Tag Capturer - System Design Document

Prepared By	:	Anthony Lam

Report No	:	CUHK-TagCapturer-SDD
Version	:	1.0
Issue Date	:	30 Jul 2007

Table of Content

Revision History

Date	Ver./Rev.	Description	Author
30 Jul 2007	V1.0	First Release	Anthony Lam

1 Executive Summary

This document is to present the architectural and technical design of the application Tag Capturer. This provides sufficient technical details for application architects, system analysts and developers to understand the design and usage.

The objective of Tag Capturer is to demonstrate the capability of extending CUHK Middleware with a third-party EPCIS.

2 Introduction

Tag Capturer is an application demonstrating the functionalities of the ALE middleware and integration with Accada's EPCIS project. It demonstrates the successful integration of CUHK Middleware with an open source EPCIS module.

The goal of EPCIS is to enable disparate applications to leverage Electronic Product Code (EPC) data via EPC-related data sharing, both within and across enterprises. Ultimately, this sharing is aimed at enabling participants in the EPCglobal Network to gain a shared view of the disposition of EPC-bearing objects within a relevant business context.

Tag Capturer is an application that demonstrates the interoperability of CUHK middleware with an open sourced EPCIS product, namely Accada EPCIS. Accada EPCIS Project is the first open source EPCIS project with its implementation available public, and claims to be compliant to the EPC Information Services (EPCIS) Version 1.0 Specification as specified in EPCglobal's Ratified Standard (Version 1.0 of April 12, 2007). Tag Capturer implements the EPCIS role in the EPC network and develops the appropriate tools that facilitate communication with an EPCIS Repository instance.

Since an EPCIS application is domain specific and requires detailed workflow in order to fulfill requirements and align with business, Tag Capturer demonstrates only the feasibility of expanding CUHK middleware with specification compliant EPCIS products.

Tag Capturer shows the integration of CUHK middleware with Accada EPCIS Capture Interface and Accada EPCIS repository to capture an EPCIS Event. Apart from the interoperability, Tag Capturer also demonstrates the usage feasibility of the following scenarios:

1. Embed EPCIS Repository into the demonstration application to add an EPCIS interface to it
2. Interactively explore any EPCIS Repository using Accada's graphical EPCIS Query Application
3. Build an EPCIS Capture Application
4. This document is intended to provide a "getting started" point for Tag Capturer. After following this technical article, users should be able to install and setup the Tag Capturer.
5. This technical article does not explain the system design and the technical details of Tag Capturer. Readers are expected to have a basic level of understanding of the EPCGlobal ALE and EPCIS standards, which may be referenced from the site http://www.epcglobalinc.org/.

2.1 References

Document	Vers.	Date	Org.
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	Jul 2007	CUHK

3 Architecture Representation

The architecture 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 development view.

4 Requirement View

Tag Capturer allows manipulation of views in order to subscribe the channel for ALE Events and receive EPC reports, operations include:

1. Create a view
2. Edit a view
3. Select a view
4. Delete a view

User first requires defining views that bind to a particular logical reader each. After defining the view, user may select the view that he wants to subscribe for EPC reports. Tag Capturer will check if reports are generated by probing the middleware, and display the details on the screen.

Tag Capturer allows alerting of new incoming EPC reports.

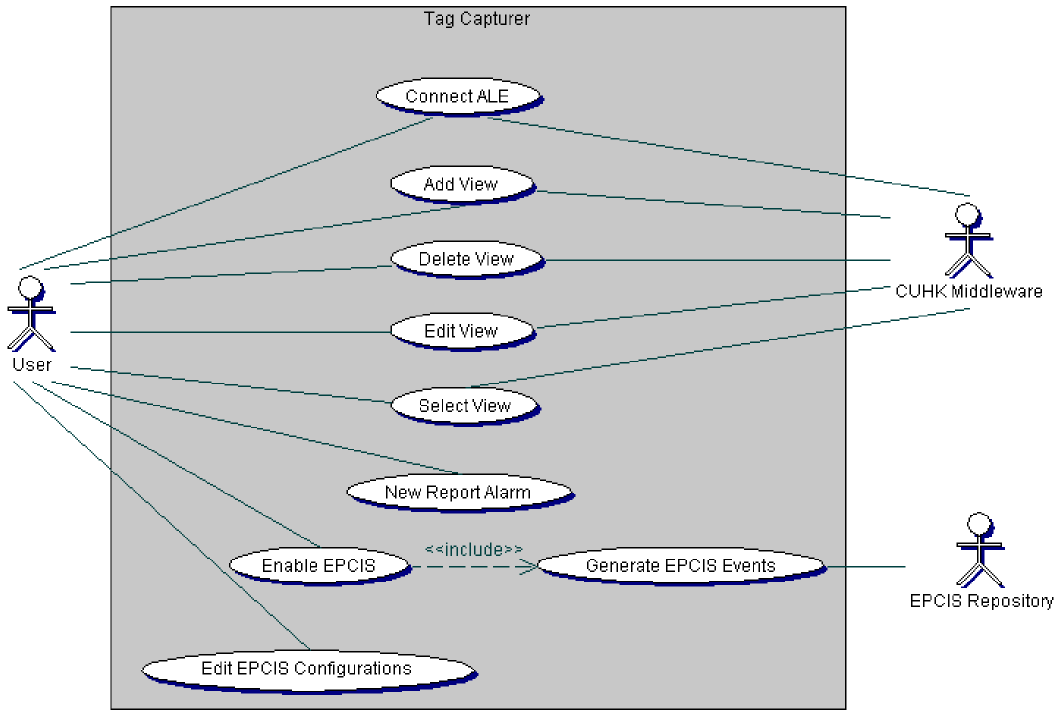
Tag Capturer allows integration of CUHK Middleware with Accada EPCIS Repository. It allows user to define the URL of the EPCIS Capture Interface.

