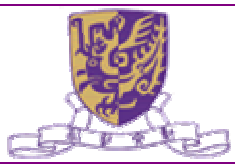


CUHK RFID Middleware - System Design Document

Prepared By : Andy Mak
Anthony Lam
Daiming Qu

Report No : RFID-SDD

Version : 1.0

Issue Date : 10 August 2007

Table of Content

Revision History

Date	Version	Description	Author
10 Aug 2007	1.0	First Release	Andy Mak Anthony Lam Daiming Qu

1 Executive Summary

Radio Frequency Identification (RFID) middleware is a new breed of software system which facilitates data communication between automatic identification equipments like RFID readers and enterprise applications. It provides a distributed environment to process the data coming from tags, filter and then deliver it to a variety of backend applications via various communication protocols including web services.

This document describes the system design of the CUHK RFID middleware. The middleware is developed based on the Application Level Events (ALE) Specification Version 1.0 from EPCGlobal, together with some CUHK specific extensions. The functions CUHK middleware provides are summarized as follows:

1. Receiving EPCs from one or more data sources
2. Accumulating data over intervals of time
3. Filtering
   - eliminate duplicate EPCs
   - filter off EPCs that are not of interest
4. Manipulating (grouping & counting) to reduce volume of data
5. Reporting

The middleware is designed as a J2EE application hosted in JBoss server. It connects databases via JDBC. The middleware interfaces with its clients via ALE standard interface – SOAP for ALE Clients while HTTP/TCP for Subscribers. The middleware also interacts with readers via reader adaptors. An application Management Console has been developed for system administration.

The middleware currently supports 4 service endpoints, namely *ALEService*, *TagDataService*, *ReaderManager* and *Notifier*. The service endpoints serve as points of communication with external clients. *ALEService* and *TagDataService* are accessible as web services by using SOAP over HTTP. The *ReaderManager* can be accessed through RMI/JRMP, while the *Notifier* communicates with subscribers via HTTP or TCP.

The middleware may need to deal with many active readers at the same time. To handle multiple tag reads simultaneously without performance impact, two database instances are used - one in memory and the other in disk.

In order to minimize the complexity of middleware and ease server implementation, the middleware implements one single neutral reader protocol, which assumes that readers are working in an autonomous mode. Since the actual physical readers do not know about the standard neutral protocol, Reader Adaptor is implemented to act as a relay between the physical reader and the middleware. If a physical reader supports autonomous mode, the adaptor simply acts as a relay. If a physical reader only supports poll mode (as in the CUHK reader), the adaptor then emulates autonomous mode by polling the reader at regular intervals (the Read Cycle). The Reader Adaptor also performs registration for the reader in the middleware.

The ALE implementation assumes readers with minimal intelligence, i.e. readers can only report tag reads and cannot do advanced processing such as pattern filtering. The core business logics of the middleware are divided into components and are implemented as Enterprise JavaBeans (EJB).

2 Introduction

Radio Frequency Identification (RFID) middleware is a new breed of software system which facilitates data communication between automatic identification equipments like RFID readers and enterprise applications. It provides a distributed environment to process the data coming from tags, filter and then deliver it to a variety of backend applications via various communication protocols including web services. Apart from providing an application-level interface for managing readers and querying RFID observations, it encapsulates applications from device interfaces. It also processes raw observations captured by the readers and sensors so that applications see only meaningful, high-level events, thereby lowering the volume of information that they need to process.

This document describes the system design of the CUHK RFID System. The implementation of CUHK RFID system is not only compatible with EPCglobal Application Level Events (ALE) specification version 1.0, but also provides CUHK specific extensions. In compliance to the specification, the middleware does not support vendor extension to the ECSpec and ECReports XML Schemas, but supports the optional FILE Notification URI for writing of ECReports in XML to a file.

Overall functional components of CUHK RFID System as well as the framework are illustrated in the following sections, which are organized as follows:

Section 3 provides the references required for understanding the rationale of the system design and the standard specifications. The relevant references are necessary as the supplements to this design document. Section 4 describes the architectural representation of the document, which basically adopts 4+1 View Model. The architectural descriptions of the system are organized from different perspectives, each of which addresses a specific set of concerns. Section 5 depicts the relations of the system with external environments; this demonstrates the interoperability and compatibility of the middleware with external systems.

Section 6 describes the functions provided by the system, and explains the details in terms of use cases. Section 7 provides the logical view of the system, which includes the architectural design and design mechanism of the system. Section 8 gives the process view of the system. This section describes the details of various interfaces, including (1) application to middleware interface, (2) middleware to application interface, (3) adaptor to middleware interface, (4) middleware to adaptor interfaces, and (5) the interfaces between middleware and management console.

According to the design considerations, Section 9 provides the implementation view of the system. This section states the details of the design in terms of class diagrams, collaboration diagrams and package diagrams.

Section 10 provides the deployment view. This section describes the environment setup and the necessary deployment procedures. Data view is provided in Section 11, which gives the design details of the involved database schema.

Last but not least, Section 12 lists the additional properties of the system.

3 References

Users are expected to acquire general knowledge of middleware and the details listed in the corresponding specifications. The relevant references are listed in the following table as the necessary supplement to this design document.

Document	Version	Date	Organisation
The Application Level Events (ALE) Specification	1.0	15 Sep 2005	EPCglobal Inc.
Accada's EPCIS Implementation Developer Guide	0.2.0	8 May 2007	Accada
EPC Information Services (EPCIS) Version 1.0 Specification	1.0	12 Apr 2007	EPCglobal Inc.
Quick Start Guide of Middleware Installation	1.0	July 2007	CUHK
CUHK ALE Middleware - Test Plan	1.0	31 Aug 2007	CUHK
CUHK ALE Middleware - Test Cases	1.0	31 Aug 2007	CUHK
Tag Capturer - System Design Document	1.0	30 Jul 2007	CUHK

4 Architecture Representation

The architectural representation will basically adopt the 4 + 1 View Model as recommended, to organize the architectural description from different perspectives, each of which addresses a specific set of concerns:

Requirement View:	describes the software requirements, functional and non-functional, illustrated by significant use cases and scenarios
Logical View:	describes the object model of the design, the system decomposition into layers and subsystems, and the dependencies between them
Process View:	describes the concurrency and synchronization aspects of the design
Implementation View:	describes the software's static organization in the development environment
Deployment View:	describes the mapping of the software onto hardware
Data View:	describes the database design for the software

It allows various stakeholders to find what they need in the software architecture. System engineers can approach it from the logical view, process view and deployment view. DBA can approach it from the data view. Project managers and software configuration managers can approach it from the implementation view.

5 Relation to External Environment

This section provides high-level descriptions of the external systems that relate with CUHK RFID Middleware. Three applications were developed together with CUHK RFID Middleware for the purpose of demonstrating the overall data flow in the RFID environment, namely Management Console, Reader Emulator and Tag Capturer.

- Management Console – provides a bird-view of the running system and provides administrative and management functions.
- Reader Emulator – provides hardware reader emulation to the middleware.
- Tag Capturer – demonstrates the basic functionalities, including EPCIS connectivity of the middleware.

Please refer to the corresponding design document for the details of each of these applications.

.

6 Requirement View

The CUHK RFID middleware implements the EPCglobal standard, "The Application Level Events (ALE) specification, version 1.0. In compliance to the specification, the middleware does not support vendor extension to the ECSpec and ECReports XML Schemas, but supports the optional FILE Notification URI for writing of ECReports in XML to a file.

Apart from the standard system functions, the middleware supports CUHK Extensions.

6.1 System Functions

The CUHK RFID Middleware provides the following functions:

1. Receive EPCs from one or more data sources
2. Accumulate data over intervals of time
3. Filtering
   - eliminate duplicate EPCs
   - filter off EPCs that are not of interest
4. Manipulate (grouping & counting) to reduce volume of data
5. Reporting

6.2 CUHK Extensions

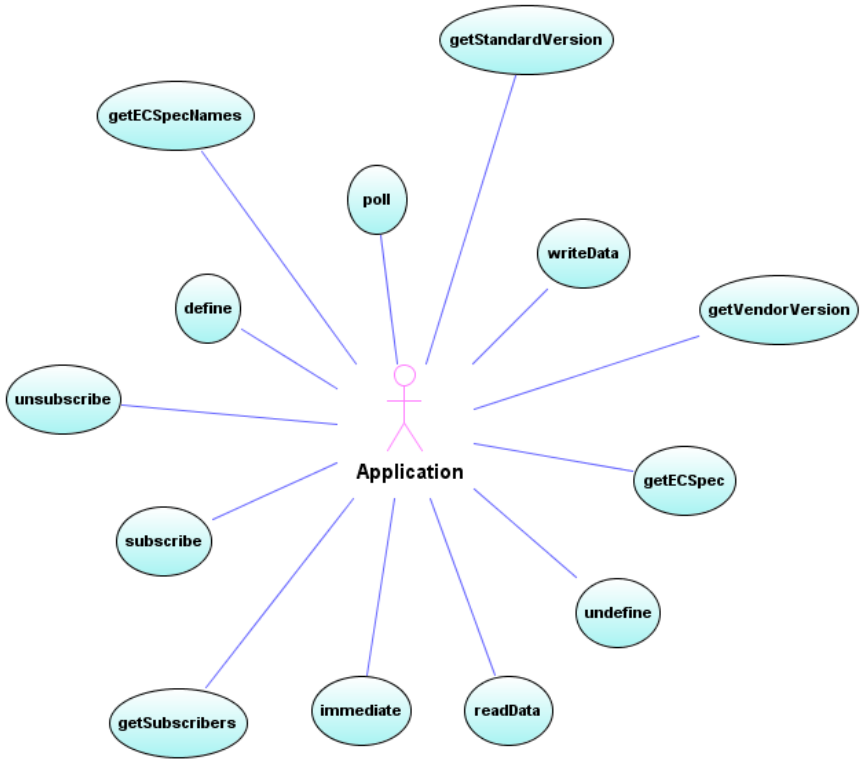
Since the ALE Specification does not specify a standard mechanism for middleware to receive EPCs from data sources, proprietary implementations may be needed to cater for various readers. In view of flexibility enhancement, it is recommended to simplify reader connections from different vendors without jeopardizing the standard compliance of the middleware. As a result, a reader to middleware protocol has been devised and the middleware implements this neutral protocol. For any other readers that communicate by using this protocol, an adaptor is needed to be developed for the translation.

To facilitate system administration, a CUHK middleware management console has been developed to allow direct connection from the middleware. The management console generates a bird's eye view of the running system and provides administrative and management functions.

The standard specifies only the manipulation of EPCs as a set of tag IDs, and does not provide any means for operating the data contained in tags. As we foresee that more and more RFID tags will be supporting data operation, especially with the use of active tags, we have extended the standard middleware to support tag data reading and writing.

6.3 Use Cases

The following diagram describes the standard operations required by the specification supported in the middleware.



Actor	DEFINITION
Application	Software applications that interact with the middleware to perform operations as defined by the ALE interface

6.3.1 Define

6.3.1.1 Description

This use case describes the definition of an ECSpec in the middleware

6.3.1.2 Actors

- Application

6.3.1.3 Preconditions

- None

6.3.1.4 Major Flow of Event

1. Application to create an ECSpec object
2. Application to define the ECSpec object with a name

6.3.1.5 Alternate Flows

- DuplicateNameException is thrown when the ECSpec name already exists
- ECSpecValidationException is thrown when the specified ECSpec is invalid
- SecurityException is thrown when the operation was not permitted due to an access control violation or other security concern
- ImplementationException is thrown when other implementation errors occur

6.3.1.6 Post-conditions

- An ECSpec with the specified name is defined in the middleware
- The ECSpec enters the Unrequested state

6.3.2 Undefine

6.3.2.1 Description

This use case describes the definition removal of an ECSpec in the middleware

6.3.2.2 Actors

- Application

6.3.2.3 Preconditions

- The ECSpec is defined
- The ECSpec is at the Unrequested state

6.3.2.4 Major Flow of Event

1. Application to remove the definition of an ECSpec object with a name

6.3.2.5 Alternate Flows

- NoSuchNameException is thrown when the ECSpec name does not exist
- SecurityException is thrown when the operation was not permitted due to an access control violation or other security concern
- ImplementationException is thrown when there were other implementation errors

6.3.2.6 Post-conditions

- The ECSpec with the specified name is undefined in the middleware

6.3.3 GetECSpec

6.3.3.1 Description

This use case describes getting an ECSpec from the middleware

6.3.3.2 Actors

- Application

6.3.3.3 Preconditions

- The ECSpec is defined

6.3.3.4 Major Flow of Event

1. Application to get the ECSpec object with a name

6.3.3.5 Alternate Flows

- NoSuchNameException is thrown when the ECSpec name does not exist
- SecurityException is thrown when the operation was not permitted due to an access control violation or other security concern
- ImplementationException is thrown when other implementation errors occur

6.3.3.6 Post-conditions

- The ECSpec with the specified name is returned

6.3.4 GetECSpecNames

6.3.4.1 Description

This use case describes getting a list of names of the ECSpecs defined the middleware

6.3.4.2 Actors

- Application

6.3.4.3 Preconditions

- None

6.3.4.4 Major Flow of Event

1. Application to query for the names

6.3.4.5 Alternate Flows

- SecurityException is thrown when the operation was not permitted due to an access control violation or other security concerns
- ImplementationException thrown when other implementation errors occur

6.3.4.6 Post-conditions

- A list of ECSpec names are returned

6.3.5 Subscribe

6.3.5.1 Description

This use case describes the subscription of an ECSpec. The ECSpec will generate event cycles as long as there is at least one subscriber. Results for each event cycle are sent to the notification URIs as provided by the subscribers.

6.3.5.2 Actors

- Application

6.3.5.3 Preconditions

- The ECSpec is defined

6.3.5.4 Major Flow of Event

1. Application to subscribe the ECSpec, providing the name of the ECSpec and a notification URI for receiving the results

6.3.5.5 Alternate Flows

- NoSuchNameException is thrown when the ECSpec name does not exist
- InvalidURIException is thrown when the URI specified for a subscriber cannot be parsed
- DuplicatedSubscriptionException is thrown when the specified ECSpec name and the subscriber URI is identical to a previous subscription that was created and not yet unsubscribed
- SecurityException is thrown when the operation was not permitted due to an access control violation or other security concerns
- ImplementationException is thrown when other implementation errors occur

6.3.5.6 Post-conditions

- The ECSpec enters the Requested or Active state when it is first subscribed

6.3.6 Unsubscribe

6.3.6.1 Description

This use case describes the un-subscription of an ECSpec.

