# Technical Architecture Guide - caAERS

## Document Change History

| Version Number | Date | Contributor | Description |
|---|---|---|---|
| V1.0 | 2007-02-13 | Joshua Phillips | Initial revision |
| V1.1 | | Ram Chilukuri | Revision |
| V1.2 | | Vinay Kumar | Revision |
| V1.3 | | Biju Joseph | Revision |
| V1.4 | | Edmond Mulaire | Revision |
| V1.5 | | Vinay Kumar | Revision |
| V1.6 | | Ram Chilukuri | Revision |
| V1.7 | 2008-01-09 | Vinay Kumar | Revision to include caAERS-AdEERS Communication |
| V1.8 | 2009-11-18 | Paul Baumgartner | Added Module 5 and Hosted Mode Security |
| V1.9 | 2010-05-26 | Paul Baumgartner | Converted to wiki |
| V1.9.1 | 2011-02-28 | Paul Baumgartner | Copy edits |

## Overview

### Overview of the Guide

This document covers the technical and design aspects of the implementation of caAERS with an overview of the requirements for context. Not covered here, but available in other project documents are:

- Detailed requirements
- Use cases
- Domain UML model
- Installation
- Administration

### About caAERS

The Cancer Adverse Events Reporting System (caAERS) is a scalable system that provides advanced adverse event recording and reporting functionality to the cancer research and care community.

caAERS is being developed to address a number of problems and challenges faced by the cancer care and research communities. The table below summarizes some of these key problems and the proposed technical solutions to address these requirements.

| Requirement | Motivation & Challenges | Solution | Technology | Reasoning |
|---|---|---|---|---|
| Adverse Event Reporting Business Rules | • Very adaptive to changes in business.<br>• Rapid provisioning.<br>• Domain-specific authoring of rules.<br>• Decoupled from Java source code | Rules Engine | Drools | • Spring integration<br>• Open-source<br>• Strong financial backing<br>• Large community |

| | | | | |
|---|---|---|---|---|
| Report Routing & Review | • Very adaptive to individual organization's needs for report review process.<br>• Platform-independent authoring<br>• Decoupled from Java source code | Workflow Engine | jBPM | • Integration with Spring, Hibernate, databases<br>• Open-source<br>• Strong financial backing<br>• Large community<br>• Multiple process definition languages |
| Adverse Event Capture & Reporting | • Run notifications on a given schedule<br>• Reliability (schedule survives server restarts and crashes)<br>• Flexible way of describing the schedule | Scheduling Engine | Quartz | • Lightweight<br>• Integration with Spring and Hibernate<br>• Supports CRON syntax<br>• Uses database to persist jobs |
| External Agency Reporting | • Conformance to reporting requirements of external agencies in a scalable manner.<br>• Decoupling from external systems' interfaces | Enterprise Service Bus | Apache ServiceMix / iHub | • Consistency: used already by the Suite Applications<br>• Standardization (JBI)<br>• Open-source |
| Portability & Platform-Independence | • Run reliably on all major platforms and databases | Platform-independent language & ORM framework | Java/Hibernate | • Proved platform-independence<br>• Portability across databases with little or no changes. |

## Software overview

caAERS consists of a Java web application that based on the J2EE Servlets infrastructure, WSRF-compliant web services that are built on top of caGrid infrastructure, and JBI components that are hosted by Apache ServiceMix JBI container.

## System overview

The following diagram shows the system overview of caAERS application

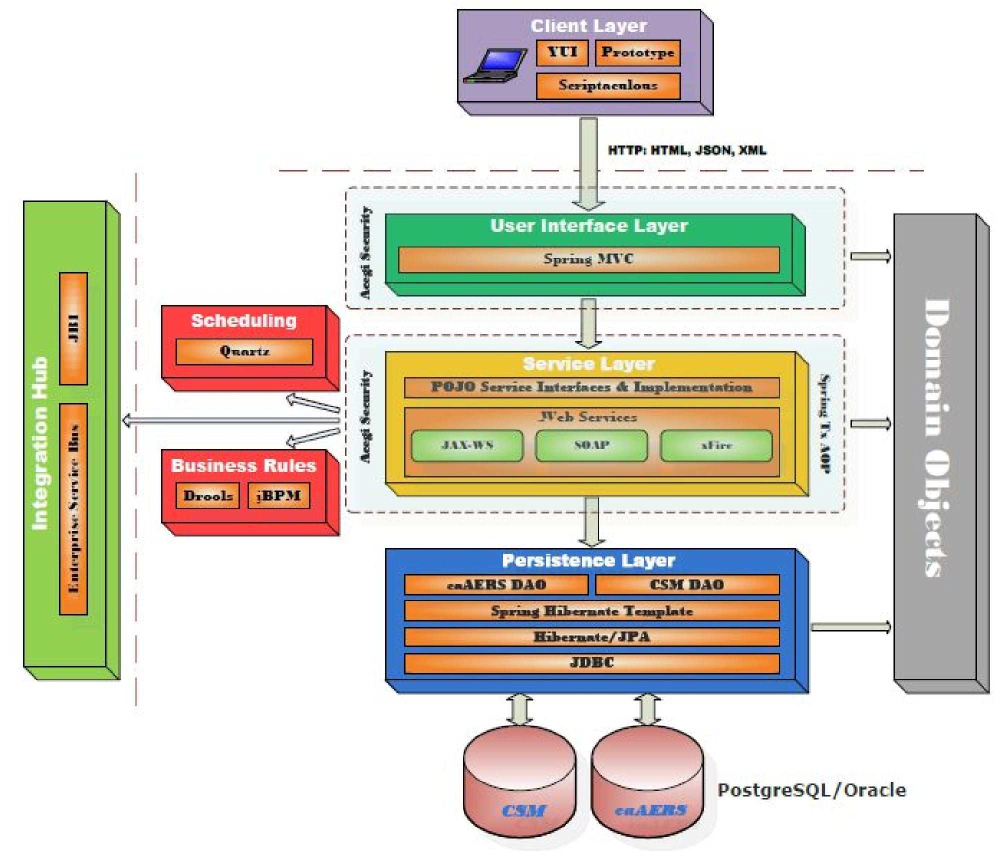


Figure 1: caAERS overview

## System requirements

### Required Software

| *Software Name* | *Version* | *Description* | *URL* |
|---|---|---|---|
| Java SE Development Kit | JDK 5.0 | The Java SE development kit with JRE, compilers and debuggers | http://java.sun.com/javase/downloads/index.jsp |
| Apache Tomcat | 5.5.17 or higher | Servlet container for JSP | http://tomcat.apache.org/download-55.cgi |
| Apache Ant | 1.6 or higher | Java-based build tool | http://ant.apache.org/bindownload.cgi |
| Apache ServiceMix | 3.0 | JBI (JSR 208) container | http://incubator.apache.org/servicemix/home.html |

Table 1_ *Required software and technology*

### Database Requirements

caAERS requires access to a properly-configured database. The application is built with database-independence in mind and has been tested on the following:

| ***Database Name*** | ***Version*** | ***Description*** | ***URL*** |
|---|---|---|---|
| PostgreSQL | 8 | PostgreSQL is a powerful, open source relational database system. | http://http://www.postgresql.org/download/ |
| Oracle | 9i | Industry-leading, commercial database product. | http://www.oracle.com/technology/software/products/oracle9i/index.html |

For more details, please see the Installation Guide.

## Design and architecture

caAERS is built in layers, with most communication occurring between adjacent layers only, and communication consisting of passed domain instances or other model objects. For more detail, see Chapter 4.

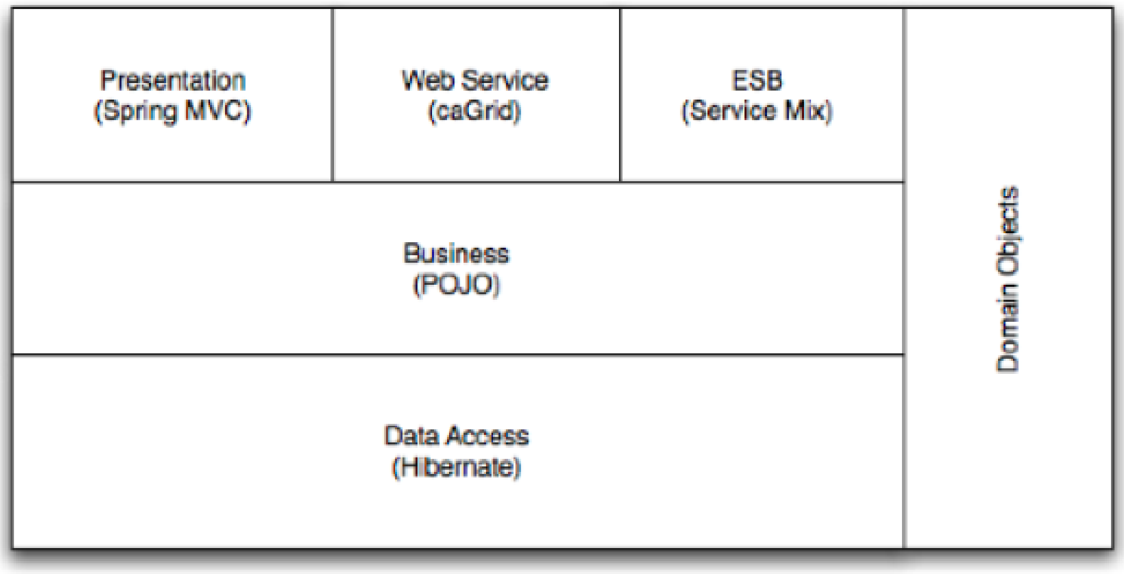


Figure 2: caAERS Architectural Layers

## caAERS Requirements (Use Case View)

This is a brief overview. For more information, please see the Software Requirement Specifications and the caAERS Use cases
The diagram below is a high-level view of all of the caAERS use cases.

