Cibet Control Framework

Tutorial 1.0

1. Content

2. Introduction

For simplicity we assume for all examples that Cibet functionality should be applied on a web archive application. The test examples Tutorial1.war … Tutorial6.war can be downloaded ready-built from here:

http://www.logitags.com/cibet/tutorial/Tutorial1.war

http://www.logitags.com/cibet/tutorial/Tutorial2.war

http://www.logitags.com/cibet/tutorial/Tutorial3.war

http://www.logitags.com/cibet/tutorial/Tutorial4.war

http://www.logitags.com/cibet/tutorial/Tutorial5.war

http://www.logitags.com/cibet/tutorial/Tutorial6.war

The junit test classs can be downloaded here:
http://www.logitags.com/cibet/download.html.

3. Archiving

3.1Archiving state changes of a JPA entity

Business Case:
We have a business entity Person which has a 1:n relationship to entity Address. A history of all modifications of a Person or Address should be kept.

Application: Tutorial1.war

Test: com.logitags.cibet.tutorial.Tutorial1.archive1()

Sensor: JPA

Actuator: ARCHIVE

Solution:

1. Include following dependency to your web application:

```
<dependency>
        <groupId>com.logitags</groupId>
        <artifactId>cibet-jpa</artifactId>
        <version>${version}</version>
</dependency>
```

2. ARCHIVE actuator requires a database. Create the Cibet database tables from the sql scripts in cibet-core.jar!sql

3. Next we need some persistence units:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation=http://java.sun.com/xml/ns/persistence
        http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd version="2.0">

        <persistence-unit name="Cibet" transaction-type="JTA">
                <provider>org.hibernate.ejb.HibernatePersistence</provider>
                <jta-data-source>java:/MyDatasource</jta-data-source>
                <class>com.logitags.cibet.actuator.archive.Archive</class>
                <class>com.logitags.cibet.actuator.common.Controllable</class>
                <class>com.logitags.cibet.core.EventResult</class>
                <class>com.logitags.cibet.resource.ResourceParameter</class>
                <class>com.logitags.cibet.resource.Resource</class>
                <class>com.logitags.cibet.sensor.ejb.EjbResource</class>
                <class>com.logitags.cibet.sensor.http.HttpRequestResource</class>
                <class>com.logitags.cibet.sensor.jdbc.driver.JdbcResource</class>
                <class>com.logitags.cibet.sensor.jpa.JpaResource</class>
                <class>com.logitags.cibet.sensor.jpa.JpaQueryResource</class>
                <class>com.logitags.cibet.sensor.pojo.MethodResource</class>

                <properties>
                  <property name="hibernate.format_sql" value="true" />
                  <property name="hibernate.transaction.jta.platform" value=
                            "org.hibernate.service.jta.platform.internal.JBossAppServerJtaPlatform" />
                </properties>
        </persistence-unit>

        <persistence-unit name="APPL-UNIT" transaction-type="JTA">
                <provider>com.logitags.cibet.sensor.jpa.Provider</provider>
                <jta-data-source>java:/MyDatasource </jta-data-source>
                <class>com.logitags.cibet.tutorial.Person</class>
                <class>com.logitags.cibet.tutorial.Address</class>
                <properties>
                  <property name="com.logitags.cibet.persistence.provider" value=
                            "org.hibernate.ejb.HibernatePersistence" />
                  <property name="hibernate.transaction.jta.platform" value=
                            "org.hibernate.service.jta.platform.internal.JBossAppServerJtaPlatform" />
                </properties>
        </persistence-unit>

</persistence>
```

The Cibet persistence unit is used internally by Cibet to persist its entities.

The APPL-UNIT persistence unit is the application's PU to persist entities Person and Address. Here we set the JPA sensor: The provider must be the Cibet provider and the original provider is set in property *com.logitags.cibet.persistence.provider*.

4. The Cibet persistence unit must be made available in JNDI. Add to your web.xml:

```
   <persistence-unit-ref>
     <persistence-unit-ref-name>java:comp/env/Cibet</persistence-unit-ref-name>
     <persistence-unit-name>Cibet</persistence-unit-name>
   </persistence-unit-ref>
```

5. Add the Cibet context filter to web.xml. It starts and ends the Cibet context and synchronizes it with the http session:

```
        <filter>
                <filter-name>cibetContextFilter</filter-name>
                <filter-class>com.logitags.cibet.context.CibetContextFilter</filter-class>
        </filter>

        <filter-mapping>
                <filter-name>cibetContextFilter</filter-name>
                <url-pattern>/*</url-pattern>
```

```
        </filter-mapping>
```

6. Finally, we need the Cibet configuration. Create a cibet-config.xml file in the classpath, for example in Tutorial1.war!WEB-INF/classes and/or add the following snippet. The setpoint defines that all inserts, deletes and updates of a Person or Address entity should be archived.

```
<?xml version="1.0" encoding="UTF-8"?>
<cibet xmlns="http://www.logitags.com" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.logitags.com
       http://www.logitags.com/cibet/cibet-config_1.3.xsd">

  <setpoint id="setpoint-1">
    <controls>
      <event>INSERT, DELETE, UPDATE</event>
      <target>com.logitags.cibet.tutorial.Person,
              com.logitags.cibet.tutorial. Address</target>
    </controls>
    <actuator name="ARCHIVE"/>
  </setpoint>

</cibet>
```

Effect:
Each time a Person or Address object is inserted, deleted or updated like in

```
   @PersistenceContext(unitName = "APPL-UNIT")
   protected EntityManager applEman;

   public void persist(Person person) {
       applEman.persist(person);
   }
```

an entry in table cib_archive is written. In order to display the history entries of a Person with Id 4973 the following method could be applied:

```
List<Archive> history = ArchiveLoader.loadArchivesByPrimaryKey(
                        Person.class.getName(), “4973”);
```

If Person objects have been updated you may be interested in what has been updated between the Archive entries. The following method takes a list of Archives as parameter returns an ordered map with the Archive object as key and a list of Difference objects as value. The list of Differences could be empty if the Archive is not from an UPDATE event.

```
Map<Archive, List<Difference>> differencesMap = ArchiveLoader.analyzeDifferences(history);

Iterator<Entry<Archive, List<Difference>>> iter = differencesMap.entrySet().iterator();
while (iter.hasNext()) {
  Entry<Archive, List<Difference>> e = iter.next();
  log.info(“Archive ” + e.getKey().getArchiveId + " has " + e.getValue().size() + "
    differences";
  for (Difference dif: e.getValue()){
    log.info(“Property ” + dif.getCanonicalPath + “: old value=” + dif.getOldValue() +
             “, new value=” + dif.getNewValue());
  }
}
```

3.2Archiving http requests

Business Case:
All requests to URL with path http:// localhost:8788/Tutorial2/call shall be monitored and historized.

Application: Tutorial2.war

Test: com.logitags.cibet.tutorial.Tutorial2.archive2()

Sensor: HTTP-FILTER

Actuator: ARCHIVE

Solution:

1. Include following dependency to your web application:

```
        <dependency>
                <groupId>com.logitags</groupId>
                <artifactId>cibet-core</artifactId>
                <version>${version}</version>
        </dependency>
```

2. ARCHIVE actuator requires a database. Create the Cibet database tables from the sql scripts in cibet-core.jar!sql

3. Next we need the Cibet persistence unit for the persistence of Archive entries:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation=http://java.sun.com/xml/ns/persistence
        http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd version="2.0">

        <persistence-unit name="Cibet" transaction-type="JTA">
                <provider>org.hibernate.ejb.HibernatePersistence</provider>
                <jta-data-source>java:/MyDatasource</jta-data-source>
                <class>com.logitags.cibet.actuator.archive.Archive</class>
                <class>com.logitags.cibet.actuator.common.Controllable</class>
                <class>com.logitags.cibet.core.EventResult</class>
                <class>com.logitags.cibet.resource.ResourceParameter</class>
                <class>com.logitags.cibet.resource.Resource</class>
                <class>com.logitags.cibet.sensor.ejb.EjbResource</class>
                <class>com.logitags.cibet.sensor.http.HttpRequestResource</class>
                <class>com.logitags.cibet.sensor.jdbc.driver.JdbcResource</class>
                <class>com.logitags.cibet.sensor.jpa.JpaResource</class>
                <class>com.logitags.cibet.sensor.jpa.JpaQueryResource</class>
                <class>com.logitags.cibet.sensor.pojo.MethodResource</class>

                <properties>
                  <property name="hibernate.format_sql" value="true" />
                  <property name="hibernate.transaction.jta.platform" value=
                        "org.hibernate.service.jta.platform.internal.JBossAppServerJtaPlatform" />
                </properties>
        </persistence-unit>

</persistence>
```