6.3.6.2 Actors

- Application

6.3.6.3 Preconditions

- The ECSpec is defined
- The application has previously subscribed the ECSpec with the notification URI

6.3.6.4 Major Flow of Event

1. Application to unsubscribe the ECSpec, providing the name of the ECSpec and a notification URI for receiving the results

6.3.6.5 Alternate Flows

- NoSuchNameException is thrown when the ECSpec name does not exist
- NoSuchSubscriberException is thrown when the subscriber does not exist
- InvalidURIException is thrown when the URI specified for a subscriber cannot be parsed
- SecurityException is thrown when the operation was not permitted due to an access control violation or other security concern
- ImplementationException is thrown when other implementation errors occur

6.3.6.6 Post-conditions

- The ECSpec enters the Unrequested state when it is last unsubscribed

6.3.7 Poll

6.3.7.1 Description

This use case describes polling of an ECSpec. The poll call is similar to subscribing and then unsubscribing immediately after one event cycle is generated except that results are returned from poll instead of being sent to a notification URI.

6.3.7.2 Actors

- Application

6.3.7.3 Preconditions

- The ECSpec is defined

6.3.7.4 Major Flow of Event

1. Application provides the name of the ECSpec to poll it

6.3.7.5 Alternate Flows

- NoSuchNameException is thrown when the specified ECSpec name does not exist
- SecurityException is thrown when the operation was not permitted due to an access control violation or other security concerns
- ImplementationException is thrown when other implementation errors occur

6.3.7.6 Post-conditions

- ECReports are returned

6.3.8 Immediate

6.3.8.1 Description

This use case describes the 'immediate' action of an ECSpec. The call is like as defining an ECSpec, performing a single poll operation and then undefining it.

6.3.8.2 Actors

- Application

6.3.8.3 Preconditions

- None

6.3.8.4 Major Flow of Event

1. Application to create an ECSpec object
2. Application to invoke the 'immediate' action by passing the ECSpec

6.3.8.5 Alternate Flows

- ECSpecValidationException is thrown when the specified ECSpec is invalid
- SecurityException is thrown when the operation was not permitted due to an access control violation or other security concerns
- ImplementationException is thrown when other implementation errors occur

6.3.8.6 Post-conditions

- ECReports are returned

6.3.9 GetSubscribers

6.3.9.1 Description

This use case describes the 'getSubscribers' action, which returns the lists of subscribers (the notification URIs) of an ECSpec.

6.3.9.2 Actors

- Application

6.3.9.3 Preconditions

- The ECSpec is defined

6.3.9.4 Major Flow of Event

1. Application provides the name of the ECSpec to get the subscriber list

6.3.9.5 Alternate Flows

- NoSuchNameException is thrown when the ECSpec name does not exist
- SecurityException is thrown when the operation was not permitted due to an access control violation or other security concerns
- ImplementationException is thrown when there was other implementation errors

6.3.9.6 Post-conditions

- A list of notification URIs are returned

6.3.10 Get Vendor Version

6.3.10.1 Description

This use case describes the action which returns a string that identifies the vendor extensions this implementation provides.

6.3.10.2 Actors

- Application

6.3.10.3 Preconditions

- None

6.3.10.4 Major Flow of Event

1. Application to get the vendor version

6.3.10.5 Alternate Flows

- None

6.3.10.6 Post-conditions

- A string specifying the vendor version is returned

6.3.11 GetStandardVersion

6.3.11.1 Description

This use case describes the action which returns a string that identifies the version of ALE specification this implementation complies with.

6.3.11.2 Actors

- Application

6.3.11.3 Preconditions

- None

6.3.11.4 Major Flow of Event

1. Application to get the standard version

6.3.11.5 Alternate Flows

- None

6.3.11.6 Post-conditions

- A string specifying the standard version is returned

6.3.12 readData

6.3.12.1 Description

This use case describes the action which reads data from specific tags.

6.3.12.2 Actors

- Application

6.3.12.3 Preconditions

- None

6.3.12.4 Major Flow of Event

1. Application creates a read-data request for a specific tag, with the offset address, and data length.

6.3.12.5 Alternate Flows

- Application gets error code if the tag is not detected in any reader.

6.3.12.6 Post-conditions

- Application receives the data from the specific tag.

6.3.13 writeData

6.3.13.1 Description

This use case describes the action which writes data to specific tag.

6.3.13.2 Actors

- Application

6.3.13.3 Preconditions

- None

6.3.13.4 Major Flow of Event

1. Application creates a write-data request for a specific tag, with the offset address and data length.

6.3.13.5 Alternate Flows

- Application gets error code if the tag is not detected in any reader.

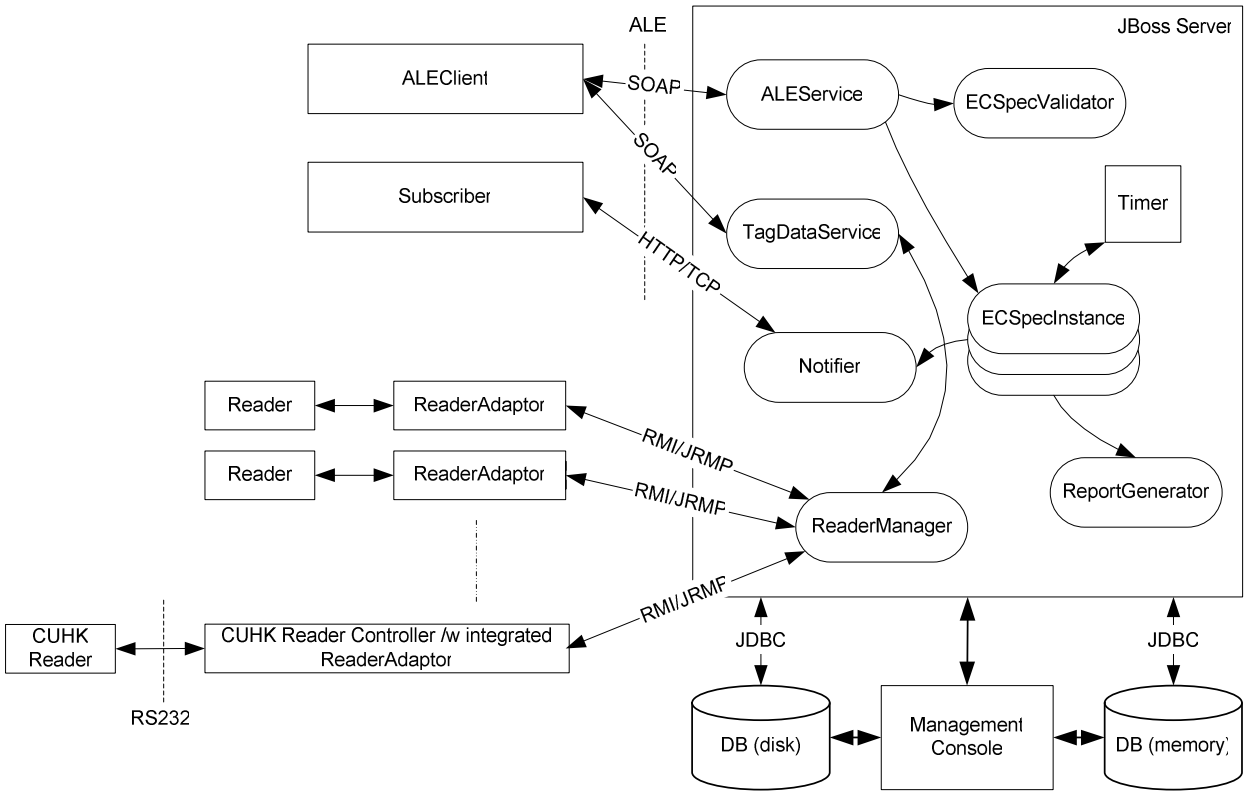
6.3.13.6 Post-conditions

- Data is written to the tag.

7 Logical View

7.1 Architectural Design

The middleware is a J2EE application hosted in JBoss server. It connects databases via JDBC. The middleware interfaces with its clients via ALE standard interface: SOAP for ALEClient, and HTTP/TCP for Subscriber. The ALEClient invokes operations in the middleware using ALE API. The Subscriber is the listener for reporting results. The middleware also interacts with Readers, through the ReaderAdaptor. There is also a Management Console for system administration.



ALEService is a stateless session bean, realized as *ALEServiceBean*. It implements the ALE main API class as defined in ALE specification section 8.1. It is exposed as a web service. The clients access it via SOAP.

TagDataService is a stateless session bean, realized as *TagDataServiceBean*. It implements the CUHK's tag data read/write extension to the middleware. It is exposed as a web service. The clients access it via SOAP.

ECSpecValidator is a standard java utility class. It validates ECSpec according to rules defined in ALE specification section 8.2.11.

ECSpecInstance is an entity bean, realized as *ECSpecInstanceBean*. It represents an ECSpec defined in the middleware. It also models the lifecycle of the ECSpec by supporting the Unrequested, Requested and Active state transition. It works with Timer to handle state transition triggered by timeout.

ReportGenerator is a stateless session bean. It generates ECReports for a given Event Cycle.

Notifier is a message driven bean. It performs HTTP, TCP and FILE notifications.

Timer is a built-in timer service in J2EE 1.4. It supports the various operations in the ALE specification related to timeout.

ReaderManager is a stateless session bean. It is exposed as an EJB service endpoint. It allows reader registration and aggregates tag reads for middleware through interfacing with *ReaderAdaptors*, using RMI/JRMP. Tag reads are stored in the in-memory database for centralized & shared usage by the system.

ReaderAdaptor is a driver program, which interfaces with the native driver of a reader. It performs tag reads on the actual reader and submits the reads to *ReaderManager* via RMI/JRMP. It makes sure that EPCs sent to the middleware in a read cycle are distinct by removing duplicated reads. It also performs reader registration.

7.1.1 Service Endpoints

The middleware currently has 4 service endpoints, namely *ALEService*, *TagDataService*, *ReaderManager* and *Notifier*. The service endpoints serve as the point of communication with external clients. *ALEService* and *TagDataService* are assessable as web services using SOAP over HTTP. *ReaderManager* can be accessed through RMI/JRMP, while *Notifier* communicates with subscribers using HTTP or TCP.

7.1.2 Database Storage

The middleware may need to deal with many active readers at the same time. To handle many tag reads simultaneously without performance impact, two database instances, one in memory, anther in disk, are used.

The in-memory database is used to store tag reads. The middleware does not provide persistence for raw tag reads. Persistence of EPC data is not required in the ALE layer as instructed in the ALE Specification.

7.1.3 Reader Adaptation

The middleware implements one single neutral reader protocol, which assumes that readers are working in an autonomous mode. That means, the readers will fire tag reads to the middleware at a particular frequency when they are active. This is to minimize the complexity of middleware and ease server implementation to support for various reader protocols.

Since the actual physical readers do not know about the standard neutral protocol, *ReaderAdaptor* is implemented to act as a relay between the physical reader and the middleware. To be specific, the adaptor communicates with the physical reader using proprietary protocol and relays the EPC reads to the middleware using the standard protocol. If a physical reader supports autonomous mode, the adaptor simply acts as a relay. If a physical reader only supports poll mode (as in the CUHK reader), the adaptor will emulate autonomous mode by polling it in regular intervals (the Read Cycle).

The *ReaderAdaptor* also performs registration for the reader in the middleware.

The adaptor does not require a 1-1 mapping to the reader. If multiple readers are mapped to an adaptor, they are being treated and managed by the middleware as a single unit.

The adaptor is placed NEAR the reader, and AWAY from the middleware. This approach helps reduce the overhead during polling. If the adaptor is placed NEAR the middleware, then for every poll, the traffic will be generated across the network to the reader, which is inefficient. Since the adaptor is working in autonomous mode, meaning that the middleware-to-adaptor traffic is minimal, therefore, an adaptor placing NEAR the reader, says, in the same LAN, would be optimal. It is expected that the volume of adaptor-to-reader traffic is much more than that for the adaptor-to-middleware traffic.

7.1.4 Reader Intelligence

The ALE implementation assumes readers with minimal intelligence, i.e. readers can only report tag reads and cannot do advanced processing such as pattern filtering.

7.2 Design Mechanism

The core business logics of the middleware are divided into components and are implemented as Enterprise JavaBeans (EJB).

7.2.1 Entity Beans

An entity bean represents a business object in a persistent storage mechanism. So most entity bean APIs are setters and getters, and the underlying data persistence logics are handled inside the function.

7.2.1.1 ECSpecInstanceBean

This bean handles all logics related to a ECSpec. Since state in ECSpec is temporal, the bean also implements *TimedObject* from J2EE specification.

Package

- cuhk.ale.ejb

Implemented Interfaces

- javax.ejb.EntityBean
- javax.ejb.TimedObject

J2EE Remote interface of ECSpecInstanceBean

- Nil (to avoid RMI overhead for entity bean)

J2EE Local interface of ECSpecInstanceBean

- ECSpecInstanceLocal

Function list

Return Type	Declaration
Void	addNotificationURL(java.lang.String specName, java.lang.String url)
Void	cancelTimer(ECSpecInstanceBean.TimerType type)
Void	createTimer(ECSpecInstanceBean.TimerType type, long duration)
Void	deleteNotificationURL(java.lang.String specName, java.lang.String url)
ECSpecInstancePK	ejbFindByPrimaryKey(ECSpecInstancePK pk)
Void	ejbTimeout(javax.ejb.Timer timer)
epcglobal.ale.ECSpec	getECSpec()
ECSpecInstanceValue	getECSpecInstanceValue()
java.util.List	getNotificationURLs(java.lang.String specName)
long	getPreviousEndTime()
long	getPreviousStartTime()
java.lang.String	getSpecName()
long	getStartTime()
int	getState()
int	getStateVersion()
void	setECSpec(epcglobal.ale.ECSpec value)
void	setECSpecInstanceValue(ECSpecInstanceValue value)

void	setPreviousEndTime(long l)
void	setPreviousStartTime(long l)
void	setSpecName(java.lang.String value)
void	setStartTime(long l)
void	setState(int value)
void	setStateVersion(int value)
void	startTriggerReceived()
void	stopTriggerReceived()
void	subscribe(java.lang.String notificationUrl)
void	Unsubscribe(java.lang.String notificationUrl)

7.2.2 Session Beans

Session Beans implement business tasks.