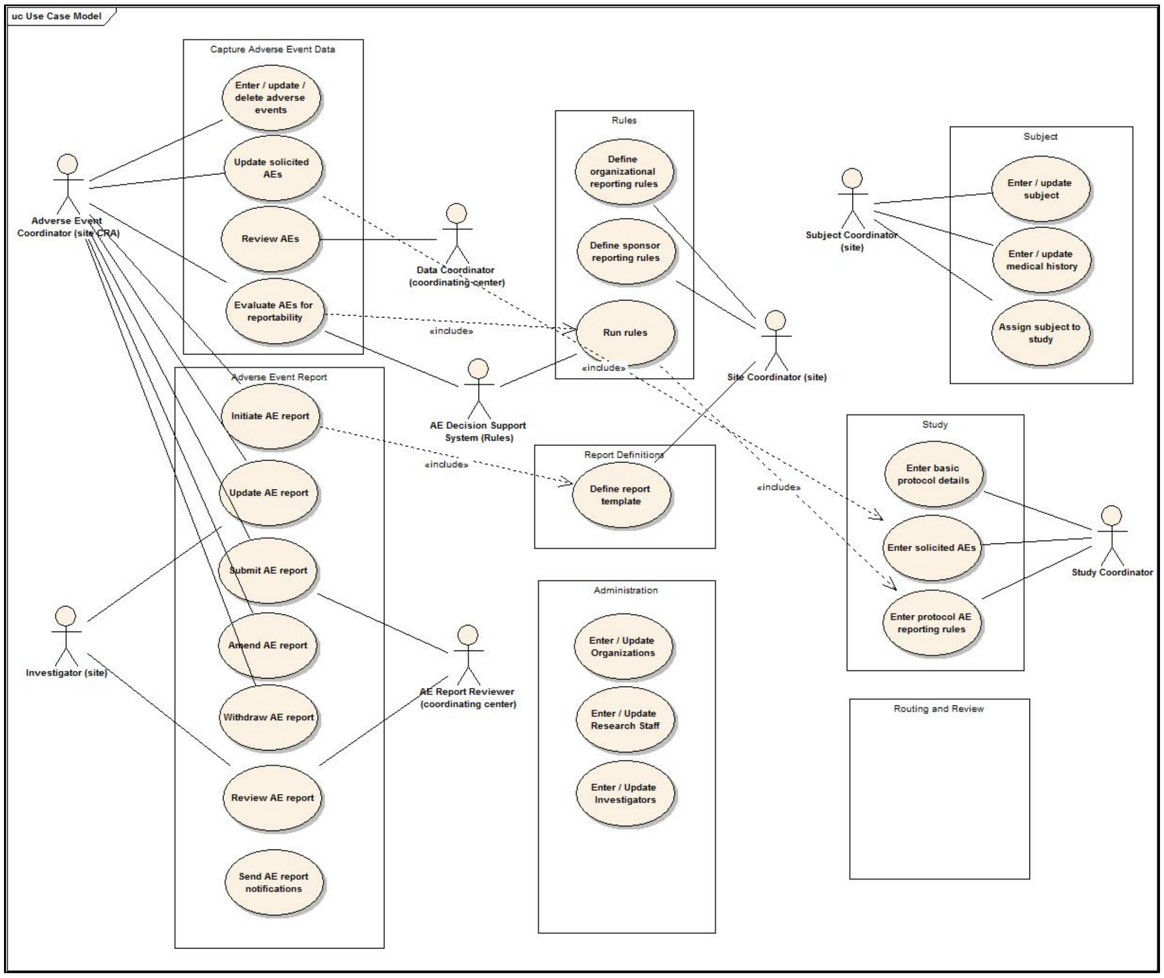

uc Use Case Model
Capture Adverse Event Data
Enter / update / delete adverse events
Update solicited AEs
Review AEs
Evaluate AEs for reportability
Adverse Event Coordinator (site CRA)
Data Coordinator (coordinating center)
Rules
Define organizational reporting rules
Define sponsor reporting rules
Run rules
«include»
«include»
AE Decision Support System (Rules)
Site Coordinator (site)
Subject
Enter / update subject
Enter / update medical history
Assign subject to study
Subject Coordinator (site)
Adverse Event Report
Initiate AE report
Update AE report
Submit AE report
Amend AE report
Withdraw AE report
Review AE report
Send AE report notifications
Investigator (site)
AE Report Reviewer (coordinating center)
Report Definitions
Define report template
«include»
Administration
Enter / Update Organizations
Enter / Update Research Staff
Enter / Update Investigators
Study
Enter basic protocol details
Enter solicited AEs
Enter protocol AE reporting rules
«include»
Study Coordinator
Routing and Review

Figure 3: High-level caAERS use cases

## Module 1 - Adverse Event Data Capture

This module captures Adverse Events when they occur. Based on information entered through a web interface, the system captures the severity of the adverse event and provides instructions for further reporting. Internal reporting capabilities allow the CRA to follow submissions, Quality Assurance to review them, and the Principal Investigator(s) to monitor toxicities and address further reporting requirements.

## Module 2 - Interface Between AE Capture Tool & Local Clinical Trials Databases

Module 2 facilitates communication between the database created in Module 1 and the participating institution's clinical trials (CT) database. A pre-defined set of basic study participant demographics and protocol participation data elements is provided to the caAERS via XML formatted output. In return, AE data that have been submitted on-line to caAERS is exported to the local database via XML formatted output, thereby eliminating the need for duplicate entry.

## Module 3 - Vocabulary Mapping Service

Module 3 utilizes mappings created in our CTMS Metadata/Vocabulary project to support customized reports and forms. Through discussions with the AE SIG participants, it was decided that the initial build of caAERS will encompass only the CTC - MedDRA mapping necessary to support searching on AE category and term during data capture AND for reporting. This will also include mechanisms to access/edit mapping of codes and support future CTC Coding versions.The SIG felt that mapping of other vocabularies was a low priority and was not needed to meet core functionality. Therefore, a high level mapping using SNOMED and LOINC will be addressed prior to the development of Module 8 (Assistance in Grading of Qualitative AEs).

## Module 4 - External Agency Reporting

Module 4 expands the functionality of Module 1 to electronically communicate SAEs to participating entities/systems such as AdEERS and providing generic alert messages to national cooperative groups and industrial sponsors involved with NCI funded protocols.

### Module 5 - Routing and Review

The design for the implementation of this use case is documented at the following url: https://wiki.nci.nih.gov/x/jBay

### Hosted Mode Security

The design for the implementation of this use case is documented at the following url: https://wiki.nci.nih.gov/x/nxey

## caAERS Architecture

### Overview

caAERS consists of a Java web application that based on the J2EE Servlets infrastructure, WSRF-compliant web services that are built on top of caGrid infrastructure, and JBI components that are hosted by Apache ServiceMix JBI container.
The architecture of the caAERS is described below using the 4 +1 view paradigm.

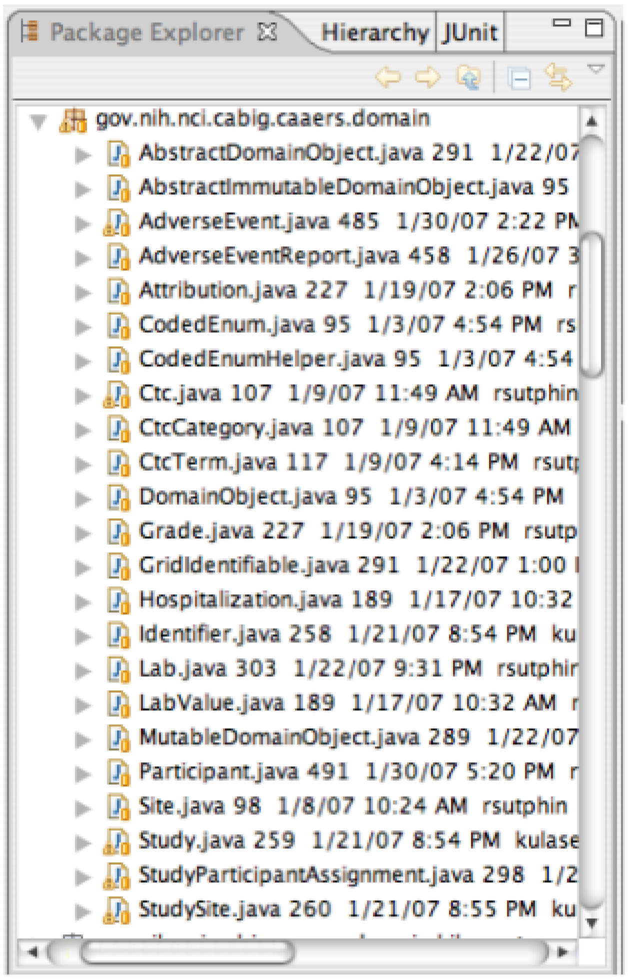
### Architectural Layers (Logical View)

It has 6 fundamental layers - domain, data access, business logic, presentation, web service, and ESB - and several ancillary packages, all glued together using dependency injection.

