Time Tracker Service Detail 1.0 Component Specification

1. Design

The Service Detail custom component is part of the Time Tracker application and provides additional detail to the invoice for Time Entries. This component handles the persistence and other business logic required by the application.

1.1 Design Patterns

Strategy
This pattern is used to make setting of DAO class pluggable.

Façade
ServiceDetailManager use this pattern. Actually these classes provide ability to run complex functionality (DAO, EJB) by using single methods.

DAO
This pattern is used to provide pluggable access to database.

Business Delegate
This pattern is used by managers to delegate all functionality to session beans.

Service Locator
This pattern is used by manager to locate needed session beans

1.2 Industry Standards

JDBC 2.0

EJB 3.0

SQL

1.3 Required Algorithms

Component does not provide some complex algorithm. Code examples in methods documentation and sequence diagrams are enough to how to implement component's requirements. Here is only information how correspond tables fields and object model classes and information how to create AuditHeader.

Used table may be found into create_db.sql.
Corresponding between service_detail and InvoiceServiceDetail:

service_detail field	InvoiceServiceDetail
service_detail_id	id attribute(from TimeTrackerBean)
time_entry_id	timeEntry attribute, which contains TimeEntry instance with id given in table
invoice_id	invoiceId attribute
rate	rate attribute
amount	amount attribute
creation_date	creationDate attribute(from TimeTrackerBean)
creation_user	creationUser attribute(from TimeTrackerBean)
modification_date	modificationDate attribute(from TimeTrackerBean)
modification_user	modificationUser attribute(from TimeTrackerBean)

Example of simple queries:
Insert into service_detail table:

```
INSERT INTO service_detail(service_detail_id, time_entry_id, invoice_id,
rate, amount, creation_date, creation_user, modification_date,
modification_user) VALUES (?,?,?,?,?,?,?,?,?)

creation_date, modification_date should be set to current date
modification_user should be set to creation_user
rate value should be retrieved from Time Tracker Project 3.2, amount
should be calculated by using TimeEntry instance and rate.

Delete row by id:
DELETE * FROM service_detail WHERE service_detail_id =?

Update row by id:
UPDATE service_detail SET time_entry_id=?, invoice_id =?, rate =?, amount
= ?, modification_date = ?, modification_user = ?
modification_date should be set to current date
amount should be calculated by using TimeEntry instance and rate.

Retrieve data from service_detail by id:
SELECT * FROM service_detail WHERE service_detail_id =?

Retrieve data from service_detail by invoice id:
SELECT * FROM service_detail WHERE invoice_id =?
```

Creation of AuditHeader

1. Create new instance of AuditHeader through the non-argument constructor

AuditHeader attributes	Values from InvoiceServiceDatail and table
entityId	id attribute(from TimeTrackerBean)
tableName	service_detail
companyId	id attribute(from TimeTrackerBean)
actionType	either INSERT, DELETE or UPDATE depending of the action type
applicationArea	TT_INVOICE
resourceId	id attribute(from TimeTrackerBean)
creationUser	creationUser attribute(from TimeTrackerBean)

2. Create AuditDetail object for each column involved in SQL Statement.

Operation	oldValue	newValue
INSERT	Null	New value of column
DELETE	Old value should be retrieved from table before deleting	Null
UPDATE	Old value should be retrieved from table before updating	New value of column

3. Create an array containing list of AuditDetail
4. Update the detail of AuditHeader

Usage Configuration Manager and EJB

EJB specification does not allow using some I/O operation inside beans. In this case we can use Configuration Manager directly. But we may use fact that ConfigManager class is singleton and load all needed namespaces before running of EJB container. This decision does not break rules from EJB specification and it is used in this component.

Transactions management

Transaction should be provided by session bean. For this target bean should be properly configured:

```
<container-transaction>
```

```
        <method>
              <ejb-name>ServiceDetailBean</ejb-name>
              <method-name>*</method-name>
        </method>
        <trans-attribute>Required</trans-attribute>
</container-transaction>
```

Now container will be managing transaction by itself, but we need notify it when it should be used rollback. In this case if some operations failed e.g. DAO class throw some exception, this exception should be caught, called method of bean context setRollbackOnly() and re-thrown again. Transaction rollback EJB container provides automatically.

1.4 Component Class Overview

Class InvoiceServiceDetail
This class extends from TimeTrackerBean and represents object model for invoice_detail table. This is used to add and update data from database. This instance returns all methods which retrieves data from database.

Class is not thread safe, because it is mutable.

Package com.topcoder.timetracker.invoice.servicedetail
Interface ServiceDetailManager
This interface defines all operation which may be done on service_detail table. Operations like adding, removing, updating and retrieving entry or group of entries. This interface will be used by user directly and it represents all functionality on service details.

Classes which will implement this interface should be thread safe.

Class ServiceDetailManagerImpl
This is default realization of ServiceDetailManager interface. This class realized J2EE patterns Business Delegate and Service Locator. Each method of this class look up for LocalServiceDetail by using JNDIUtils and run its corresponding method.