7.2.2.1 ALEServiceBean

This bean handles all front tier logics for ALE standard. The bean implements all the APIs as listed in Section 8 in The Application Level Events (ALE) Specification, Version 1.0. All RMI calls (including SOAP) should be directed to these functions in order to realize ALE functions.

Package

- cuhk.ale.ejb

Implemented Interfaces

- javax.ejb.SessionBean

J2EE Remote interface of ALEServiceBean

- ALEService

J2EE Local interface of ALEServiceBean

- ALEServiceLocal

Function list

Return Type	Declaration
Void	define(java.lang.String specName, epcglobal.ale.ECSpec spec)
epcglobal.ale.ECSpec	getECSpec(java.lang.String specName)
java.util.List	getECSpecNames()
java.lang.String	getStandardVersion()
java.util.List	getSubscribers(java.lang.String specName)
java.lang.String	getVendorVersion()
epcglobal.ale.ECReports	immediate(epcglobal.ale.ECSpec spec)
epcglobal.ale.ECReports	poll(java.lang.String specName)
Void	startTrigger(java.lang.String specName)
Void	stopTrigger(java.lang.String specName)
Void	subscribe(java.lang.String specName, java.lang.String notificationURI)
Void	undefine(java.lang.String specName)
Void	unsubscribe(java.lang.String specName, java.lang.String notificationURI)

7.2.2.2 ReportGeneratorBean

The bean is called in each event cycle for each ECSpec to generate the ECReport.

Package

- cuhk.ale.ejb

Implemented Interfaces

- javax.ejb.SessionBean

J2EE Remote interface of ReportGeneratorBean

- ReportGenerator

J2EE Local interface of ALEServiceBean

- ReportGeneratorLocal

Function list

Return Type	Declaration
epcglobal.ale.ECReports	generateReports(cuhk.ale.EventCycle eventCycle)

7.2.2.3 ReaderManagerBean

The bean receives the events submitted from the adaptor into the middleware.

Package
- cuhk.ale.ejb

Implemented Interfaces
- javax.ejb.SessionBean

J2EE Remote interface of ReaderManagerBean
- ReaderManager

J2EE Local interface of ReaderManagerBean
- ReaderManagerLocal

Function list

Return Type	Declaration
Void	submitALEPhysicalReaderInfo(java.lang.String readerID,cuhk.ale.PhysicalReaderInfo physicalReaderInfo)
Void	submitALETags(java.lang.String readerID,java.util.List tags)

7.2.2.4 TagDataServiceBean

This bean provides the middleware the support of tag data reading and writing. These functions are not specified in the EPCGlobal middleware standard but is designed and implemented as CUHK extensions to the middleware.

Package
- cuhk.ale.ejb

Implemented Interfaces
- javax.ejb.SessionBean

J2EE Remote interface of TagDataServiceBean
- TagDataService

J2EE Local interface of TagDataServiceBean
- TagDataServiceLocal

Function list

Return Type	Declaration
byte[]	readData(cuhk.ale.reader.DataSpec data)
Int	writeData(cuhk.ale.reader.DataSpec data)

7.2.3 Message Driven Beans

A Message Driven Bean is similar to a session bean, except it responds to a JMS message rather than a RMI event.

7.2.3.1 NotifierBean

This bean performs HTTP, TCP and FILE notification. All ECReports to be delivered are sent to this bean internally via JMS as *cuhk.ale.NotifyRequest* requests. These requests are then directed to the *NotifierQueue* for delivery to corresponding subscribers.

Package

- cuhk.ale.ejb

Implemented Interfaces

- javax.ejb.MessageDrivenBean
- javax.jms.MessageListener

7.2.3.2 TagWriteActivatorBean

This bean supports asynchronous reads and writes through the messaging support provided by JMS. They are described in details in the Process View section of the document.

Package

- cuhk.ale.ejb

Implemented Interfaces

- javax.ejb.MessageDrivenBean
- javax.jms.MessageListener

7.2.4 EPCGlobal Classes

The EPCGlobal classes are well defined in the specification. According to the specification, the ECSpec and ECReport should be able to be represented in XML format, and the ALE API should be accessible via SOAP. Therefore, instead of modelling the EPCGlobal classes directly in Java, they are developed with the foundation of XML to Java object transformation & SOAP accessibility. The JAX-WS software library is used to facilitate the process.

7.2.4.1 JAX-WS

JAX-WS stands for Java API for XML Web Services. JAX-WS is a technology for building web services and clients that communicate using XML. According to the EPCGlobal specification, the remote procedure calls to the ALE middleware are represented by the XML-based protocol SOAP. These calls and responses are transmitted as SOAP messages (XML files) over HTTP.

The SOAP messages are complex, and they need to be compliant to the envelope structure, encoding rules, and conventions defined by SOAP specification. The remote procedure calls and responses need to follow the Web Service Description Language (WSDL) definitions from EPCGlobal ALE specification which specifies an XML format for describing the ALE service as a set of endpoints operating on messages.

With JAX-WS, the complexities of SOAP messages are hided. The developer does not generate nor parse SOAP messages. It is the JAX-WS runtime system that converts the API calls and responses to and from SOAP messages. The JAX-WS also provides tools to generate the Java classes involved from the WSDL definitions and XML schemas.

7.2.4.2 Classes Generation

The EPCGlobal classes are generated using JAX-WS with the following steps:

1. Extract the XML Schemas and WSDL definition from EPCGlobal ALE Specification, resulting in *ALE.xsd*, *EpcGlobal.xs*d and *aleservice.wsdl*.
2. Modify the *ALE.xsd* to remove some optional elements related XML Extensibility.
3. Customize the XML binding mainly for package names and to convert XML date time to Java date time.
4. Input the *aleservice.wsdl* to the '*wsimport*' in JAX-WS to generate JAX-WS portable artefacts, such as Service Endpoint Interface (SEI), Service, Exception class mapped from *wsdl:fault* (if any), Async Reponse Bean derived from response wsdl:message (if any) & JAXB generated value types (mapped Java classes from schema types). Since the artefacts are generated from the wsdl, therefore, the SOAP implementation will be EPCGlobal compliant.

As a result, two packages are created – *epcglobal.ale* for ALE defined objects such ECSpec and ECReports, and *epcglobal.ale.soap* for SOAP call and response. The logics in the middleware are programmed to use the XML-bind Java objects.

8 Process View

8.1 Application to Middleware Interfaces

Upper level applications can access the middleware through the SOAP interfaces. There are two main sets of interfaces: the ALE standard service interfaces and the CUHK tag data services interfaces. The SOAP interfaces use HTTP as the transport protocol.

To facilitate application integration, the interfaces come with WSDL definitions, which are XML-based descriptions for web services defined according to W3C WSDL standard (*http://www.w3.org/TR/wsdl/*). Application clients can read the WSDL to determine what functions are available on the server. The clients can then use SOAP to actually call one of the functions listed in the WSDL.

The WSDL contains 4 major sections:

1. wsdl:messages - represents the data being transmitted; all request and response object are defined here.
2. wsdl:portType - represents all the operations, and each operation refers to an input and output message.
3. wsdl:binding - specifies the protocols and data formats for the operations and messages.
4. wsdl:service - defines the communication endpoints.

8.1.1 ALE Service Interfaces

These are the standard interfaces of the middleware as defined in ALE specification section 8.1. The server side of the interfaces is implemented by the *ALEService* EJB component.

Once the server is started, the WSDL can be accessed via:
http://<server_name>:8080/ale-ws/aleservice?wsdl

The following operations are supported:

- define(specName:string, spec:ECSpec) : void
- undefined(specName:string) : void
- getECSpec(specName:string) : ECSpec
- getECSpecNames() : List
- subscribe(specName:string, notificationURI:string) : void
- unsubscribe(specName:string, notificationURI:string) : void
- poll(specName:string) : ECReports
- getSubscribers(specName:String) : List
- getStandardVersion() : string
- immediate(spec:ECSpec) : ECReports
- getVendorVersion() : string

The details of the operation can be found in ALE specification Section 8.1.

8.1.2 Tag Data Service Interfaces

These are CUHK's extensions to the middleware standard interfaces to support tag data reads and writes. The server side of the interfaces is implemented by the TagDataService EJB component.

Once the server is started, the WSDL can be accessed via:

http://*<server_name>*:8080/ale-ws/aletagdata?wsdl

The following operations are supported:

- readData(data:DataSpec) : byte[]
- writeData(data:DataSpec) : int

DataSpec is the key to support tag data services. DataSpec is a data structure which contains tag id, format id, data address, data length, and data bytes. The format id is an integer dictating the addressing scheme to be used. The value of 0 represents byte addressing, which is the simplest addressing scheme that every byte has its own address. The value of 1 represents page addressing, which the data memory is addressable by using page number, and reads and writes always start from a page.

byte[] readData(DataSpec data)

Based on the data (DataSpec), the function gets a list of reader ids which has read the tag id within a predefined time period (default is 10s). With the reader id list, the server activates all the readers by sending the read tag command to the readers, for that particular read id, and data address, data length, and format id. The first reader replied to the query is considered as a tag read hit, and the result will be return to the function callee.

int writeData(DataSpec data)

The write-into-tag operation can be time consuming and unreliable. Therefore, we separate the request and response in two asynchronous messages.

The application user starts by issuing writeData function call via SOAP. *TagDataServiceBean* then takes this request, and it will locate the related readers via *TagDataServiceDAOImpl*. *TagDataServiceBean* will locate the list of readers, and generate a GUID (global unique identifier) for this particular request.

With the GUID, *TagDataServiceBean* registers for a callback, *TagDataServiceCallback*. Later *ReaderManagerBean* will use this callback to notify the write-into-tag result.

At the same time, *TagDataServiceBean* send the *TagWriteActivateRequest* to an internal queuing module, which will asynchronously send the write-into-tag request together with GUID to all reader adaptors.

Once a tag write is successful, the reader adaptor will send a response with the GUID to *ReaderManagerBean*, which in turns will use the predefined callback corresponding to the GUID. *TagDataServiceBean* will be notified, and return the result to the function callee.

Returns

0 : success
-1 : other failures
-2 : no readers access to the tag

8.2 Middleware to Application Interfaces

The communication from middleware to the application follows the standard notification mechanisms, as described in the Section 9 of ALE specification. The middleware supports the HTTP, TCP and FILE notifications.

8.3 Adaptor to Middleware Interfaces

These are the interfaces between the *ReaderAdaptor* and the *ReaderManager* component in the middleware. The *ReaderManager* is implemented as a stateless session bean using EJB. The *ReaderAdaptor* should be implemented as an EJB client using the exported library adaptor-client.jar. The protocol in between is RMI/JRMP.

The following methods are defined.

public void submitALEPhysicalReaderInfo(String readerID, PhysicalReaderInfo physicalReaderInfo) throws Exception

The adaptor uses this method to submit to the ALE the connected hardware reader's information. Since the physical reader must be defined in the ALE before the adaptor can submit tags to ALE, the adaptor should call this method to do reader registration on the first time connecting to the server. This method can be also be used to update the reader information in the ALE, by doing another submission to an already-defined reader. The *PhysicalReaderInfo* is a class with the following attributes: manufacturer, model and IP address.

public void submitALETags(String readerID, List<String> tags) throws Exception

This adaptor uses this method to submit the tags read in every read cycle to ALE. The *ReaderAdaptor* should call this method regularly (i.e. at a particular frequency) to submit tags to the server. In a read cycle, the tags submitted should be distinct without duplicates and in the "Tag URI" format defined EPCGlobal Tag Data Specification (e.g. "urn:epc:tag:gid-96:21.300.1"). An exception would be thrown if the readerID is not yet defined in the ALE.

public void submitReadData(String readerID, int status, byte[] data, String guid)

This adaptor uses this method to submit the tag data that the reader has been requested to read. In a tag-read cycle, commands are sent to reader from middleware to read tag data stored in a particular tag in a particular reader. Since the command is an asynchronous call, the *submitReadData* method is used as a callback for the reader adaptor to communicate with the middleware for the data read.

public void submitWriteData(String readerID, int status, String guid)

This adaptor uses this method to submit the tag-data-write status that the reader has been requested to write. In a tag-write cycle, commands are sent to reader from middleware to write tag data to in a particular tag in a particular reader. Since the command is an asynchronous call, the *submitWriteData* method is used as a callback for the reader adaptor to communicate with the middleware for the status of the tag-write.

8.4 Middleware to Adaptor Interfaces

public void read(String readerID, String tagID, DataSpec spec, String guid) throws RemoteException

The middleware uses this method to request the adaptor to read user data from the physical tag.

public void write(String readerID, String tagID, DataSpec spec, String guid) throws RemoteException

The middleware uses this method to request the adaptor to write user data to the physical tag.

8.5 Interfaces between Middleware and Management Console

There is no specific interface between the middleware and the management console. Indeed, the management console manipulates the middleware database directly, so as to perform operations such as adding readers, viewing connection diagrams, etc.

The management console accesses the database using the user '*rfidmc*'. The default is to grant all the privileges to this user in localhost access with password '*rfidmc*':

GRANT ALL ON aledb.* TO 'rfidmc' identified by 'rfidmc';
GRANT ALL ON aledb.* TO 'rfidmc'@127.0.0.1 identified by 'rfidmc';

9 Implementation View

9.1 Platform Considerations

Among many available platform products in the market, MySQL and JBoss are selected for the implementation. This section describes the reasons of using these platforms briefly.

9.1.1 MySQL

MySQL is the world's most popular open-source database. Its advantages include fast performance, high reliability, high usability and high quality technical support.

MySQL has a good support for different OS, such as Windows, Linux, HP-UX, AIX, etc. Moreover, it also support common database APIs such as ODBC and JDBC which ease the development tasks.

According to a real case study on Friendster, MySQL works smoothly with data of total size over 7TB, 100 millions of row, without service degrading. Considering the fact that middleware should be of much less scale than the above, the use of MySQL is a feasible solution.