### Domain

The caAERS domain objects are implemented as plain java beans. They are contained in the *gov.nih.nci.cabig.caaers.domain* java package. The domain model is documented in a UML class diagram, which can be accessed at:
https://ncisvn.nci.nih.gov/svn/caaersappdev/docs/models/caAERS_model.eap

The implementation model is based on a domain analysis model which has been harmonized with the BRIDG model.
The domain objects in the implementation model are mapped to tables in the database (commonly referred to as Object Relational Mapping) using Hibernate Annotations. For more information on Hibernate, please see: http://hibernate.org/5.html
The following graphic enumerates the domain objects:

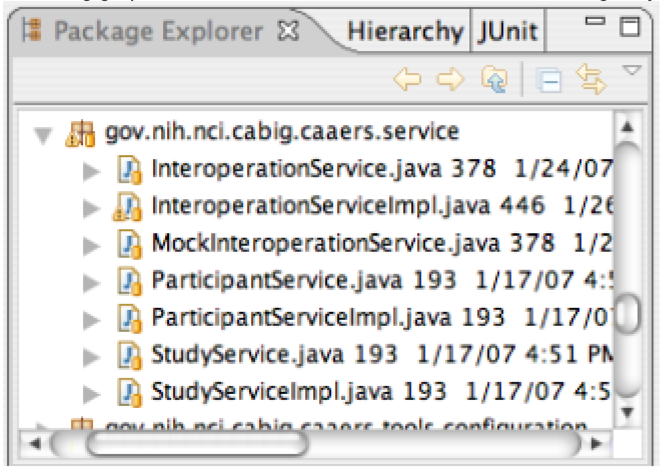


```
gov.nih.nci.cabig.caaers.domain
  AbstractDomainObject.java 291  1/22/07
  AbstractImmutableDomainObject.java 95
  AdverseEvent.java 485  1/30/07 2:22 PM
  AdverseEventReport.java 458  1/26/07 3
  Attribution.java 227  1/19/07 2:06 PM  r
  CodedEnum.java 95  1/3/07 4:54 PM  rs
  CodedEnumHelper.java 95  1/3/07 4:54
  Ctc.java 107  1/9/07 11:49 AM  rsutphin
  CtcCategory.java 107  1/9/07 11:49 AM
  CtcTerm.java 117  1/9/07 4:14 PM  rsut
  DomainObject.java 95  1/3/07 4:54 PM
  Grade.java 227  1/19/07 2:06 PM  rsutp
  GridIdentifiable.java 291  1/22/07 1:00
  Hospitalization.java 189  1/17/07 10:32
  Identifier.java 258  1/21/07 8:54 PM  ku
  Lab.java 303  1/22/07 9:31 PM  rsutphin
  LabValue.java 189  1/17/07 10:32 AM  r
  MutableDomainObject.java 289  1/22/07
  Participant.java 491  1/30/07 5:20 PM  r
  Site.java 98  1/8/07 10:24 AM  rsutphin
  Study.java 259  1/21/07 8:54 PM  kulase
  StudyParticipantAssignment.java 298  1/2
  StudySite.java 260  1/21/07 8:55 PM  ku
```

*Figure 4: Domain classes*

## Business Logic

Business logic based on a single domain object (commonly referred to as domain logic) and its attributes is implemented within the domain object itself, following good object-oriented design principles.
Business logic involving multiple domain objects is implemented externally, generally in the business service tier.
Following graphic enumerates all the classes in the business logic layer:

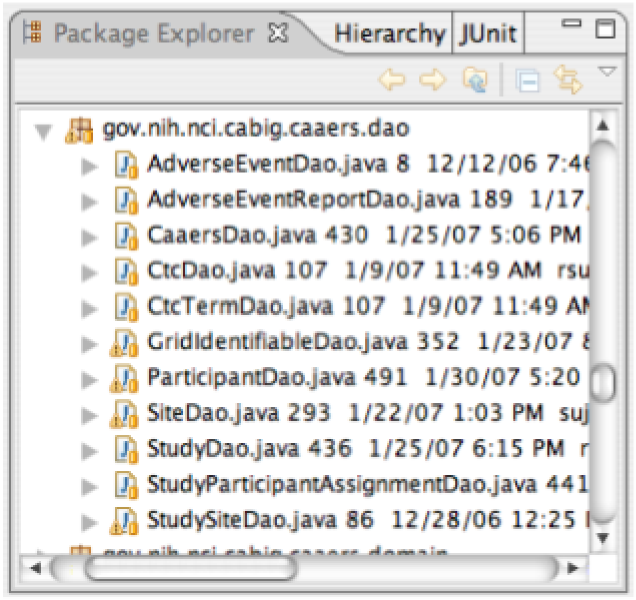


```
gov.nih.nci.cabig.caaers.service
  InteroperationService.java 378  1/24/07
  InteroperationServiceImpl.java 446  1/26
  MockInteroperationService.java 378  1/2
  ParticipantService.java 193  1/17/07 4:5
  ParticipantServiceImpl.java 193  1/17/07
  StudyService.java 193  1/17/07 4:51 PM
  StudyServiceImpl.java 193  1/17/07 4:5
gov.nih.nci.cabig.caaers.tools.configuration
```

*Figure 5: Business logic classes*

## Data access

The data access layer in caAERS is implemented using the *Data Access Object (DAO)* J2EE design pattern. The domain objects are created, updated and queried exclusively via DAOs. A DAO essentially encapsulates the logic necessary to create, retrieve update, and delete a domain object from the database. Implementation of DAO pattern promotes database portability. The DAOs in caAERS are based on the Spring Framework's Hibernate abstraction layer. Use of this abstraction layer greatly simplifies the data access by eliminating the need to write low level and repetitive plumbing code using the API provided by Hibernate framework.
The DAOs in caAERS are contained in the *gov.nih.nci.cabig.caaers.dao* java package. The graphic below enumerates all the DAO classes in caAERS.



```
gov.nih.nci.cabig.caaers.dao
    AdverseEventDao.java 8  12/12/06 7:4
    AdverseEventReportDao.java 189  1/17,
    CaaersDao.java 430  1/25/07 5:06 PM
    CtcDao.java 107  1/9/07 11:49 AM  rsu
    CtcTermDao.java 107  1/9/07 11:49 AM
    GridIdentifiableDao.java 352  1/23/07
    ParticipantDao.java 491  1/30/07 5:20
    SiteDao.java 293  1/22/07 1:03 PM  su
    StudyDao.java 436  1/25/07 6:15 PM  r
    StudyParticipantAssignmentDao.java 441
    StudySiteDao.java 86  12/28/06 12:25
```

*Figure 6: DAO classes*

## Presentation

The presentation layer can further be sub divided into control and view layers, which are described below.

## Control

The control layer is made up of two closely-collaborating types of classes: *commands* and *controllers*. Commands encapsulate the data and behavior associated with a user operation, generally including one or more domain objects. Controllers handle mapping incoming requests onto command instances and providing the command output back to the view layer.
caAERS uses Spring MVC (which is itself based on Servlets) to dispatch web requests to the appropriate controllers.

## View

The view layer uses JSP 2.0 and takes full advantage of tag libraries, both provided by Spring and custom developed, to reduce boilerplate code. Design templating is handled using SiteMesh, which takes the full HTML pages generated from the JSPs and decorates it with the common site elements.
JavaScript components (using Scriptaculous) and Ajax (using DWR) are used to provide a richer user experience.

## Web Service

caAERS exposes several web service interfaces. These interfaces are intended to be invoked by other CTMS suite applications, and also external systems. They have been implemented as caGrid services.
Since CCTS applications are exchanging common data types, interfaces among applications are often identical, or very similar. To facilitate interoperability and code reuse, we have taken the following approach to implementing these services. Each web service consists two components: a skeleton, and an implementation.

The skeleton is generated by Introduce. It fulfills the generic responsibilities of a caGrid service (i.e. binding of SOAP message to objects, advertisement, security). The skeleton also defines an interface that it encapsulates the actual business logic. Spring is used to instantiate (and wire together) an appropriate implementation of this interface. The skeleton just delegates to this implementation. The service implementation is responsible for interacting with the external API defined by the application.
The following diagram depicts this structure.