In the scope of EPCIS, Tag Capturer is a Capture Application that may be enabled to send the received EPC Reports to EPCIS Repository via the Capture Interface in terms of EPC Events. According to the EPCIS specification, the EPCIS Events are sent to the Capture Interface via XML messages on HTTP. As a result, conversion of EPC Reports into corresponding EPCIS Events in XML format is needed. For illustration purpose, OBSERVE Event is generated for simulating the scenario of detecting a new EPC Tag from a Tag Reader.

Responses from the Capture Interface indicate the statuses of submitting EPCIS Events to the EPCIS Repository. For every submission of EPCIS Event, its corresponding event status is received and interpreted, and is updated in the main screen of Tag Capturer.

4.1 Use Cases

The diagram below illustrates the use cases of Tag Capturer that provides functional features as discussed above:



Following actors has been identified:

Actor	DEFINITION
User	User that use the Tag Capturer to examine the functions provided by CUHK Middleware.
CUHK Middleware	The middleware that generates EPC Reports
EPCIS Repository	The product that complies with EPCIS Specification and provides a repository for EPCIS Events.

4.1.1 Connect ALE

4.1.1.1 Description

- This use case describes establishing connection with CUHK Middleware.

4.1.1.2 Actors

- User
- CUHK Middleware

4.1.1.3 Preconditions

- ALE Middleware is set up properly and ready to serve request

4.1.1.4 Major Flow of Event

1. User starts the application
2. Application connects CUHK Middleware
3. Application listens from CUHK Middleware for incoming ALE reports

4.1.1.5 Alternate Flows

- Exception will be thrown if the port for CUHK Middleware is invalid

4.1.1.6 Post-conditions

- A connection to CUHK Middleware is established

4.1.2 Add A View

4.1.2.1 Description

- This use case describes the creation of view.

4.1.2.2 Actors

- User
- CUHK Middleware

4.1.2.3 Preconditions

- A logical reader is available
- A mapping of logical reader with the physical reader is available
- ALE Middleware is set up properly and ready to serve request

4.1.2.4 Major Flow of Event

1. User input the necessary information to bind a view with a logical reader
2. Application creates an instance of view supporting class with assigned details
3. Application adds the view definition via the middleware and insert details in the database

4.1.2.5 Alternate Flows

- Exception will be thrown if duplicate view name is found in the database

4.1.2.6 Post-conditions

- A view is defined and details are persisted in the database

4.1.3 Edit A View

4.1.3.1 Description

- This use case describes the editing of view.

4.1.3.2 Actors

- User
- CUHK Middleware

4.1.3.3 Preconditions

- A view has been defined and available for edit
- A mapping of logical reader with the physical reader is available
- ALE Middleware is set up properly and ready to serve request

4.1.3.4 Major Flow of Event

1. User select the view to be altered
2. User modify the necessary information to bind a view with a logical reader
3. Application creates an instance of view supporting class with assigned details
4. Application adds the view definition via the middleware and insert details in the database

4.1.3.5 Alternate Flows

- Exception will be thrown if duplicate view name is found in the database
- Exception will be thrown if logical reader does not exists in the middleware

4.1.3.6 Post-conditions

- A view is modified and details are persisted in the database

4.1.4 Select A View

4.1.4.1 Description

- This use case describes selection of defined view.

4.1.4.2 Actors

- User
- CUHK Middleware

4.1.4.3 Preconditions

- A view has been defined in the middleware

4.1.4.4 Major Flow of Event

1. User selects the desired view for receiving EPC reports
2. Display screen changes according to the selected view

4.1.4.5 Alternate Flows

- N/A

4.1.4.6 Post-conditions

- A view is selected and the display screen is switched

4.1.5 Delete A View

4.1.5.1 Description

- This use case describes the deletion of view.

4.1.5.2 Actors

- User
- CUHK Middleware

4.1.5.3 Preconditions

- A view has been defined in the middleware

4.1.5.4 Major Flow of Event

3. User input the necessary information to bind a view with a logical reader
4. Application creates an instance of view supporting class with assigned details
5. Application adds the view definition via the middleware and insert details in the database

4.1.5.5 Alternate Flows

- Exception will be thrown if duplicate view name is found in the database

4.1.5.6 Post-conditions

- A view is defined and details are persisted in the database

4.1.6 Edit EPCIS Configurations

4.1.6.1 Description

- This use case describes modification of EPCIS Configurations.