9.1.2 Jboss

JBoss (AS) is the world's most widely used Java Application server. It is an open-source (GPL) J2EE certified platform, which is widely supported by the community developers.

JBoss provides a wide range of J2EE features, which may be missing in other well known J2EE platform, such as cluster, caching and persistence. Also, it is one of the first industrial-grade Java application servers that support J2EE 5.0 and EJB 3.0.

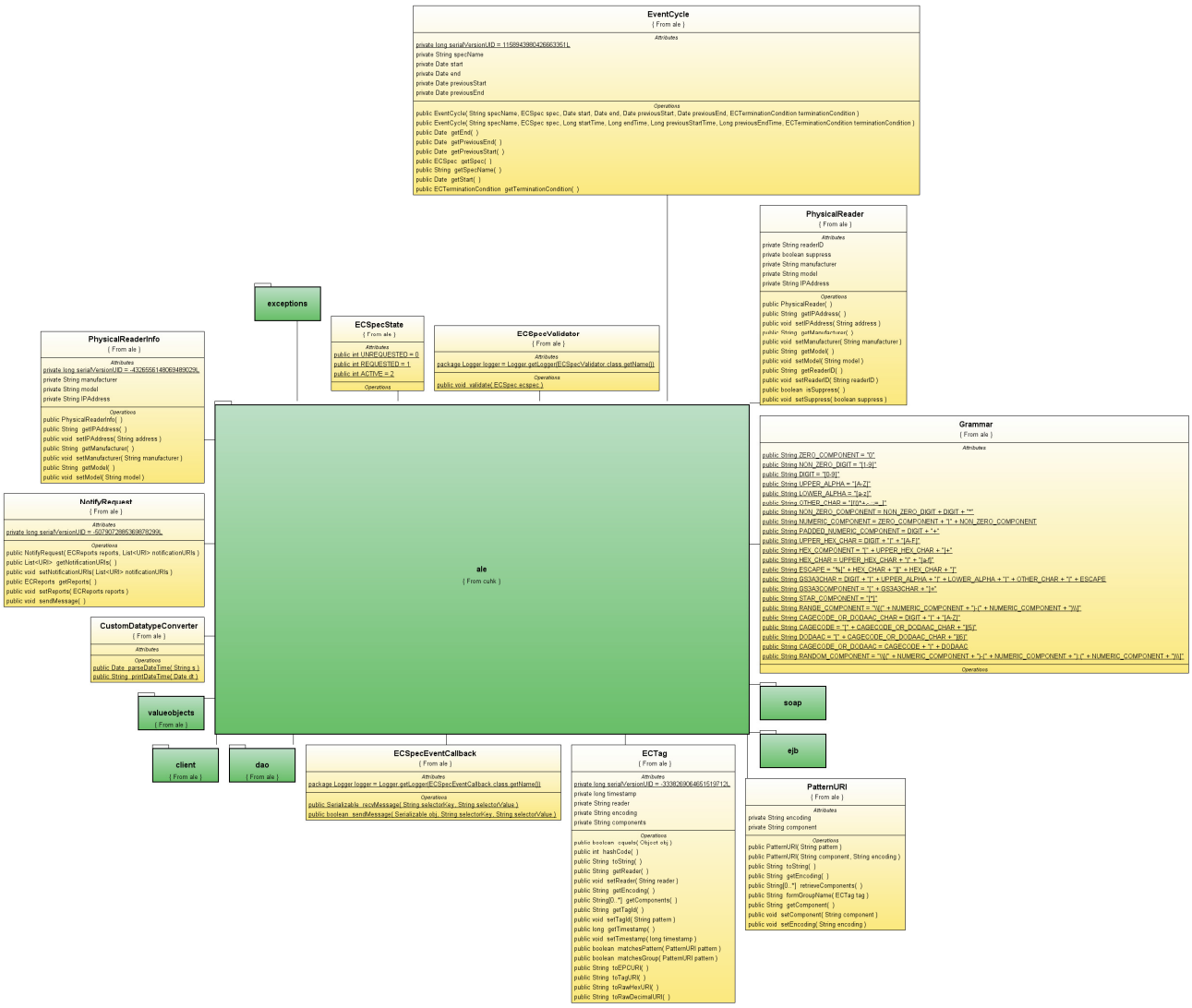
Considering performance, one of the useful benchmark could be found in *http://jbento.oscj.net/httpsession2.html*. In the benchmark the capability of thread handling, which is an essential part of application server, is measured. JBoss runs with less than 25% CPU utilization, more than 250 tps throughput, less than 10ms response time, with 250 clients collected.

9.2 Class Diagram

The overall categorization of classes is discussed in the Package Diagram section in Development View. Here the methods and properties of the classes within the packages are shown in details.

Classes that are added by CUHK to support the ALE implementation are described in the following sections. For details of ALE standard classes, please refer to ALE specification.

9.2.1 cuhk.ale



CustomDatatypeConverter – utility class to support JAX-WS conversion so the date time specified in the schema can be converted properly into proper Java Date class.

ECSpecEventCallback – call back methods to support ALE service bean's immediate and poll method

ECSpecState – represents the possible states of the ECSpec: Unrequested, Requested, Active.

ECSpecValidator – to validate ECSpec

ECTag – tags received and processed in the middleware

EventCycle – the event cycle of the ECSpec

Grammar – utility class that contains all the regular expressions necessary to validate the tag formats

NotifyRequest – internal class to support *ECReports* notification

PatternURI – encapsulate the EPCGlobal's Pattern URI attributes and methods

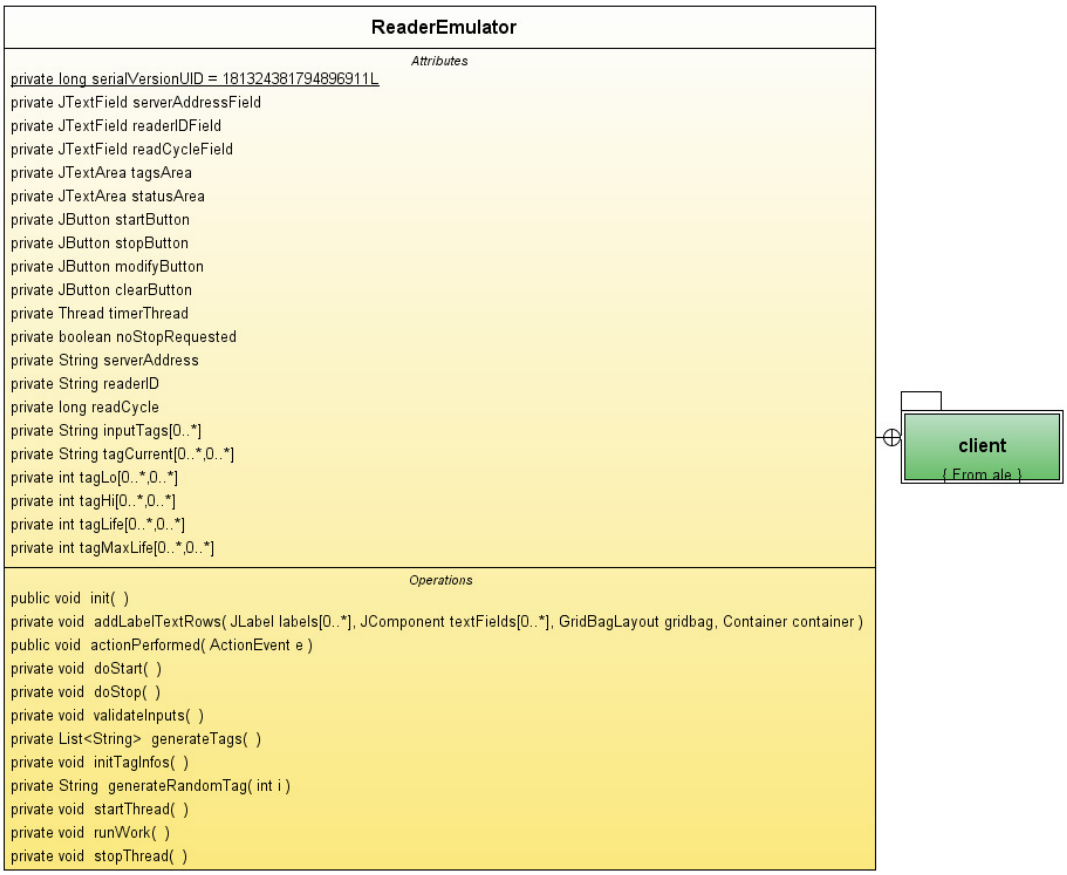
PhysicalReader – represents a physical tag reader

PhysicalReaderInfo - for use in *submitALEReaderInfo* in ReaderManager

TagDataServiceCallback – call back class to support CUHK specific tag data read/write

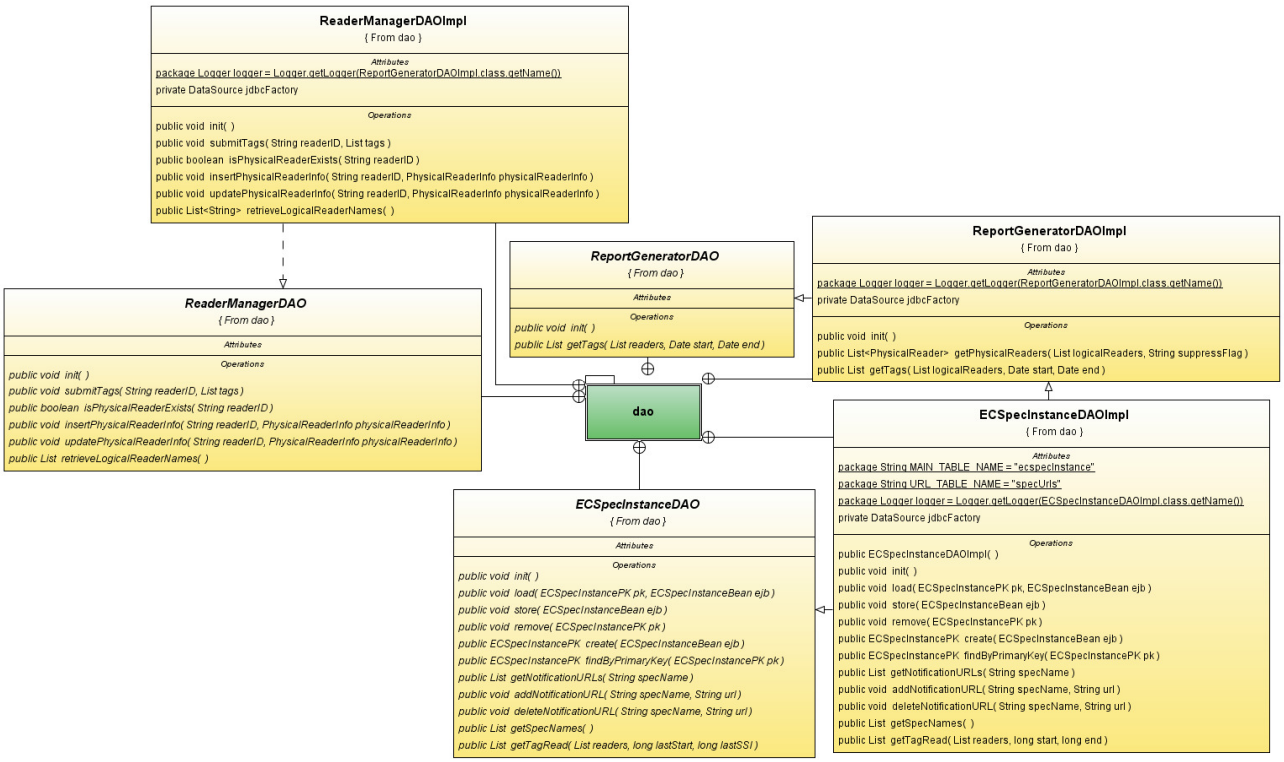
TagWriteActivateRequest – internal class to support asynchronous tag write request

9.2.1.1 cuhk.ale.client



ReaderEmulator – reader emulator client

9.2.1.2 cuhk.ale.dao



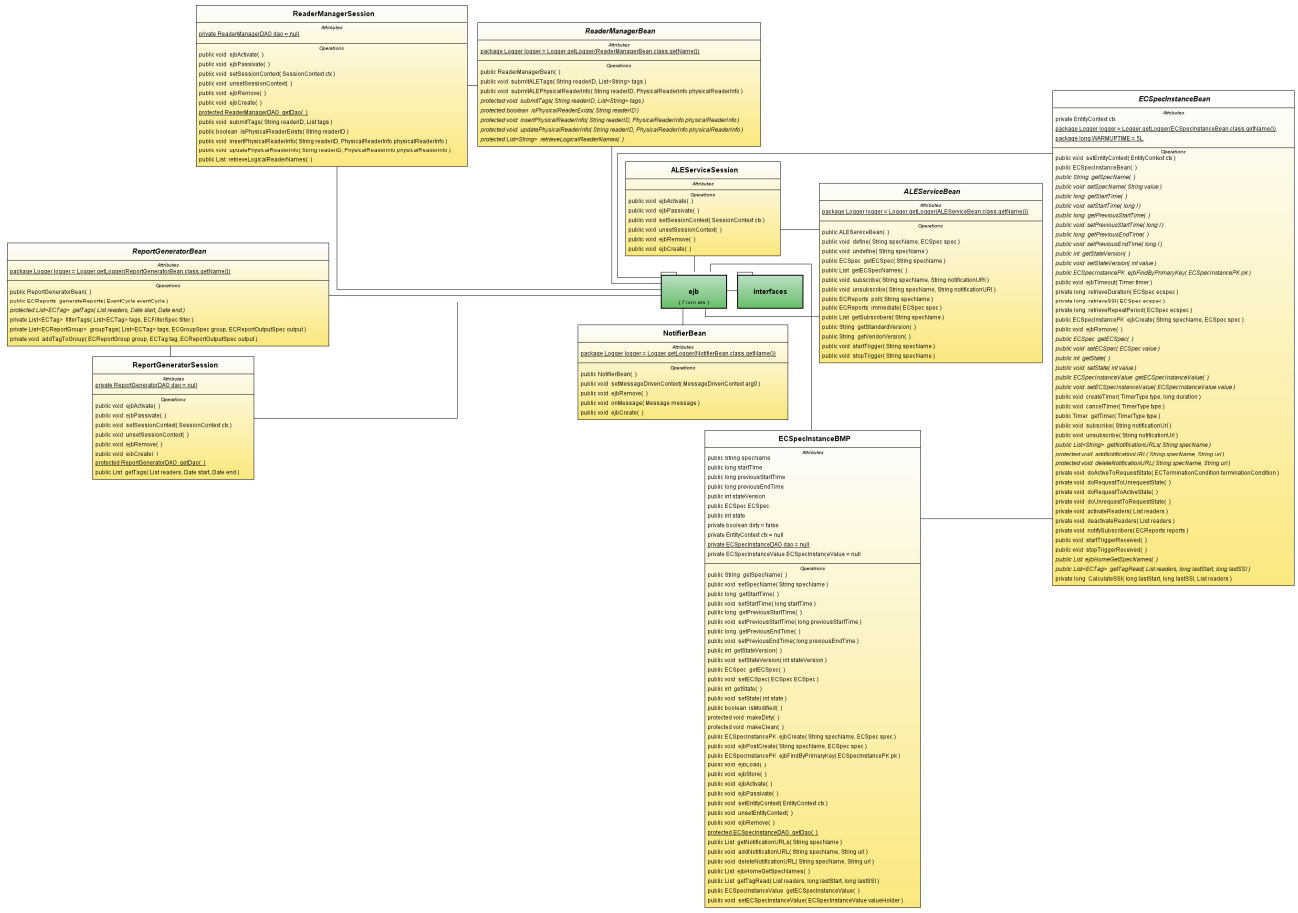
ECSpecInstanceDAOImpl – DAO implementation for manipulating *ECSpecInstance* and SpecURLs