Class is not thread safe, because it is immutable.

Package com.topcoder.timetracker.invoice.servicedetail.dao
Interface ServiceDetailDAO
This interface defines all operation which may be done on service_detail table. Operations like adding, removing, updating and retrieving entry or group of entries. This instance should be saved in session beans which will manage transaction. Implementations of this class should never use transaction by themselves.

Classes which will implement this interface may not be thread safe.

Package com.topcoder.timetracker.invoice.servicedetail.dao.impl
Class ServiceDetailDAOImpl
This is implementation of ServiceDetailDAO interface. This class realizes all operation with database table and also provides ability to audit operation. Class has as attributes DBConnectionFactory to create connection to database and IDGenerator to generate unique id for service detail. Class contains special attribute – useBatch which is used to indicate mode of usage of batch operation. If it is true we allowed using batch operation, if false we do not allowed doing it. This attribute was added because not all drivers support batch operations.

Class is immutable, but it is not thread safe. Methods on this class do not use transaction and in this case they may injure data in database. This class should run with some transaction manager, like EJB in this component. Using this class directly may cause a lot of problems so do not do it.

Package com.topcoder.timetracker.invoice.servicedetail.ejb
Interface LocalServiceDetail

This represent local interface for ServiceDetailBean. This interface will be looked up by ServiceDetailManager implementation. This extends from EJBLocalObject interface.

Interface LocalServiceDetailHome
This represent home interface for ServiceDetailBean. Not used in current design. This extends from EJBLocalHome interface

Class ServiceDetailBean
This class is used to provide transactions. It simply gets ServiceDetailDAO from manager and run its corresponding method. If executing is failed it simply use method setRollbackOnly() to notify container that it should be rollback. Class contains as attribute ServiceDetailDAO, which is initialized in method ejbCreate().

Note all session beans classes and interfaces are thread safe. This is achieved by using of EJB container.

1.5 Component Exception Definitions

Component uses standard exceptions:
IllegalArgumentException
SQLException
NamingException
EJBException
CreateException

Custom exceptions:

Class InvoiceServiceDetailException
This exception should never be thrown. This is used only as a parent for all rest eception.

Class is thread safe because it is immutable

Class ConfigurationException
This exception should be thrown when problems with component configuration appear. It encapsulates any possible problems with configuration.

Class is thread safe because it is immutable

Class TransactionCreationException
This exception should be thrown if appears problem to look up session bean. This exception has neutral name to keep back user about usage EJB

Class is thread safe because it is immutable

Class DataAccessException
This exception should be thrown if appears problem with database like failed to create connection. This exception is parent for all other exceptions connected with DAO class.

Class is thread safe because it is immutable

Class EntityNotFoundException
This exception should be thrown if detail is not found. Id of this exception may be or id of detail or id of invoice.

Class is thread safe because it is immutable

Class InvalidDataException

This exception should be thrown if data which should be inserted to database table is incorrect. For example if amount is null.

Class is thread safe because it is immutable

Class BatchExecutionException
This exception should be thrown if batch execution failed. Arrays of ids stored in this class allows to get concrete ids of details for each batch operation fails

Class is thread safe because it is immutable

1.6 Thread Safety

Component is not thread safe. It contains classes which are mutable and classes which work in database in not safe manner.

InvoiceServiceDetail is not thread safe because it is mutable and do not synchronize access to mutable attributes.

ServiceDetailDAOImpl is not thread safe because it does not use transaction and may injure data in database. It should run with some transaction manager, like EJB in this component. This component uses DAO in thread safe manner.

Thread safety of session beans is guaranteed by EJB container.

In this case for achieving thread safety, methods of InvoiceServiceDetail should be synchronized.

2. Environment Requirements

2.1 Environment

J2EE 1.4

2.2 TopCoder Software Components

Base Exception 1.0
This component is used as parent for all component exceptions.

Configuration Manager 2.1.5
This component is used for configuration of component.

Object Factory 2.0
This component is used for creation configurable instances of ServiceDetailDAO.

DB Connection Factory 1.0
This component is used to create connection to database in DAO classes

ID Generator 3.0
This component is used to generate unique id of database table’s fields.

Time Tracker Audit 3.2
This component is used to audit information about operations (add/update/delete).

Time Tracker Common 3.1
Its class TimeTrackerBean is used as a parent of InvoiceServiceDetail

Time Tracker Project 3.2
This component is used to retrieve rate of service detail

Time Tracker User 3.2

This component is used to retrieve rate of service detail

Time Entry 3.2
This component is used to retrieve and save time entry for service detail

JNDI Context Utility 1.0
This component is used to lookup session bean

Time Tracker Invoice 1.0
Its class Invoice is used in InvoiceServiceDetail instance as attribute.