4.1.6.2 Actors

- User

4.1.6.3 Preconditions

- N/A

4.1.6.4 Major Flow of Event

1. User select EPCIS Configuration option in the application
2. User input the URL of the Capture Interface
3. Application saves the URL of the Capture Interface

4.1.6.5 Alternate Flows

- N/A

4.1.6.6 Post-conditions

- The URL of the Capture Interface is modified

4.1.7 Alarm on New Reports

4.1.7.1 Description
- This use case describes the notification of receiving new reports.

4.1.7.2 Actors
- User
- CUHK Middleware

4.1.7.3 Preconditions
- A view has been defined in the middleware and subscribed by selecting the view
- ALE Middleware is set up properly and ready to serve request

4.1.7.4 Major Flow of Event
1. User selects the option in the application
2. Application listens to the middleware for new EPC reports generated. Information in the EPC reports is extracted and displayed on the corresponding screen of the view

4.1.7.5 Alternate Flows
- N/A

4.1.7.6 Post-conditions
- Details of new reports will be displayed in the corresponding screen of the defined view

4.1.8 Enable EPCIS

4.1.8.1 Description
- This use case describes enabling notification to EPCIS Repository.

4.1.8.2 Actors
- User

4.1.8.3 Preconditions
- A view has been defined in the middleware and subscribed by selecting the view
- ALE Middleware is set up properly and ready to serve request

4.1.8.4 Major Flow of Event
1. User selects the option in the application
2. Application listens to the middleware for new EPC reports generated. Information in the EPC reports is extracted and displayed on the corresponding screen of the view

4.1.8.5 Alternate Flows
- N/A

4.1.8.6 Post-conditions
- Details of new reports will be displayed in the corresponding screen of the defined view

4.1.9 Generate EPCIS Event

4.1.9.1 Description

- This use case describes sending EPCIS Events to EPCIS Repository via Capture Interface.

4.1.9.2 Actors

- CUHK Middleware
- EPCIS Repository

4.1.9.3 Preconditions

- A view has been defined in the middleware and subscribed by selecting the view
- ALE Middleware is set up properly and ready to serve requests
- EPCIS Repository is set up properly and ready to serve requests
- Alarm on New Reports has been selected

4.1.9.4 Major Flow of Event

1. User selects the desired view for receiving EPC reports
2. Application listens for EPC reports from ALE Middleware
3. For every received EPC report, application transforms the report into EPCIS Event
4. Application sends the EPCIS Event to EPCIS Repository via Capture Interface
5. Application receives responses from the Capture Interface
6. Application interprets the responses and updates the corresponding screen of the selected view

4.1.9.5 Alternate Flows

- Exception will be thrown if the Capture Interface cannot be reached

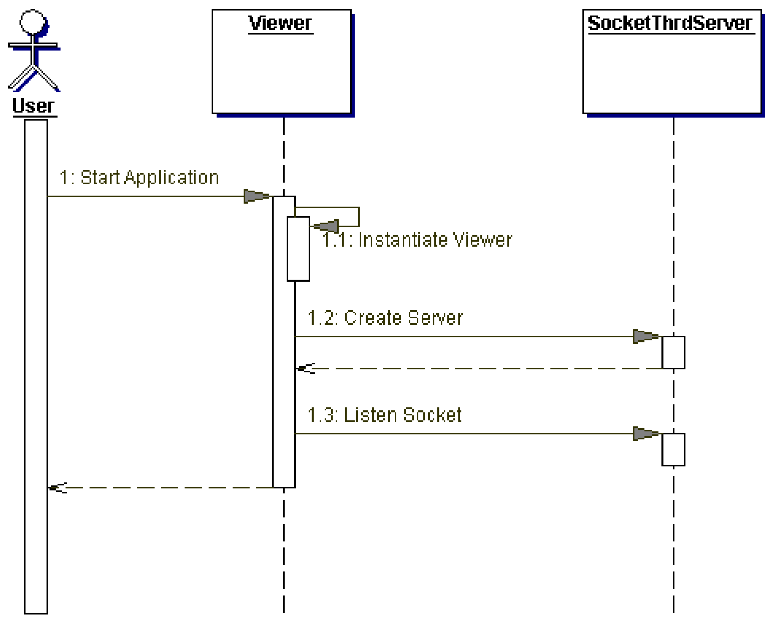
4.1.9.6 Post-conditions

- The EPCIS events are sent to the EPCIS Repository
- Responses from Capture Interface are interpreted and the corresponding record status is updated

