that observes the server part of the enterprise application may change the operation mode if the application server fails or restores after an outage.

Framework Configuration As different enterprise application scenarios have also different requirements concerning offline operation, the J2OO framework provides a variety of configuration parameters allowing the customization of the framework. Consequently, the J2OO framework is usable for a wide range of application scenarios based on different architecture variants. Especially, the application configurations described in section 4.3 are supported by the framework.

5.2 Framework Overview

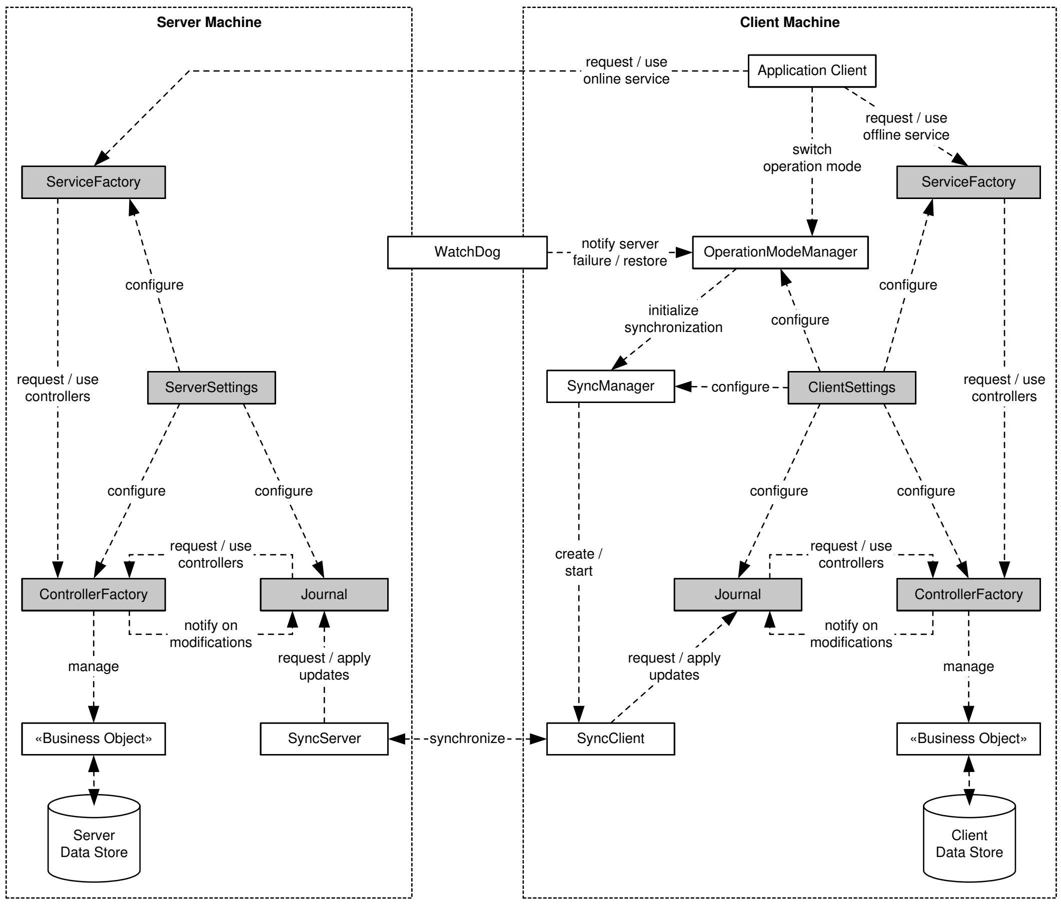


Figure 5.1: *Main components of the J2OO framework.*

Figure 5.1 illustrates the main components of the J2OO framework applied to a rich client architecture scenario[1]. Depending on the current operation mode, the application client uses either the service factory on the server machine (for online operation) or the one on the client machine (for offline operation). The service factory creates service instances

[1]While the rich client architecture is well-suited for illustrating the concepts, the J2OO framework is applicable also to fat client and web application architectures.

that implement parts of the business logic and uses the controller factory to retrieve an appropriate controller instance that is able to manage a specific type of business object required by the service. The controller instance notifies the journal about each data creation, modification or deletion. For each enterprise application, exactly one journal instance exists on the client and one on the server side. The journal maintains version information for each business object in the data store and is able to generate a list of updates that need to be sent to the remote side during synchronization. A synchronization server component on the server machine listens to synchronization requests made by the synchronization client on each client machine.

On the client machine, the operation mode manager implements the functionality to switch between different operation modes. The synchronization manager controls the initiation of synchronization processes and may either be used directly by the application client on demand or by the operation mode manager for synchronization during operation mode switches. A watchdog component is able to observe the server part of the enterprise application and to initiate an operation mode transition if the server fails or restores again.

Both the client and the server part of the enterprise application are configured using application-specific settings. On the server machine, the concrete implementation of the service factory, the controller factory and the journal are configured by the server settings. On the client machine, additional settings for the operation mode manager and the synchronization manager are defined by the client settings.

5.3 Framework Interfaces & Classes

The J2OO framework defines a number of interfaces and classes to be implemented, extended or simply used by an enterprise application that provides offline operation based on the J2OO framework. A description of all framework interfaces, classes and methods including the Java source code with JavaDoc [Sun03b] documentation is available on the resource CD outlined in appendix A.

5.4 Data Modification Tracking

The J2OO framework uses a mechanism referred to as the journal to track object modifications, both on the server machine for online operation and on the client machine during offline operation. Each type of business object, that has to be synchronized between the client and the server data store, must implement the `ISynchronizable` interface and is herein referred to as a synchronizable object. Whenever the enterprise application creates a new synchronizable object, it registers the object with the corresponding local journal. The journal maintains version information for each synchronizable object registered. Whenever the application subsequently modifies or deletes an existing synchronizable object, the controller instance that actually modified the business object informs the local journal about the change. Thereon, the journal updates the version information for this object accordingly.

5.4.1 Journal

The default implementation for the `IJournal` interface contains a logical clock realized as an incrementing number, a journal identifier (also referred to as the replica identifier) and a list of `UpdateInfo` objects for currently registered synchronizable objects. The update list acts as a modification history, holding all updates in the exact order they were applied to the data store. Additionally, the journal creates a forward index map that maps `ISyncId` instances to the corresponding `UpdateInfo` object and a reverse index map that holds the latest `UpdateInfo` object for each `ISynchronizable` object registered. These maps are stored transiently and are recreated when the journal is read from the persistent store. Figure 5.2 illustrates the internal structure of the Journal implementation.

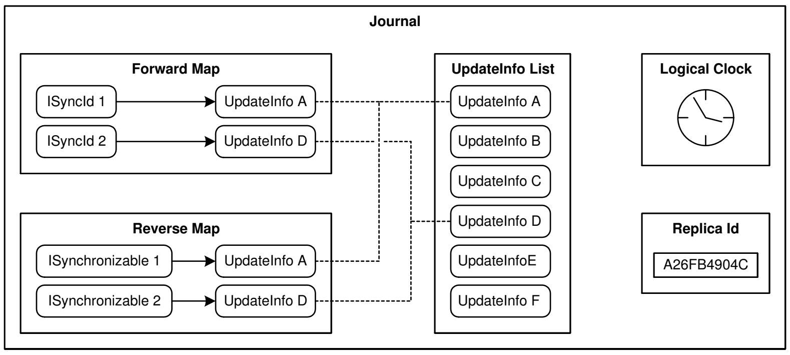


Figure 5.2: *The internal structure of the Journal implementation.*

Each time the version information for a particular synchronizable object changes because of an update of the object's state, a new `UpdateInfo` object containing the updated version information is added to the update list. The existing `UpdateInfo` object is not deleted from the update list in order to preserve the original order of object modifications. In addition, the new `UpdateInfo` instance is associated with the appropriate object in the forward and reverse index map.

5.4.2 Update Information

Updates for synchronizable objects registered with the local journal are represented within the journal and transferred during synchronization as `UpdateInfo` instances. An instance of `UpdateInfo` contains a `ISyncId` object that uniquely identifies the synchronizable object across multiple replicas on different machines, the identifier of the journal that created the `UpdateInfo` object, information on the synchronizable object affected and the current version associated with the synchronizable object. The J2OO framework distinguish between two `UpdateInfo` subclasses:

ContentUpdateInfo The `ContentUpdateInfo` object represents the creation of a new or the modification of an existing synchronizable object. A `ContentUpdateInfo` instance contains a reference to the object itself and includes the affected object when being transferred to the remote replica. `ContentUpdateInfo` objects are created when a new object is added to the journal or an already registered object is updated.

DeletionUpdateInfo An instance of `DeletionUpdateInfo` represents a deletion tombstone for a previously registered, but actually deleted synchronizable object. A `DeletionUpdateInfo` does not contain a reference to the object deleted, but contains the type and the unique `ISynchronizable` identifier of the deleted object. `DeletionUpdateInfo` objects are created when a registered object is deleted.

Depending on the `UpdateInfo` object in the local journal, the one received from the remote replica and the version vector of each `UpdateInfo` object, the actions described in figure 5.3 are taken.

		Received UpdateInfo	
Local UpdateInfo		**ContentUpdateInfo**	**DeletionUpdateInfo**
none		add object locally create ContentUpdateInfo with remote version	createDeletionUpdateInfo with remote version
ContentUpdateInfo	local version later than or equal to remote version	ignore received update (old news)	
	local version earlier than remote version	update local object to state of remote object create ContentUpdateInfo with remote version	delete local object create DeletionUpdateInfo with remote version
	local version conflicting with remote version	reconcile confflict update local object to state of resolved object create ContentUpdateInfo with merged version	reconcile conflict delete local object / keep merged object create DeletionUpdateInfo / ContentUpdateInfo with merged version
DeletionUpdateInfo	local version later than or equal to remote version	ignore received update (old news)	
	local version earlier than remote version	re-introduce object locally create ContentUpdateInfo with remote version	create DeletaionUpdateInfo with remote version
	local version conflicting with remote version	reconcile conflict keep deletion / re-introduce object locally create DeletionUpdateInfo / ContentUpdateInfo with merged version	create DeletionUpdateInfo with merged version

Figure 5.3: *Actions to be taken when applying an update to the local data store, depending on the type of received update, the type of the local update and the relation between the two participating versions.*

5.4.3 Object Versioning

Substantial to the journal is an appropriate mechanism to track and compare different object versions. While section 4.6 describes a simple approach based on explicit object states, the J2OO framework uses a vector-based approach offering sophisticated version comparison and conflict detection. This approach enables the provisioning of fast synchronization.

According to [Coh03], the J2OO framework provides a vector-based implementation of the IVersion interface. Fundamental is the concept of version vectors [Sto83] to compare two versions of the same object and to detect version conflicts. A version vector consists of multiple vector components, one for each synchronization replica that is aware of the particular object. A vector component on its part consists of the replica identifier and the value of the logical clock on the corresponding replica at the time the version vector is being created. Figure 5.4 shows several examples of version vectors as used by the framework. The notation 'A:1' describes a vector component for the replica A with the logical clock value 1 on the replica A at the time the version vector has been created. Dotted areas in figure 5.4 indicate vector components for replicas that are not aware of the particular synchronizable object. Object O1, for example, is known by replica S, A and B, but not by replica C and D. The version vector implementation used by the framework does not contain vector components for replicas that are not aware of a particular synchronizable object.

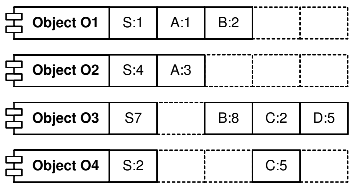


Figure 5.4: *Version vector examples containing version components for each replica aware of the particular synchronizable object.*

When a particular object is created, updated or removed either on the server or on a client, the logical clock of the corresponding journal is incremented and the version for that object is updated to the new local clock value. Only the vector component representing the local journal is modified. In addition, a version vector is only modified if the associated object is being updated. This guarantees that objects having different states are also tagged with different version vectors and objects with the same version vector have an identical state. Based on that, version vectors of different versions of the same object may be compared in order to determine the relation between the two versions. One version vector may either be equal to, earlier than, later than or conflicting with a second version vector. Let v be a version vector and $v[i]$ the logical clock value for replica i in version vector v. Two version vectors v_1 and v_2 are equal, if $v_1[i] = v_2[i]$ for every replica identifier i. One version vector v_1 is earlier than a second version vector v_2, and v_2 is later than v_1, if $v_1 \neq v_2$ and $v_1[i] \leq v_2[i]$ for every replica identifier i. Two version vectors v_1 and v_2 are conflicting, if at least two replica identifiers i and j exist such that $v_1[i] < v_2[i]$ and $v_1[j] > v_2[j]$.

Whenever a conflict between two version vectors is detected, a new version vector resulting from merging the conflicting version vectors and incrementing the local journal's logical clock is created and assigned to the synchronizable object affected by the conflicting updates. The merged version vector v_m resulting from two conflicting version vectors v_1 and v_2 satisfies the conditions $v_m[i] = \max(v_1[i], v_2[i])$ for every replica identifier i not equal to the local replica identifier and $v_m[j] = \max(v_1[j], v_2[j]) + 1$ for the local replica identifier j.

Until a client connects to and synchronizes with the server for the first time, the server does not contain a component for that particular client in the version vector since the server is not aware of the client's journal identifier. During synchronization, the server adds a vector component for the connecting client to each version vector associated to an object exchanged with the client. If an object has already been synchronized between n clients and the server, the version vector for that object contains n+1 components.