2.3 Third Party Components

None

3. Installation and Configuration

3.1 Package Name

com.topcoder.timetracker.invoice.servicedetail
com.topcoder.timetracker.invoice.servicedetail.dao
com.topcoder.timetracker.invoice.servicedetail.dao.impl
com.topcoder.timetracker.invoice.servicedetail.ejb

3.2 Configuration Parameters

Configuration of manager:

Parameter	Description	Values
connection_string	JNDI connection string which is used to look up session bean. Required.	Non-empty string

Configuration of DAO class:

Parameter	Description	Values
connection_namespace	The namespace used to create DBConnectionFactory object. Required	Non-empty string
connection_name	The connection name used to obtain db connection from DBConnectionFactory. Required	Non-empty string
id_generator	Represent name for IDGenerator which should be used to service_detail_id field of service_detail table. Required	Non-empty string
factory_namespace	The namespace for ObjectFactory which is used to create TimeEntryManager, ProjectWorkerUtility and AuditManager. Required	Non-empty string
audit_name	The audit manager name used to obtain AuditManager instance from ObjectFactory. Required	Non-empty string
time_entry_name	The time entry name used to obtain TimeEntryManager	Non-empty string

	instance from ObjectFactory. Required	
project_worker_manager_name	The project worker utility name used to obtain ProjectWorkerUtility instance from ObjectFactory. Required	Non-empty string
user_manager_name	The user manager name used to obtain UserManager instance from ObjectFactory. Required	Non-empty string
use_batch	This parameter is used by ServiceDetailDAO. It indicate mode of usage of batch operation. If it is true we allowed using batch operation, if false we do not allowed doing it. This attribute was added because not all drivers support batch operations.	boolean value

3.3 Dependencies Configuration

It should be configured Configuration Manager 2.1.5, DB Connection Factory 1.0 and ID Generator 3.0, Search Builder 1.3.

Note that DB Connection Factory should be configured by JNDI reference to data source. Like:

```
<Property name="DefaultJNDI">
<Property name="producer">

<Value>com.topcoder.db.connectionfactory.producers.JNDIConnectionProducer
</Value>
</Property>
      <Property name="parameters">
            <Property name="jndi_name">
                  <Value>java:comp/env/jdbc/InformixDataSource</Value>
            </Property>
      </Property>
</Property>
```

InformixDataSource is name of data source which should be loaded (see example in ifx-ds.xml). Only in this case session beans will provide transaction.

ejb_jar.xml should contain two environment variables – factory_namespace and dao_name. See example of file in design distribution.

Before running this component should be preloaded all needed namespaces for Configuration Manager.

4. Usage Notes

4.1 Required steps to test the component

See below.

4.2 Required steps to use the component

1. You need to install Informix and JBoss
2. Put JDBC driver for Informix to directory {JBOSS_HOME}\server\default\lib
3. Put ifx-ds.xml to directory {JBOSS_HOME}\ server\default\deploy
4. Deploy you component into JBoss. Example of ejb_jar.xml and jboss.xml may be found in design distribution
5. Run server. Now you application is ready to use.

4.3 Demo

Here is shown base demo for entries. Demo for statuses is actually the same except it does not work with batch operations and reject reasons.

```
// create instance of manager
ServiceDetailManager manager = new ServiceDetailManagerImpl();

/*Lets imagine that we have in database two already configured entries
with ids 1 and 2*/

//get the details
InvoiceServiceDetail firstDetail = manager.retrieveServiceDetail(1L);

InvoiceServiceDetail secondDetail = manager.retrieveServiceDetail(2L);

//Or by using batch operation; result should be the same
long[] detailsArray = new long[2];
detailsArray[0] = 1;
detailsArray[1] = 2;

InvoiceServiceDetail[] details =
manager.retrieveServiceDetails(detailsArray);

firstDetail = InvoiceServiceDetail[0];

secondDetail = InvoiceServiceDetail[1];

//we may create some Detail
InvoiceServiceDetail thirdDetail = new InvoiceServiceDetail();

//lets set it attributes like in firstDetail by using just simple setters.

//add new Detail to data store and perform audit
manager.addServiceDetail(thirdDetail, true);

//now we may ensure that dao generate new id for this Detail
System.out.println("" + thirdDetail.getId());

//Now it is possible to delete it and perform audit
manager.deleteServiceDetail(thirdDetail.getId(), true);

//It is possible to update something in details
```

```
firstDetail.setInvoiceId(10);

//Now we may update it in database and do not perform audit
manager.updateServiceDetail(firstDetail, false);

/*We may also look for details by invoice id. This array should contain
detail with the same id as in firstDetail*/
InvoiceServiceDetail[] details1 =
manager.retrieveServiceDetails(10);

/*except retrieve, delete and update one or several details we have
methods which allows to retrieve or delete all details */

//retrieve all details
InvoiceServiceDetail[] details2 = manager.retrieveAllServiceDetails();

//delete all details and do not perform audit
manager.deleteAllServiceDetails(false);
```

5. Future Enhancements

“*Invoice reporting, aging of invoices and additional steps I the workflow process will likely evolve.”*
In general this future enhancement treats to all invoice components, which part is this component.