ReaderManagerDAOImpl – DAO implementation of the *ReaderManager*, which is responsible manipulating tag events, reader registration, etc

ReportGeneratorDAOImpl – DAO implementation to support for the ReportGenerator which is responsible for resolving logical to physical reader mapping, gathering tag events between event cycle, etc

TagDataServiceDAOImpl – DAO implementation to support CUHK specific tag data read/write

9.2.1.3 cuhk.ale.ejb



ALEServiceBean – as described in the previous sections

ECSpecInstanceBean – as described in the previous sections

NotifierBean – as described in the previous sections

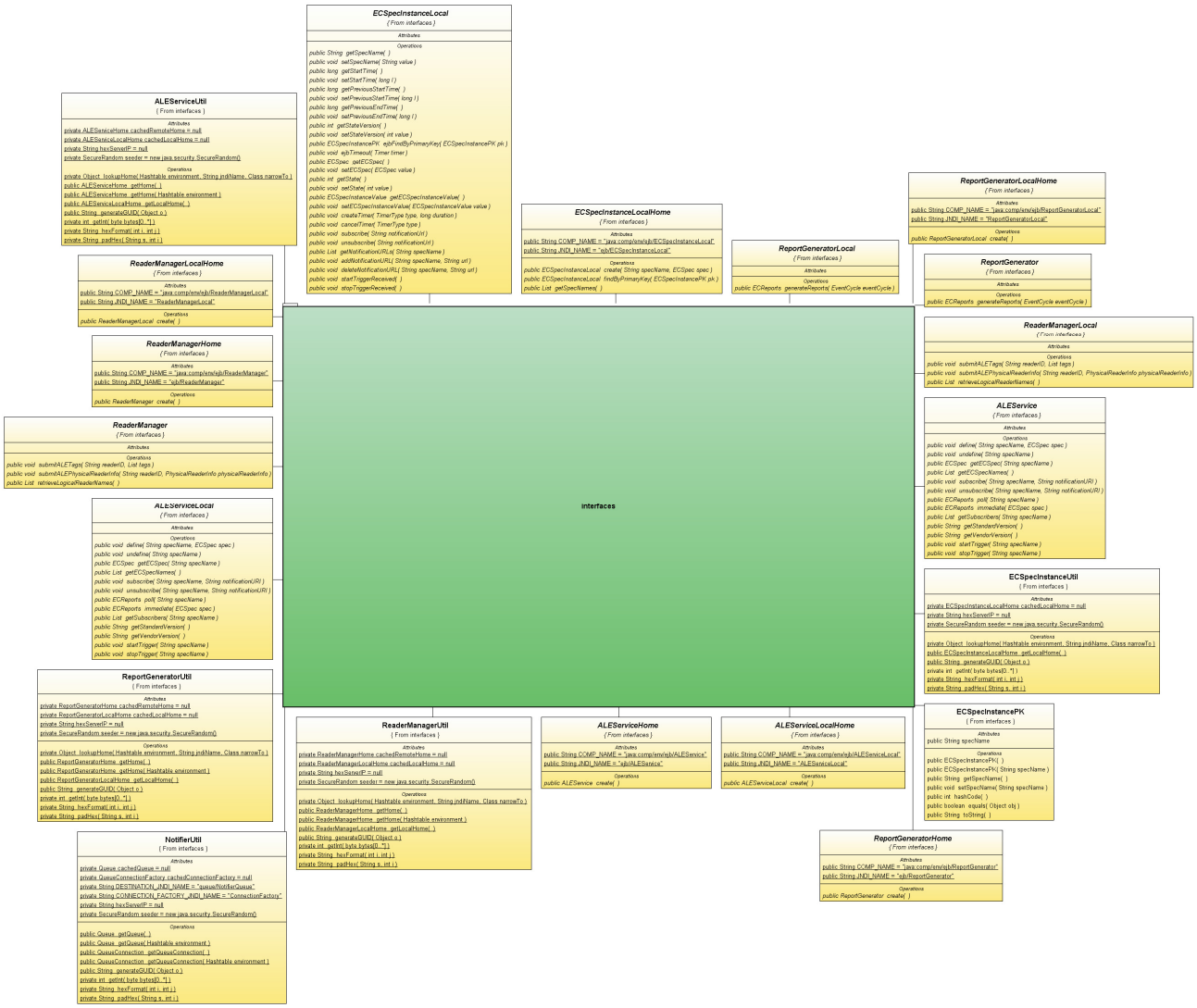
ReaderManagerBean – as described in the previous sections

ReportGeneratorBean – as described in the previous sections

TagDataServiceBean – as described in the previous sections

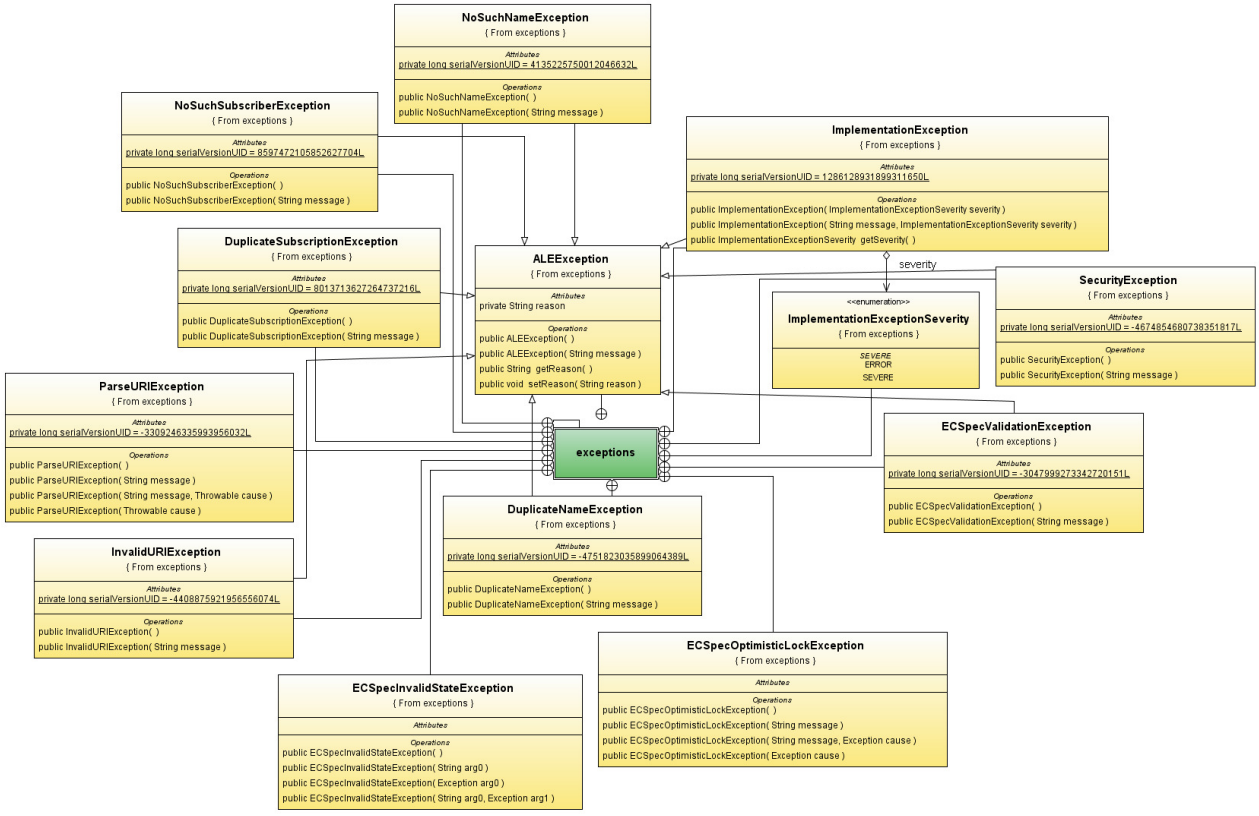
TagWriteActivatorBean – as described in the previous sections

9.2.1.4 cuhk.ale.ejb.interfaces



This package contains the home interfaces, remote interfaces and utility classes to support the EJB life cycles.

9.2.1.5 cuhk.ale.exceptions



ALEException – exception occurred in the middleware; the ancestor of other exceptions occurred in the middleware

DuplicateNameException – exception representing the specified ECSpec name already exists

DuplicateSubscriptionException – exception representing the specified ECSpec name and subscriber URI is identical to a previous subscription that was created and not yet unsubscribed

ECSpecInvalidStateException – exception representing the ECSpec has entered into an invalid state

ECSpecOptimisticLockException – exception representing unexpected locking problem of ECSpec in the middleware

ECSpecValidationException – exception representing the specified ECSpec is invalid

ImplementationException – a generic exception thrown by the implementation for reasons that are implementation-specific

ImplementationExceptionSeverity – an enumeration whose values are either ERROR or SEVERE, used in the ImplementationException

InvalidURIException – exception representing the URI specified for a subscriber cannot be parsed

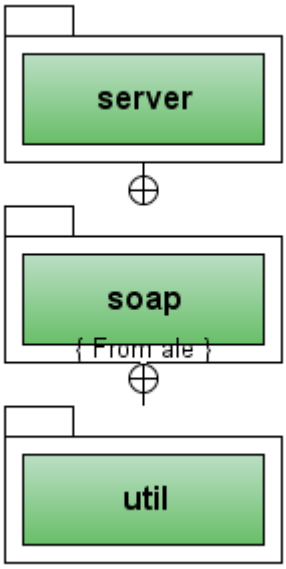
NoSuchNameException – exception representing the specified ECSpec name does not exist

NoSuchSubscriberException – exception representing the specified subscriber does not exist

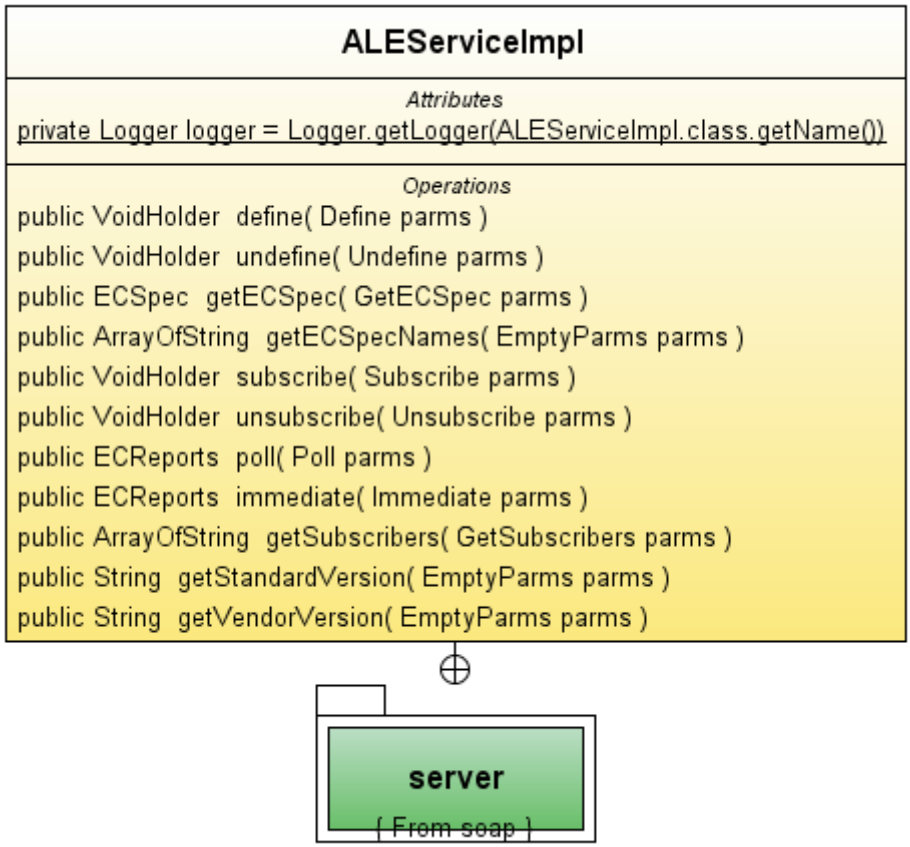
ParseURIException – exception representing errors in parsing the tag URI

SecurityException – exception representing the operation was not permitted due to an access control violation or other security concern

