IM Application Logic 1.0 Component Specification

1. Design

The IM Application Logic component performs the backend logic for a client-server chat application. The logic is event-driven. It provides implementations for all the necessary interfaces in order to handle the events properly. These implementations will be plugged into the appropriate places during the initialization of the application.

Design provides implementations of the ChatStatusEventListener interface for “user status change” and “session status change”. It also provides the implementation of ChatSessionEventListener interface for “change of user in session”, and implementation of ServiceHandler interface to handle the service logic including “To-be-serviced” and “Servicing”. These implementations simply follow the desired logic of the requirements in section 1.2 in requirement specification.

Design also provides the command line interface for Inactivity detection, which can be plugged into the Job Scheduling component at configurable time intervals.

Design provides one common logging class -IMLogger for the logging of this component. This is provided to make the logging more convenient. User can specify the log name of the Log instance and the DateFormat used to format the date time. It is aggregated in every event listener or handler. Null value means that the user doesn’t want to log the operations. Please refer to 1.3.1 for more details of the logging.

1.1 Design pattern

None

1.2 Industry Standards

None

1.3 Required Algorithm

This component does not have complex algorithms. But there are some simple guidelines to follow.

1.3.1 Logging

Logging is not stated in every function of the design. The common logic is stated below.

1) The logging should be done using the *IMLogger..Log(…….).* The IMLogger is provided for the common usage of the logging, which is initialized in the constructor of every listener and handler.

2) Only the important operations of this component are logged. For example, the “session status change” is logged.

3) This section states how to log the operations. For example, in the following method:

```
Public void TestFunction(long userId, Object arg){
        // Validate the argument
        Try{
                // Update the user(Id 10) status to be “ONLINE”

                // Update the session(Id 100) status to be “OPEN”,
                        // user(Id 10) status to be  “OFFLINE”

                // Do other things.
        }
        Catch(Exception){
```

```
        }
}
```

After the logging is added to the function, it will be:

```
Public void TestFunction(Object arg){
        // Validate the argument
        Try{
                // Update the user(Id 10) status to be "ONLINE"
                logger.log(Level.INFO,
                        "Update user status to ONLINE",
                        New String[]{"User – 10"});

                // Update the session(Id 100) status to be "OPEN",
                        // user(Id 10) status to be  "OFFLINE"
                logger.log(Level.INFO,
                        "Update user status to OFFLINE, Session status to be OPEN",
                        New String[]{"User – 10", "Session – 100"});

                // Do other things.
        }
        Catch(Exception e){
                logger.log(LEVEL.ERROR, e.Message, null);
        }
}
```

4) Logged message. The logged message will be something like:

"Action: Update User Status; Time: 2007-03-15 16:00:00; Entities Affected: User – 10, Session – 100".

Or

"Action: Send EnterChatMessage; Time: 2007-03-15 16:00:00; Entities Affected: No".

1.3.2 Status and Entity

The status and entity are maintained by Id. Comparing two statuses **should compare the Ids of them**, instead of the Names.

User Status	
ID	**Name**
101	ONLINE
102	BUSY
103	OFFLINE

Session Status	
ID	**Name**
201	NEW
202	PENDING
203	OPEN
204	CLOSE

Entity	
ID	**Type**
1	User

1.4 Component Class Overview

Package com.cronos.im.logic
Class UserStatusEventListener
This class implements the ChatStatusEventListener interface. It provides method statusChanged to deal with the “user status changed” event. This class is directly initialized by providing the member field arguments. This class will be plugged into the ChatUserStatusTracker class in the future.

Class is thread-safe, as it is immutable.

Class SessionStatusEventListener
This class implements the ChatStatusEventListener interface. It provides method statusChanged to deal with the “session status changed” event. This class is directly initialized by providing the member field arguments. This class will be plugged into the ChatSessionStatusTracker class in the future.

Class is thread-safe, as it is immutable.

Class UserSessionEventListener
This class implements the ChatSessionEventListener interface. It provides method to deal with the events of “User Requested”, “User Added” and “User Removed”. This class can be directly initialized by providing the member field arguments. The acknowledge time value for the responder to accept the request can also be loaded from the configuration. This class will be plugged into the ChatSessionManager class in the future.

Class is thread-safe, as it is immutable.

Class IMServiceHandler
This class implements the ServiceHandler interface. It provides method to deal with the service logic of “To-be-serviced” and “Servicing”, which are the two phases of the service. This class can be initialized from the configuration file or be directly initialized by providing the member field arguments. This class will be plugged into the ServiceEngine class in the future.