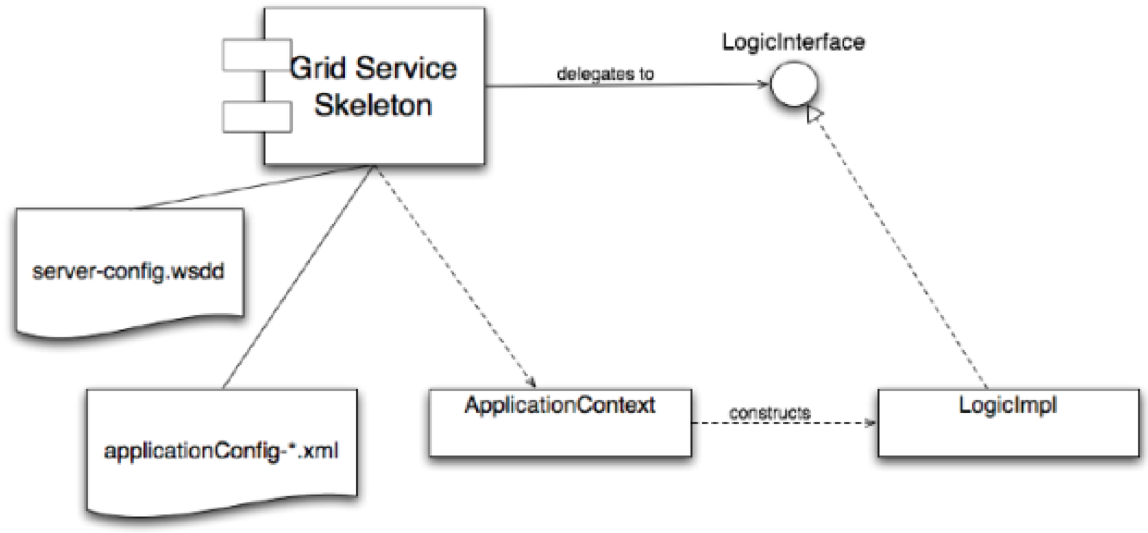


*Figure 7: Grid service framework*
In this way, the same grid service skeleton can be used in various CTMS suite applications, and each can provide it's own implementation.

## Enterprise Service Bus (ESB)

caAERS integrates with other systems through an ESB. We are using the Apache ServiceMix ESB, which implements the Java Business Integration (JBI) standard. The main components of a JBI-compliant ESB include binding components (BCs), service engines (SEs), and the normalized message router (NMR).
BCs are responsible for translating between protocol- or transport-specific messages to "normalized" messages. The structure of a normalized message defined by the abstract service definition in the WSDL of the target service provider that is exposed by the binding component. SEs provide services such as content-based routing, translation among message exchange patterns (MEPs), or message format translation. The NMR routes messages among BCs and SEs based service type and the operation being executed. It is also responsible for fulfilling the requested quality of service (QoS).
The diagram below shows typical interactions among caAERS and other systems in the CTMS suite of applications. In one scenario, the LabViewer application invokes a web service. In CTMSi prototype, we had a mixture of grid and web services. I am not sure if all the web services are being converted to grid services. We will update this when we resubmit this document in iteration 8.endpoint that is exposed by the SOAP BC. The SOAP BC normalizes the message and places in on the NMR which routes it to the EIP (Enterprise Integration Pattern) SE. The EIP SE routes the message to the Auth SE which uses the basic (username and password) credentials to obtain a grid proxy through the caGrid GAARDS infrastructure. It then populates the normalized message's JAAS subject with the principal and credentials obtained from the proxy. The EIP then places the message onto the NMR, which routes it to the Grid Service BC. The Grid Service BC then invokes the caAERS CreateCandidateAdverseEventService grid service.

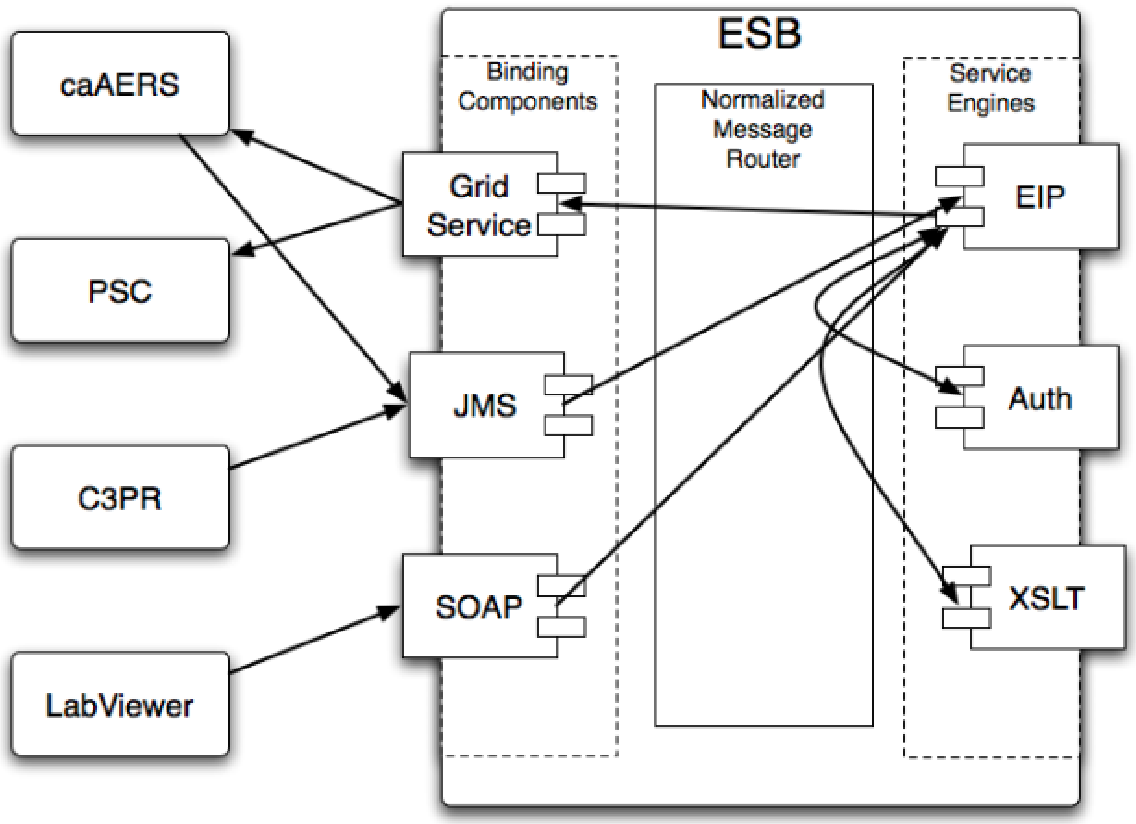


*Figure 8: Integration with external systems through ESB*

## caAERS AdEERS Communication

caAERS is capable of sending AE reports to AdEERS. However, the systems are independent of each other so a platform agnostic and context-free communication approach was developed. Messages are also required to be transmitted reliably and securely.
Since AdEERS has already published the necessary WSDLs, the caAERS AdEERS communication infrastructure implements SOAP messaging. Messaging between caAERS and AdEERS is facilitated using ESB [servicemix implementation discussed above]. The various message queues are used in ESB to send and consume the messages back and forth berween these two systems. The Web Services binding component within ESB calls the web service. The following diagram shows the flow of messages through the ESB.
**LW**-- //**AEReportMessageSenderResult QueueListenerWeb Service JMS queue (Inbound Component) EIP Router (transform and send)(transformer)12345A5B6789Extension 110Pre Message Activity**
*Figure 9: caAERS AdEERS communication using ESB.*
[1] AEReportMessageSender will turn AE Report into XML message and send it to JMS queue.
[2] EIP [Enterprise Integration Pattern] router listens to messages in the JMS queue [In bound component] and processes it.
[3] EIP [Enterprise Integration Pattern] router delegates the message to the Lightweight Transformer container.
[4] LW sends transformed XML back to EIP router.
[5A&B] EIP sends XML to outbound component. The outbound component is a proxy for the actual AEReportMessageSender.
[6&7] Outbound Component detaches caAERS report ID and sends message to webservice of AdEERS and receives reply SYNCHRONOUSLY.
[8] Outbound component attaches the report ID and forwards reply to results queue.
[9] Listener(Message Consumer) on caAERS side will listen to the results queue and consume the return message.
[10] Message consumer process the message and notify the results to reporter via email
[11] In case of successful submission AdEERS ticket number is stored in caAERS database.