4. Make the Cibet persistence unit available in JNDI. Add to your web.xml:

```
   <persistence-unit-ref>
     <persistence-unit-ref-name>java:comp/env/Cibet</persistence-unit-ref-name>
     <persistence-unit-name>Cibet</persistence-unit-name>
```

```
   </persistence-unit-ref>
```

5. The HTTP-FILTER sensor is a servlet filter. Add the CibetFilter to web.xml:

```
        <filter>
                <filter-name>cibetFilter</filter-name>
                <filter-class>com.logitags.cibet.sensor.http.CibetFilter</filter-class>
        </filter>

        <filter-mapping>
                <filter-name>cibetFilter</filter-name>
                <url-pattern>/*</url-pattern>
        </filter-mapping>
```

6. Finally, we need the Cibet configuration. Create a cibet-config.xml file in the classpath, for example in Tutorial2.war!WEB-INF/classes and/or add the following snippet. The setpoint defines that all requests to URL starting with http://localhost:8788/Tutorial2/ca should be archived.

```
<?xml version="1.0" encoding="UTF-8"?>
<cibet xmlns="http://www.logitags.com" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.logitags.com
       http://www.logitags.com/cibet/cibet-config_1.3.xsd">

  <setpoint id="setpoint-2">
    <controls>
      <event>INVOKE</event>
      <target>http://localhost:8788/Tutorial2/ca*</target>
    </controls>
    <actuator name="ARCHIVE"/>
  </setpoint>

</cibet>
```

Effect:
Each time a request is sent to URL beginning with http:// localhost:8788/ Tutorial2/ca an entry in table cib_archive is written. In order to display the archives of all executed requests to a specific URL the following method could be applied:

```
List<Archive> history = ArchiveLoader.loadArchives("http://localhost:8788/Tutorial2/call");
```

4. Dual Control

4.1Applying Four Eyes Control on a method call

Business Case
The business method changeConfig() in class TutorialServlet2 changes a configuration and informs all registered clients about the change. The business case is critical and must be supervised by a second user before it is executed.

Application: Tutorial2.war

Test: com.logitags.cibet.tutorial.Tutorial2.fourEyes1()

Sensor: ASPECT

Actuator: FOUR_EYES

Solution:

1. Include following dependency to your web application:

```
        <dependency>
                <groupId>com.logitags</groupId>
                <artifactId>cibet-core</artifactId>
                <version>${version}</version>
        </dependency>
```

2. FOUR_EYES actuator requires a database. Create the Cibet database tables from the sql scripts in cibet-core.jar!sql

3. Next we need the Cibet persistence unit which is responsible to persist dual control business cases in table cib_controllable:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation=http://java.sun.com/xml/ns/persistence
        http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd version="2.0">

        <persistence-unit name="Cibet" transaction-type="JTA">
                <provider>org.hibernate.ejb.HibernatePersistence</provider>
                <jta-data-source>java:/MyDatasource</jta-data-source>
                <class>com.logitags.cibet.actuator.archive.Archive</class>
                <class>com.logitags.cibet.actuator.common.Controllable</class>
                <class>com.logitags.cibet.core.EventResult</class>
                <class>com.logitags.cibet.resource.ResourceParameter</class>
                <class>com.logitags.cibet.resource.Resource</class>
                <class>com.logitags.cibet.sensor.ejb.EjbResource</class>
                <class>com.logitags.cibet.sensor.http.HttpRequestResource</class>
                <class>com.logitags.cibet.sensor.jdbc.driver.JdbcResource</class>
                <class>com.logitags.cibet.sensor.jpa.JpaResource</class>
                <class>com.logitags.cibet.sensor.jpa.JpaQueryResource</class>
                <class>com.logitags.cibet.sensor.pojo.MethodResource</class>