5 Logical View

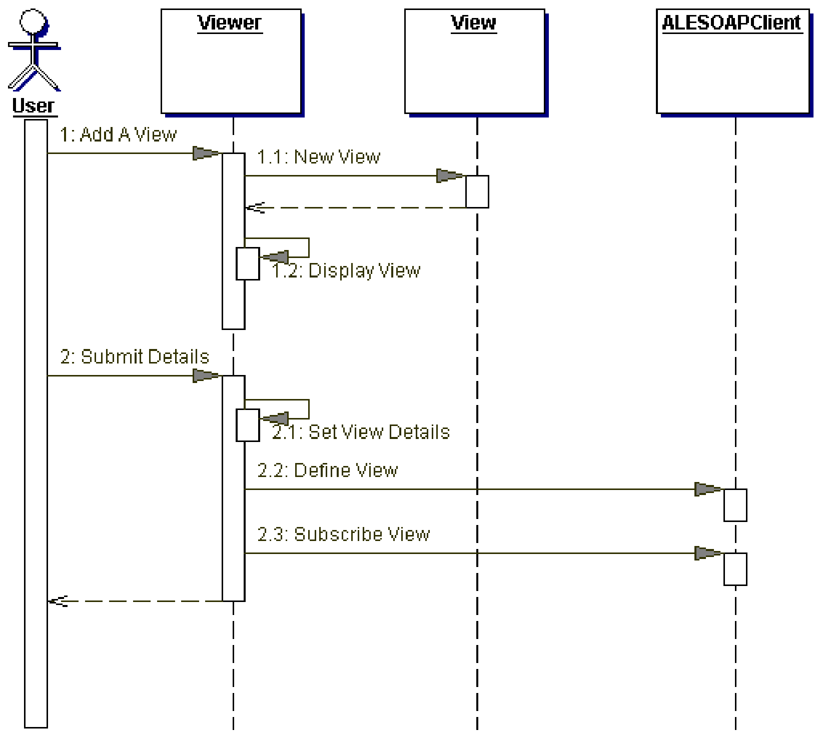
5.1 Use Case Realization

5.1.1 Connect ALE



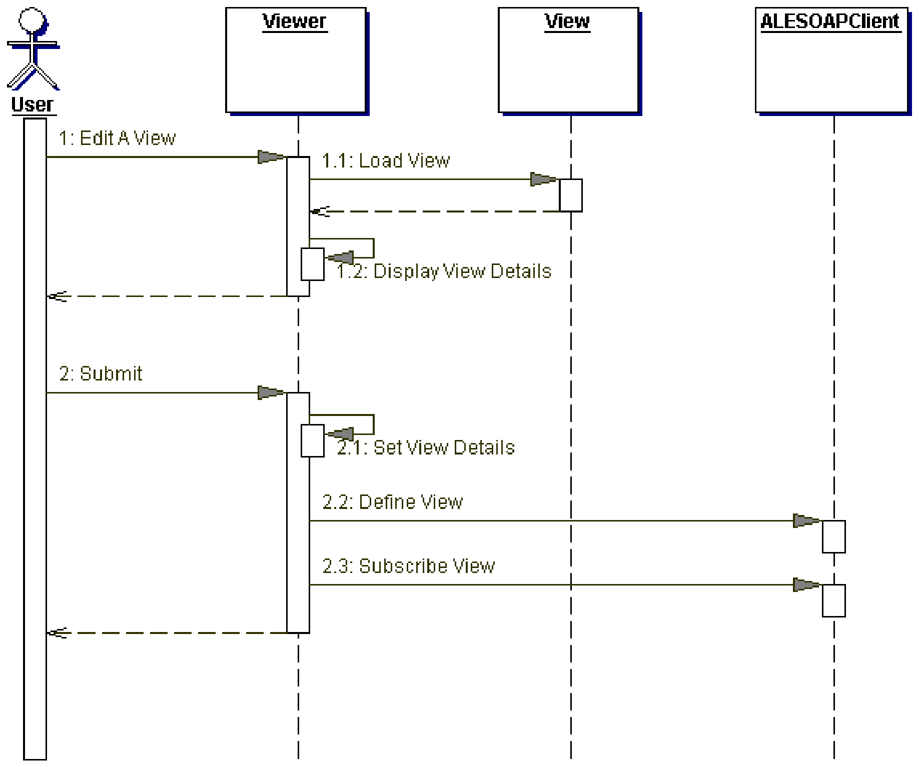
The above sequence diagram describes the use case in section 4.1.1 – Connect ALE. Firstly, the User starts the Application, which instantiates the Viewer object. The Viewer then creates an object SocketThrdServer that serves as a helper class to connect CUHK Middleware. The instantiation of SocketThrdServer establishes direct connection to hook CUHK Middleware. The Viewer then listens to the socket established with CUHK Middleware for any ALE reports generated.

5.1.2 Add A View



The above sequence diagram describes the use case in section 4.1.2 – Add A View. Firstly, the User selects the ‘Add A View’ option on the Application. The Viewer object creates a View object that collects the details. The View object is instantiated and a dialog is popped up for the User to input the details of the View. After the User has filled in all the details, a submit action is triggered by the User. The details of the View object are extracted and set. The Viewer then defines the view by sending the View object to CUHK Middleware via ALESOAPClient. After the View has been defined, the Viewer then subscribes to the defined View. By subscribing the View, the Viewer starts listening for any incoming ALE Reports that are generated for the defined View by CUHK Middleware.

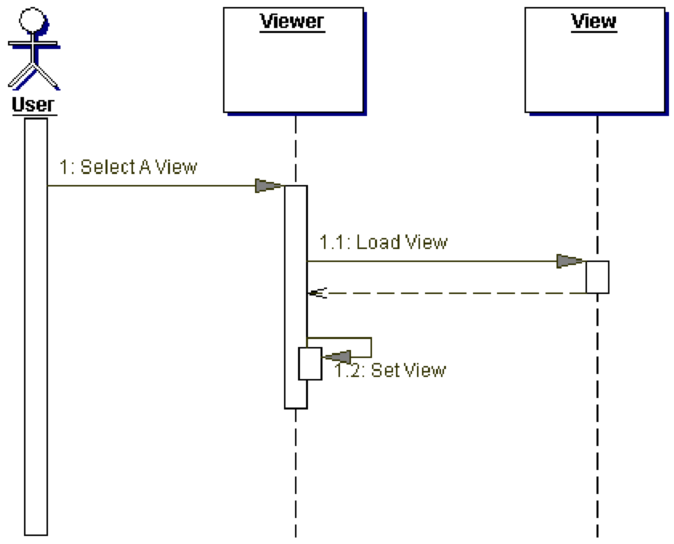
5.1.3 Edit A View



The above sequence diagram describes the use case in section 4.1.3 – Edit A View. Firstly, the User selects the 'Edit A View' option on the Application. The Viewer object the loads the View that has been instantiated in the memory. The details of the selected View are then displayed on a dialog box.