Each replica maintains a summary version that indicates the earliest version the journal is aware of. The summary version is represented by a version vector and is equal to or later than the version of each object the replica is aware of. The summary version of replica A is used by a remote replica B to determine the updates that have to be sent to journal A. As updates are always sent and applied to journal A in the exact order they occurred on the remote journal B, updates with a version earlier than the summary version of journal A do not need to be transferred from journal B to journal A as they would represent old news.

The vector-based version tracking approach allows to determine the set of updates, that has to be sent to a remote replica, only using the summary version of the remote replica. In contrast to other version tracking mechanisms, such as revision-based approaches that use a replica-independent revision number as version information[2], there is no need to propagate the version of every object stored in the local data store to the remote replica.

5.5 Data Synchronization

While section 4.6 describes the relevance and requirements of data synchronization with respect to online and offline operation of enterprise applications, this section illustrates the synchronization solution integrated in the J2OO framework that satisfies these requirements. The proposed solution is partly based on concepts described in [Coh03], which have been adapted to meet the specific requirements for synchronization between multiple clients and a central server.

The J2OO framework provides a solution to synchronize two data stores that are distributed across two interconnected machines. One machine acts as the central server hosting the complete application data, whereas the other machine represents the client machine that runs the application client and hosts the parts of the application data required for offline operation.

5.5.1 Synchronization Model

Figure 5.5 illustrates the substantial building blocks of the synchronization solution and highlights their collaboration. The synchronization manager handles the initiation of the synchronization process. Triggered by the user or by another component of the J2OO framework or the enterprise application, the synchronization manager creates a synchronization client instance that is able to communicate with the synchronization part on

[2]The Concurrent Versions System (CVS) [Ced03] is an example for a system that uses a revision-based tracking mechanism.

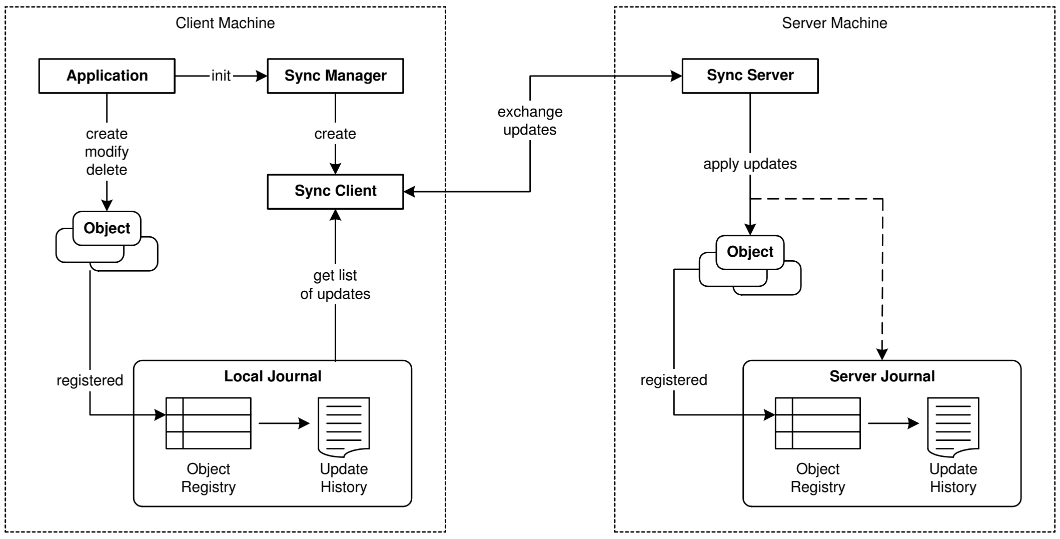


Figure 5.5: *Substantial parts of the synchronization solution provided by the J2OO framework.*

the server machine. The synchronization manager supports both synchronous and asynchronous synchronization. While synchronous synchronization blocks the user interface and may display a synchronization dialog until the synchronization process has finished, asynchronous synchronization is accomplished in a background thread and therefore does not block the user interface. Synchronous synchronization is useful when running the application in the 'on-demand' configuration, after logging in, before switching to offline operation or after switching back to online operation. Asynchronous synchronization is used to periodically synchronize data in the 'be-prepared' configuration in order to avoid interrupting the user by a blocked user interface.

The enterprise application may configure a list of synchronization events for which a synchronization process has to be initiated, for example during a transition from offline to online operation or on user demand. A synchronization process is started by the synchronization client on the client machine and not by the synchronization server. This results from the request-response-driven communication model implemented by the protocols supported by the J2EE platform. The client first requests a summary version from the server indicating the current overall version of the server data store. The client uses this information to create a list of pending and conflicting updates for locally created, modified or deleted objects, that have to be sent to the server side. The synchronization solution implemented by the J2OO framework only transfers new updates to the remote machine, referred to as fast synchronization. After resolving potential conflicts and applying the updates to the central data store, the server returns a list of updates newer than or conflicting with the summary version provided by the client. The client thereon applies these updates to its local data store. During the actual synchronization, update information objects containing the object version, an update identifier and optionally the object itself are transferred between the two participating machines. Figure 5.6 illustrates the synchronization protocol used between the client and the server. This protocol may either be processed on top of HTTP(S), RMI-IIOP or a vendor-specific communication protocol. The J2OO framework provides default implementations for both HTTP(S) and RMI/IIOP-based synchronization.

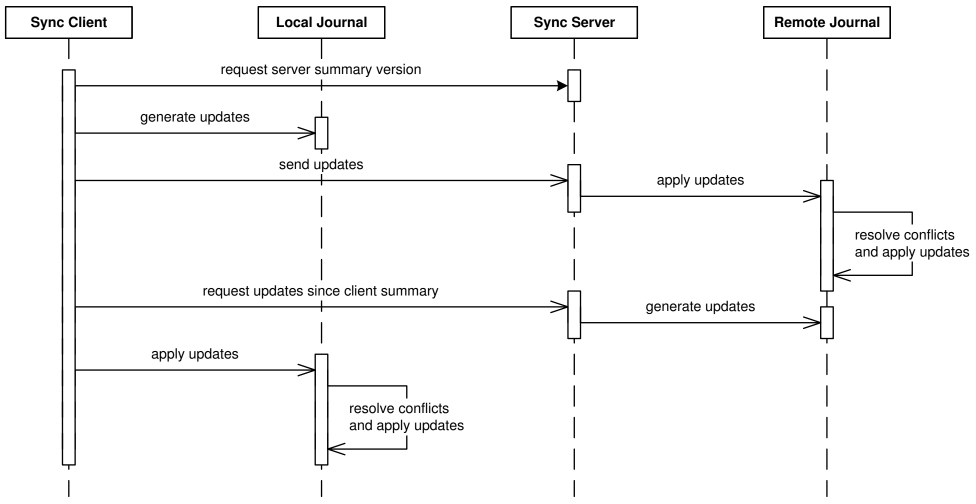

Sync Client
Local Journal
Sync Server
Remote Journal
request server summary version
generate updates
send updates
apply updates
resolve conflicts and apply updates
request updates since client summary
generate updates
apply updates
resolve conflicts and apply updates

Figure 5.6: *Synchronization protocol used between the synchronization client and server.*

5.5.2 Selective Synchronization

As described in section 4.6.1, a synchronization solution is required to support selective synchronization in order to limit the amount of data stored locally on the client machine and to prohibit unauthorized access to business objects in offline operation. The synchronization solution of the J2OO framework supports the concept of implicit data collections described in section 4.5. Both the server and the client part of the synchronization process may be configured with a list of data collections that specify which updates for modified objects have to be sent to the synchronization partner. While the client usually sends the complete list of updates to the server side, the latter may define a list of data collections depending on the connecting user and its privileges. The set of data collections has to be coordinated with and respect the application security policy. Objects not included in at least one data collection are not synchronized with the remote data store. On the server side, an application-specific implementation of the `IDataCollectionFactory` interface is used to determine the list of data collections configured for a particular user. Figure 5.7 illustrates the main J2OO framework classes that realize the synchronization solution.

The synchronization solution of the J2OO framework implements a mechanism to resolve the out-of-collection problem mentioned in section 4.6.5. The client sends a list containing the type and unique identifier for each object stored locally on the client machine to the server side. Thereon, the server checks the contents of the list received against the data collections currently configured for the particular user and creates a deletion command for each object that is no longer included in any data collection. The server sends the list of deletion commands back to the client machine where each deletion command is applied to the local journal. Each deletion command triggers the journal to remove a specific synchronizable object from the client's local data store. Depending on the

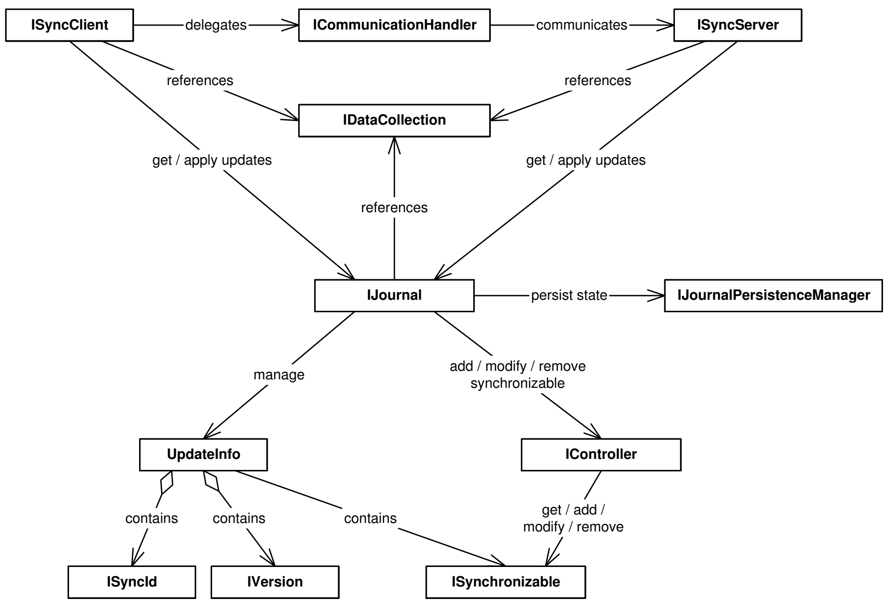


Figure 5.7: *Classes and interfaces of the J2OO framework synchronization solution.*

expected frequency of changes to the set of data collections configured for a particular user, the mechanism to remove out-of-collection objects from the client data store must be triggered more or less often, for example once after each login to the application client for rarely changing data collections, or during each synchronization process for highly dynamic data collection configurations

5.6 Operation Mode Transition

The operation mode manager component manages all transitions between different operation modes. The J2OO framework defines three distinct operation modes:

IDLE The operation mode IDLE refers to the initial phase after the enterprise application has been started. In this operation mode, calls to the service factory, the controller factory and the journal instance are not permitted. The operation mode IDLE may be used to perform tasks that do not rely on application services and where the the target operation mode is not yet defined, such as displaying a login dialog or a dialog that lets the user choose whether to start the application client in online or offline operation

ONLINE The operation mode ONLINE refers to the phase when the enterprise application is connected to the J2EE application server. In this operation mode, the online implementation for service factory and controller factory is used[3]. Object modifications are tracked by the server-side journal.

[3]Depending on the enterprise application architecture, the service factory and/or the controller factory implementation for online and offline operation may not differ.

OFFLINE The operation mode OFFLINE describes the mode of an enterprise application running in offline operation, disconnected from the J2EE application server. In this operation mode, the offline implementation for service factory and/or controller factory is used. Additionally, the local instance of journal is used to track object modifications.

An operation mode transition may either be initiated by the user on demand or by the watchdog component on a server failure or restore. Figure 5.8 illustrates the possible transitions between operation modes and their initiator actions. If the switch to a new operation mode fails, the previous operation mode is first tried to be restored. If this also fails, the operation mode IDLE is started.

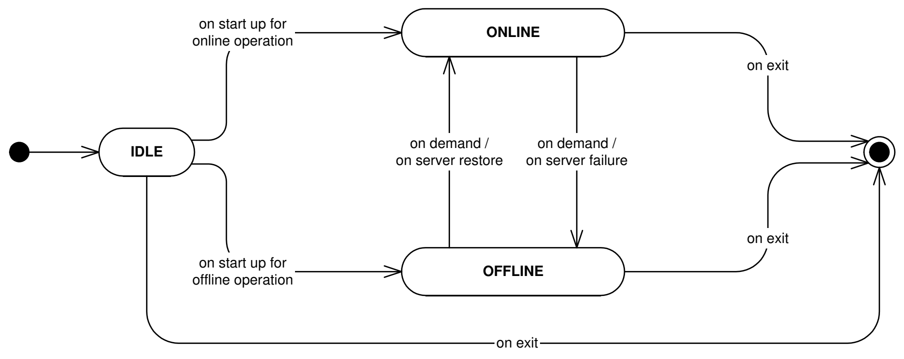


Figure 5.8: *J2OO framework operation modes and transitions between operation modes.*

During a transition to online or offline operation, the operation manager component uses an operation mode-dependent implementation of the authentication service in order to authenticate the user for the new operation mode. In addition, the synchronization manager is called to initiate a synchronization process for specific synchronization events.

5.7 Configuration

The J2OO framework must be configured and customized to reflect the actual enterprise application architecture and scenario in order to work properly. This is done by providing application-specific implementations for the `ClientSettings` and `ServerSettings` framework classes. Both classes define methods to retrieve several factory implementations and the implementation of the local journal. In addition, both settings classes define additional parameters to be used either on the client or on the server machine.