9.2.1.6 cuhk.ale.soap



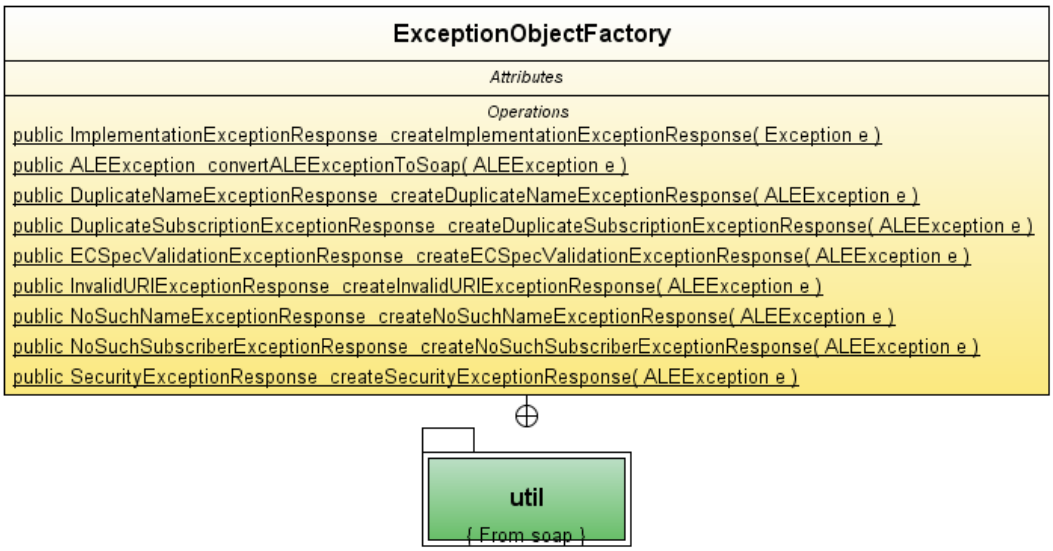
9.2.1.7 cuhk.ale.soap.server



ALEServiceImpl – end point implementation for the ALE web service

TagDataServiceImpl – end point implementation for the CUHK specific tag data service

9.2.1.8 cuhk.ale.soap.util

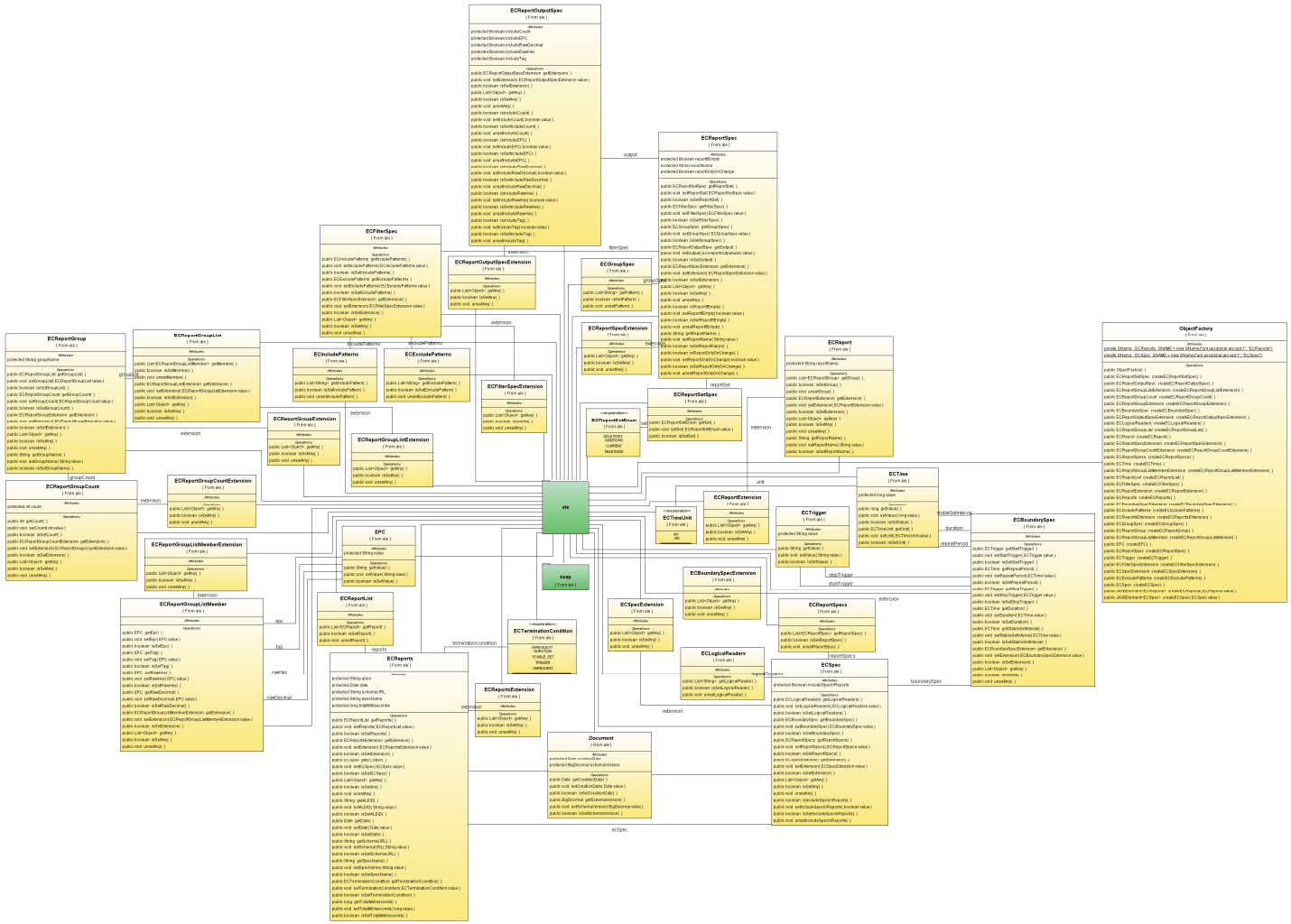


ExceptionObjectFactory – a factory for generating exceptions used in the web services

9.2.1.9 cuhk_ale_valueobjects

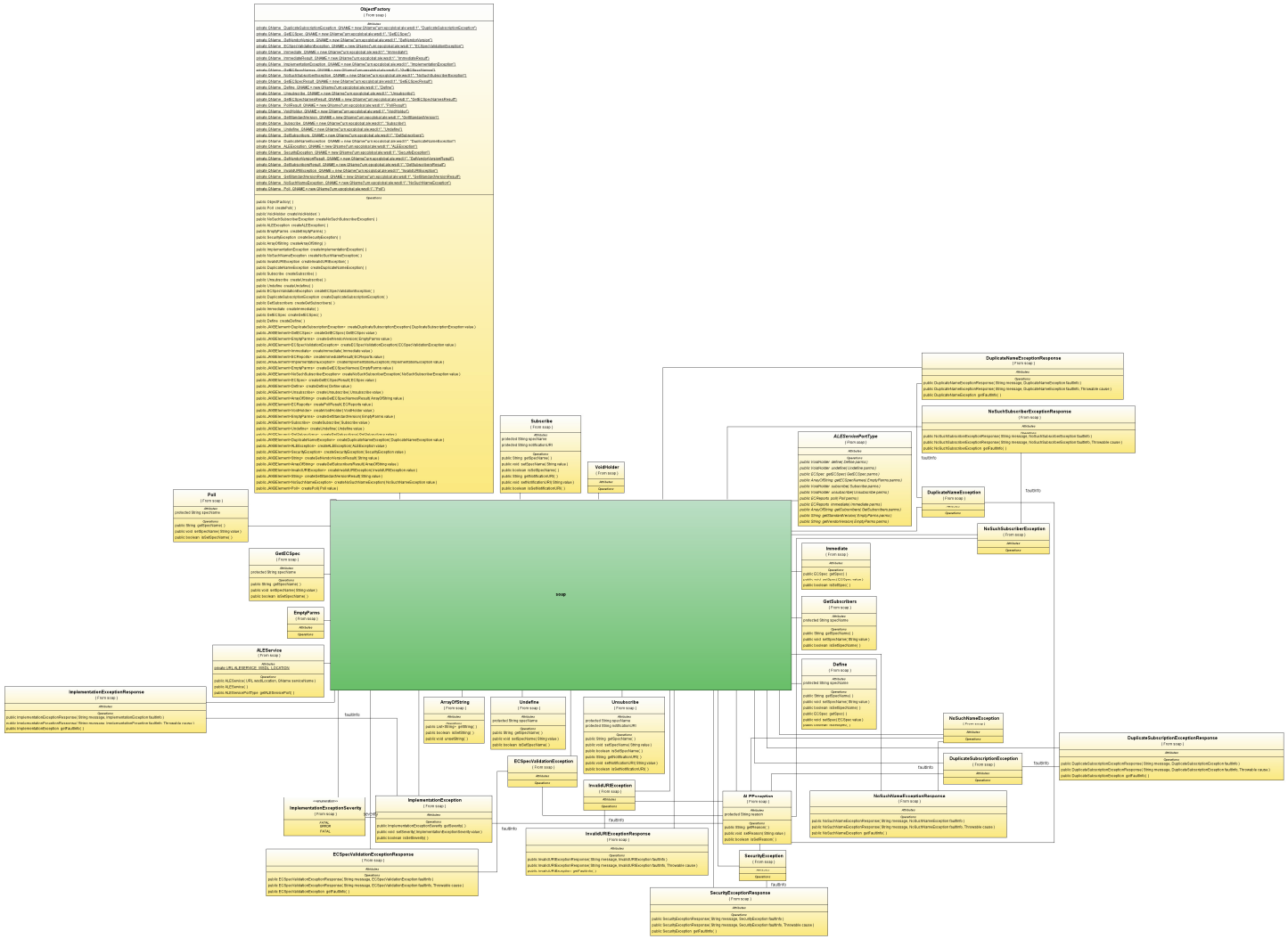
ECSpecInstanceValue – class used to represent the state of an *ECSpecInstance* object. This value object is not connected to the database in any way. It is just a normal object used as a container for data from an EJB.

9.2.2 epcglobal.ale



For those standard classes defined in the ALE specification, please refer to the specification for details.

9.2.2.1 epcglobal.ale.soap



Classes automatically generated by the JAX-WS from the ALE web services schema.

9.3 Collaboration Diagram

Here the collaboration diagrams of the main ALE API operations, together with the CUHK extended tag data reading and writing are shown. The major entities involved in each operation are depicted.

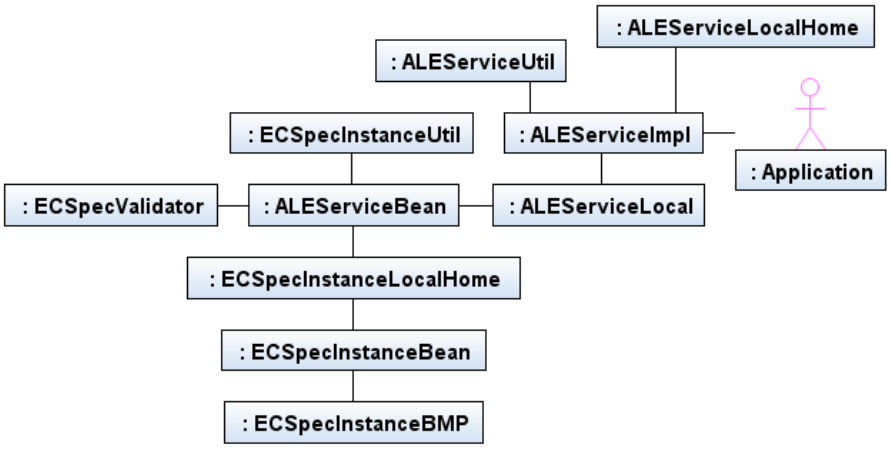
We will have a more detailed explanation for the 'define' operation. The entities and procedures involved are also similar for other operations, as they follow the EJB standard invocation procedures.

9.3.1 Define

The application starts invoking business logic remotely through SOAP by using *ALEServiceImpl*. The *ALEServiceImpl* is a class implementing the web services server processes and provides the entry points for all the standard ALE operations such as 'define', 'subscribe', etc.

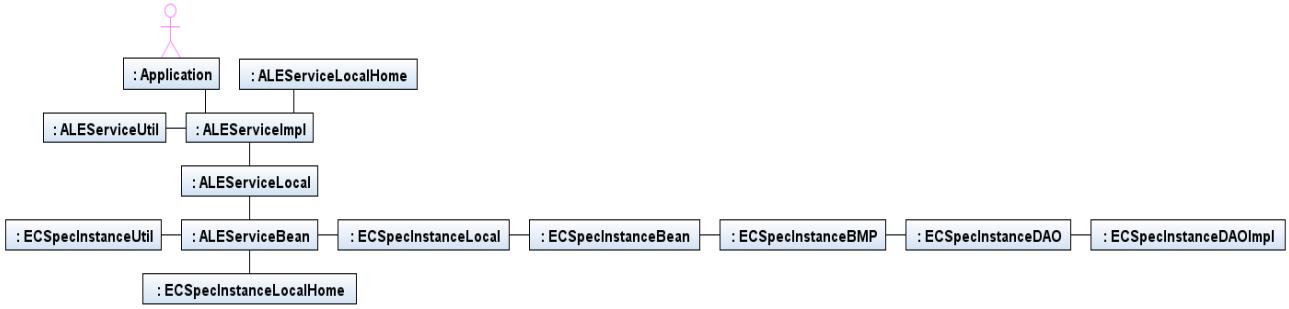
The *ALEServiceImpl* class then gets the home interface (*ALEServiceLocalHome*) of the *ALEServiceBean* through the *ALEServiceUtil* class. Through the home interface, the *ALEServiceImpl* then creates an instance of the *ALEServiceBean* session bean and then access the bean through the local interface *ALEServiceLocal*. With the local interface, the *ALEServiceImpl* can access to the business methods exposed by the bean.

The 'define' operation creates an ECSpec in the middleware. First the ECSpec is validated through *ECSpecValidator* and then the *ALEServiceBean* will create another instance of entity bean (*ECSpecInstanceBean*) to store the ECSpec. The procedures underway are similar as before and involve the use of utility and home interface classes.