Class is thread-safe, as the access to the mutable variables is ensured to be thread-safe.

Class MessagePoolDetector
This class provides command line utility of “Inactivity Detection”. Namespace passed from the command line will be used to instantiate this class. Job Scheduling component will creates job to run this command line utility at a configurable time interval.

Class is thread-safe, as it is immutable.

Class IMLogger
This class provides command logging function for this component. All the logging should be done using this class. Instance of this class is passed into the constructor when initializing the listeners or handlers.

Class is thread-safe, as it is immutable.

1.5 Component Exception Definitions

Component uses one standard exception: **IllegalArgumentException**.
This component also provides one custom exception:

Class IMConfigurationException
This exception should be thrown by IMServiceHandler and UserSessionEventListener classes when the configuration is incorrect. For example, the configuration for the responderWaitTime is negative.

This component also uses the exceptions derived from the interface in other components:

ServiceHandleException
Defined in "com.topcoder.service" package and thrown by IMServiceHandler class.

Other exceptions in other functions should be ignored but logged.

1.6 Thread Safety

This component is thread-safe. Almost all the classes are immutable. Only the IMServiceHandler class is mutable, but the access to the mutable elements is ensured to be thread-safe, so this class is thread-safe.
The dependency components used in this component are ensured to be thread-safe.

2. Environment Requirements

2.1 Environment

- Development language: Java 1.4
- Compile target: Java 1.4

2.2 TopCoder Software Components

Base Exception 1.0 - Used for standard TopCoder exception inheritance

Configuration Manager 2.1.5 - Used for configuration of component

Object Factory 2.0.1 – Used for initialize configured objects. Like ChatMessagePool.

Job Scheduling 3.0 - Used to execute the MessagePoolDetector.

Service Engine 1.0 – Used as the service engine of this component. For example, the IMServiceHandler will be plugged into this component.

Chat User Profile 2.0 – Used to load the user profile

Chat Session Manager 1.0 – Used to manage the session

Chat Status Tracker 1.0 – Used to track the status of user and session

Chat Message Pool 1.1 – Used as the message pool in this component

Sales IM Messenger 1.0 – Used as the Messenger of this component. For example, the AskForChatMessage is defined in this component.

Logging Wrapper 1.2 – Used to log the message.

NOTE: The default location for TopCoder Software component jars is../lib/tcs/COMPONENT_NAME/COMPONENT_VERSION relative to the component installation. Setting the tcs_libdir property in topcoder_global.properties will overwrite this default location.

2.3 Third Party Components

No third party components used.

3. Installation and Configuration

3.1 Package Name

com.cronos.im.logic

3.2 Configuration Parameters

3.2.1 Configuration for IMServiceHandler(Optional)

Default namespace: com.cornos.im.logic.IMServiceHandler

Parameter	Description	Values
responder_wait_time	Time value for the service handler to wait before the responder accepts the request.	Required and must be Not-negative integer value.
check_response_interval	Time interval for the service handler to sleep while waiting the responder to accept the request.	Required and must be Not-negative integer value.

3.2.2 Configuration for UserSessionEventListener(Optional)

Default namespace: com.cornos.im.logic.UserSessionEventListener

Parameter	Description	Values
acknowledge_time	Time value for the responder to accept the request.	Required and must be Not-negative integer value.

*3.2.3 Configuration for the MessagePoolDetector**(Required)***

Default namespace: com.cronos.im.logic.MessagePoolDetector

Parameter	Description	Values
factory_namespace	Namespace used to create ObjectFactory instance.	Required and not-empty, Any valid ObjectFactory namespace.
chat_user_status_tracker_namespace	Namespace used to instantiate	Optional and Not-empty.