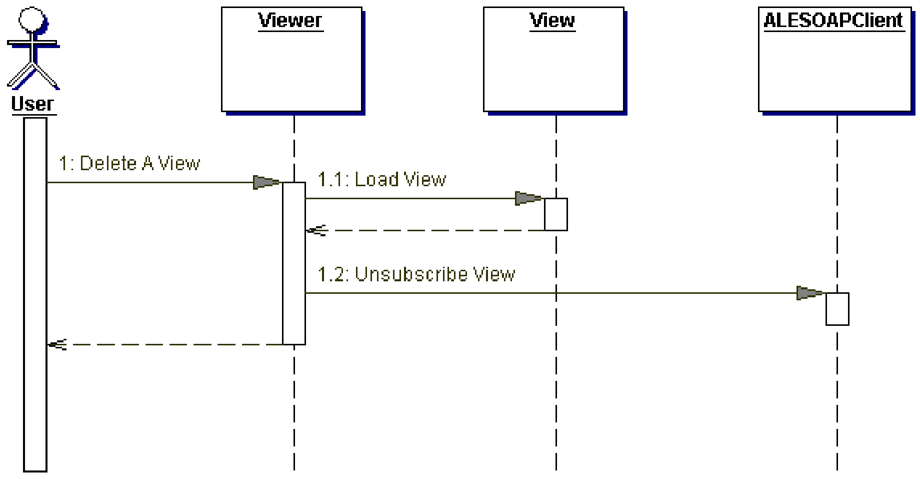
The User modifies the details of the view and then submits the changes by pressing on the submit button on the dialog. The Viewer then creates a View objects with the latest information. The Viewer then defines the View in the CUHK Middleware via ALESOAPClient. After the view has been defined, the Viewer then subscribes to the modified view. By subscribing the View, the Viewer starts listening for any incoming ALE Reports that are generated for the defined View by CUHK Middleware.

5.1.4 Select A View



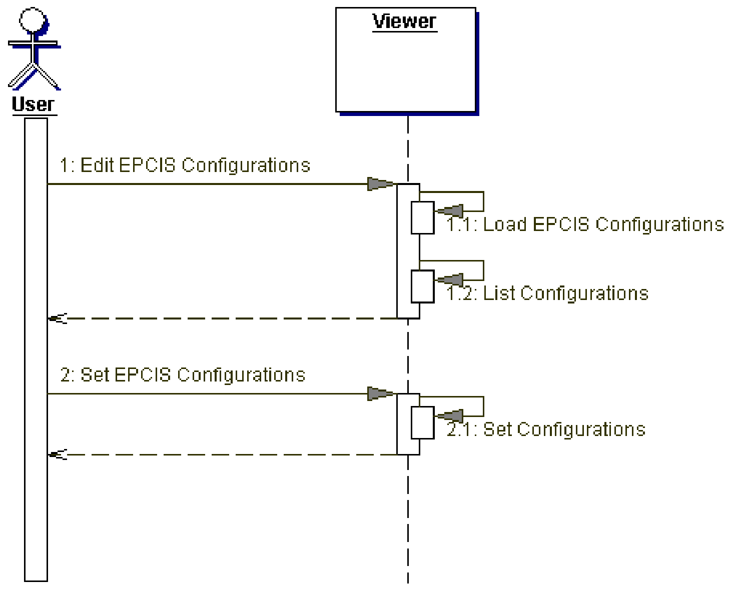
The above sequence diagram describes the use case in section 0 – Select A View. Firstly, the User selects the 'Select A View' option on the Application. The Viewer object loads the View that has been instantiated in the memory. The application switches to the selected view that shows the details of any incoming ALE reports received.

5.1.5 Delete A View



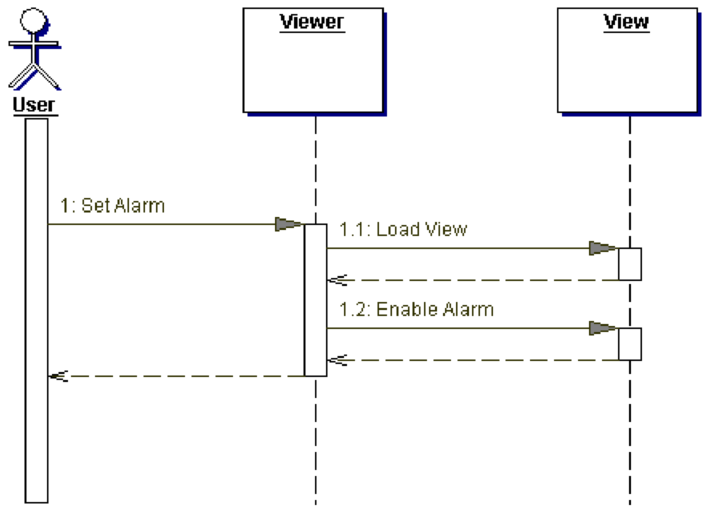
The above sequence diagram describes the use case in section 4.1.5 – Delete A View. Firstly, the User selects the 'Delete A View' option on the Application. The Viewer object the loads the View that has been instantiated in the memory. The Viewer deletes the view by submitting an action to ALESOAPClient object to unsubscribe the view from the list.

