1. General Structure

NYU Parallel And Distributed Systems Group’s (PDSG) Sample Military-Oriented Component-Based Application (SMOCBA) serves as an example of component-based implementation of a general-purpose online intelligence information system. Functionally, SMOCBA serves as a repository of reconnaissance data, and provides functionality to retrieve, process, and store entries (items) of the data, according to user-specified preferences.

Logically, SMOCBA consists of four sub-modules, serving the following functions:

- **Item Creator (Producer) Module** – module that produces items of reconnaissance data and stores them in the database. This module might or might not have its own web portal. For simplicity, in the first stages of application development, this module is absent. We assume that data is preloaded into the database.

- **Analysts Module** – provides functionality for analysts to log on into the system, search items based on desired attributes, process, add comments, and delete items. This module has web portal. The client behavior of this sub-module should be represented by fairly balanced READ-WRITE transactional activities on massively shared data. The component architecture is represented by a relatively large number of non-trivial components, so that component access paths of client requests are of length >= 2.

- **Transformers Module** – analysts submit items for heavy computational processing to the Transformer components in an asynchronous fashion. Once computation is finished, the result is sent back to the analyst. Transformers Module does not have web portal.

- **Viewers Module** – provides functionality for consumers of reconnaissance data to log on into the system, and query the repository for information of interest. Data access is READ-ONLY. This module has some processing capabilities to adapt (filter) the presented data according to the viewer’s rank (e.g., soldier, or general), or access rights.

2. Database Tier

Data items (items of reconnaissance data) are just chunks binary data, e.g. images (JPEG, GIF). Here is the description of the tables in the database.

**Table Items**: storage for items

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>INTEGER UNSIGNED</td>
<td>Primary Key</td>
</tr>
<tr>
<td>name</td>
<td>VARCHAR(20)</td>
<td>name of item</td>
</tr>
<tr>
<td>body</td>
<td>MEDIUMBLOB</td>
<td>body of the item</td>
</tr>
<tr>
<td>type</td>
<td>INTEGER NOT NULL</td>
<td>auxiliary; value is from the predefined set (0 - 9), e.g. 0 – JPEG, 1 – GIF, etc.</td>
</tr>
<tr>
<td>category</td>
<td>VARCHAR(30) NOT NULL</td>
<td>category of the item</td>
</tr>
<tr>
<td>region</td>
<td>VARCHAR(30) NOT NULL</td>
<td>region of item’s origin</td>
</tr>
<tr>
<td>creation_date</td>
<td>DATETIME NOT NULL</td>
<td>when item was created</td>
</tr>
<tr>
<td>last_modif_date</td>
<td>DATETIME NOT NULL</td>
<td>when item was last modified by an analyst</td>
</tr>
<tr>
<td>last_modif_id</td>
<td>VARCHAR(20) NOT NULL</td>
<td>username of analyst who last modified item</td>
</tr>
<tr>
<td>rating_sum</td>
<td>INTEGER NOT NULL</td>
<td>analysts rate the item, say, in the range of 0 – 100</td>
</tr>
<tr>
<td>num_comments</td>
<td>INTEGER NOT NULL</td>
<td>number of comments added to this item</td>
</tr>
</tbody>
</table>

**Table Comments**: analytic comments are added to items by the analysts; comment may contain processed body of the item (binary data), or remark (text comment), or both.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>INTEGER UNSIGNED</td>
<td>Primary Key</td>
</tr>
<tr>
<td>item_id</td>
<td>INTEGER UNSIGNED</td>
<td>references item it belongs to</td>
</tr>
</tbody>
</table>
body MEDIUMBLOB // processed body of the original item; may be null
remark VARCHAR(255) NOT NULL // text comment; may be null
type INTEGER NOT NULL // type of the comment, from the interval (0..9), whatever it means
category VARCHAR(30) NOT NULL // category of the item
region VARCHAR(30) NOT NULL // region of item’s origin
creation_date DATETIME NOT NULL
creator_id VARCHAR(20) NOT NULL // username of the analyst created this comment
last_modif_date DATETIME NOT NULL
last_modif_id VARCHAR(20) NOT NULL // username of the analyst, who last modified this comment
security_level INTEGER NOT NULL // the minimum rank of the viewer required to display the comment to
him/her (see Viewers Table)

Table Analysts:

username VARCHAR(20) // Primary Key
password VARCHAR(20)
name VARCHAR(100)

Table Viewers:

username VARCHAR(20) // Primary Key
password VARCHAR(20)
name VARCHAR(100)
rank INTEGER NOT NULL // rank of the viewer, sergeant – 0, ..., general – 9.