                <properties>
                  <property name="hibernate.format_sql" value="true" />
                  <property name="hibernate.transaction.jta.platform" value=
                             "org.hibernate.service.jta.platform.internal.JBossAppServerJtaPlatform" />
                </properties>
        </persistence-unit>

</persistence>
```

4. Make the Cibet persistence unit available in JNDI. Add to your web.xml:

```
   <persistence-unit-ref>
     <persistence-unit-ref-name>java:comp/env/Cibet</persistence-unit-ref-name>
     <persistence-unit-name>Cibet</persistence-unit-name>
   </persistence-unit-ref>
```

5. Add the Cibet context filter to web.xml. It starts and ends the Cibet context and synchronizes it with the http session.

```
        <filter>
                <filter-name>cibetContextFilter</filter-name>
                <filter-class>com.logitags.cibet.context.CibetContextFilter</filter-class>
        </filter>
```

```
        <filter-mapping>
               <filter-name>cibetContextFilter</filter-name>
               <url-pattern>/*</url-pattern>
        </filter-mapping>
```

6. The ASPECT sensor can be placed as an annotation or non-intrusively as an aspectJ aspect. Add @CibetIntercept annotation to method ConfigurationService.changeConfig(). Alternatively, create a file aop.xml in application.war!WEB-INF/classes with the following content:

```
<aspectj>
  <aspects>
    <concrete-aspect name="com.myApp.configurationChange.aspect"
                     extends="com.logitags.cibet.sensor.pojo.CustomAspect">
      <pointcut name="cibetIntercept" expression="target(com.logitags.cibet.tutorial.TutorialServlet2)
               &amp;&amp; execution(public String changeConfig())"/>
    </concrete-aspect>
  </aspects>
</aspectj>
```

7. Finally, we need the Cibet configuration. Create a cibet-config.xml file in the classpath, for example in Tutorial2.war!WEB-INF/classes and/or add the following snippet. The setpoint defines that all calls of the changeConfig method in class TutorialServlet2 are not executed directly but must be released by a second user.

```
<?xml version="1.0" encoding="UTF-8"?>
<cibet xmlns="http://www.logitags.com" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.logitags.com
       http://www.logitags.com/cibet/cibet-config_1.3.xsd">

  <setpoint id="setpoint-3">
    <controls>
      <event>INVOKE</event>
      <target>com.logitags.cibet.tutorial.TutorialServlet2</target>
      <method>changeConfig</method>
    </controls>
    <actuator name="FOUR_EYES"/>
  </setpoint>

</cibet>
```

8. The aspect must be weaved into the classes. This can be done at runtime. Download aspectjweaver and start the application (server) with
   set "JAVA_OPTS=%JAVA_OPTS% -javaagent:path-to\aspectjweaver.jar "

Effect:
The FOUR_EYES actuator requires that a user is authenticated when executing the controlled business case. Therefore we first login a user. The user is stored in the Cibet context within the http session. When the method TutorialServlet2.changeConfig() is called it is not executed but an entry in table cib_controllable is written. In order to display method calls which attend release the following method could be applied:

```
List<Controllable> openList =
DcLoader.findUnreleased("com.logitags.cibet.tutorial.TutorialServlet2");

for (Controllable dc : openList) {
  log.info("created by " + dc.getCreateUser() + " at " + dc.getCreateDate());
  log.info("Class: " + dc.getResource().getTarget() + ", method: " +
           ((MethodResource)dc.getResource()).getMethod());
}
```

Now another user must login. The first user is not allowed to release the method call. The second user can now release the method call with:

```
openList.get(0).release(“accepted”);
```

or reject it with

```
openList.get(1).reject(“No!”);
```

4.2Applying Four Eyes Control on a URL

Business Case
HTTP requests on URL /secured change security configuration. The business case is critical and must be checked and released by a second user before it is executed.

Application: Tutorial2.war

Test: com.logitags.cibet.tutorial.Tutorial2.fourEyes2()

Sensor: HTTP-FILTER

Actuator: FOUR_EYES

Solution:

1. Include following dependency to your web application:

```
        <dependency>
                <groupId>com.logitags</groupId>
                <artifactId>cibet-core</artifactId>
                <version>${version}</version>
        </dependency>
```

2. FOUR_EYES actuator requires a database. Create the Cibet database tables from the sql scripts in cibet-core.jar!sql

3. Next we need the Cibet persistence unit:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation=http://java.sun.com/xml/ns/persistence
        http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd version="2.0">

        <persistence-unit name="Cibet" transaction-type="JTA">
                <provider>org.hibernate.ejb.HibernatePersistence</provider>
                <jta-data-source>java:/MyDatasource</jta-data-source>
                <class>com.logitags.cibet.actuator.archive.Archive</class>
                <class>com.logitags.cibet.actuator.common.Controllable</class>
                <class>com.logitags.cibet.core.EventResult</class>
                <class>com.logitags.cibet.resource.ResourceParameter</class>
                <class>com.logitags.cibet.resource.Resource</class>
                <class>com.logitags.cibet.sensor.ejb.EjbResource</class>
                <class>com.logitags.cibet.sensor.http.HttpRequestResource</class>
                <class>com.logitags.cibet.sensor.jdbc.driver.JdbcResource</class>
                <class>com.logitags.cibet.sensor.jpa.JpaResource</class>
                <class>com.logitags.cibet.sensor.jpa.JpaQueryResource</class>
                <class>com.logitags.cibet.sensor.pojo.MethodResource</class>
```

```
            <properties>
              <property name="hibernate.format_sql" value="true" />
              <property name="hibernate.transaction.jta.platform" value=
                        "org.hibernate.service.jta.platform.internal.JBossAppServerJtaPlatform" />
            </properties>
        </persistence-unit>

</persistence>
```

4. Make the Cibet persistence unit available in JNDI. Add to your web.xml:

```
   <persistence-unit-ref>
      <persistence-unit-ref-name>java:comp/env/Cibet</persistence-unit-ref-name>
      <persistence-unit-name>Cibet</persistence-unit-name>
   </persistence-unit-ref>
```

5. The HTTP-FILTER sensor is a servlet filter. Add the CibetFilter to web.xml:

```
        <filter>
                <filter-name>cibetFilter</filter-name>
                <filter-class>com.logitags.cibet.sensor.http.CibetFilter</filter-class>
        </filter>

        <filter-mapping>
                <filter-name>cibetFilter</filter-name>
                <url-pattern>/*</url-pattern>
        </filter-mapping>
```

6. Finally, we need the Cibet configuration. Create a cibet-config.xml file in the classpath, for example in Tutorial2.war!WEB-INF/classes and/or add the following snippet. The setpoint defines that requests to URL http://localhost:8788/Tutorial2/secured are not executed directly but postponed until a second user has released it.

```
<?xml version="1.0" encoding="UTF-8"?>
<cibet xmlns="http://www.logitags.com" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.logitags.com
       http://www.logitags.com/cibet/cibet-config_1.3.xsd">

  <setpoint id="setpoint-4">
    <controls>
      <event>INVOKE</event>
      <target>http://localhost:8788/Tutorial2/secured</target>
    </controls>
    <actuator name="FOUR_EYES"/>
  </setpoint>

</cibet>
```

Effect:
The FOUR_EYES actuator requires that a user is authenticated when executing the controlled business case. Therefore we first login a user. The user is stored in the Cibet context within the http session. When the URL http://localhost:8788/ Tutorial2/secured is called it is not executed but an entry in table cib_controllable is written. In order to display requests which attend release the following method could be applied:

```
List<Controllable> openList =
DcLoader.findUnreleased("http://localhost:8788/Tutorial2/secured");

for (Controllable dc : openList) {
  log.info("created by " + dc.getCreateUser() + " at " + dc.getCreateDate());
  log.info("Class: " + dc.getResource().getTarget());
}
```