5.1.6 Edit EPCIS Configuration


User
Viewer
1: Edit EPCIS Configurations
1.1: Load EPCIS Configurations
1.2: List Configurations
2: Set EPCIS Configurations
2.1: Set Configurations

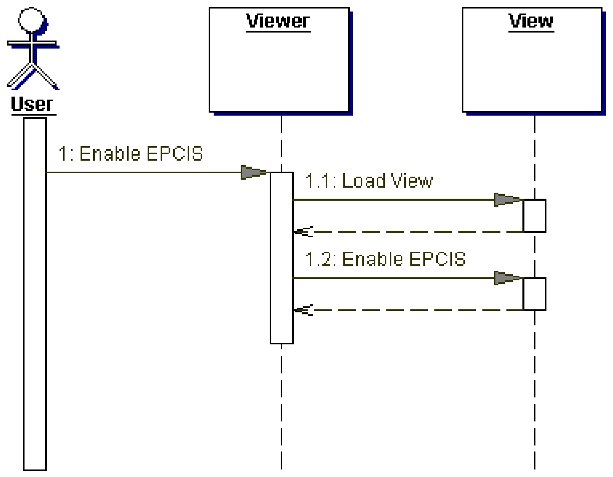
The above sequence diagram describes the use case in section 0 – Edit EPCIS Configuration. The Viewer has been initialized with default EPCIS configurations stated in a property file when the Application starts. The User selects the edit option on the Application. The Viewer then loads the EPCIS configurations from memory and displays on a dialog. The User modifies the details on the dialog and submits the changes. The Viewer then changes the EPCIS configurations in the memory.

5.1.7 Alarm on New Report



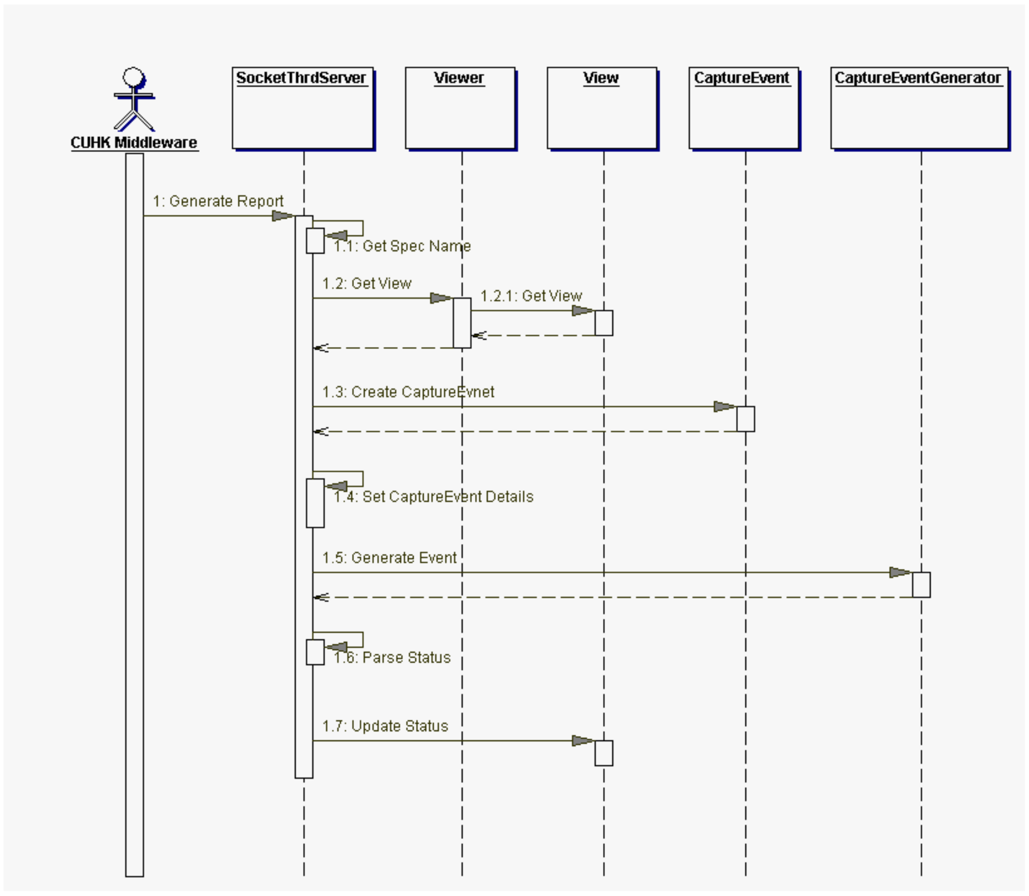
The above sequence diagram describes the use case in section 4.1.7 – Alarm on New Report. The User selects “Alarm on New Reports” on the Application. The Viewer loads the instance of the selected view and set a flag in the View object.