## Rules Engine

Rules that govern SAE reporting are complex and could vary from sponsor to sponsor, study to study and healthcare site to healthcare site. When certain criteria are satisfied a specific type of report should be submitted to variety of organizations like the funding sponsor, coordinating center etc. That is, the type of SAE report that needs to be submitted to a sponsor or to any other interested agency may depends on a number of criteria. These criteria may also differ from study to study. Some studies may have much more strict criteria and some others may define relaxed criteria to reduce the number of SAE reports. Attributes from domain objects such as Study, AdverseEvent, etc. form the basis for defining the criteria. A rule is a set of criteria with one or more actions, which get executed when all the criteria in that rule are satisfied.
There are numerous use cases in caAERS, which dictate some kind of notification to be sent to interested parties by the system. In other cases, the system might be required to invoke some action/actions when certain pre-defined conditions are met. These conditions are based on the state

of the domain objects. When a domain object achieves a predefined state then system has to perform preconfigured action(s). Definition of these conditions may change from time to time and/or from adopter to adopter, which introduces a lot of agility in business logic. Implementing the business rules using the classical approach of coding them as part business logic layer will necessitate changing the code and re-deployment of the application when rules change, which is not desirable and will lead to a maintenance nightmare.
With the background of the problem statement stated above, caAERS team has implemented Java Rules (http://jcp.org/en/jsr/detail?id=94) by integrating an open source rules engine. The adoption of this approach will give the desired flexibility that is needed in caAERS to define rules, conditions and plug-gable actions. This will ease the process of introducing new rules or retiring old rules without redeploying the entire application. The provisioning of rules and actions will be accessible to users rather than just system administrators.
As a part of this effort, a Business Rule Management System (BRMS) has been developed and deployed as a service. The rules module provides following high level features:

1. Web provisioning tool for rules ( create, update and delete)
2. Run time environment for execution of rules
3. Grouping the rules based on their types
4. Categorization of rules
5. Invoking rules for assessing the adverse event
6. Invoking rules for report scheduling and notification

Following is the high level architecture for rules engine

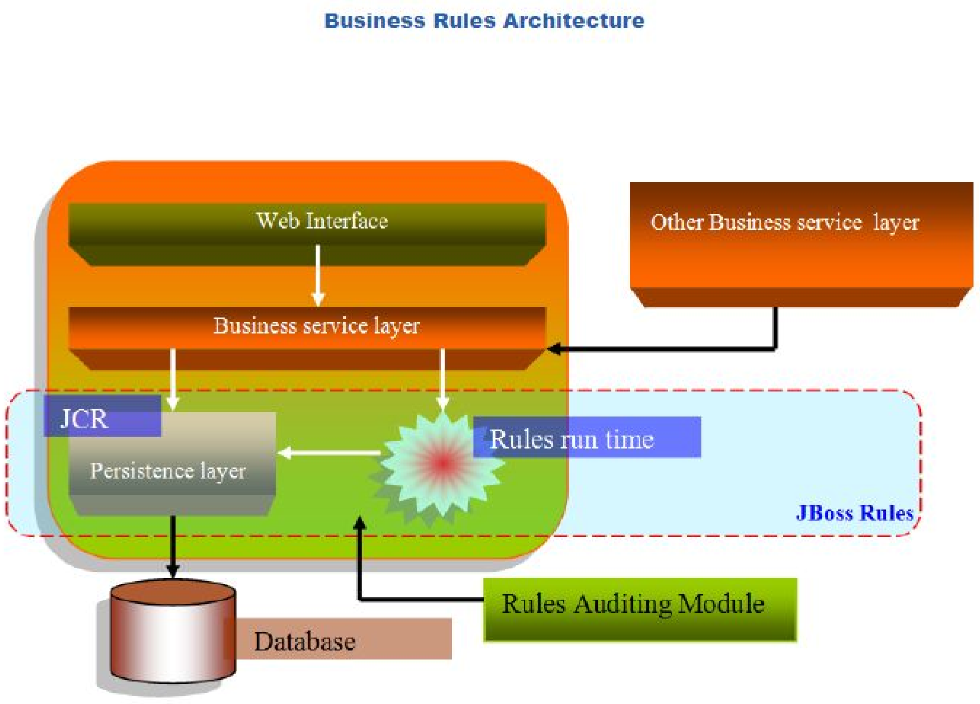


Figure 10: High Level Architecture- Rules Engine_
The rules module is composed of several subsystems including web interface, business service layer, persistence layer, rules run-time, database and rules auditing module. Using web interface users provision and deploy the rules. The rules module defines one business service interface, which is used by web interface. The business service interface is called RulesEngineService. This interface provides various methods to provision, deploy and fire the rules.
Rules repository is a place where rules are persisted, regardless of their format. In caAERS application, a Java Content Repository ( http://jcp.org/en/jsr/detail?id=170) has been adopted to persist rules. Content repository is a generic "data store" that can be used for storing both text and binary data. One key feature of a content repository is that you don't have to worry about how the data is actually stored. Data could be stored in a RDBMS or a file system or as an XML document. Relational and object databases lack many data management features required by modern applications, such as versioning, rich data references, inheritance or fine grained security. The Java Content Repository API (JSR-170) defines a standard for accessing content repositories from java code and promises to greatly simplify java database programming. caAERS team has adopted JackRabbit, which is a fully compliant open source JCR implementation.
**Spring integration for JackRabbit**
To facilitate the use of Jackrabbit in caAERS, Sping JCR module has been used. It has several advantages and provides following artifacts:
**JcrTemplate** which allows execution of **JcrCallback** and exception handling (transforming checked JCR exceptions into unchecked Spring DAO exceptions). The template implements most of the methods from the JCR Session and can be easily used as a replacement. Moreover the template is aware of thread-bound sessions which can be used across several methods, functionality very useful when using a transactional repository.
**RepositoryFactoryBean** which configures, starts and stops the repository instances. As the JSR-170 specification does not address the way the repository should be configured, implementations vary in this regard. The support contains predefined **FactoryBeans** for Jackrabbit and an

abstract base class which can easily support other repositories.
**SessionFactory** which unifies the **Repository**,**Credentials** and **Workspace** interfaces and allows automatic registration of listeners and custom namespaces.
Spring declarative transactional support for repositories that implement the (optional) transactional feature.
**OpenSessionInView** interceptor and filter which allow the usage of the same session per thread across different components. Along with **JcrTemplate**, the retrieval, closure and management of the JCR session is externalized and totally transparent to the caller.
**Rules Auditing Module**
The rules auditing module logs the execution summary of the rules for every request. It has been implemented using standard rules execution listeners. The log files provide information that is sufficient to confirm if the intended rules were executed or not.
**Export and Import of rules**
Rules can be exported from one instance or version of the caAERS application and imported into another instance or version. Each rule set is exported into a separate XML file, which then can be used as an input to import rules into another instance of caAERS. This feature is very useful in distributing rules defined by a sponsor to multiple instances of caAERS. A sponsor can use the BRMS in caAERS to create rules, export them into XML format and then distribute them to all sites which are conducting studies sponsored by it.

## Reporting and Scheduling

Adverse events that are to be reported in expedited fashion require that a report containing them must be delivered within a stipulated time frame. This requires caAERS to provide functionality to configure the 'when, what, to-whom' aspects of expedited adverse event reporting. The reporting module provides functionality to create report definitions which contain information regarding the mandatory fields, delivery options and reminder notifications. The reporting module is seamlessly integrated with the Quartz framework to schedule notifications. The notification framework of Spring is used in-order to send the internal reminder notifications.
The high-level layered architecture of the reporting and scheduling module is shown below:-
Delivery End Points
(E-mail, web service, message Queue) Report Submission
(PDF and XML) Expedited Reporting Rules Engine Web Interface Report Definitions Quartz Scheduling Framework Notification FrameworkcaAERS Reporting & SchedulingusesDefined usingusesusesSchedules Notifications
*Figure 11: Reporting and Scheduling module*

## Dependency injection (Architectural Pattern)

The objects in different layers are connected or wired together using an architectural pattern called dependency injection/inversion of control. This means that, instead of a DAO object acquiring a data source from JNDI, or a controller finding a DAO using some sort of service locator, each object declares the objects (commonly referred to as collaborators) it depends on based on its own requirements. All the collaborators are instantiated by an external configuration tool; all their dependencies are satisfied by that tool when they are created.
Inter-layer dependencies are oriented such that higher levels depend on lower levels, but never vice versa. For instance, a service may depend on a DAO, but a DAO will never depend on a service.
caAERS uses the Spring Framework (version 2.0), specifically Spring IOC container for dependency injection, with the dependencies declared in a series of XML files included alongside the deployed classes. For more information about the Spring Framework, please see http://www.springframework.org/documentation.
The following diagram illustrates the wiring of various objects that belong to different layers in caAERS:

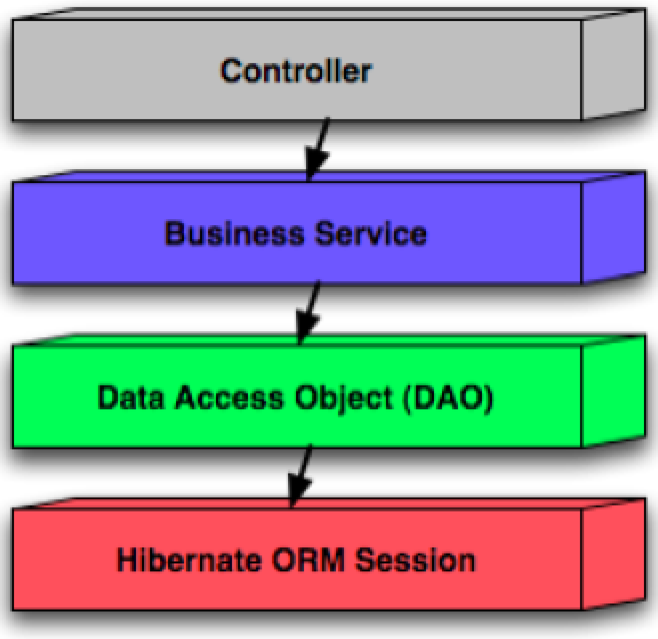


*Figure 12: Spring wiring of caAERS web app components*

## Process View

The sequence diagram below shows a portion of the interactions among the User, Browser, DWR, Spring MVC, and Core components that occur during the creation of an Adverse Event Report.