Another user can now release the http request with

```
openList.get(0).release(“accepted”);
```

or reject it with

```
openList.get(1).reject(“No!”);
```

4.3Applying Two-Man-Rule Control on a JPA entity

Business Case
Modifications of the Person name is critical and must be checked and released by a second user before it is executed. As an additional requirement it is necessary that both, the initiating and the releasing user are present when the name change is released. The application is secured by Apache Shiro.

Application: Tutorial3.war

Test: com.logitags.cibet.tutorial.Tutorial3.twoManRule1()

Sensor: JPA

Actuator: TWO_MAN_RULE

Solution:

1. Include following dependencies to your web application:

```
        <dependency>
                <groupId>com.logitags</groupId>
                <artifactId>cibet-jpa</artifactId>
                <version>${version}</version>
        </dependency>

        <dependency>
                <groupId>com.logitags</groupId>
                <artifactId>cibet-shiro</artifactId>
                <version>${version}</version>
        </dependency>
```

2. TWO_MAN_RULE actuator requires a database. Create the Cibet database tables from the sql scripts in cibet-core.jar!sql

3. Next we need the persistence units. The Cibet persistence unit is used internally by Cibet to persist the dual control event.
   The APPL-UNIT persistence unit is the application’s PU to persist Person entities. Here we set the JPA sensor: The provider must be the Cibet provider and the original provider is set in property *com.logitags.cibet.persistence.provider*.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation=http://java.sun.com/xml/ns/persistence
        http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd version="2.0">
```

```
        <persistence-unit name="Cibet" transaction-type="JTA">
                <provider>org.hibernate.ejb.HibernatePersistence</provider>
                <jta-data-source>java:/MyDatasource</jta-data-source>
                <class>com.logitags.cibet.actuator.archive.Archive</class>
                <class>com.logitags.cibet.actuator.common.Controllable</class>
                <class>com.logitags.cibet.core.EventResult</class>
                <class>com.logitags.cibet.resource.ResourceParameter</class>
                <class>com.logitags.cibet.resource.Resource</class>
                <class>com.logitags.cibet.sensor.ejb.EjbResource</class>
                <class>com.logitags.cibet.sensor.http.HttpRequestResource</class>
                <class>com.logitags.cibet.sensor.jdbc.driver.JdbcResource</class>
                <class>com.logitags.cibet.sensor.jpa.JpaResource</class>
                <class>com.logitags.cibet.sensor.jpa.JpaQueryResource</class>
                <class>com.logitags.cibet.sensor.pojo.MethodResource</class>

                <properties>
                  <property name="hibernate.format_sql" value="true" />
                  <property name="hibernate.transaction.jta.platform" value=
                            "org.hibernate.service.jta.platform.internal.JBossAppServerJtaPlatform" />
                </properties>
        </persistence-unit>

        <persistence-unit name="APPL-UNIT" transaction-type="JTA">
                <provider>com.logitags.cibet.sensor.jpa.Provider</provider>
                <jta-data-source>java:/MyDatasource </jta-data-source>
                <class>com.logitags.cibet.tutorial.Person</class>
                <class>com.logitags.cibet.tutorial.Address</class>
                <properties>
                  <property name="com.logitags.cibet.persistence.provider" value=
                            "org.hibernate.ejb.HibernatePersistence" />
                  <property name="hibernate.transaction.jta.platform" value=
                            "org.hibernate.service.jta.platform.internal.JBossAppServerJtaPlatform" />
                </properties>
        </persistence-unit>

</persistence>
```

4. Make the Cibet persistence unit available in JNDI. Add to your web.xml:

```
   <persistence-unit-ref>
     <persistence-unit-ref-name>java:comp/env/Cibet</persistence-unit-ref-name>
     <persistence-unit-name>Cibet</persistence-unit-name>
   </persistence-unit-ref>
```

5. Add the Cibet context filter to web.xml. It starts and ends the Cibet context and synchronizes it with the http session.