5.1.8 Enable EPCIS



The above sequence diagram describes the use case in section 4.1.8 – Enable EPCIS. The User selects “Send to EPCIS” option on the Application. The Viewer loads the instance of the selected view and set a flag in the View object. The flag indicates sending of EPCIS Event to EPCIS Repository with the pre-defined EPCIS Configurations when an ALE report is received.

5.1.9 Generate EPCIS Event



The above sequence diagram describes the use case in section 4.1.9 – Generate EPCIS Event.

The Application has been started to listen for ALE reports generated by CUHK Middleware via SocketThrdServer object. When a report is received, the SocketThrdServer will get the name of the report specification from the report. The corresponding view is then loaded according to the spec name. The configuration of the loaded view is checked if a 'Send to EPCIS' flag has been set of not. If the flag is set to on, EPCIS Event will be generated according to the details retrieved from the ALE report. CaptureEvent object is created to carry the details of Capture Event.

The generated CaptureEvent object is sent to the EPCIS Repository via CaptureEventGenerator object. The CaptureEventGenerator is instantiated in the SocketThrdServer by using the EPCIS configuration details as specified by the Application.

The CaptureEventGenerator sends the CaptureEvent object to the EPCIS Repository via EPCIS Capture Interface by HTTP requests, and the HTTP responses are received. The responses contain the EPCIS status, which will be parsed by the SocketThrdServer. The status is then sent to the corresponding View object, and the View object will update its corresponding tables to reflect the results. The Viewer displays the details of the View object and the User finally sees the results on the Application.

6 Process View

Since Tag Capturer aims to demonstrate the integration of CUHK Middleware and EPCIS Repository, no special handling of concurrency and synchronization aspects in the design.