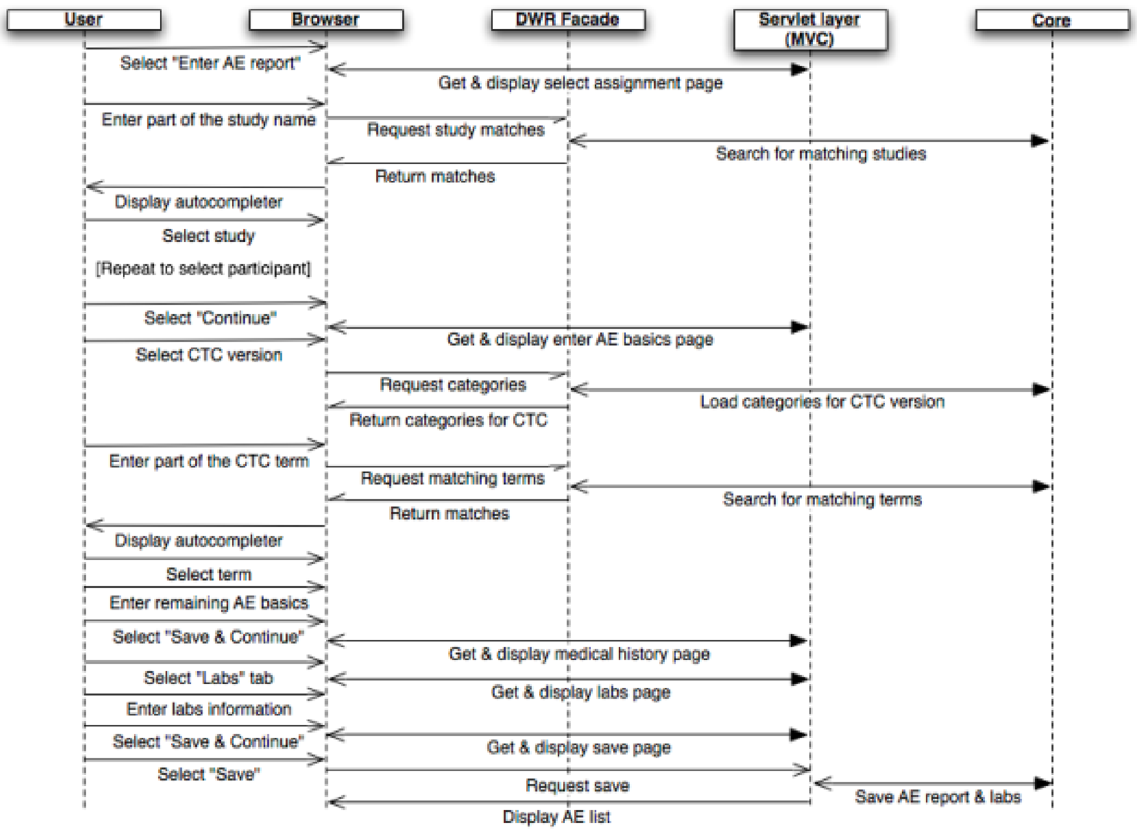


*Figure 13: AE Report flow*

## Component View

The component view shows the grouping of the components in a system and their structural relationship with each other. Components are considered autonomous, encapsulated units within a system or subsystem. One of the purposes of this view is to help aid in understanding how the overall system is built.
Below is the UML component diagram for caAERS:

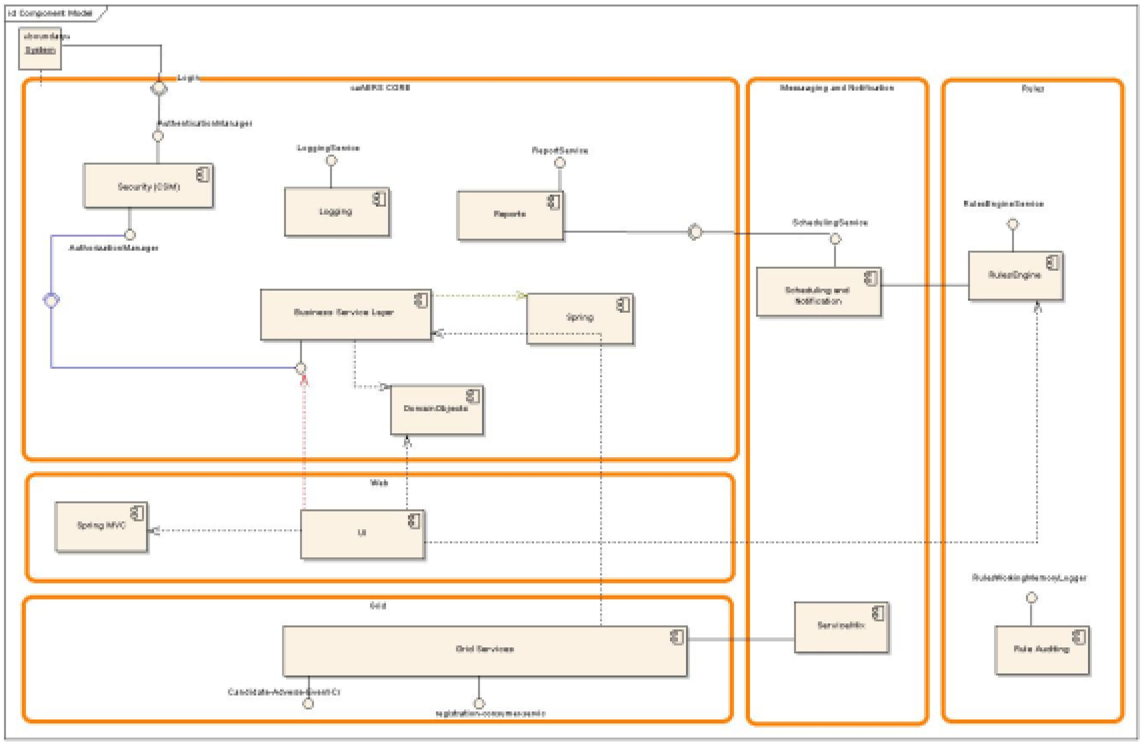


*Figure 14: Component diagram*

**Security**: This component provides authentication and authorization mechanism for the caAERS system. It exposes two interfaces called authentication manager and authorization manager. For implementation of this component Common Security Module (CSM) has been used.
**Grid Services**: This component exposes some of the business interfaces as grid services. These grid services use the business services of caAERS and are the part of data exchange mechanism between caAERS and external systems.
**Business Service Layer:** This component has the business services defined by caAERS application. These services are used by other components within caAERS application.
**Report Service:** This component is used to manage a wide variety of reports. It will also be used to submit reports to different destinations.
**Spring Framework**: Spring inversion of control (IOC) bean container is used for wiring objects that exist in different architectural layers of caAERS system (i.e., spring MVC controllers with business services; business services with DAOs and DAOs with data source). The Hibernate abstraction layer and the generic data access exception hierarchy is also used.
**UI:** This component consumes the business service interfaces. It also leverages domain objects and is implemented using the spring MVC framework.
**Domain Objects**: This component contains all the domain object classes of the caAERS system.
**Messaging**: This sub system has several components such as ServiceMix, Notification System, Message Broker and Scheduling system. ServiceMix is used to implement the workflow involving data exchange with other systems. Various grid services have been strung together using the ServiceMix for implementing the data exchange amongst different CTMS modules.
**Scheduler and Notification**: It is responsible for scheduling the notifications within caAERS. It has been developed using popular open source framework called Quartz.
**Rules**: The rules sub system essentially consists of a rules engine and a business service as an exposed interface. It is used for maintaining and deploying rules.
**Rules Auditing**: This component provides functionality for logging the execution of rules.

## Deployment View

The following diagram depicts the current deployment of caAERS in our DEV environment.