```
        <filter>
                <filter-name>cibetContextFilter</filter-name>
                <filter-class>com.logitags.cibet.context.CibetContextFilter</filter-class>
        </filter>

        <filter-mapping>
                <filter-name>cibetContextFilter</filter-name>
                <url-pattern>/*</url-pattern>
        </filter-mapping>
```

6. Finally, we need the Cibet configuration. Create a cibet-config.xml file in the classpath, for example in Tutorial3.war!WEB-INF/classes and/or add the following snippet. The setpoint states that EntityManager.merge actions of Person entities are set under dual control in case the name has changed. If other Person attributes are modified the merge is executed directly.

```
<?xml version="1.0" encoding="UTF-8"?>
<cibet xmlns="http://www.logitags.com" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.logitags.com
       http://www.logitags.com/cibet/cibet-config_1.3.xsd">
```

```
  <setpoint id="setpoint-5">
    <controls>
      <event>UPDATE</event>
      <target>com.logitags.cibet.tutorial.Person</target>
      <stateChange>name</stateChange>
    </controls>
    <actuator name="TWO_MAN_RULE"/>
  </setpoint>

</cibet>
```

Effect:
The TWO_MAN_RULE actuator requires that a user is authenticated when executing the controlled business case. Therefore we first login a user. The user is stored in the Cibet context within the http session. When now the state attribute of a Person is modified it is not postponed but when the name of a Person is changed it is not executed but an entry in table cib_controllable is written.
In order to display requests which attend release the following method could be applied:

```
List<Controllable> openList = DcLoader.findUnreleased();

for (Controllable dc : openList) {
  log.info("created by " + dc.getCreateUser() + " at " + dc.getCreateDate());
  log.info("Class: " + dc.getResource().getTarget());
}
```

For releasing the name change, a second user must be authenticated while the first user is still logged in. As the application is secured with Apache Shiro this can be done with the ShiroService provided by Cibet. The second user may enter his credentials in a dialogue and then execute the following code. Note that the EntityManager must be given to the release method because it is a persistence release.

```
UsernamePasswordToken token = new UsernamePasswordToken(user, password);
ShiroService.logonSecondUser(token);
openList.get(0).release(applUnitEntityManager, "accepted");
```

5. Scheduling

5.1Scheduling an EJB call on the server side

Business Case
When clients call the remote EJB SimpleRemoteEjbImpl, this EJB should not be executed directly but with a delay. The scheduling is controlled by the server.

Application: Tutorial4.war

Test: com.logitags.cibet.tutorial.Tutorial4.schedule1()

Sensor: EJB

Actuator: SCHEDULER

Solution:

1. Include following dependency to your web application:

```
        <dependency>
                <groupId>com.logitags</groupId>
                <artifactId>cibet-core</artifactId>
                <version>${version}</version>
        </dependency>
```

2. SCHEDULER actuator requires a database. Create the Cibet database tables from the sql scripts in cibet-core.jar!sql

3. Next we need the Cibet persistence unit:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation=http://java.sun.com/xml/ns/persistence
        http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd version="2.0">

        <persistence-unit name="Cibet" transaction-type="JTA">
                <provider>org.hibernate.ejb.HibernatePersistence</provider>
                <jta-data-source>java:/MyDatasource</jta-data-source>
                <class>com.logitags.cibet.actuator.archive.Archive</class>
                <class>com.logitags.cibet.actuator.common.Controllable</class>
                <class>com.logitags.cibet.core.EventResult</class>
                <class>com.logitags.cibet.resource.ResourceParameter</class>
                <class>com.logitags.cibet.resource.Resource</class>
                <class>com.logitags.cibet.sensor.ejb.EjbResource</class>
                <class>com.logitags.cibet.sensor.http.HttpRequestResource</class>
                <class>com.logitags.cibet.sensor.jdbc.driver.JdbcResource</class>
                <class>com.logitags.cibet.sensor.jpa.JpaResource</class>
                <class>com.logitags.cibet.sensor.jpa.JpaQueryResource</class>
                <class>com.logitags.cibet.sensor.pojo.MethodResource</class>

                <properties>
                  <property name="hibernate.format_sql" value="true" />
                  <property name="hibernate.transaction.jta.platform" value=
                            "org.hibernate.service.jta.platform.internal.JBossAppServerJtaPlatform" />
                </properties>
        </persistence-unit>

</persistence>
```