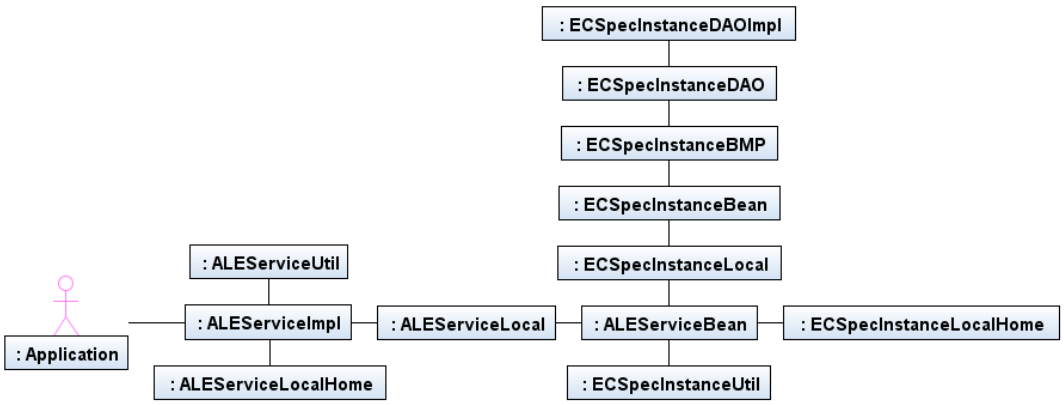
9.3.2 Undefine

Internal flows are similar to the ‘define’ operation.



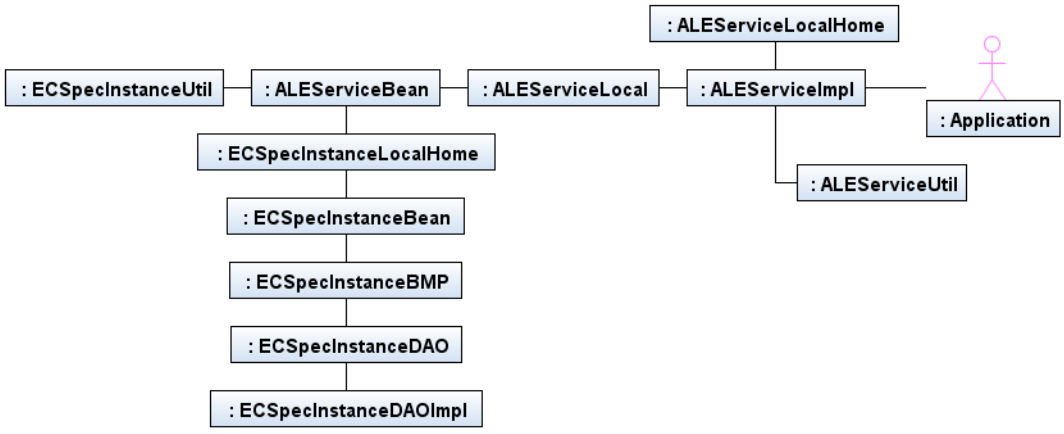
9.3.3 GetECSpec

Internal flows are similar to the ‘define’ operation.



9.3.4 GetECSpecNames

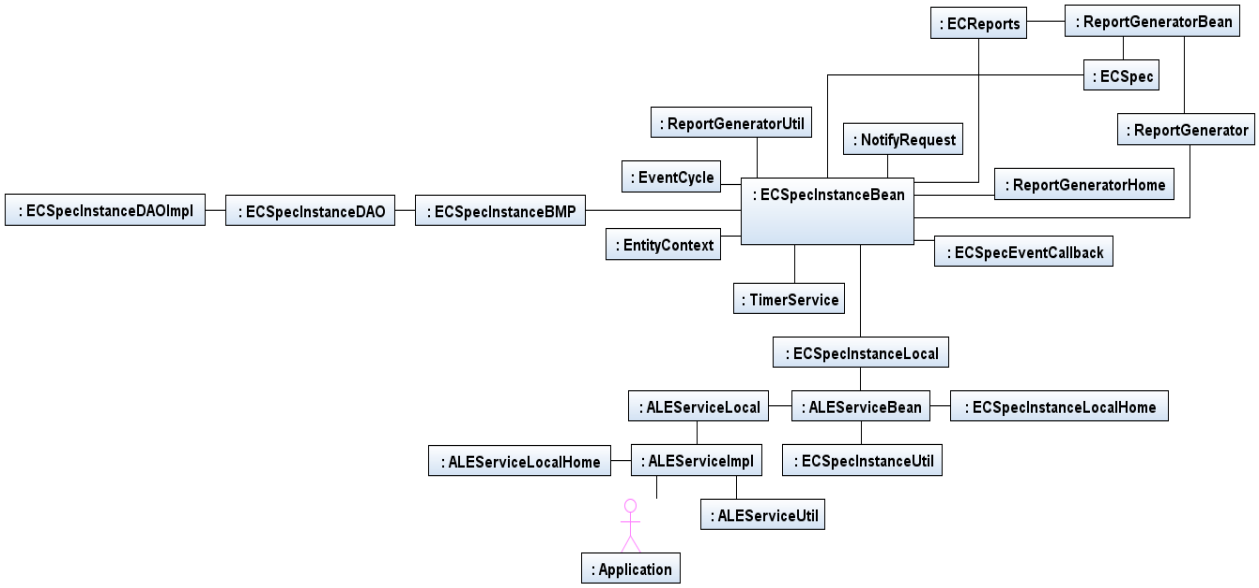
Internal flows are similar to the ‘define’ operation.



9.3.5 Subscribe

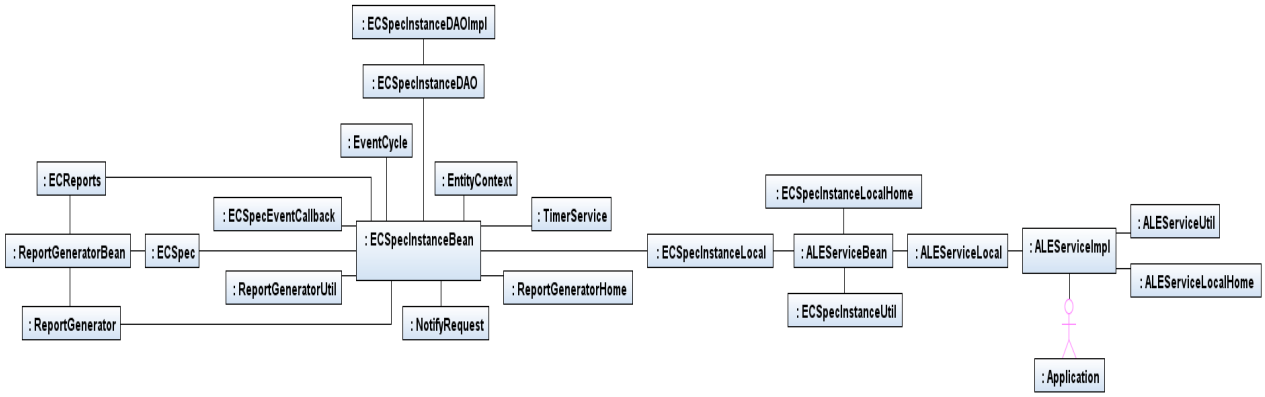
In the 'subscribe' operation, *ALEServiceBean* will register for timer events through *TimerSerivce* so as to provide time-based event triggering. For each event cycle, the *ReportGenerator* is triggered to provide notifications to the subscribers.

Internal flows are similar to the 'define' operation.



9.3.6 Unsubscribe

Internal flows are similar to the 'define' operation.



9.3.7 Poll

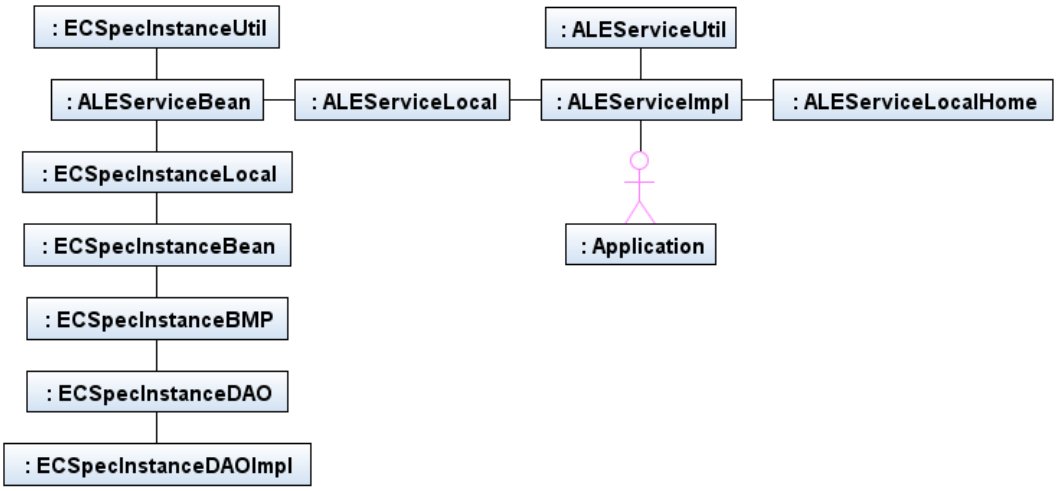
The *poll* call is implemented as subscribing and then unsubscribing immediately after one event cycle is generated except that results are returned from *poll* instead of being sent to a *notifcationURI*.

9.3.8 Immeidate

The *immediate* call is implemented as defining an *ECSpec*, performing a single *poll* operation and then undefining it.

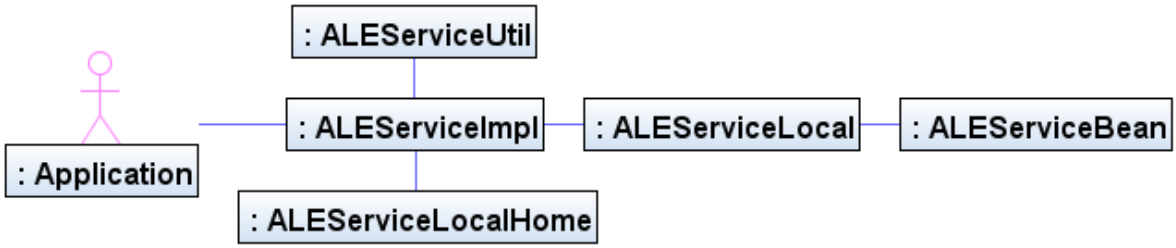
9.3.9 getSubscribers

Internal flows are similar to the 'define' operation.



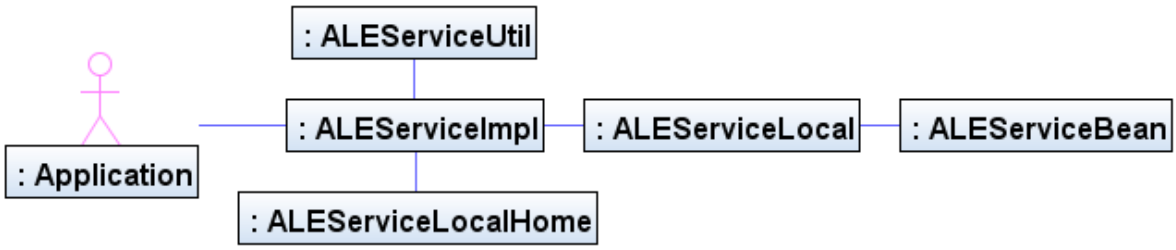
9.3.10 getStandardVersion

Internal flows are similar to the 'define' operation.



getVendorVersion

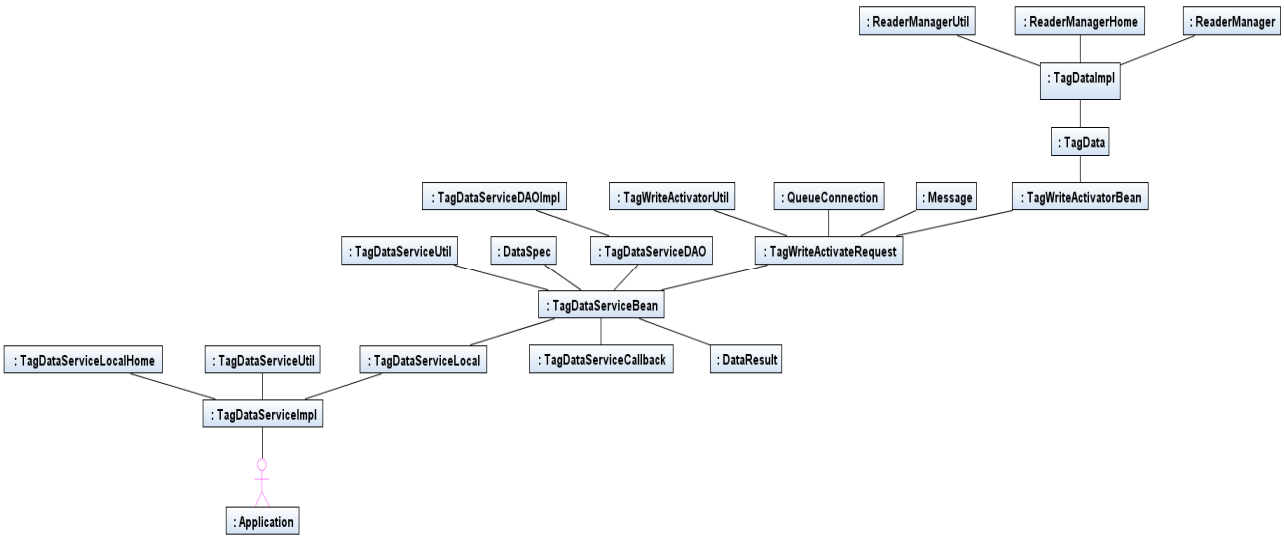
Internal flows are similar to the 'define' operation.



9.3.11 Tag Data Read/Write

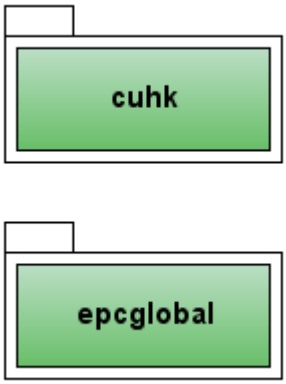
Here, the application uses *TagDataServiceImpl* instead of *ALEServiceImpl*.

The *TagDataServiceImpl* is a class implementing the web services server processes and provides the entry points for all the CUHK extended operations such as tag data read and write.



9.4 Package Diagram

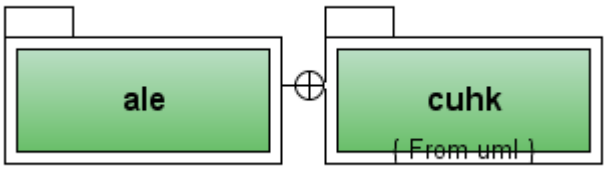
The middleware is developed into 2 Java packages: *cuhk* and *epcglobal*.