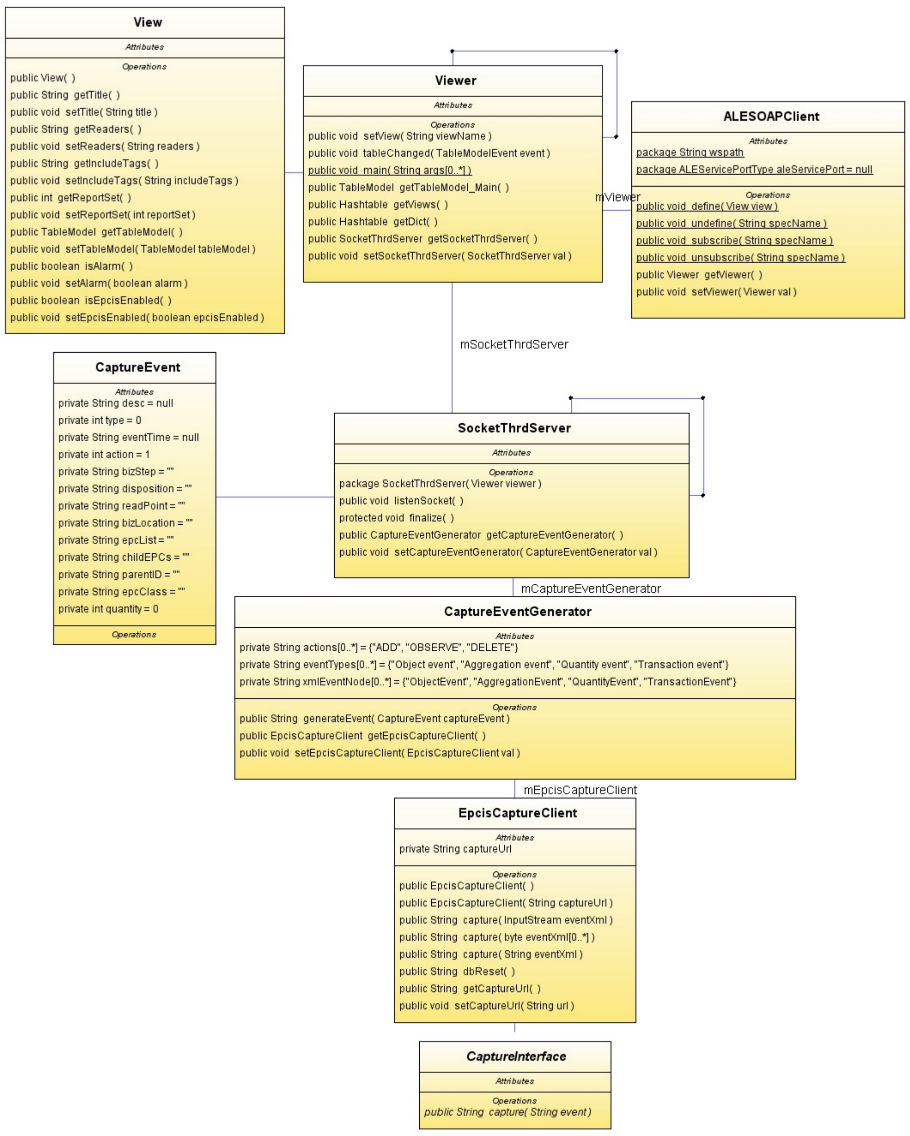
7 Implementation View

7.1 Package

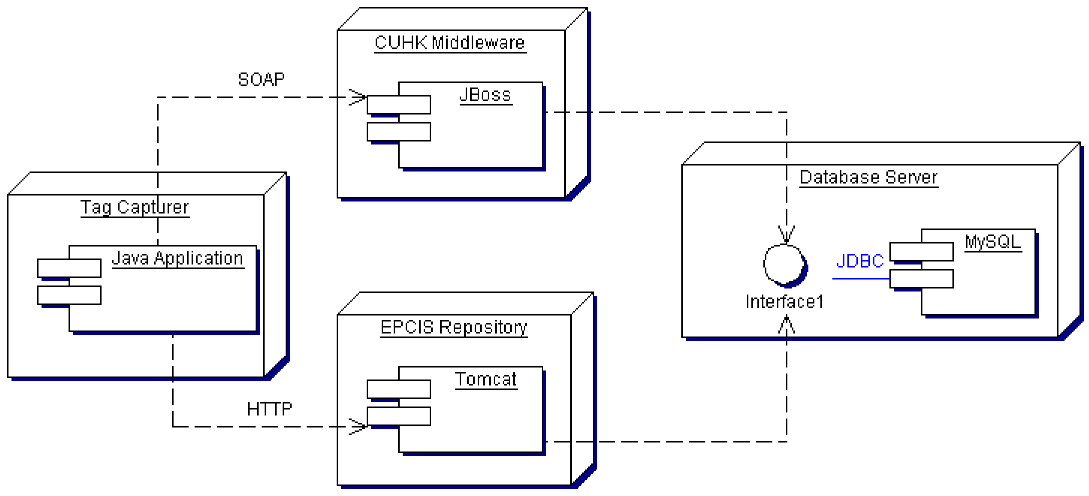
Tag Capturer is an application that demonstrates the integration of CUHK Middleware and third-party EPCIS Repository, the packaging is as simple as one layer, and in one package, namely tagcapturer.

7.2 Class Diagram

The following is the class diagram for TagCapturer:


View
Attributes
Operations
public View( )
public String getTitle( )
public void setTitle(String title)
public String getReaders( )
public void setReaders(String readers)
public String getIncludeTags( )
public void setIncludeTags(String includeTags)
public int getReportSet( )
public void setReportSet(int reportSet)
public TableModel getTableModel( )
public void setTableModel(TableModel tableModel)
public boolean isAlarm( )
public void setAlarm(boolean alarm)
public boolean isEpcisEnabled( )
public void setEpcisEnabled(boolean epcisEnabled)
Viewer
Attributes
Operations
public void setView(String viewName)
public void tableChanged(TableModelEvent event)
public void main(String args[0..*])
public TableModel getTableModel_Main( )
public Hashtable getViews( )
public Hashtable getDict( )
public SocketThrdServer getSocketThrdServer( )
public void setSocketThrdServer(SocketThrdServer val)
ALESOAPClient
Attributes
package String wspath
package ALEServicePortType aleServicePort = null
Operations
public void define(View view)
public void undefine(String specName)
public void subscribe(String specName)
public void unsubscribe(String specName)
public Viewer getViewer( )
public void setViewer(Viewer val)
mViewer
mSocketThrdServer
CaptureEvent
Attributes
private String desc = null
private int type = 0
private String eventTime = null
private int action = 1
private String bizStep = ""
private String disposition = ""
private String readPoint = ""
private String bizLocation = ""
private String epcList = ""
private String childEPCs = ""
private String parentID = ""
private String epcClass = ""
private int quantity = 0
Operations
SocketThrdServer
Attributes
Operations
package SocketThrdServer(Viewer viewer)
public void listenSocket( )
protected void finalize( )
public CaptureEventGenerator getCaptureEventGenerator( )
public void setCaptureEventGenerator(CaptureEventGenerator val)
mCaptureEventGenerator
CaptureEventGenerator
Attributes
private String actions[0..*] = {"ADD", "OBSERVE", "DELETE"}
private String eventTypes[0..*] = {"Object event", "Aggregation event", "Quantity event", "Transaction event"}
private String xmlEventNode[0..*] = {"ObjectEvent", "AggregationEvent", "QuantityEvent", "TransactionEvent"}
Operations
public String generateEvent(CaptureEvent captureEvent)
public EpcisCaptureClient getEpcisCaptureClient( )
public void setEpcisCaptureClient(EpcisCaptureClient val)
mEpcisCaptureClient
EpcisCaptureClient
Attributes
private String captureUrl
Operations
public EpcisCaptureClient( )
public EpcisCaptureClient(String captureUrl)
public String capture(InputStream eventXml)
public String capture(byte eventXml[0..*])
public String capture(String eventXml)
public String dbReset( )
public String getCaptureUrl( )
public void setCaptureUrl(String url)
CaptureInterface
Attributes
Operations
public String capture(String event)

8 Deployment View



9 Data View

Tag Capturer does not own a database to manage, but depends on CUHK Middleware and EPCIS Repository. The two systems keep their own database for maintaining application specific data.

10 System Properties

Tag Capturer loads a property file that specifies default values of parameters during initialization.

The sample property file is given as below:

```
server.host=localhost
server.port=8080
client.host=localhost
client.port=6666
epcis.captureurl=http://localhost:8081/epcis-repository-0.2.0/capture
```