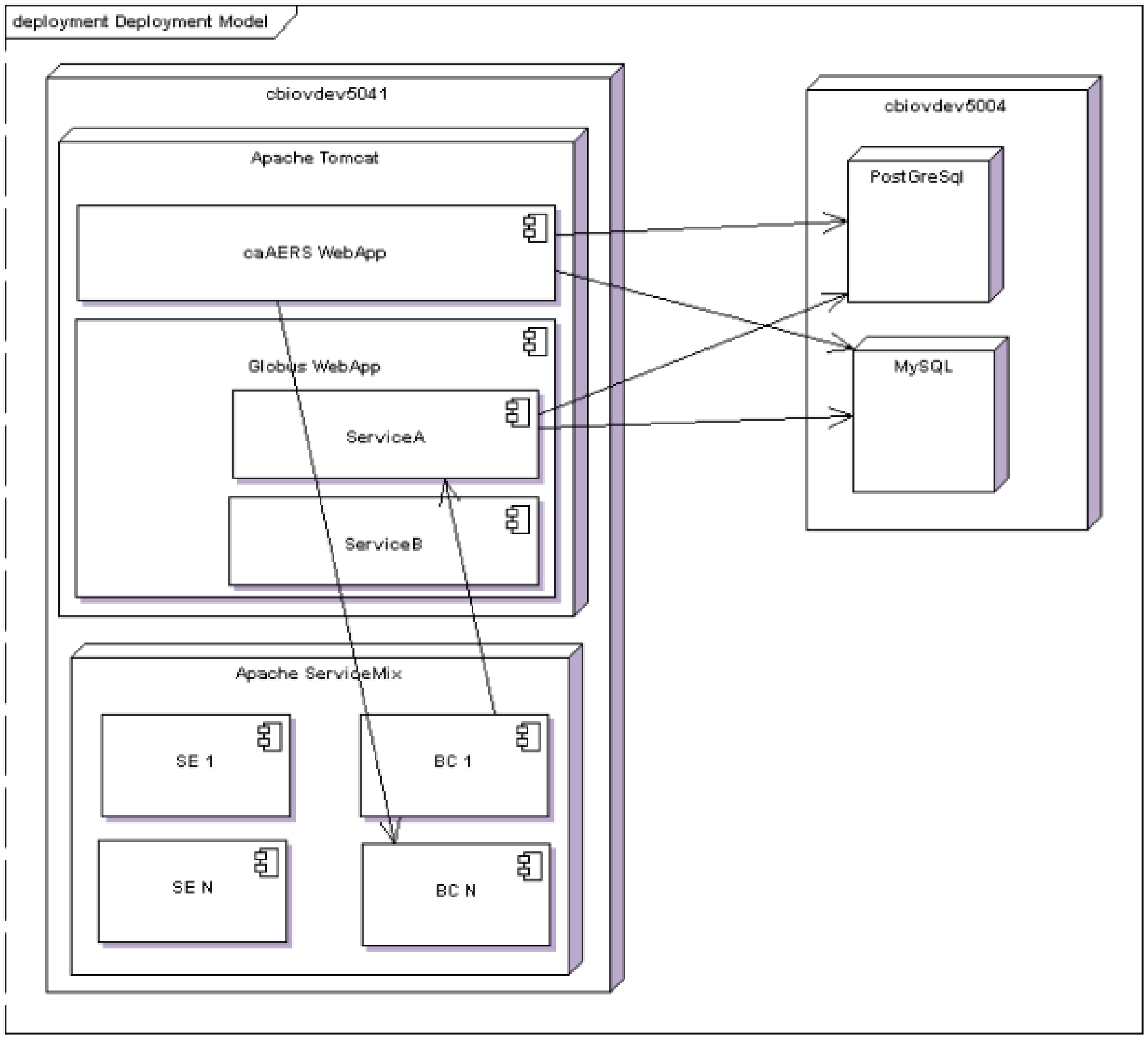


*Figure 15: DEV environment*

The caAERS web application and the caGrid services are hosted in the same instance of Apache Tomcat. Binding components and service engines are hosted within a single instance of Apache ServiceMix. The caAERS web application and the caGrid services all communicate with database instances on cbiovdev5004. caAERS application data resides in a PostGreSql instance. But, the caGrid authorization component uses CSM, which requires MySQL or Oracle. So, grid-related authorization policy is stored in the MySQL instance.

## Security

The requirements of the caAERS security architecture are to ensure the privacy and integrity of all network communication, and to enforce authentication and authorization before granting access to operations and data. This architecture is also required to enforce authentication and authorization of users and services that may be outside of the security authority of the host organization.

Privacy and integrity of data in transit through the network are ensured by using Transport Level Security (TLS). Access to data at rest in a database is protected using basic (username/password) authentication mechanisms. Authentication is enforced using a combination of Public Key Infrastructure (PKI) and basic authentication. Authorization policy is constructed and maintained using NCICB's Common Security Module (CSM). Authorization policy is enforced using custom components in the web application layer and the caGrid Authz module, which enables local and external (grid-wide) policy to be enforced in parallel.

PKI-based authentication works as follows. Users and services are provisioned with X509v3 certificates that have been signed by widely-trusted certificate authorities. During negotiation of security communication between parties, party A authenticates party B by verifying that the certificate of party B has been signed by a certificate authority that party A trusts. Both uni- and bi-directional (mutual) authentication can be required.

Basic authentication involves looking up the requesting parties credentials (username and password) in some credential provider. CSM supports LDAP and RDBMS credential providers.

Local users have a username and password that have been provisioned by the hosting organization and are authenticated using basic authentication. Grid users have a proxy certificate (short-lived X509 certificate) and are authenticated using PKI authentication.

The following sections describe how security issues are addressed at each layer of the caAERS architecture.

## Web Application

The caAERS web application runs in a servlet container. Web browser clients make requests to the container, and code running in the container sends and receives messages from a JMS broker.
To secure communication between the web browser and the web application, the servlet container is configured to accept connections over HTTPS, and the caAERS web application is configured to require that pages containing sensitive data can only be accessed over HTTPS. In this way, all data communicated between the browser and the application are encrypted.
To secure communication between the web application and the JMS broker, the broker is configured to accept SSL connections and the client is configured to use a SSL connection to the broker.

### Authentication

X509 certificates are used to authenticate the servlet container to the web browser client using PKI. Similarly, the JMS brokers certificate is used to authenticate the broker to the caAERS JMS client.
Local users must provide username and password credentials which are validated by CSM against the credential provider (LDAP or RDBMS). Authentication of local users is enforced in the web application as follows.
All HTTP requests in the Spring MVC framework go through the DipatcherServlet, which serves as a front controller. It uses a HandlerMapping object to figure out the Controller class that should be used to handle the request. The mapping between the URL and the controller class is configured in the spring-servlet.xml file. The DispatcherServlet also retrieves all the interceptors that are configured for the controller before delegating the request to the controller class. A LoginCheckInterceptor is used to check if the user is logged in and has a valid active session. If the login check fails, the user is redirected to the login page.
Grid users must provide a proxy certificate which is validated using PKI. Authentication of grid users is enforced by the WEBSSO module. See the WEBSSO section for details.

### Authorization

Both role-based and identity-based authorization. Role-based authorization policy applies in two ways. First is access to URLs - roles are assigned access permissions to the pages they need to function. Second is attribute-based - some pages mutate based on the roles granted to the logged-in user. Certain links or forms will only appear if a user has a role that grants him or her access to the target page.
caAERS also uses identity-based authorization for specific resources. For example, it is possible to limit access an individual item to one or more users.
Authorization is enforced through a combination of a Spring interceptor, custom JSP tags, and the caGrid Authz component.
As described previously, the Spring DispatcherServlet passes an incoming request to the chain of interceptors before delegating to the Controller. One of these interceptors is the URLAccessCheckInterceptor, which verifies if the logged in user is authorized to access the page that he is requesting, based on his role. If the user does not have privileges to access a page then the access denied page is displayed. The URLAccessCheckInterceptor uses the CSM API to determine the roles of the logged in user.
Similarly, custom JSP tags control the rendering of a page based on the user's role, so that only sections which expose functionality and data to which the user is authorized are displayed.
This works fine for local users because they can be provisioned with roles using the organization's CSM. However, it is impractical to provision all possible grid users. Instead, the Authz component is plugged into CSM to enable grid-based authorization policy to be enforced in parallel with local policy. See the caGrid User's Guide for more information about the Authz component.

## Data Access

caAERS uses the Spring Hibernate abstraction layer to manage persistence to an RDBMS. Both Oracle and PostGreSQL are supported.
The DB connection pool is configured to use encrypted connections to the RDBMS.
Authentication
The caAERS data source must provide a username and password to establish database connections. And the database server must provide a certificate that is trusted by the caAERS application.
Authorization
Authorization is managed within the RDBMS.

## caGrid Services

Each of the caAERS grid services uses standard Globus/caGrid security mechanisms
The servlet container in which the grid services run is configured to use HTTPS.
Authentication
Mutual PKI-based authentication of grid service and grid client is implemented by the Globus Grid Security Infrastructure (GSI).
Authorization
Local authorization policy is mapped to grid-wide authorization policy using the conventions defined by caGrid Authz component. Grid-based authorization policy is enforced within the grid services using Authz Policy Decision Point (PDP) approach. See the caGrid wiki for further information (http://www.cagrid.org/mwiki/index.php?title=CaGrid:How-To:IntegrateCSMAuthorizationPolicy)

## ESB

caAERS interacts with external services and clients across the ESB through binding components.
Authentication
As discussed previously, the caAERS web application sends and receives messages through a JMS broker. This broker is hosted within the ESB as a JMS binding component. While the JMS binding component is required to authenticate itself to clients, the clients are not required to authenticate themselves to the JMS. Therefore, it is the responsibility target services to authenticate all requests.
caAERS grid services receive requests across the ESB through out-bound grid service binding components. Mutual, PKI-based authentication is used.
Within the ESB, among service engines, no authentication is performed. However, in order to facilitate interoperability among in-bound and out-bound components provided by other members of the CTMS suite of applications, as standard representation of grid credentials is assumed. An X509 proxy certificate is used as the security token that enables SSO among applications in the CTMS suite, both at the web application layer and the grid service layer. To represent this proxy in the normalized message that is routed through the ESB, we populate the JAAS security subject of the normalized message in the following way.

- A GridIdentifierPricipal (from the caXchange code base), which represents the grid identity of the caller, is added to the set of principals.
- The private key of the certificate is added to the set of private credentials.
- The certificate chain is added to the set of public credentials.