	ChatUserStatusTracker instance.	Any valid namespace.
service_engine_namespace	Namespace used to instantiate ServiceEngine instance.	Optional and Not-empty. Any valid namespace.
chat_session_manager_key	Key used to create ChatSessionManager instance.	Required and not-empty, Any valid key.
chat_session_manager_identifier	Identifier to create ChatSessionManager instance.	Optional and Not-empty. Any valid identifier.
Message_pool_key	Key used to create MessagePool instance.	Required and not-empty, Any valid key.
Message_pool_identifier	Identifier to create MessagePool instance.	Optional and Not-empty. Any valid identifier.
whether_log	Whether logging in this class is performed.	Optional and default to be “Yes” Yes/No
log_name	Log name to load the Log instance.	Optional. Any valid log name.
date_format_key	Key used to create DateFormat instance.	Optional and not-empty, Any valid key.
date_format_identifier	Identifier to create DateFormat instance.	Optional and Not-empty. Any valid identifier.
inactive_time_interval	Time value for the Inactive Detector to identify whether the message pool is inactive. The message pool that has not been pulled for longer than “inactive_time_interval” is considered to be inactive.	Required and must be Not-negative integer value.
schedule_time_interval	Interval time used by Job Scheduling to execute the MessagePoolDetector. This configuration is configured in this component but used by other component.	Required and must be Not-negative integer value.

3.2.4 Command line parameter for MessagePoolDetector

Parameter	**Description**	**Values**
-namespace	Namespace to load the configuration.	Optional and Not-empty. Any valid namespace.
-help	Print help message	
-h	Print help message	

3.3 Dependencies Configuration

The configuration for the dependencies should be configured correctly before this component is used. For example, In MessagePoolDetector configuration, the ChatSessionManager instances should be configured in the ObjectFactory configuration first. And the ServiceEngine, ChatUserStatusTracker should be configured in the corresponding namespace first.

And the Logging Wrapper component should be configured before the logger is enabled.

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Follow Dependencies Configuration.
- Execute 'ant test' within the directory that the distribution was extracted to.

4.2 Required steps to use the component

Extract the component distribution.

4.3 Demo

4.3.1 Usage of UserStatusEventListener and SessionStatusEventListener

```
// Demo the usage of UserStatusEventListener.
// The UserStatusEventListener is plugged into the ChatUserStatusTracker component.
ChatUserStatusTracker userStatusTracker = <<Initialize the ChatUserStatusTracker>>

MessagePool msgPool = <<Initialize the MessagePool>>

// Use default logger.
IMLogger logger = new IMLogger (new SimpleDateFormat());

// Initialize the user status event listener.
ChatStatusEventListener userStatusEventListener = new UserStatusEventListener(userStatusTracker,
msgPool, logger);

userStatusTracker.addChatStatusListener(userStatusEventListener);

// When the user status is change, the userStatusEventListener.statusChanged will be invoked.

// Demo the usage of SessionStatusEventListener
ChatSessionStatusTracker sessionStatusTracker = <<Initialize the ChatSessionStatusTracker>>

ChatSessionManager sessionManager = new ChatSessionManagerImpl();
Messenger messenger = <<Initialize the Messenger>>

// Initialize the session status event listener.
ChatStatusEventListener sessionStatusEventListener = new SessionStatusEventListener(
        sessionStatusTracker, sessionManager, messenger, logger);

sessionStatusTracker.addChatStatusListener(sessionStatusEventListener);

// When the session status is changed, the sessionStatusEventListener.statusChanged
```

```
// will be invoked.
```

4.3.2 Usage of UserSessionEventListener

```
UserSessionEventListener sessionEventListener = <<Initialize the UserSessionEventListener>>

ChatSessionManager sessionManager = <<Initialize the ChatSessionManager>>

sessionManager.addChatSessionEventListener(sessionEventListener);
```

4.3.3 Usage of IMServiceHandler

```
IMServiceHandler serviceHandler = <<Initialize the IMServiceHandler>>

ServiceEngine serviceEngine = new ServiceEngine(serviceHandler, …….);
```

4.3.4 Usage of MessagePoolDetector

```
//Command Line:
MessagePoolDetector –namespace:com.cronos.im.logic.MessagePoolDetector
MessagePoolDetector

// Job Scheduling
Scheduler dailyScheduler = new Scheduler(“Daily Job Scheduler”);
Job inactiveDetectorJob = new Job(“Message Pool Detector”,
                    new GregorianCalendar(2007, 01, 01, 01, 00, 00),
                    new GregorianCalendar(2009, 01, 01, 01, 00, 00),
                    1,
                    Calendar.DATE,
                    dailyScheduler.JOB_TYPE_EXTERNAL,
                    <<path of the MessagePoolDetector command line exe>> + "MessagePoolDetector");

dailyScheduler.addJob(inactiveDetectorJob);
dailyScheduler.start();
```

5. Future Enhancements

More status tracker event listener, session event listener or service handler will be implemented to fulfill more complex logic.