The concrete implementations of these settings classes have to be installed when the enterprise application is deployed or started. As the J2OO framework is used both on the client and the server side and reuses most of its classes in both environments, the mechanism specifying the concrete settings implementations must also be supported in both environments. The J2OO framework uses the Java Naming and Directory Interface (JNDI) to register the full qualified name of the settings implementation class with a

JNDI context in each environment. JNDI is part of the J2EE specifications and is also supported by newer versions of the Java 2 Standard Edition. The abstract `Settings` class provides a static method that reads the settings implementation class name from the JNDI context and creates an instance of that class. Other classes that need to read configuration settings may retrieve the settings instance using this static method.

In addition to the actual configuration, an enterprise application may register itself as listener to several framework components such as the `OperationModeManager` or the `SyncManager` in order to get informed on events like operation mode switches, synchronization process initiations or failures and data modifications.

5.8 Framework Integration

Most of the J2OO framework configuration parameters and integration points are covered by the actual implementation of the `ClientSettings` and `ServerSettings` framework classes. These settings classes have to provide several application-specific implementations for J2OO framework interfaces and classes:

Service Factory and Services Each service provided by the enterprise application both for online and offline operation has to implement the `IService` interface. In addition, a concrete implementation of the `ServiceFactory` class has to be specified in the settings implementation that is aware of creating application-specific services. Depending on the enterprise application architecture used, a separate implementation of `ServiceFactory` may be required for online and offline operation.

Controller Factory, Controllers and Business Objects Each business object in an enterprise application that needs to be synchronized between the client and the server machine has to implement the `ISynchronizable` interface and thereon implements the declared methods to update the object's state and reconcile conflicts between different versions of the same business object. In addition, associated with each synchronizable object, a controller instance implementing the `IController` interface must be provided. These controllers are both used by the journal during synchronization and by the enterprise application in order to add, modify and delete business objects. `IController` instances are responsible for notifying the local journal instance on synchronizable object creations, modifications and deletions. The J2OO framework provides an abstract controller implementation, `AbstractController`, that handles the notification of modifications to the journal instance. Finally, an implementation of the abstract `ControllerFactory` class that determines the concrete controller instance to be used for a specific `ISynchronizable` type has to be provided and specified within the settings implementations. Depending on the enterprise application architecture used, a separate implementation of `ControllerFactory` may be required for online and offline operation.

Journal The concrete `IJournal` implementation has to be defined in the settings implementation. If the enterprise application uses the default implementation of the `IJournal` interface, the `Journal` class, an environment-specific implementation of the `IJournalPersistenceManager` interface that is responsible for managing the persistence of the internal journal state has to be configured for the `Journal` instance. The J2OO framework provides two default implementations for the `IJournalPersistenceManager` interface: one to be used within an EJB container that persists the state of the journal as an Entity Bean and one that persists the state of the journal to the file system intended to be used on the client machine.

Synchronization Within the `ClientSettings` implementation, the enterprise application client has to specify an implementation of the `ICommunicationHandler` interface that is responsible for communication with the synchronization server on the server machine. The J2OO framework provides both `SyncClient` and `SyncServer` classes and default implementations for the `ICommunicationHandler` interface suitable for synchronization using the HTTP(S) and RMI/IIOP communication protocol. Additionally, the enterprise application client has to specify a list of `SyncTrigger` instances that represents the synchronization triggers on which a synchronization process has to be initiated. The J2OO framework defines a number of `SyncTrigger` instances that cover the most important situations where the initiation of a synchronization process is reasonable, for example before switching to offline operation, after switching to online operation, after login, or when a synchronizable object has been modified. For the `ServerSettings` implementation, the enterprise application needs to implement the `IDataCollectionFactory` interface that is responsible for creating a list of `IDataCollection` instances used to restrict the data to be synchronized with the client machine.

Authentication Authentication, both for online and offline operation, is initiated by the `OperationModeManager` class during the transition to a new operation mode. The `OperationModeManager` class relies on an implementation of the `IAuthentication-Service` interface that is determined by the `ServiceFactory` implementation specified for a particular operation mode. The credentials required for authentication have to be stored in the `SecurityContext` class and must implement the `ICredentials` interface. The `OperationModeManager` retrieves the current credentials from the `SecurityContext` and passes them to the login method of the `IAuthentication-Service` implementation.

Watch Dog If the enterprise application uses the watchdog functionality of the J2OO framework to observer server availability and to initiate the transition to offline operation on server outages, an appropriate implementation of the `IWatchDog` interface has to be provided. This implementation completely encapsulates the business logic for server availability checks. The `IWatchDog` implementation may also rely on server-side watchdog components.

Application Client Startup As the enterprise application client relies on the environment defined by the J2OO framework, this environment has to be prepared initially and set up by the framework. This is done by the `ApplicationLauncher` class that is configured with the name of the concrete `ClientSettings` implementation and the enterprise application client main class. The application client main class must implement the `IApplication` interface which specifies a start method. Before this start method is called by the `ApplicationLauncher` class, no J2OO framework classes and settings are allowed to be called because the environment is not yet completely set up.

Section 6

Architectures for Offline J2EE Applications

This section describes architectures for offline-capable enterprise applications based on the J2EE platform and the Java programming language. The AddressBook enterprise application provided on the resource CD (Appendix A) implements a simple address book application based on the offline fat client, the offline rich client and the offline web application architecture discussed in the following sections. The sample application demonstrates the differences regarding infrastructure requirements on the client machine, the procedure during operation mode transition and partitioning of application logic between the client and the server. The AddressBook application uses the J2OO framework described in section 5. Additional information on starting and exploring the AddressBook application may be found in appendix B.

6.1 Offline-Architecture for Fat Client Applications

Providing an offline-capable architecture[1] for fat client applications as described in section 2.5.2 is strongly influenced by the fact that the complete application logic and data already resides on the client side. This simplifies the reuse of application functionality for offline operation since no further effort is required to transfer the application logic to the client machine. Figure 6.1 shows the essential parts of the offline-capable architecture for fat client enterprise applications. For offline operation, an additional mechanism for storing data locally is required. The concept of an abstract controller factory may be used to substitute the business object persistence mechanism when switching to a different operation mode. While the controller instances for online operation access the server-side database, instances for offline operation may either use a locally installed database management system or store business objects to the file system.

[1] 'Offline-capable architecture' and 'offline architecture' both denote a J2EE-supported architecture for enterprise applications that may run both in online and offline operation.

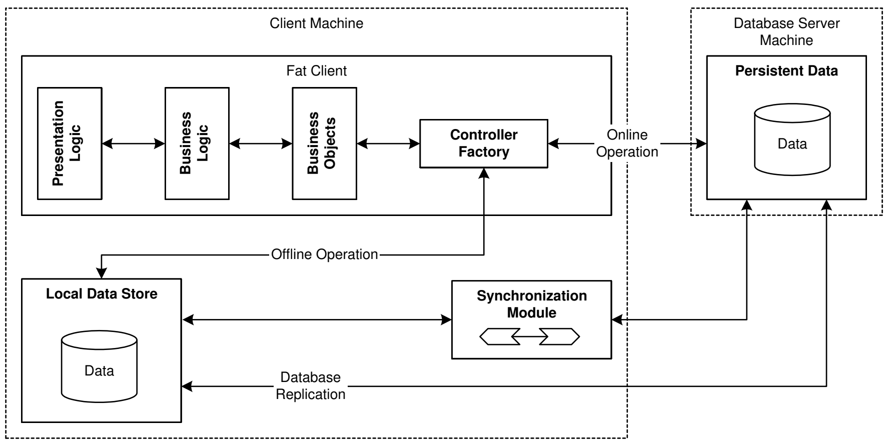


Figure 6.1: *Architecture for offline-capable fat client applications.*

6.1.1 Synchronization

The synchronization process is required to reconcile the local data store with the database on the server machine. Depending on the technology used to store data on the client, the following approaches are conceivable.

Database Replication If the client uses a JDBC-aware database system as local data store for offline operation, replication may be achieved using a vendor-specific replication tool either based on JDBC or on a proprietary communication protocol. Integration of such a replication tool into the fat client application may be crucial in order to support synchronization during application runtime. A completely external replication tool may not allow the Java-based application to trigger a synchronization process and therefore, synchronization has to be initiated outside of the enterprise application. Using a replication tool delegates the responsibility of tracking changes and reconciling conflicts to the tool and frees the enterprise application from these tasks. Although, this may limit the flexibility concerning conflict resolution and application-specific synchronization handling.

Synchronization Module If the local data store implementation is based on files or the database management system used does not offer a replication mechanism, a separate synchronization module is required. Since the server machine may offer neither a J2EE-compliant application server nor a Java runtime environment, the server machine is not able to play an active role during synchronization. All the processing has to take place on the client machine. Therefore, the synchronization module running on the client machine has to compare the business objects from the local data store and the ones from the server database, to merge the two versions and to apply the resulting modifications either to the local data store, to the server database or to both stores. The usage of the journal-based synchronization approach described in section 5.5 has to be enhanced in a manner that the database is able to keep track of the changes during application operation, since these changes must be managed centrally and not distributed across multiple clients. One feasible approach is to provide a JDBC-based journal implementation that runs in the runtime environment on the client machine, but propagates each modification notification directly to the database on the server machine. If supported by the database management system, a solution based on database-inherent trigger mechanisms that automatically maintain update information on each modification applied to the business objects stored in the database liberates the fat client from maintaining the server journal. A degenerated journal implementation only responsible for generating and applying updates may be used during synchronization.

Summarized, the fact that the server does not offer a Java runtime environment requires additional effort and a specialized solution regarding the data synchronization mechanism.

6.1.2 Operation Mode Transition

The actions to be performed during the transition between online and offline operation are limited to synchronization and to the replacement of the data store currently active for the enterprise application. While the synchronization is described in section 6.1.1, the replacement of the data store may be achieved by configuring the controller factory for offline operation, so that subsequently calls to controller instances access the local data store.

6.1.3 Security

The fat client application implements security-related features on the client machine and therefore does not rely on J2EE technologies for enforcing the application security policy. Besides the location of the authentication and authorization data, the application security is operation mode-independent and does not differ from online to offline operation. Authentication and authorization may be performed using JAAS or any other, potentially application-specific mechanism. To access the server-side data storage, the fat client uses the credentials of a database account. It may be useful to introduce a database account on a per-user base rather than a global database user in order to provide the flexibility for restricting access to the database depending on a particular user.

For authentication in offline operation, an offline authentication approach, similar to the one described in section 4.8.3, is required. The fat client itself is responsible for storing the authentication data in an appropriate manner in the local data store in order to enable the offline login module to verify to user credentials provided during offline authentication.

6.2 Offline-Architecture for Rich Client Applications

Figure 6.2 shows an overview of the offline architecture for rich client enterprise applications. Compared to a fat client architecture, the client part of a rich client application

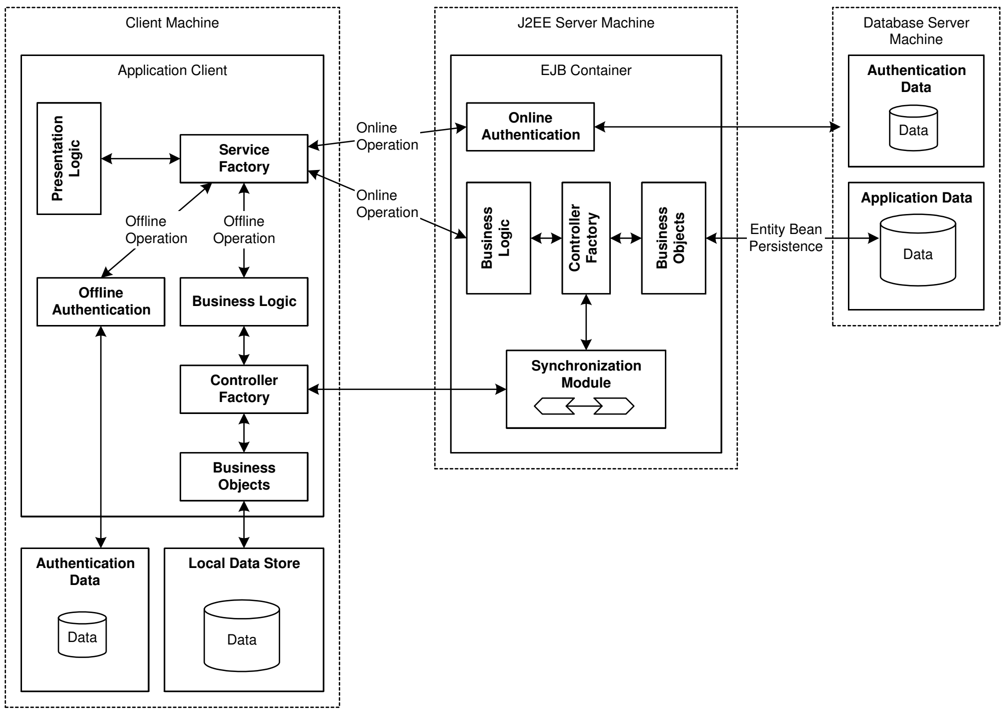


Figure 6.2: *Architecture for offline-capable rich client enterprise applications.*

only consists of the presentation logic located on the client machine, and excludes the business logic and the persistence management for business objects. In order to run the rich client application in offline operation, the missing application logic and business object management have to be made available on the client machine. According to the concepts described in section 4.7.2, the business logic and management of business objects may be reused by implementing both a service and controller factory that determines the correct service implementation for each service depending on the current operation mode. If running in online operation, the enterprise application is configured with a service factory that delivers a service stub for services implemented as Session Beans in the business logic layer on the server machine. The controller factory returns a controller implementation that handles business objects implemented as Entity Beans. In offline operation, the service factory returns a local service implementation for each service that is available for offline operation and the controller factory creates controllers aware of persisting business objects to the local data store. Figure 6.2 does not show a potentially co-existing web container on the server side. A web container may be integrated if the application client is using the HTTP(S) protocol for communication. If a web container is part of the rich client architecture, the client machine is also required to host a HTTP(S)-aware web container. This leads to additional infrastructure requirements for the client machine.