Authorization
caAERS uses Apache ServiceMix as the ESB implementation. ServiceMix is an implementation of the Java Business Integration (JBI) standard (JSR 208). Security in JBI is based on the JAAS framework, and ServiceMix supports enforcement of role-based authorization policy based on the principals associated with JAAS security principal of the normalized message. Since we are currently using the ESB to facilitate integration among external services, rather than hosting business logic in service engines, we do not plan to make use of this feature.

## WEBSSO

The websso component of caAERS is used by web applications in the CTMS suite to implement SSO at the web application layer. It is designed to be easily integrated into an existing web application. The following describes how this component works.
First, some background. In caGrid, authentication is achieved through the use of X.509 certificates. Therefore, each user needs a certificate in order to access a secure resource on the grid. The caGrid components that enable each user to obtain a certificate are the Dorian service and the AuthenticationService service. AuthenticationService plays the role of an Identity Provider (IdP). Dorian plays the role of an Identity Federation Service (IFS).
A user obtains a certificate by following these steps:

1. User provides IdP username and password
2. IdP returns a singed SAML assertion attesting to the user's identity
3. User provides SAML assertion to IFS
4. IFS returns a certificate

The certificate returned by the IFS is a proxy certificate, which we commonly refer to as a grid proxy. It is basically just a short-lived certificate (usually valid for only 12 hours).
The websso component implements SSO across web applications using these grid proxies. There are four components involved in this implementation:

- gov.nih.nci.cabig.ctms.web.sso.GridProxyFilter
- gov.nih.nci.cabig.ctms.web.sso.GrixProxyRetrievalStrategy
- gov.nih.nci.cabig.ctms.web.sso.GridBasicAuthenticationClient
- gov.nih.nci.cabig.ctms.web.sso.GridProxyValidator

### GridProxyFilter

This is a servlet filter which will use an instance of GridProxyRetrievalStrategy to check for the presence of a grid proxy in the request or session scopes. If not found, it will redirect to a configured page. If found, it will continue chain processing. This filter can operate in "strict" or "non-strict" modes. In non-strict mode, the filter will continue chain processing even if a proxy is not found. This permits use of the application by users that do not have grid identities.
This filter uses Spring to instantiate the desired implementation of GridProxyRetrievalStrategy.

### DefaultGridProxyRetrievalStrategy

This implementation of GridProxyRetrievalStrategy looks for a proxy first in the request and then the session under a configured parameter/attribute name. If a proxy is found, it delegates to an instance of GridProxyValidator to validate the proxy.
It can also me configured to check for username/password credentials under configurable parameter/attribute names. If these credentials are

found, this class will delegate to an instance of GridBasicAuthenticationClient to obtain a grid proxy.
If a grid proxy is located (or newly obtained) and successfully validated, it is placed under a configurable attribute name in the session.
DefaultGridProxyRetrievalStrategy is the class that should be extended to work with a local authentication mechanism. In caAERS, it is gov.nih.nci.cabig.caaers.tools.accesscontrol.CaaersGridProxyRetrievalStrategy that interacts with this systems ApplicationSecurityManager to set the current user to the grid identity obtained from the proxy.

## GridBasicAuthenticationClient

This is simple API that handles interaction with the IdP and IFS. The single method looks like this:
/**

- @param username
- @param password
- @return String representation of grid proxy
- @throws AuthenticationErrorException if an error is encountered during authentication
  */
  String authenticate(String username, String password) throws AuthenticationErrorException;

  ### GridProxyValidator

  This class the JGlobus ProxyPathValidator to verify that the proxy is valid and check that it has been signed by one of the "trusted" certificates and has not been revoked. The convention used by Globus is to store trusted certificates under $HOME/.globus/certificates. So, this class checks the proxy against certificates in that directory by default. But custom directories for trusted certificates and certificate revocation lists (CRLs) can be provided.

  ### SyncGTS

  SyncGTS is the caGrid component that automates the process of keeping a node (machine) synchronized with the grid trust fabric. That means, keeping the adding/remove trusted certificates and CRLs from the local file system. In order for the websso component to work properly, syncGTS must be installed on the machine that hosts the web application.

  ### Integration Steps

  Three steps are required to integrate this approach into a web application. First, GridProxyFilter must be configured in web.xml. Second, the existing authentication mechanism must ensure that grid proxy ends up in the session under a pre-defined attribute name. Third, "hot links" between applications must include the grid proxy as a pre-defined parameter in the request.
  The websso component has been integrated into caAERS by extending DefaultGridProxyRetrievalStrategy to interact with the ApplicationSecurityManager. Essentially, it uses ApplicationSecurityManager to put the grid identity of the caller into the session. Also, the LoginController first checks if the caller already exists in the session before attempting to authenticate to the local credential provider. Finally, "hot links" to other CTMS suite applications are implemented by intercepting JavaScript click events on anchor tags that have a CSS class "sso", and submitting a form (implemented as a custom JSP tag) that contains a hidden field whose value is the string representation of the grid proxy. In this way, the grid proxy is included as a parameter of an HTTP POST to the target application. # References

# References

## Document Text Conventions

The following table shows various typefaces to differentiate between regular text and menu commands, keyboard keys, and text that you type. This illustrates how conventions are represented in this guide.

| *Convention* | *Description* | *Example* |
|---|---|---|
| **Bold & Capitalized Command Capitalized command > Capitalized command** | Indicates a Menu command<br>Indicates Sequential Menu commands | **Admin > Refresh** |

| | | |
|---|---|---|
| text in small caps | Keyboard key that you press | Press enter |
| text in small caps + text in small caps | Keyboard keys that you press simultaneously | Press shift + ctrl and then release both. |
| Special typestyle | Used for filenames, directory names, commands, file listings, source code examples and anything that would appear in a Java program, such as methods, variables, and classes. | URL_definition ::= url_string |
| **Boldface type** | Options that you select in dialog boxes or drop-down menus. Buttons or icons that you click. | In the Open dialog box, select the file and click the **Open** button. |
| *Italics* | Used to reference other documents, sections, figures, and tables. | *caCORE Software Development Kit 1.0 Programmer's Guide* |
| ***Italic boldface type*** | Text that you type | In the New Subset text box, enter ***Proprietary Proteins.*** |
| **Note:** | Highlights a concept of particular interest | **Note:** This concept is used throughout the installation manual. |
| **Warning!** | Highlights information of which you should be particularly aware. | **Warning!** Deleting an object will permanently delete it from the database. |
| {} | Curly brackets are used for replaceable items. | Replace {root directory} with its proper value such as c:\cabio |

*Table 1. 2 Document Conventions*

## Technical Manuals/Articles

1. caAERS Knowledge Center https://cabig-kc.nci.nih.gov/CTMS/KC/index.php/CaAERS
2. National Cancer Institute. "caCORE 2.0 Technical Guide", ftp://ftp1.nci.nih.gov/pub/cacore/caCORE2.0_Tech_Guide.pdf
3. Java Bean Specification: http://java.sun.com/products/javabeans/docs/spec.html
4. Foundations of Object-Relational Mapping: http://www.chimu.com/publications/objectRelational/
5. Object-Relational Mapping articles and products: http://www.service-architecture.com/object-relational-mapping/
6. Hibernate Reference Documentation: http://www.hibernate.org/hib_docs/reference/en/html/
7. Basic O/R Mapping: http://www.hibernate.org/hib_docs/reference/en/html/mapping.html
8. Java Programming: http://java.sun.com/learning/new2java/index.html
9. Javadoc tool: http://java.sun.com/j2se/javadoc/
10. JUnit: http://junit.sourceforge.net/
11. Extensible Markup Language: http://www.w3.org/TR/REC-xml/
12. XML Metadata Interchange: http://www.omg.org/technology/documents/formal/xmi.htm

## Glossary

.

| ***Term*** | ***Definition*** |
|---|---|
| API | Application Programming Interface |
| caBIG | cancer Biomedical Informatics Grid |
| CSM | Common Security Module |
| DAO | Data Access Objects |
| HTTP | Hypertext Transfer Protocol |
| JAR | Java Archive |
| Javadoc | Tool for generating API documentation in HTML format from doc comments in source code (http://java.sun.com/j2se/javadoc/) |
| JDBC | Java Database Connectivity |
| JSP | JavaServer Pages |

| | |
|---|---|
| JUnit | A simple framework to write repeatable tests (http://junit.sourceforge.net/) |
| ORM | Object Relational Mapping |
| RDBMS | Relational Database Management System |
| SDK | Software Development Kit |
| SQL | Structured Query Language |
| UML | Unified Modeling Language |
| UPT | User Provisioning Tool |
| URL | Uniform Resource Locators |
| WAR | Web Application Archive |
| XMI | XML Metadata Interchange (http://www.omg.org/technology/documents/formal/xmi.htm) - The main purpose of XMI is to enable easy interchange of metadata between modeling tools (based on the OMG-UML) and metadata repositories (OMG-MOF) in distributed heterogeneous environments |
| XML | Extensible Markup Language (http://www.w3.org/TR/REC-xml/) - XML is a subset of Standard Generalized Markup Language (SGML). Its goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML |

## Copyright and License