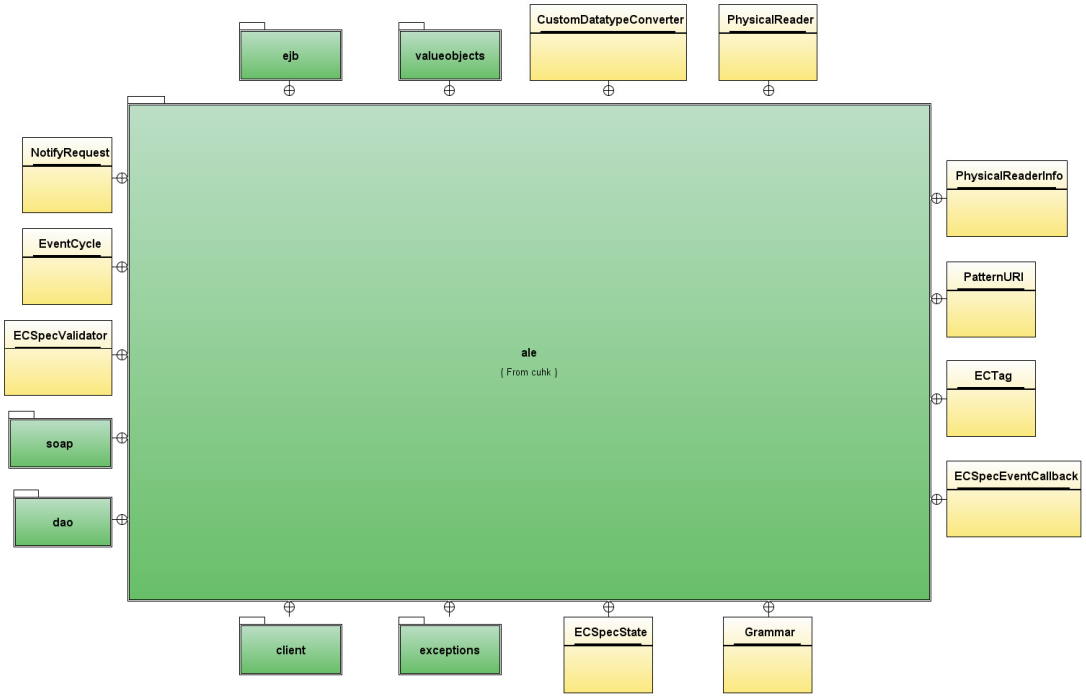


9.4.1 cuhk

9.4.1.1 cuhk.ale

The "*cuhk*" package only contains the "*ale*" package and it consists of the cuhk implementation of the middleware, some supporting classes, and utilities.





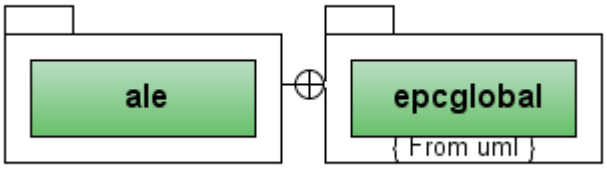
These are packages inside *cuhk.ale*:

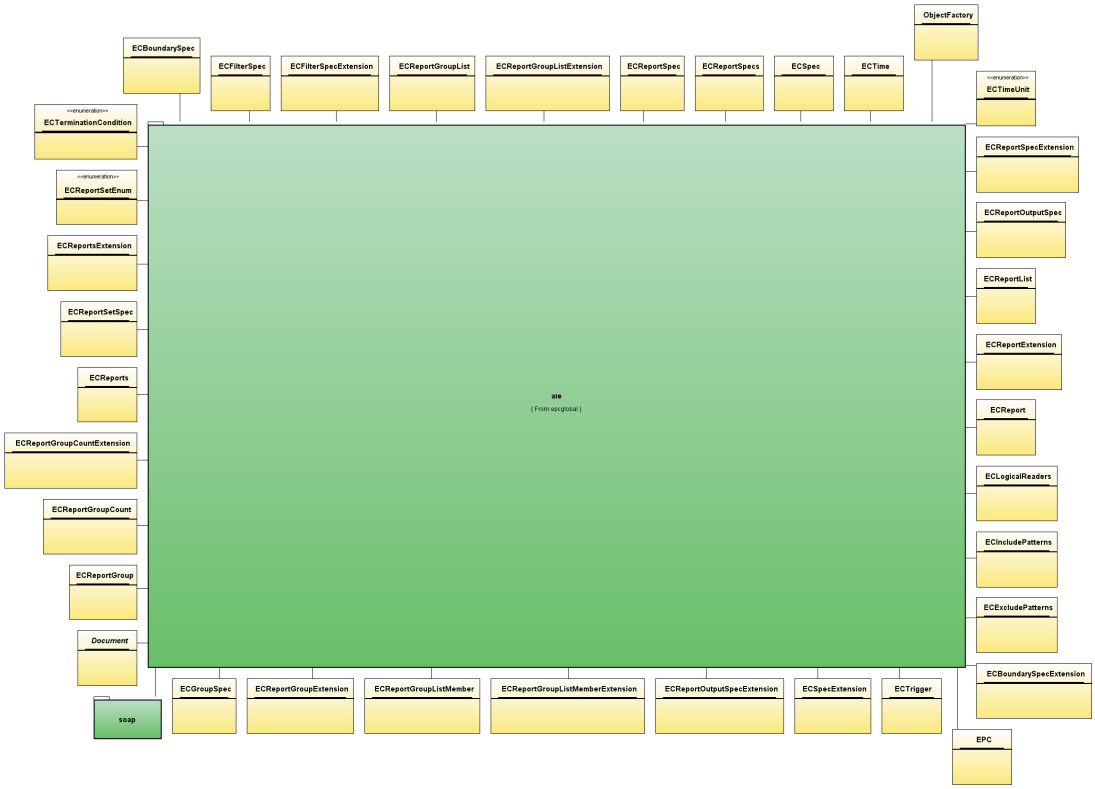
- *cuhk.ale.client*: clients of the middleware, which are considered to be an integral part of the middleware solution, for example, Reader Emulator, are placed here.
- *cuhk.ale.dao*: contains Data Access Objects, which are responsible for database storage and retrieval of data.
- *cuhk.ale.ejb*: the core of the middleware is implemented using J2EE technology, and is realized as different kinds of beans: session, entity, and message. They will be discussed in details in the Logical View section of the document.
- *cuhk.ale.exceptions*: exceptions in the middleware.
- *cuhk.ale.soap*: contains the implementation classes for the soap connectivity of the middleware.
- *cuhk.ale.valueobjects*: contains object used as a container for data in EJB, and the data stored is not connected to database.

9.4.2 epcglobal

9.4.2.1 epcglobal.ale

The "*epcglobal*" package also only contains the "*ale*" package and it consists of the classes defined by the EPCGlobal, which are the key data components of the middleware.





These are packages inside *epcglobal.ale*:

- *epcglobal.ale.soap*: contains the classes for the data exchange for the soap connectivity of the middleware

10 Deployment View

10.1 Environment

10.1.1 Setup

Our standard development environments are:

- Eclipse 3.1 + JBossIDE 1.5 (the IDE for development)
- Apache Ant 1.6.5 (a software tool for automating software build processes)
- JBoss 4.0.4 GA (an open source J2EE application server)
- Java Development Kit (JDK) 1.5 (for the Java development)
- Java API for XML Web Services (JAX-WS) 2.0 (library for Java Web Services)
- MySQL 5.0 (the database)
- XDoclet 1.2.3 with XJavaDoc 1.5 (for attribute-oriented code generation)
- JDBC Library 3.1 for MySQL (for database connectivity)
- Log4j 1.2.14 (a Java based logging utility)

And make sure the following environment variables are set:

- ANT_HOME
- JAVA_HOME
- JAXWS_HOME
- JBOSS_HOME
- JUNIT_HOME
- XDOCLET_HOME
- JUNIT_HOME
- LOG4J_HOME

10.1.2 Build Procedures

Check out all the codes and binaries, the required procedures are listed as below:

- Clean all code: run “ant clean“
- Configure the environment: run “ant configure”
- Generate all source codes: run “ant gen-src”
- Generate the Java Documentation: run “ant javadoc”
- EJB jar building: run “ant package-ejb“
- WAR jar building: run “ant package-war”
- EAR jar building: run “ant package-ear”
- SOAP library jar building: run “ant package-soap“
- ECSpec Editor building: run “ant package-editor“
- Reader Emulator building: run “ant package-emulator”

10.1.3 Deployment Procedures

The web services artefacts are packaged in the WAR file with the WSDL definitions, schema documents and the endpoint implementation. The WAR file is served by the Tomcat inside JBoss. It acts as the middleman between the EJB components and the SOAP client as the endpoint implementation will call ALE EJB services internally. Therefore, the WAR and the EJB files are packaged into an EAR file, as a J2EE application. And the resulting EAR file can be deployed to JBoss.

The compiled and built packages are stored under the bin folder in the project directory:

- bin/lib/ale.ear (middleware service)
- bin/lib/adaptor-client.jar (library files for adaptor client)
- bin/lib/ale-soap.jar (library files for ALE SOAP client)

For deployment of middleware, copy ale.ear to JBoss, under

- $JBOSS_HOME\server\ale\deploy

For adaptor to ALE interfacing, copy the adaptor-client.jar to the classpath of the adaptor, see also Adaptor to ALE interface in Process View.

For application to ALE interfacing, copy the ale-soap.jar to the classpath of the application, see also SOAP Application APIs in Process View

10.1.4 Logging

All middleware logs are stored in {jboss.home}\server\ale\log\.

The adaptor*.log stores all logs related to the adaptor operations. The server*.log stores all logs related to the internal processing of the middleware.

Daily log rotation is enabled with pattern *yyyy-MM-dd* which rollover at midnight of each day.

11 Data View

The middleware gets tag data using a reader list which is populated from the logical to physical reader mapping, giving a list of logical reader IDs. There is on/off settings for both logical and physical readers, indicated by the '*suppress'* flag. There may be case that the logical reader list specified by the user results in a list of all suppressed readers. In this case, the tag read in the event cycle will be empty.

11.1 Table LOGICALREADER

The LOGICALREADER table is storing logical readers defined in the middleware.

Column name	Type	Description
logicalreader_id *	varchar	The ID of the logical reader
suppress	boolean	// 0=reader on; 1=reader off

Primary Key: logicalreader_id

11.2 Table READER

The READER table is storing hardware reader information defined in the middleware.

Column name	Type	Description
reader_id *	varchar	The ID of the reader
suppress	boolean	// 0=reader on; 1=reader off
manufacturer	varchar	The name of the manufacturer
model	varchar	The model of the reader
ipaddress	varchar	The IP address of the reader

Primary Key: reader_id

11.3 Table READERMAPPING

The READERMAPPING table stores the mapping between logical readers and hardware readers.

Column name	Type	Description
logicalreader_id *	varchar	The ID of the logical reader
reader_id *	varchar	The ID of the reader

Primary Key: logicalreader_id, reader_id
Foreign Key: logicalreader_id (of LOGICALREADER), reader_id (of READER)

11.4 Table ECSPECINSTANCE

The ECSPECINSTANCE table is for manipulation of ECSpec via a J2EE entity bean.

Column name	Type	Description
specName *	varchar	The ECSpec name defined.
state	Int	The current state of ecspec
stateVersion	int	(internal) optimistic locking state.
previousStartTime	bigint	Start time for the previous event cycle
previousEndTime	bigint	End time for the previous event cycle
startTime	bigint	Start time for the current event cycle
specXML	blob	Normalized ECSpec XML in raw format

Primary Key: specName

Note:
1. This table should be only modified by JBoss application server.
2. Other field inside the table but not listed here are for internal debugging purpose.

11.5 Table SPECURLS

The SPECURLS table is for storing notification URLs. It is of a 1:M mapping with the ECSPECINSTANCE table.

Column name	Type	Description
specName *	varchar	The ECSpec name for the notification url.
NotificationUrl	varchar	The notification url.

Foreign Key: specName (of ECSPECINSTANCE)

Note:
1. This table should be only modified by JBoss application server.

11.6 Table READ_EVENT

The READ_EVENT table records the event received from the hardware reader in each read cycle.

Column name	Type	Description
event_id *	bigint	Auto-generated id for the event
reader_id	varchar	The id of the hardware reader
timestamp	datetime	Timestamp for the event received

Primary Key: event_id
Foreign Key: reader_id (of READER)

11.7 Table READ_TAG

The READ_TAG table stores the tags received from the hardware reader in a event.

Column name	Type	Description
event_id *	bigint	Auto-generated id for the event
tag_id	varchar	Tag ID in EPCGlobal Tag URI format

Primary Key: event_id, tag_id
Foreign Key: event_id (of READ_EVENT)

12 System Properties

The CUHK RFID middleware is based on J2EE technologies and leverages many of its important qualities.

12.1 Extensibility

The system is built around a modular architecture. All the core functionalities of the system are built as EJB components, e.g. *ReportGenerator* as a stateless session bean, *ECSpecInstance* as an entity bean. New features can be added to the system by writing new beans, without rewriting existing pieces; and existing pieces can be modified to fit changing needs quickly and efficiently.

12.2 Scalability

Clustering allows one to add more server hardware to handle more requests and is important for high traffic applications. Our system is configured and deployed on JBoss, which comes with clustering support and its support is transparent to applications. This means the system can be setup to run in a cluster by changing a few configurations, without changing the programming codes. By making such configurations, those server instances can detect each other and automatically form a cluster.

12.3 Portability

The whole system is developed in Java, and can be run on different types of machines without changes, such as recompilation or tweaks to the source codes. And the MySQL database that we are using is also available in major platforms, such as Windows, Linux, Solaris, FreeBSD, Mac OS, etc. Therefore, the whole RFID system setup is portable across platforms.

12.4 Reliability

The system can be configured to run on several parallel servers as cluster nodes. The load is distributed across different servers, and even if any of the servers fails, the application is still accessible via other cluster nodes. This makes the system fail-safe.

The system is also configured with a persistence setup, which means that even the JBoss application server is stopped manually; a restart of the server will resume all the working state of the system without any failures.