6.2.1 Synchronization

Depending on the enterprise application design, the synchronization module communicates either with an EJB component for EJB-centric designs or with a Servlet component for web-centric components. It is useful to base the synchronization module on as much existing application logic as possible in order to profit from existing functionality and security-related features. For example, it is a reasonable approach to reuse the controller factory when modifying business objects during synchronization and using the security constraints implemented therein instead of directly manipulating the data in the corresponding data store. The J2OO framework synchronization solution described in section 5.5 respects these requirements and offers a configurable synchronization solution for rich client applications.

6.2.2 Operation Mode Transition

The application logic for offline operation may either already be included in the application client used in online operation or may be downloaded when switching to offline operation. The 'be-prepared'-configuration described in section 4.3.2 requires the former method because no additional effort can be carried out when coincidentally switching to offline operation triggered by a server failure. The 'on-demand'-configuration from section 4.3.1 supports both manners. It is even conceivable to provide a dedicated offline application client whose appearance and functionality is adapted to the requirements of the offline mode of a particular enterprise application. In either case, both the online- and offline-specific application client components may be pre-installed on the client machine or may be downloaded entirely on-demand using an appropriate network download service such as JNLP. In addition, the service factory has to be replaced by the offline-specific implementation that creates offline services using the offline controller factory for accessing business data.

6.2.3 Security

Offline authentication may be performed by implementing a local authentication service that uses locally stored credentials to verify the login data provided by the user. Because of its wide support both in the standard Java and J2EE environment, a JAAS-based authentication system as described in section 4.8.3 may be useful. Again, the service factory may be used to determine whether the local or the server authentication service has to be used. Since authorization is performed on the server machine in either way, using offline authorization is optional, depending on the particular enterprise application scenario. Offline authentication and authorization may be useful to enhance the user's experience by disabling application features that the user is not authorized to use, but should not replace server-side authentication or authorization, as mentioned in the offline security guidelines in section 4.8.1.

6.3 Offline-Architecture for Web Applications

The web application architecture is the most server-centric architecture within the J2EE-proposed architectures. The enterprise application runs completely on the server machine, using a generic presentation engine, a web browser, on the client machine. Whereas this architecture simplifies development, operation and maintenance because of the centralized approach for online operation, running a web application in offline operation requires intense preparation of the client machine. Not only the complete application logic and parts of the data must be transferred to the client machine, but also additional infrastructure services may have to be installed first. Figure 6.3 shows the architecture for offline-capable web applications.

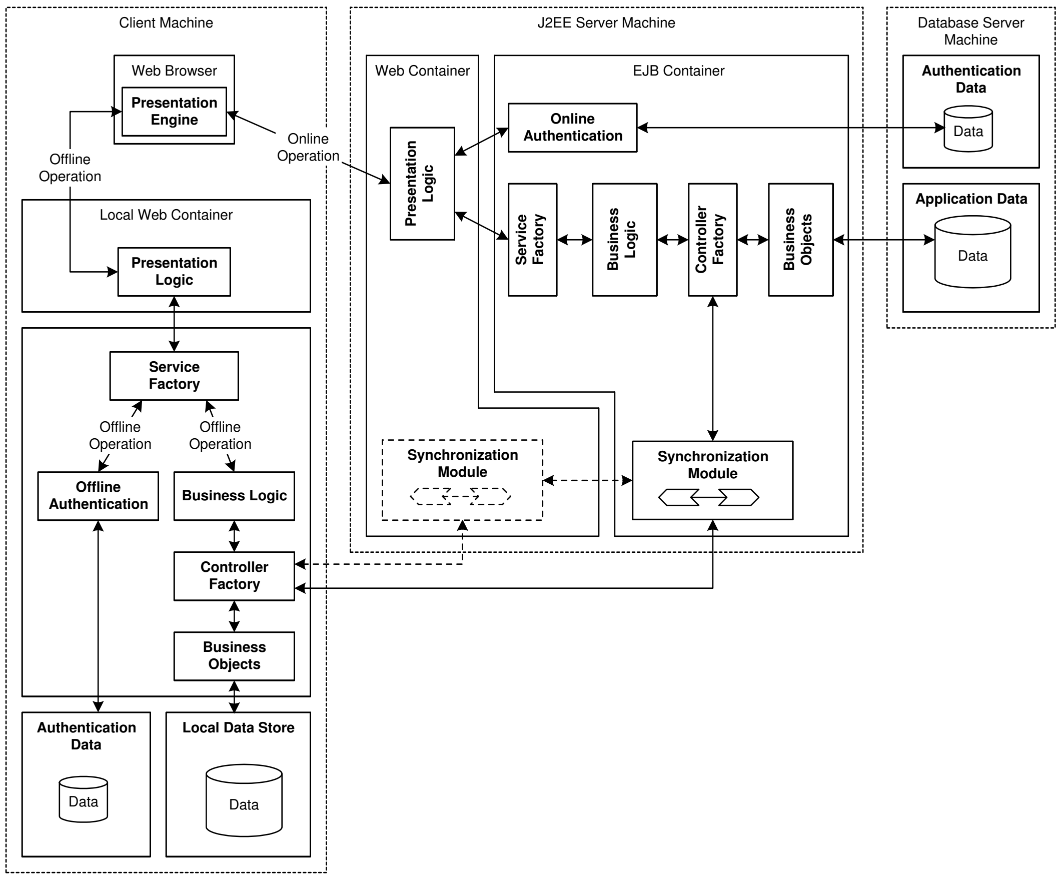


Figure 6.3: *Offline architecture for offline-capable web applications.*

A local web container that contains the presentation logic and responds to web browser requests is required for offline operation, since a web browser only supports the HTTP(S) protocol for communication. Business logic and business object management may be solved similarly to the approach for offline-capable rich client applications if an EJB-centric application design is presumed. Unlike the rich client application architecture, not the client component, but the web component implementing the presentation logic within the web container uses the service factory. In a web-centric application design, parts of the business logic including the service factory may be moved directly in the web container.

The local web container may either be pre-installed on the client machine or may be downloaded and installed from the server machine on demand. Section 6.3.2 proposes a JNLP-based procedure to install a complete offline web application including the required infrastructure on demand. Whereas on the server machine, web components typically consist both of Servlet components to process requests and JSP components to display the user interface, JSP components used in offline operation may be pre-compiled to Servlet components on the server machine in order to reduce the technology requirements on the client machine. Thereon, the client machine may run a lightweight web container only supporting the Servlet API, but not the constitutive JSP API. A lightweight web container may differ significantly from a complete J2EE-compliant web container with respect to size, download and installation time, but also regarding performance and scalability.

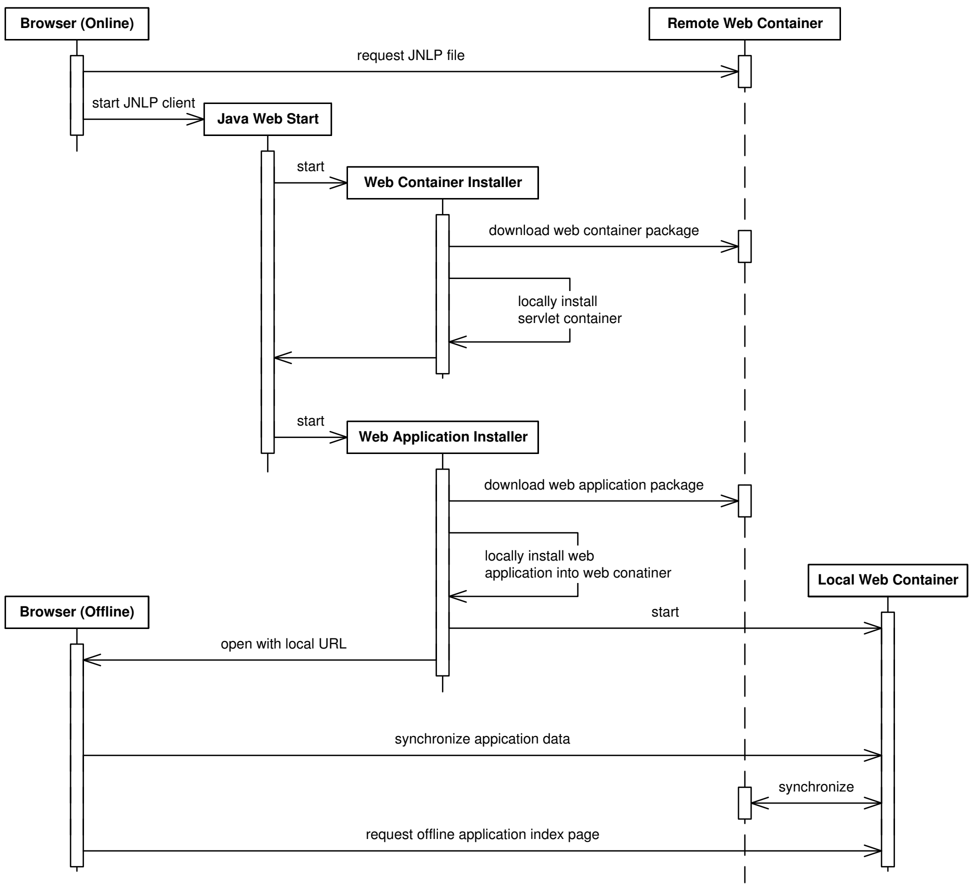
6.3.1 Synchronization

Since the synchronization solution is influenced primarily by the design of the business logic and business objects and not by the presentation logic, the synchronization for web applications does not differ substantially from the one used for rich client applications. The client part of the synchronization is implemented by a web component residing in the local web container. Consequently, the user may initialize a synchronization process by referring to an appropriate URL using the web browser or the application by calling the synchronization client web component directly. Alternatively, a signed Java applet component running in the web browser may be used for the client-side synchronization component.

6.3.2 Operation Mode Transition

The extended infrastructure requirements for running a web application in offline operation may either be pre-installed on the client machine or may be downloaded and installed on demand. In the latter approach, the infrastructure components do not exist on the client machine until the first transition to offline operation is initialized. This requires an appropriate procedure that prepares the client machine during the operation mode transition but leads to the advantage that only client machines actually required to run the web application in offline operation are installed with the infrastructure prerequisites. Additionally, newer versions of both the enterprise application and the infrastructure services may be downloaded and installed using the same procedure.

Within the Java environment, JNLP offers services to download and install applications and services on demand. Figure 6.4 illustrates the schematic sequence of installing infrastructure and enterprise application components locally on the client machine and starting the web application in offline operation. While running the web application in online operation, the user chooses to switch to offline operation and requests a JNLP configuration file from the web container by referring to an appropriate HTTP URL link in the online web application client. The web browser launches the Java Web Start [Sun03d] client, the reference implementation of JNLP, referred to as the JNLP client. This JNLP client parses the provided JNLP configuration file and starts the web container installer first. This installer is a custom Java program, responsible for downloading a local web container made available as JAR file on the server-side web container and for installing the



Browser (Online)
Remote Web Container
request JNLP file
start JNLP client
Java Web Start
start
Web Container Installer
download web container package
locally install servlet container
start
Web Application Installer
download web application package
locally install web application into web conatiner
Local Web Container
Browser (Offline)
start
open with local URL
synchronize appication data
synchronize
request offline application index page

Figure 6.4: *Installation sequence of an offline web application.*

local web container on the client machine. Thereon, the JNLP client launches the web application installer that is responsible for downloading and installing the offline web application, also prepared as JAR file, into the local web container. Finally, the JNLP client starts the local web container and points a new web browser window to the local URL of the offline web application. Before displaying the web application's start page, the local web container synchronizes the local data store with the data store on the server. This approach assumes that the local data store is implemented based on files, thus no local database management system is required on the client machine, but a lightweight database management system may have been downloaded and installed locally using a procedure similar to the one used to install the local web container.

Since the web browser itself is not capable of automatically referring to a different URL if the online web application cannot be accessed, a transparent, dynamic transition to the offline web application is not feasible. After the web browser presents a 'page-not-found' error, the user may decide to point the browser to the URL of the offline web application. Also, if the local web container and the offline web application is not yet installed locally, a switch to offline operation on a server or communication failure is not possible. Therefore, the 'be-prepared'-configuration described in section 4.3.2 is not completely supported for offline web applications.

6.3.3 Security

As for rich client applications, authentication and authorization for web applications must be performed on the server machine running the online web application, but may be adapted in offline operation in order to enhance the user experience. In contrast to rich client applications, the web browser used as client component for web applications does not offer a standard way to locally store user credentials during online operation and to reuse them to silently re-authenticate the user for offline operation. Thus, the application user has to re-authenticate when switching between online and offline operation in web applications.

Section 7

Demonstration Application

This section describes the demonstration application implementing a simple, but business relevant use case for an enterprise application that supports both an online and offline operation mode. The use case is similar to the one described in section 3.2. Besides acting as 'proof of concept' application for the J2OO framework described in section 5, this application allows the demonstration of different offline operation modes and shows activities happening behind the graphical user interface visible to the standard user. Although the use case is designed for simplicity, it covers most aspects of an enterprise application listed in section 2.2, such as a multi-tier architecture approach, multiple users in different roles, and application-specific security constraints. The demonstration application is provided as Java source code with building scripts and runtime prerequisites on the resource CD (Appendix A). Additional information on running and exploring the demonstration application may be found in appendix C.