3. WEB Tier (Functionality observed by web clients - analysts and viewers).

Atomic READ operations performed by clients (analysts and viewers):

- Item Queries: all sorts of queries for items, based on attributes (the complete list is to be defined);
- Comment Queries: all sorts of queries for comments of a single item (the complete list is to be defined)
- Coarse-grained READ of an Item: read the whole item from the corresponding Entity EJB (returns the Model object containing all the data of the item, including all comments)
- Fine-grained READs of an item: getters of single item attributes - getRegion, getNumberOfComments, etc.
- Fine-grained READs of a comment: getters of single comment attributes – getType, getCreator, etc.

Atomic WRITE operations performed by analysts:

- Set item’s rating;
- Set item’s security level;
- Add new comment;
- Delete a comment;
- Set the new body of a comment;
- Set new text remark of a comment;
- Set comment’s type;
- Set comment’s security level.

Functionality of the Viewers Module Web Portal

The functionality of the web portal for viewers is more or less straightforward. Viewer should log in, than query the system in a READ-ONLY fashion. A processing intermediate component is present, which limits the items viewed by the viewer according to his/her rank.
## Functionality of Analysts Module Web Portal

Analyst logs in and starts with queries. The result of a query is not the list of whole Model objects of matching items (with all the state), but rather some limited subset of the state (id, region, category, and number of comments; and without the body and comments). The analyst can manipulate with Items in two ways. He/she can modify an item and changes are immediately stored in the database, or he/she can choose items of interest and add them to his/her “basket of analysis”. At this point these items (the state) are stored in a stateful component (Stateful Session EJB). Analyst performs different tasks with the stored items (sending body of an item to a transformer component, finding average rating, adding a comment, etc., the complete list is to be defined). At any point of time analyst might “store/release” the changes he/she made with the items.

### 4. EJB Tier

This is the component architecture of the application. Below we describe functionality of each component in detail.

**Analyst, Viewer Entity Beans**

Standard CMP 2.0 entity beans corresponding to the database tables.
Item and Comment Entity Beans
Entity beans corresponding to the Items and Comments database tables. Both beans expose getter and setter methods of various granularities, from single attributes to the whole Model object. Beans are implemented in CMP 2.0.

Repository stateless session EJB
Facade to the Item and Comment beans for analysts, exposes getter and setter methods of various granularities, also exposes query methods. All components should never access Item and Comment beans directly; they should always use Repository facade bean.

Catalog stateless session EJB
Facade to the Item and Comment beans for viewers, it exposes getter and query methods of all granularities. The difference with the Repository bean is that it does not have setter (write) methods, and the set of getter methods might end up being different than that of the Repository bean.

Viewer Controller stateful session bean
Keeps viewer’s personal information (rank, etc.). Also processes the results of queries, returned by the Catalog component, filtering out the items and comments, which the viewer should not see due to his/her rank. Viewer Controller uses Viewer entity bean to authenticate viewers.

Analyst Controller stateful session bean
The main Analyst Controller component. Keeps last viewed Item and Comment, and the analyst’s personal information. Controller uses the Analyst entity bean to authenticate analysts.

Basket of Analysis stateful session bean
The most sophisticated component. Keeps the items, that analyst has put to the “basket of analysis”. Exposes different business non-trivial (not getter or setter) methods (like getAverageRating(), addCommentToAllItems()) that require some processing/modification of the data kept internally.

Transformers Module EJBs
Sending body of a data item (e.g., image) for processing to the Transformer component is implemented by using standard Asynchronous Remote Facade design pattern. Transformer component is stateless session bean, with a message-driven bean (MDB in the figure) attached to it. Analyst Controller component sends entity (body of an item) to process, in a JMS message to the predefined Queue/Topic (not shown on the figure), where message driven bean picks it up and feeds it to the attached Transformer Component. JMS message also contains serializable handle of the Analyst Controller session bean. Upon completion of the processing, Transformer component instantiates access to the Analyst Controller component using the serializable handle, and ships processed entity back to it. In the future, the following functionality will be implemented: if Transformer fails to instantiate access to the Analyst Controller session bean (e.g., session completed before processing ended) and if additional information was provided by the Analyst Controller (which Item and Comment processed entity should belong to), than Transformer tries to store the processed entity in the database, using the Repository component. If no such information is available, than Transformer is forced to discard the processed entity. However, various other scenarios of data exchange could be supported, in different methods of the Transformer component. Data (image) processing is implemented using the Java Advanced Imaging API (JAI) library.