7.1 Application Use Case

Related to section 3.2, the demonstration application realizes an expense management tool. Figure 7.1 shows the use case diagram for the expense management application. Employees may acquire and manage expenses disbursed for company's concerns and are only allowed to browse their own expense records. Each expense record entered must be approved by the employee's manager. Managers may therefore browse and approve the expense records of each of their subordinates, but managers are not allowed to modify or remove an employee's expense record. An administrator is able to create, modify and remove employees, to set up the relation between managers and subordinated employees and to manage the permissions for each user. Besides this base functionality, additional features are imaginable such as generating monthly reports of expenses for all subordinated employees or maintaining employee profiles.

Figure 7.1 highlights the parts of application functionality that are available both in online and offline operation and the ones only provided for online operation (shaded in gray

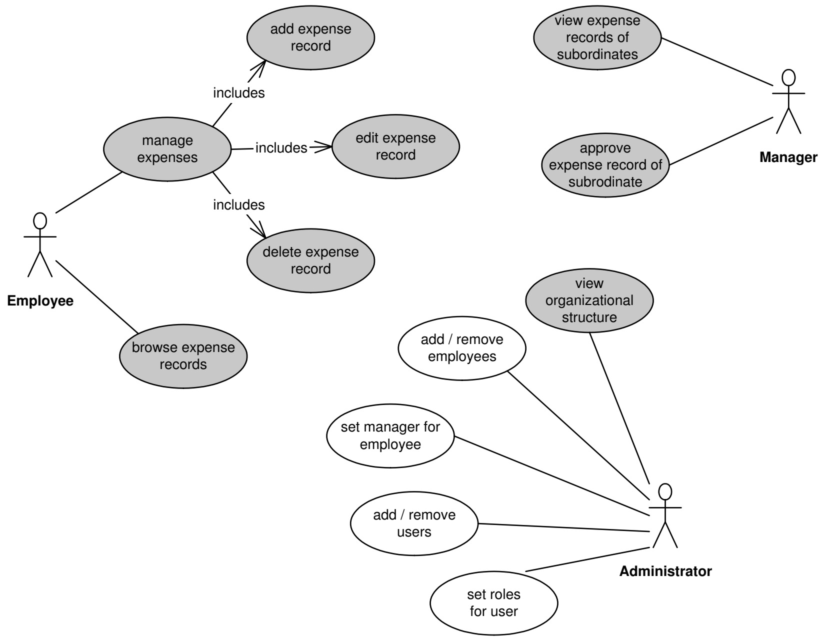


Figure 7.1: *Use case diagram for the expense management application scenario. Areas shaded in gray represent use cases that are only provided for online operation.*

in figure 7.1). Managing user accounts and the organization structure, as well as generating data-intense reports may not be appropriate or reasonable during offline operation, as shown in section 4.7. Although, browsing the company's organizational structure or displaying pre-generated and stored reports may be possible in offline operation for selected scenarios.

7.2 Architecture and Design

The demonstration application is based on an offline-capable rich client architecture as described in section 6.2. On the server side, an EJB container with an attached relational database management system is used to handle business objects and to provide the business logic. The client application realized as JNLP-enabled rich client communicates directly with the EJB components on the server machine using the RMI-IIOP protocol. The application's design follows the approach proposed in section 4.7.2 and uses a service-based design. The application functionality is grouped in services, each service implementing a relevant part of the business logic. Each service uses one or multiple controller instances that are responsible for creating, retrieving and modifying business objects. On the server side, both services and controllers are wrapped in Session Beans. Business objects are implemented as value objects and are encapsulated in Entity Beans on the server side. In offline operation, value objects are managed by a generic controller instance that is responsible for persisting any type of business objects to the file system. Figure 7.2 illustrates the main components of the demonstration application such as participating services, controllers and business objects and highlights their interaction.

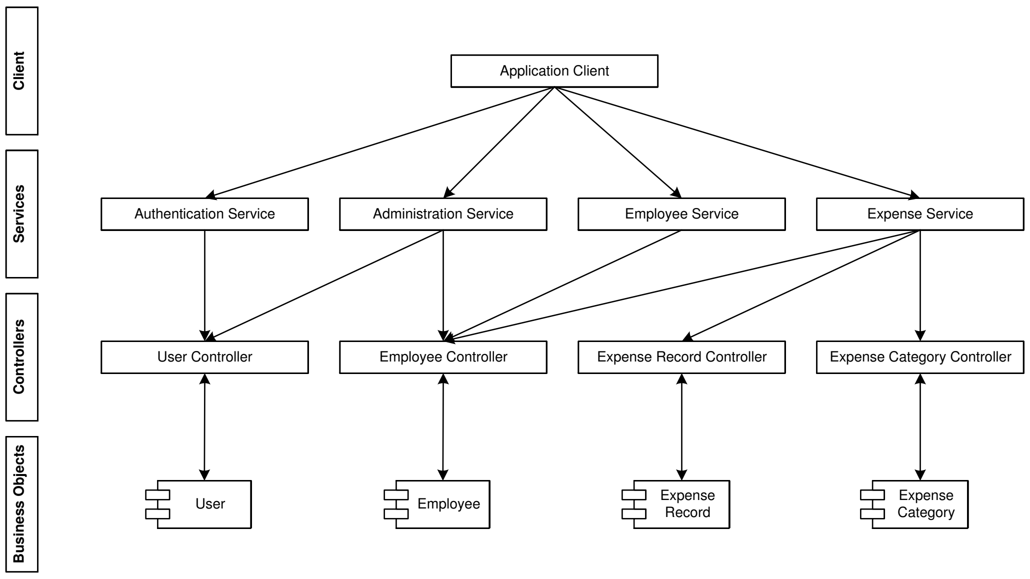


Figure 7.2: *Main components of the demonstration application and their collaboration.*

The following provides a description of the relevant components:

Client The graphical part of the application running on the client machine calls service implementations directly. Controllers and business objects are not directly accessible by the client.

Services The authentication service provides methods for user login and logout. The administration service dedicated to users owning the administrator privilege offers the functionality to retrieve, add and remove users, to set their privileges and password and to retrieve a list of all security roles. The employee services specifies methods to add and remove employees, to set employee details and to retrieve collections of employees depending on different criteria, such as all existing employees or subordinates for a specific manager. The expense service offers the core expense management functionality such as creating, modifying and removing expense records, managing expense categories and approving existing expense records.

Controllers A controller instance exists for each type of business object. All controllers implement the same controller interface defining methods to add a new, modify or delete an existing or return a specific or all existing business objects of a particular type.

Business Objects The user business object represents an application user able to login to the expense management tool. It contains the username, password and a list of privileges with the security roles. The employee object describes a company's appointee, defining the name, an underlying application user and a responsible manager. An expense record represents a single expense, having a date, description, amount and an expense category associated with it. Each expense record belongs to a single employee, referred to as the expense owner.

7.3 Security

The demonstration application implements the security concepts described in sections 4.8.1 and 4.8.3, following the 'accept-as-insecure' approach. Therefore, all security relevant checks are performed on the server side. Offline authentication using digitally signed and locally stored user credential objects are only used to protect access to the application client and to customize the application client's look and feel. The server enforces the application security policy during synchronization of modified data. The application client defines security roles required to execute particular actions such as displaying the company's organizational structure or browsing a list of expense records for a particular employee. Within the application client, these restrictions are only used to adapt the look and feel of the user interface and to enable or disable application features for the user's convenience.

As proposed in section 4.8.1, the minimal amount of data required for offline operation is transferred to and stored on the client machine. Depending on the user's privileges and preferences, the synchronization server decides which data is sent to the client during synchronization. This decision mechanism is implemented using a combination of data collections which are part of the framework described in section 5, and programmatic security constructs mentioned in section 2.4.3.

Access to services and business objects on the server machine is realized using the standard J2EE security concepts: declarative and programmatic security. Whereas access to several service methods is controlled completely by the fact that a user owns a specific security role or not, other service methods may be accessed with different security roles, but on different access levels or may adapt their behaviour depending on the particular user currently logged in, known as instance-based authorization. A call to the administration service for example is restricted completely to users owning the 'administrator' privilege, where methods directly modifying an expense record business object are accessible both for user with the 'employee' or the 'manager' role: employees may modify expense details like date, description or amount, but only managers are allowed to modify the approved-flag of an expense record. Instance-based authorization is required if a manager requests a list of all subordinated employees. The resulting list returned to the caller depends not only on the caller's privileges, but mainly on the specific identity of the caller.

Security constraints are enforced either on the service level, on the controller/business object layer or on both. Since the synchronization process accesses and manipulates business objects not using the provided application services, but directly using controller instances, securing access to service methods is not sufficient. As postulated in section 4.6.1, the synchronization process must not tamper with business objects regarding the application security policy. In order to avoid duplicated security constraint checks that both lead to duplicated code and decreased performance, security checks may be moved back as far as appropriate towards the controllers and business objects.

7.4 Synchronization

Synchronization within the demonstration application is realized by integrating the synchronization solution described in section 5 and by using the EJB-based default implementation for the synchronization client, server and communication handler. The application uses the synchronization manager responsible for initiating and controlling the synchronization process. Synchronization is requested at several points within the application, for example after logging in, during transition to a new operation mode, or after the server restores again after a failure. Depending on the application configuration defined, the synchronization manager is configured for specific synchronization events and only starts synchronization processes actually triggered by one of the configured synchronization events.

An application-specific inspector feature allows the user to inspect the state of the local journal and the data currently stored on the local machine. If running in online operation, the inspector shows the internal state of the server journal and all data currently residing in the server data store. The inspector feature observes which data has been modified in which data store and highlights synchronizable objects that are not yet synchronized with the remote replica. In addition, a log window displays actions and potential error messages triggered by the underlying J2OO framework using the listener concepts of the framework.

7.5 Application Configurations

The demonstration application supports both the 'on-demand' and the 'be-prepared' configuration described in section 4.3. Each configuration is represented by a separate settings implementation that specifies the required parameters such as service and controller factory implementations, journal implementation and synchronization events.

7.5.1 'On-Demand' Configuration

When running the expense tool application with the 'on-demand' configuration, the user explicitly chooses whether to login in for online or offline operation. Depending on the choice, the user is required to authenticate either with the credentials database on the server or with the locally stored, digitally signed user object described in section 4.8.3. Logging in for offline operation requires the expense management application to have previously synchronized the offline authentication credentials to the local data store. After successfully authenticating, the user works in the chosen operation mode until deciding to switch to another operation mode. During the transition sequence, the synchronization process is initiated in order to download data required for offline operation onto the local machine or to upload data modified in offline operation to the server. In addition, application functionality not available in the new operation mode is disabled, other functionality exclusively provided for the new operation mode is enabled. All windows displaying application data are refreshed in order to show the data actually available in the specific operation mode. Switching between operation modes does not require the user to re-enter the authentication credentials since the demonstration application caches the credentials on login and uses these credentials when authenticating for the new operation mode. When running the demonstration application in online operation, the user may initiate the synchronization process on demand. The 'on-demand' configuration does not offer enhanced mechanisms for error handling as required by the 'be-prepared' configuration, especially not regarding server or connectivity problems. Occurring problems are simply displayed to the user. If an error occurs during operation mode switch, the previous operation mode is tried to be recovered. If this fails for any reason, the user will be remain in the logged out state.

7.5.2 'Be-Prepared' Configuration

The 'be-prepared' configuration, as described in section 4.3.2, transparently handles connectivity problems leading to offline operation of the application. Therefore, there is no explicit choice whether to login in for online or offline operation. The application itself tries to login with the authentication data on the server. If this fails because the server is not accessible or not responsive, the application automatically switches to offline operation and re-tries the authentication using the provided user credentials with the previously locally stored authentication objects. During work with the application in the default operation - online operation - each request is first sent to the server-side service on a 'try-catch' base, re-sending the request with the local service implementation if the server is down.

As the server or the communication infrastructure between the client and the server may be interrupted at any time, continuous synchronization is initialized right after logging in. The synchronization is executed asynchronously since the graphical user interface should not be blocked during the periodical synchronization. Each user action leading to modification of data triggers a new synchronization process. If the user interface remains idle for a certain time, a background synchronization timer initiates the synchronization process as data may be modified on the server, resulting from actions applied by other connected application clients.

A watchdog component running in the background periodically checks for server availability. If the server is not available or a communication timeout occurs, the watchdog initializes a switch to offline operation. Thereon, the watchdog remains listening to server responses on alive-check requests. As soon as the server is up and running again, a switch to online operation is initialized. Whereas during a failover to offline operation, synchronization may no be performed since the server will not respond any synchronization requests, an explicit initiation of the synchronization process is performed when switching back to online operation. This is recommended since data modifications during offline operation must be expected. These modifications should be synchronized with the server side as soon as possible. When running in offline operation, the synchronization background thread and the background timer are suspended, also the synchronization on data modifications are suppressed. In the demonstration application, the 'be-prepared' configuration does not offer to initiate the synchronization process on demand since this is not required in this transparent run mode.

Both the background synchronization and the watchdog functionality are provided by the J2OO framework and are configured by the demonstration application using an appropriate settings implementation for the 'be-prepared' configuration.

Section 8

Conclusions

This section concludes the thesis with a discussion of the concepts proposed and results achieved.

8.1 J2EE and Offline Operation

Basically, J2EE is an online-specific technology that builds on a reliable, permanent connection between the client components on the client machine and the web, EJB and back-end components on the server machine(s). This is contrary to offline operation where an enterprise application must be able to complete its tasks without support from the server side. In case the client machine does not run a J2EE-compliant application server, most J2EE-specific technologies are not available on the client side and must be replaced by other standard Java-based technologies. This fact leads to a trade-off between pure, properly designed J2EE enterprise applications intended for online operation and offline-capable enterprise applications that try to abstract from the J2EE-specific parts and to reuse application components for both operation modes. The more J2EE-specific technologies used on the server side for online operation, the more expensive is the provisioning of offline operation since a larger part of the server-side components must be re-factored or re-implemented to be able to run on the client machine.

Porting existing enterprise applications towards offline-capability may lead to a significant effort depending on the underlying design of the existing enterprise application. Core business logic and business object representation have to be detached from J2EE-specific technologies such as EJBs or have to be re-implemented for offline operation. Against the background of a well structured, highly modularized and maintainable system with minimal duplicated code, re-implementing existing parts of an enterprise application for offline operation is not the recommended option. When building an offline-capable enterprise application from scratch, an adequate, offline-capable architecture as proposed in section 6 reduces efforts resulting from offline-specific issues. In conjunction with a modular application design discussed in section 4.7.2 that abstracts from J2EE-specific technologies, this approach provides both online and offline operation based on a common code base.

Nonetheless, since J2EE offers a wide range of support for enterprise applications, generally dispensing with any J2EE-specific features is not recommended. Concepts such as persistence management, database connection pooling and transaction management provided by the J2EE containers facilitate and accelerate the development of enterprise applications. As online operation is the prevalent operation mode for most offline-capable enterprise applications, performance, security and scalability issues are especially relevant for the J2EE-based online operation mode.

8.2 Impact of the Enterprise Application Architecture

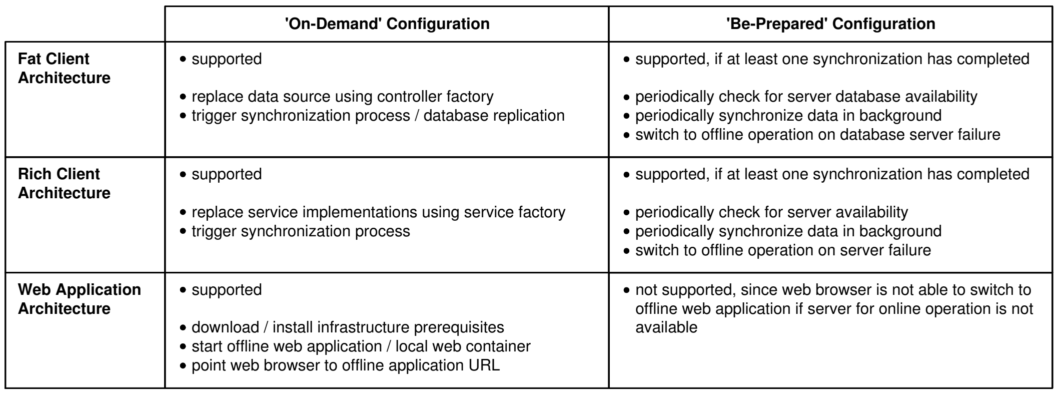
Comparing the enterprise application architectures specified by the J2EE platform and the resulting solutions for offline-capable enterprise applications proposed in section 6, the following items may be summarized regarding the architectural impact on the offline-capable enterprise application solution:

Offline Infrastructure Prerequisites The more server-centric the architecture and thus, the thinner the application client is for online operation, the more infrastructure is required on the client machine in order to run the enterprise application in offline operation. While the fat client architecture - the most client-centric architecture described - only requires a mechanism for locally storing the business data, the architecture for web applications - the most server-centric architecture - depends on a local, HTTP(S)-capable web container for presentation and interaction handling of the web browser-based user interface.

Data Synchronization The solution for application data synchronization is widely independent from the actual enterprise application architecture since synchronization mainly relies on the business data layer, that is similar in all architectural variants discussed. By providing an architecture-specific communication integration, the synchronization solution proposed in section 5 may be applied to each of the discussed architectural scenarios. Although, for scenarios where the server machine does not provide a Java runtime environment, such as if using the fat client architecture, the data modification tracking and synchronization must be managed by the application client. As this important part regarding the central data store is managed peripherally, problems in connection with concurrent modifications of business objects and their journal information must be expected. Specific architectures may offer the possibility for alternative synchronization solutions, such as the database replication approach for offline-capable enterprise application architectures that uses a database management system on the client machine which is compatible with the one running on the server machine.

Operation Mode Transition Procedure While the procedure executed during an operation mode transition is mainly controlled by the application configuration used within the enterprise application, the architecture significantly influences the first few steps of the procedure. Before being able to locally store and synchronize application data between the client and server data store and to start the enterprise application in offline mode, the transition procedure has to set up and prepare the additional infrastructure required on the client machine. Depending on the architecture, this may range from starting an already existing local database management system for the fat client scenario to downloading, installing and configuring a complete web container with the offline enterprise application for the web application architecture. The transition procedure may further be detached from the application architecture by pre-installing all infrastructure prerequisites on the client machines or simplified, if the environment on the client machine has already been set up during a previous operation mode switch.

Application Configurations Not all application configurations proposed in section 4.3 are supported in the same manner for all architecture variants. Figure 8.1 summarizes the configurations supported for each architecture and lists potential preconditions and actions required during the operation mode transition for each configuration.



	'On-Demand' Configuration	'Be-Prepared' Configuration
Fat Client Architecture	• supported • replace data source using controller factory • trigger synchronization process / database replication	• supported, if at least one synchronization has completed • periodically check for server database availability • periodically synchronize data in background • switch to offline operation on database server failure
Rich Client Architecture	• supported • replace service implementations using service factory • trigger synchronization process	• supported, if at least one synchronization has completed • periodically check for server availability • periodically synchronize data in background • switch to offline operation on server failure
Web Application Architecture	• supported • download / install infrastructure prerequisites • start offline web application / local web container • point web browser to offline application URL	• not supported, since web browser is not able to switch to offline web application if server for online operation is not available

Figure 8.1: *Supported application configurations depending on the enterprise application architecture, their preconditions and actions during operation mode transition.*

Security Realization of enterprise application security for online operation is entirely based on the server-side security concepts. For the rich client and web application architecture, the described J2EE security concepts are suitable. For the fat client architecture, server-side security is limited to privileges supported by the database management system. For offline operation, the security solution depends mainly on offline-specific security issues and the offline security approach (section 4.8.1) selected rather than on the architectural aspects. The technology used for the local data store and the access control mechanisms provided by the operating system highly affects the effort required to secure the client environment. Only a well-protected operating system, as well as variety of additional boundary conditions that are fulfilled may allow for an adequate security level on the client machine.

To summarize: although the fat client architecture supports the development of offline-capable enterprise applications, it leads to monolithic applications that do not benefit from the J2EE technologies provided. In addition, tracking changes in the central data store is complicated. Web applications rely completely on server-side J2EE concepts and use a wide range of J2EE-specific features, but pose significant infrastructure requirements for the client machine to run the web application in offline operation. Enterprise applications based on the rich client architecture rely mainly on server-side technologies for core business logic and business objects, but offer an appropriate client-side environment that facilitates the support of an offline operation mode.

8.3 J2EE Offline Framework of Application Developers

Providing an offline operation mode to an enterprise application leads to additional costs and complexity compared to a standard J2EE online enterprise application. Offline-specific issues like modification tracking, data synchronization and offline security have to be tracked down and resolved for every offline-capable enterprise application. In order to reduce the additional effort to implement an offline operation mode, application developers may be supported with practical solutions in the form of reusable libraries and implemented frameworks. By providing a framework-based, configurable solution to application developers, the development of an offline-capable enterprise application may be facilitated and accelerated.

This thesis proposes and implements a framework for offline-capable J2EE enterprise applications that is aimed to solve omnipresent offline-related issues in an general, but configurable manner. Thus, tasks like synchronization and operation mode transitions must not be re-implemented, but may simply be configured for and adapted to a specific enterprise application project. The framework described in section 5 offers a flexible way to configure parameters such as the point of time, the mode and the set of data to be synchronized and the communication protocol to be used during the synchronization process. Other aspects such as data modification tracking or update generation should be supported and resolved by the framework, but must not stringently be configurable as the default implementation is expected to be suitable for most scenarios.

8.4 Limitations concerning Application Functionality

While theoretically, arbitrary application functionality may be provided for offline operation (potentially with additional requirements concerning storage space, network traffic and performance on the client machine), it may not be reasonable to support several kinds of application features in offline operation. For example, features that either require a large amount of or especially sensitive application data to be stored on the client machine, features that are absolutely time critical related to results that have to be reflected immediately in the server data store, or actions that are not very time-consuming to complete from the user's point of view, but expensive to realize in offline operation. In addition, application functionality, that can not be provided independent from a specific J2EE technology not supported on the client machine, or that extensively relies on back-end services, may not be available for offline operation. Summarized, each feature may be audited based on a cost-benefit and risk analysis. If benefits prevail and risks are acceptable, the application feature may be supported in offline operation. Otherwise, the particular feature may be limited to online operation only.

Section 9

Outlook

This section takes a brief outlook into the future of offline operation for enterprise applications. Further efforts are suggested in order to enhance both the conceptual and programmatical solutions provided in this thesis.

The architectural and design-oriented concepts for development of offline-capable J2EE enterprise applications and the framework provided have proven their usefulness for the demonstration applications. Nonetheless, further investigations of offline-related requirements posed by specific enterprise applications scenarios may reveal additional parameters and concepts to be supported by the framework for development of offline-capable enterprise applications. It may be necessary to further enhance the prototypic framework based on feedback from offline enterprise application projects, both related to functionality and performance.

The synchronization solution proposed as part of the framework may suffer from shortcomings in real world scenarios. Especially, the performance of the synchronization process is crucial as particular scenarios may require nearly continuous synchronization in the background. The synchronization process proposed may be optimized by evaluating further state information on completed synchronization processes and modifications done since the last synchronization in order to decide whether a synchronization process has to be initiated and which steps of the process have to be executed. The vector-based versioning mechanism may not scale for enterprise applications that have large numbers of application clients and objects that need to be synchronized between different data stores. The versioning mechanism provided may therefore be replaced by a more scalable approach depending on the actual requirements. In general, there is a trade-off between the amount of object version information stored on each replica and the amount of data that has to be transferred between two replicas in order to determine the set of updates to be exchanged during synchronization.

Concurrency issues resulting from simultaneously running synchronization processes initiated by multiple application clients, object locking during synchronization, and business-specific relations and constraints between multiple synchronizable objects have to be investigated further.

Related to security issues arising from offline operation and locally stored application data, additional measurements and techniques are required in order to protect sensitive data from unauthorized access and modification on the client machine. While offline-capable enterprise applications are desired in many business sectors, an insufficient security level may inhibit the use of offline operation, especially in the financial sector. Technologies such as encrypted file systems and enhanced authentication mechanisms may enable a higher level of local data security.

In future, the relevance of offline-capable enterprise applications may change its direction. While the need for on-demand, conscious work in offline mode may be reduced based on the wider support of ubiquitous connectivity, the usage of offline operation for bypassing short, unprepared connection interruptions may become more important in the context of portable devices in mobile networks. This would also shift the relevance from well-controlled, on-demand configured operation mode switches towards the proposed 'be-prepared' configuration.

Appendix A

Resource CD

The resource CD accompanying this thesis contains the complete Java source code with JavaDoc documentation for all demonstration applications and the J2OO framework. In addition, the resource CD provides binary prerequisites necessary for building and executing the demonstration applications. The binaries include the JBoss application server, version 3.2.1, that is used as J2EE-compliant application server for running the demonstration applications.

A.1 Requirements

In order to compile, deploy and run the demonstration applications, the following technologies and tools must be supported:

- Java 2 SDK, Standard Edition, version 1.4.1 or later
  available at http://java.sun.com
- Java Applet Plug-In, version 1.4.1 or later
  available at http://java.sun.com
- Java Web Start, version 1.2
  available at http://java.sun.com/products/javawebstart
- JBoss application server, version 3.2.1
  available at http://www.jboss.org
- Ant building tool
  available at http://ant.apache.org
- Read/write access to the user directory on the local file system.

For each binary prerequisite listed above, the resource CD contains a version for the Microsoft Windows and the Linux operation system. Binaries for other operating systems may be available at the URLs listed above.

A.2 CD Layout

```
ResourceCD
  +- bin
    +- linux              Binary Prerequisites for Linux Operating Systems
    +- win32              Binary Prerequisites for Windows Operating Systems
  +- doc                  Diploma Thesis Paper, Abstract in English and German
  +- ear                  Pre-built EARs for Demonstration Applications
  +- src
    +- AddressBook        AddressBook Demonstration Application Source Code
    +- ExpenseTool        ExpenseTool Demonstration Application Source Code
    +- J2OO               J2OO Framework Source Code
    +- ObjectSigning      Object Signing and Verification Source Code
```

Appendix B

AddressBook Application

This section describes the deployment and execution of the AddressBook demonstration application. Using a simple address book scenario with limited functionality, the AddressBook application demonstrates an offline-capable J2EE enterprise application based on the three architectural solutions discussed in section 6.

B.1 J2EE Application Server Configuration

An instance of the JBoss application server must be running on the local machine, configured with the JBoss default properties listed in table B.1:

Property Name	**Property Value**
Version	3.2.1
Host Name	localhost
Naming Service Port	1099
HTTP Service Port	8080

Table B.1: JBoss application server properties.

The `JAVA_HOME` environment variable has to be set to the location of the Java runtime environment.

B.2 Java Source Compilation & Building

The Java source code of the AddressBook application is available on the resource CD in the directory `<cd>/src/AddressBook/`. Prior to building the EAR file, the content of the resource CD has to be copied to a writable directory on the local file system. `<cd>` refers to the base directory on the file system where the copy of the resource CD resides. The AddressBook enterprise application archive (EAR) file can be created with the Ant building tool using the following command:

```
ant -f <cd>/src/AddressBook/build.xml
```

The resulting EAR file `AddressBook.ear` is located in `<cd>/src/AddressBook/build/`. There is a pre-built EAR file available in `<cd>/ear/`.

B.3 Deployment

The AddressBook demonstration application is deployed by copying the EAR file to the JBoss deployment directory:

`copy <cd>\ear\AddressBook.ear <jboss>\server\default\deploy\` (Windows)

`cp <cd>/ear/AddressBook.ear <jboss>/server/default/deploy/` (Linux)

`<jboss>` refers to the JBoss installation base directory. Since the JBoss application server supports hot deployment of enterprise applications, all components of the AddressBook application are deployed automatically.

B.4 Starting the AddressBook Application

All application client variants of the AddressBook demonstration application can be started from the web page at

```
http://localhost:8080/AddressBook/
```

referred to as the AddressBook index page.

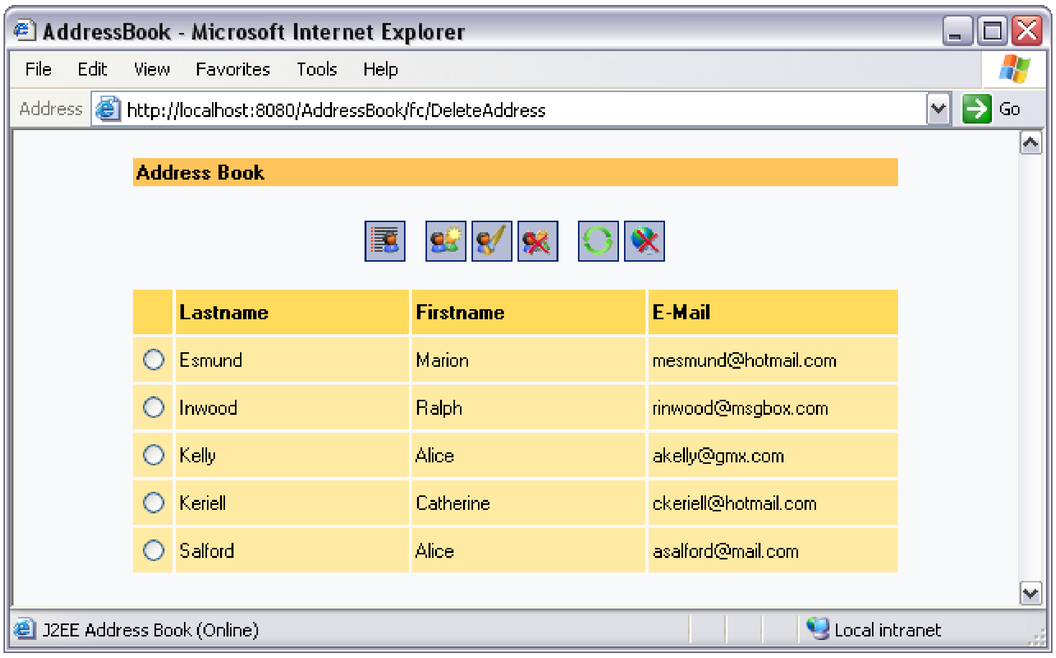
During application runtime, data required for offline operation is stored in the directory `<user.home>/.AddressBook` where `<user.home>` refers to the home directory of the system user that is configured by the Java system property `user.home`. Each application client variant maintains its own local data store that is located in the appropriate subdirectory `WebClient`, `RichClient` or `FatClient`. In addition, infrastructure prerequisites for the offline web application are also stored in separate subdirectories. When deleting the local data directory containing offline-related data, the library cache of the Java Web Start client has to be cleaned in order to enable the proper re-installation of the offline web application infrastructure prerequisites the next time the web client is started in offline operation.

B.4.1 Web Client

The AddressBook web client is started by pointing the web browser to the URL

```
http://localhost:8080/AddressBook/fc/GetAllAddresses
```

or by using the appropriate link on the AddressBook index page. Figure B.1 shows the address list screen of the AddressBook web client containing test address entries.


AddressBook - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address http://localhost:8080/AddressBook/fc/DeleteAddress

Address Book

	Lastname	Firstname	E-Mail
○	Esmund	Marion	mesmund@hotmail.com
○	Inwood	Ralph	rinwood@msgbox.com
○	Kelly	Alice	akelly@gmx.com
○	Keriell	Catherine	ckeriell@hotmail.com
○	Salford	Alice	asalford@mail.com

J2EE Address Book (Online)

Local intranet

Figure B.1: *Web client address list screen.*

Address entries can be managed using the appropriate buttons 'New Address', 'Edit Address' and 'Delete Address' in the tool bar of the web client. Data synchronization is initialized using the 'Synchronize Data' button. Clicking the button displays a new web page with an embedded Java applet implementing the synchronization client. During the loading of the Java applet, a security confirmation alert is displayed by the applet plug-in since the synchronization applet requires access to the local file system in order to store data for offline operation. The actual synchronization of the local data store with the data in the server database is started by clicking the synchronization button in the center of the screen.

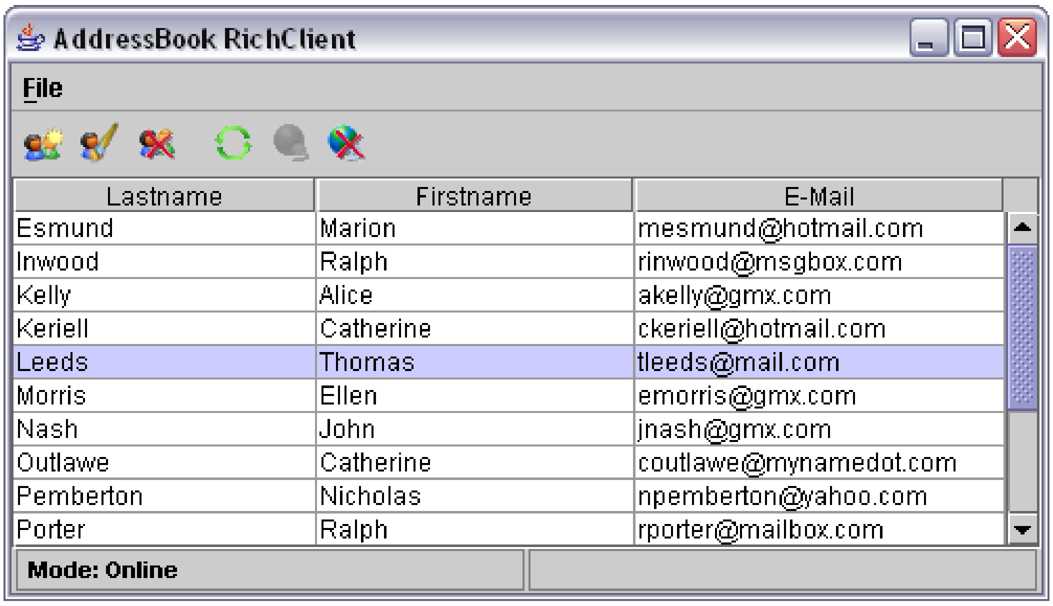
The offline version of the AddressBook web application can be installed and started either by using the appropriate link on the AddressBook index page or by clicking the 'Go Offline' button in the online web client. Both ways start the JNLP-based installation of the offline web application described in section 6.3.2 that installs a local web server and deploys the offline web application archive to the local web server. The locally installed web server is an instance of the Jakarta Tomcat server running on port 9090. During the installation procedure, several security confirmation alerts are presented since the offline installation program requires access to the file system. Finally, a new web browser window displaying the offline web client is opened. Once the infrastructure prerequisites and the offline web application are installed, the offline web client may also be started by using the appropriate link on the AddressBook index page. The offline web client functionality is identical to the one provided by the online variant except from the missing 'Go Offline' feature.

B.4.2 Rich Client

The AddressBook rich client is started by referring to the JNLP file located at the URL

```
http://localhost:8080/AddressBook/RichClient.jnlp
```

using the web browser. An appropriate link is listed on the AddressBook index page. By clicking this link, the Java Web Start client starts the graphical user interface of the rich client, as illustrated in figure B.2. Managing address entries and triggering the data synchronization works similar to the web client. Transitions between the online and offline operation mode are initialized using the 'Go Offline' or 'Go Online' button in the tool bar of the rich client. The same application client instance is used both for online and offline operation.



Lastname	Firstname	E-Mail
Esmund	Marion	mesmund@hotmail.com
Inwood	Ralph	rinwood@msgbox.com
Kelly	Alice	akelly@gmx.com
Keriell	Catherine	ckeriell@hotmail.com
Leeds	Thomas	tleeds@mail.com
Morris	Ellen	emorris@gmx.com
Nash	John	jnash@gmx.com
Outlawe	Catherine	coutlawe@mynamedot.com
Pemberton	Nicholas	npemberton@yahoo.com
Porter	Ralph	rporter@mailbox.com

Mode: Online

Figure B.2: *Rich client address list screen.*

B.4.3 Fat Client

The AddressBook fat client may also be started using JNLP by pointing the web browser to the URL

```
http://localhost:8080/AddressBook/FatClient.jnlp
```

or by clicking the appropriate link on the AddressBook index page. The look and feel of the fat client is identical to the rich client.

Appendix C

ExpenseTool Application

This section describes how to deploy and run the expense management application, herein referred to as ExpenseTool application, that demonstrates a business-relevant use case for an offline-capable J2EE enterprise application.

C.1 J2EE Application Server Configuration

An instance of the JBoss application server must be running on the local machine, configured with the JBoss default properties listed in table B.1. In addition, the server-side JAAS context has to be configured in the JBoss login configuration file

```
<jboss>/server/default/conf/login-config.xml
```

where `<jboss>` refers to the JBoss installation base directory. The following configuration entry has to be added:

```
<application-policy name="ExpenseTool">
  <authentication>
    <login-module
      code="com.canoo.diploma.expensetool.auth.server.ServerLoginModule"
      flag="required">
      <module-option name = "unauthenticatedIdentity">
        anonymous
      </module-option>
      <module-option name="principalsQuery">
        select PASSWORD from EJBUSEREJB where USERNAME=?
      </module-option>
      <module-option name="rolesQuery">
        select PRINCIPALLIST from EJBUSEREJB where USERNAME=?
      </module-option>
    </login-module>
  </authentication>
</application-policy>
```

This entry configures the source of the server-side authentication data and the login module to be used whenever the application client tries to authenticate itself with the JBoss application server or one of its containers.

The `JAVA_HOME` environment variable has to be set to the location of the Java runtime environment.

C.2 Java Source Compilation & Building

The Java source code of the ExpenseTool application is available on the resource CD in the directory `<cd>/src/ExpenseTool/`. Prior to building the EAR file, the content of the resource CD has to be copied to a writable directory on the local file system. `<cd>` refers to the base directory on the file system where the copy of the resource CD resides. The AddressBook enterprise application archive (EAR) file can be created with the Ant building tool using the following command:

```
ant -f <cd>/src/ExpenseTool/build.xml
```

The resulting EAR file `ExpenseTool.ear` is located in `<cd>/src/ExpenseTool/build/`. There is a pre-built EAR file available in `<cd>/ear/`.

C.3 Deployment

The ExpenseTool demonstration application is deployed by copying the EAR file to the JBoss deployment directory:

`copy <cd>\ear\ExpenseTool.ear <jboss>\server\default\deploy\` (Windows)

`cp <cd>/ear/ExpenseTool.ear <jboss>/server/default/deploy/` (Linux)

Since the JBoss application server supports hot deployment of enterprise applications, all components of the ExpenseTool application are deployed automatically.

C.4 Creating the Test Data Set

The ExpenseTool enterprise application requires a set of test data containing user logins and expense records. The JNLP-based test data tool that creates a predefined set of test data can be started using the appropriate link on the web page at

`http://localhost:8080/ExpenseTool/`

referred to as the ExpenseTool index page. During startup of the test data tool, a security confirmation alert is presented since the test data tool requires access to the local file system. By applying the test data set, all ExpenseTool application data, both on the server and the client machine, is reset.

C.5 Starting the Application Client

The application client of the ExpenseTool demonstration application can be started from the ExpenseTool index page.

During application runtime, data required for offline operation is stored in the directory `<user.home>/.ExpenseTool` and its subdirectories, where `<user.home>` refers to the home directory of the system user that is configured by the Java system property `user.home`. For each application user logging into the ExpenseTool application, a separate local data store with a corresponding journal is created in an appropriate subdirectory that is named by the username of the user logged in.

C.5.1 Application User Logins

The user logins listed in table C.1 are pre-configured for test purposes.

Username	**Password**	**Privileges**
employee	employee	user, sync, employee
manager	manager	user, sync, employee, manager
admin	admin	user, sync, employee, admin

Table C.1: ExpenseTool application user logins with associated privileges.

Table C.2 lists and describes the security roles used by the ExpenseTool application.

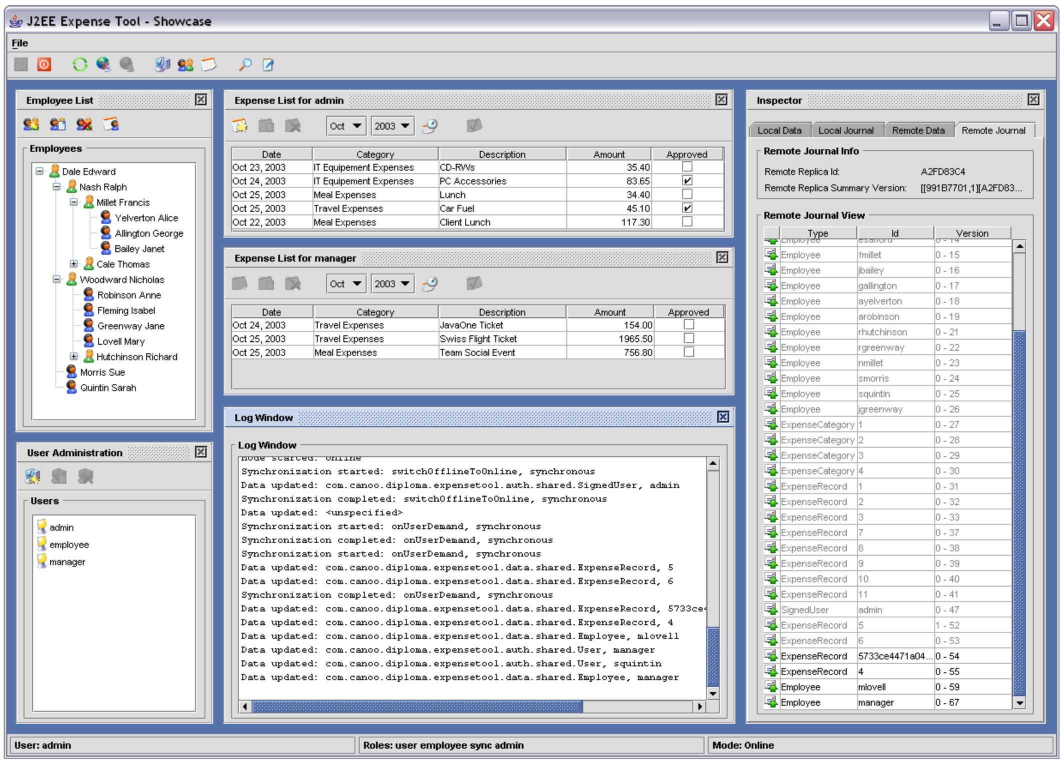
Security Role	**Description of Privileges**
user	log into the ExpenseTool application
sync	synchronize data and work in offline operation
employee	manage own expense records
manager	approve expense records of subordinates
admin	manage users and employees

Table C.2: Security roles used by the ExpenseTool application.

Both the 'on-demand' and the 'be-prepared' configuration described in section 4.3 are supported by the ExpenseTool application.

C.5.2 Starting the ExpenseTool in 'On-Demand' Configuration

The ExpenseTool application client configured for the 'on-demand' mode can be started by using the appropriate link on the ExpenseTool index page. The ExpenseTool application client is started by Java Web Start. During startup, a security confirmation alert is presented since the ExpenseTool requires access to the local file system. In the login dialog presented, the user can select to start the application client in online or offline operation. Login to offline operation is only possible after the successful completion of at least one synchronization process. After logging into the ExpenseTool with one of the user logins listed in table C.1, the application working desk is presented. Depending on the user privileges and the windows opened, the graphical user interface may look like shown in figure C.1.



Dale Edward
Nash Ralph
Millet Francis
Yelverton Alice
Allington George
Bailey Janet
Cale Thomas
Woodward Nicholas
Robinson Anne
Fleming Isabel
Greenway Jane
Lovell Mary
Hutchinson Richard
Morris Sue
Quintin Sarah

admin
employee
manager

Date	Category	Description	Amount	Approved
Oct 23, 2003	IT Equipement Expenses	CD-RWs	35.40	☐
Oct 24, 2003	IT Equipement Expenses	PC Accessories	83.65	✓
Oct 25, 2003	Meal Expenses	Lunch	34.40	☐
Oct 25, 2003	Travel Expenses	Car Fuel	45.10	✓
Oct 22, 2003	Meal Expenses	Client Lunch	117.30	☐

Date	Category	Description	Amount	Approved
Oct 24, 2003	Travel Expenses	JavaOne Ticket	154.00	☐
Oct 25, 2003	Travel Expenses	Swiss Flight Ticket	1965.50	☐
Oct 25, 2003	Meal Expenses	Team Social Event	756.80	☐

```
Synchronization started: switchOfflineToOnline, synchronous
Data updated: com.canoo.diploma.expensetool.auth.shared.SignedUser, admin
Synchronization completed: switchOfflineToOnline, synchronous
Data updated: <unspecified>
Synchronization started: onUserDemand, synchronous
Synchronization completed: onUserDemand, synchronous
Synchronization started: onUserDemand, synchronous
Data updated: com.canoo.diploma.expensetool.data.shared.ExpenseRecord, 5
Data updated: com.canoo.diploma.expensetool.data.shared.ExpenseRecord, 6
Synchronization completed: onUserDemand, synchronous
Data updated: com.canoo.diploma.expensetool.data.shared.ExpenseRecord, 5733ce
Data updated: com.canoo.diploma.expensetool.data.shared.ExpenseRecord, 4
Data updated: com.canoo.diploma.expensetool.data.shared.Employee, mlovell
Data updated: com.canoo.diploma.expensetool.auth.shared.User, manager
Data updated: com.canoo.diploma.expensetool.auth.shared.User, squintin
Data updated: com.canoo.diploma.expensetool.data.shared.Employee, manager
```

Remote Replica Id: A2FD83C4
Remote Replica Summary Version: [[991B7701,1][A2FD83...

Type	Id	Version
Employee	fmillet	0 - 15
Employee	jbailey	0 - 16
Employee	gallington	0 - 17
Employee	ayelverton	0 - 18
Employee	arobinson	0 - 19
Employee	rhutchinson	0 - 21
Employee	rgreenway	0 - 22
Employee	nmillet	0 - 23
Employee	smorris	0 - 24
Employee	squintin	0 - 25
Employee	jgreenway	0 - 26
ExpenseCategory	1	0 - 27
ExpenseCategory	2	0 - 28
ExpenseCategory	3	0 - 29
ExpenseCategory	4	0 - 30
ExpenseRecord	1	0 - 31
ExpenseRecord	2	0 - 32
ExpenseRecord	3	0 - 33
ExpenseRecord	7	0 - 37
ExpenseRecord	8	0 - 38
ExpenseRecord	9	0 - 39
ExpenseRecord	10	0 - 40
ExpenseRecord	11	0 - 41
SignedUser	admin	0 - 47
ExpenseRecord	5	1 - 52
ExpenseRecord	6	0 - 53
ExpenseRecord	5733ce4471a04...	0 - 54
ExpenseRecord	4	0 - 55
Employee	mlovell	0 - 59
Employee	manager	0 - 67

User: admin
Roles: user employee sync admin
Mode: Online

Figure C.1: *Working desk of the ExpenseTool application with several windows opened.*

The 'Inspector' window shows the state of the local journal, the data stored in the local data store on the client machine and, if running in online operation, the state of the journal and data on the server machine. Gray shaded entries in the journal table of the 'Inspector' window indicate that the synchronizable object represented by the journal entry has been synchronized with the local and the server data store. Entries in black have been modified either on the client or the server machine since the last synchronization process was completed. The 'Inspector' feature significantly decreases the performance of the ExpenseTool application when it is visible as the 'Inspector' window is updated on each data modification.

The 'Log Window' feature displays a selection of messages for J2OO framework events such as transition to a new operation mode, results of synchronization processes and data modifications. The 'Log Window' also shows the stack trace of potential exceptions.

When using the 'on-demand' configuration, the 'Go Online', 'Go Offline' and 'Synchronize' buttons may be used to switch between online and offline operation and to initialize data synchronization on demand.

C.5.3 Starting the ExpenseTool in 'Be-Prepared' Configuration

By using the appropriate link on the ExpenseTool index page, the ExpenseTool application client can be started with the 'be-prepared' configuration. Compared to the 'on-demand' configuration, there is no choice between online or offline authentication and no possibility to change the operation mode or to synchronize data on demand. The actual operation mode and data synchronization is transparently controlled by the ExpenseTool application client or the underlying J2OO framework, respectively.

A server or communication infrastructure failure may be simulated by terminating the JBoss application server process while the ExpenseTool application client is running. Seconds after the server shutdown, the application client switches to offline operation. As soon as the JBoss application server is restarted again, the application client switches back to online operation.

Bibliography

[Alu01] D. Alur, D. Malks, and J. Crupi. *Core J2EE Patterns, Best Practices and Design Strategies.* Sun Microsystems Press / Prentice Hall PTR, 2001.

[Bod$^+$02] S. Bodoff, D. Green, K. Haase, E. Jendrock, M. Pawlan, and B. Stearns. *The J2EE Tutorial.* Addison-Wesley, 2002.

[Bra$^+$00] T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler. *Extensible Markup Language (XML) 1.0 (Second Edition).* World Wide Web Consortium W3C, 2000.
http://www.w3.org/TR/REC-xml

[Ced03] P. Cederqvist et al. *Version Management with CVS.* CollabNet, Inc. 2003.
http://www.cvshome.org

[Coh03] N. H. Cohen. *IBM Research Report: Design and Implementation of the MNCRS Java Framework for Mobile Data Synchronization.* IBM Research Division, 2003.
http://www.research.ibm.com/sync-msg/RC21774.pdf

[Cow01] D. Coward. *Java Servlet Specification, Version 2.3.* Sun Microsystems Inc., 2001.
http://www.jcp.org/aboutJava/communityprocess/final/jsr053

[Dem$^+$01] L. DeMichiel, L. mit Yalinalp, and S. Krishnan. *Enterprise JavaBeans Specification, Version 2.0.* Sun Microsystems Inc., 2001.
http://java.sun.com/products/ejb/docs.html

[Ell$^+$01] J. Ellis, L. Ho, and M. Fisher. *JDBC 3.0 Specification.* Sun Microsystems Inc., 2001.
http://java.sun.com/products/jdbc/download.html

[Fow02] M. Fowler. *Patterns of Enterprise Application Architecture.* Addison-Wesley, 2002.

[Gam$^+$95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Pattern: Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995.

[Gon98] L. Gong. *Java Web Start.* Sun Microsystems Inc., 1998.
http://java.sun.com/products/jdk/1.2/docs/guide/security

[HWG03] HTML Working Group. *HyperText Markup Language (HTML) Home Page.* World Wide Web Consortium W3C, 2003.
http://www.w3.org/MarkUp

[Ken$^+$03] S. Kent, T. Polk, R. Housley, and S. Bellovin. *Public-Key Infrastructure (X.509) (pkix).* The Internet Engineering Task Force, 2003.
http://www.ietf.org/html.charters/pkix-charter.html

[Lai$^+$99] C. Lai, L. Gong, L. Koved, A. Nadalin, and R. Schemers. *User Authentication and Authorization in the Java Platform.* Sun Microsystems Inc., 1999.
http://java.sun.com/security/jaas/doc/acsac.html

[Orf97] R. Orfali. *Essential Client/Server Survival Guide, Second Edition.* John Wiley & Sons, 1997.

[Pel53] E. Pelegr-Llopart. *JavaServer Pages Specification, Version 1.2.* Sun Microsystems Inc., 2001.
http://www.jcp.org/aboutJava/communityprocess/final/jsr053

[Sch01] R. W. Schmidt. *Java Network Launching Protocol and API Specification (JSR-56), Version 1.0.1.* Sun Microsystems Inc., 2001.
http://java.sun.com/products/javawebstart/download-spec.html

[Sha01] B. Shannon. *Java 2 Platform Enterprise Edition Specification, v1.3.* Sun Microsystems Inc., 2001.
http://java.sun.com/j2ee/j2ee-1_3-fr-spec.pdf

[Shi00] R. Shirey. *Internet Security Glossary.* The Internet Engineering Task Force, 2000.
http://www.ietf.org/rfc/rfc2828.txt

[Sin$^+$02] I. Singh, B. Stearns, and M. Johnson. *Designing Enterprise Applications with the J2EE Platform.* Addison-Wesley, 2002.

[Sto83] P. Stott. *Detection of Mutual Inconsistency in Distributed Systems.* IEEE Transaction on Software Engineering, 9(3), 1983.

[Sun03a] *Java 2 Enterprise Edition, Frequently Asked Questions.* Sun Microsystems Inc., 2003.
http://java.sun.com/j2ee/faq.html

[Sun03b] *Javadoc Tool Home Page.* Sun Microsystems Inc., 2003.
http://java.sun.com/j2se/javadoc/index.html

[Sun03c] *Java Foundation Classes.* Sun Microsystems Inc., 2003.
http://java.sun.com/products/jfc

[Sun03d] *Java Web Start*. Sun Microsystems Inc., 2003.
http://java.sun.com/products/javawebstart

[Sun99] *Java Naming and Directory Interface Application Programming Interface (JNDI API)*. Sun Microsystems Inc., 1999.
ftp://ftp.javasoft.com/docs/j2se1.3/jndi.pdf

[Syn02] *SyncML Sync Protocol, version 1.1.1*. SyncML Initiative Ltd., 2002.
http://www.syncml.org