4. Make the Cibet persistence unit available in JNDI. Add to your web.xml:

```
   <persistence-unit-ref>
      <persistence-unit-ref-name>java:comp/env/Cibet</persistence-unit-ref-name>
      <persistence-unit-name>Cibet</persistence-unit-name>
   </persistence-unit-ref>
```

5. Next add some EJB interceptors either on the EJB class or specific method:

```
        @Remote
        public class SimpleRemoteEjbImpl implements SimpleRemoeEjb {

           @Interceptors({CibetContextInterceptor.class, SchedulerInterceptor.class,
                          CibetInterceptor.class })
           public String writeString(String param) {
              …
           }
        }
```

- The CibetContextInterceptor is for starting and ending the Cibet context. We have no session in this case. Each remote EJB call is independent.
- Therefore we must add another interceptor to set the scheduled date into the Cibet context. The SchedulerInterceptor postpones the EJB call for 3 sec with: Context.requestScope().setScheduledDate(Calendar.SECOND, 3);
- The EJB sensor is in fact an EJB interceptor. Add the CibetInterceptor as a third interceptor.

6. Finally, we need the Cibet configuration. Create a cibet-config.xml file in the classpath, for example in Tutorial4.war!WEB-INF/classes and/or add the following snippet. The SCHEDULER actuator starts a job that checks regularly for scheduled business cases. We tell the actuator to start this job 10 seconds after current time. The setpoint defines that invocations of the writeString(String) method in class SimpleRemoteEjbImpl should be run through the SCHEDULER actuator. The actuator postpones the call only if a scheduled date is set in the Cibet context.

```
<?xml version="1.0" encoding="UTF-8"?>
<cibet xmlns="http://www.logitags.com" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.logitags.com
       http://www.logitags.com/cibet/cibet-config_1.3.xsd">

  <actuator name=”SCHEDULER”>
    <properties>
      <timerStart>+10</timerStart>
    </properties>
  </actuator>

  <setpoint id="setpoint-6">
    <controls>
      <event>INVOKE</event>
      <target>com.logitags.cibet.tutorial.SimpleRemoteEjbImpl</target>
      <method>writeString(String)</method>
    </controls>
    <actuator name="SCHEDULER"/>
  </setpoint>

</cibet>
```

Effect:
The method writeString(String param) logs the param as INFO message and echoes it back as return value. When the EJB is called like

```
String lookupName =
       "Tutorial4/SimpleRemoteEjbImpl!com.logitags.cibet.tutorial.SimpleRemoteEjb";
SimpleRemoteEjb ejb = (SimpleRemoteEjb) initialContext.lookup(lookupName);
String answer = ejb.writeString(“Hello Ejb”);
Assert.assertEquals(null, answer);
```

the method is not executed but scheduled. The scheduler runs some seconds later and executes the method. This can be checked in the log file where the message ‘Hello Ejb’ appears and in table CIB_CONTROLLABLE, where the executionDate is set, the executionstatus set to EXECUTED and the result column contains ‘Hello Ejb’.

5.2Scheduling an EJB call on the client side

Business Case
When an application calls the remote EJB SimpleRemoteEjbImpl, this EJB should not be executed directly but with a delay. The scheduling is controlled in this case by the client application which is in this case a simple Java SE application.

Application: Tutorial4.war

Test: com.logitags.cibet.tutorial.Tutorial4.schedule2()

Sensor: EJB-CLIENT

Actuator: SCHEDULER

Solution:

1. Include following dependency to your client application:

```
        <dependency>
                <groupId>com.logitags</groupId>
                <artifactId>cibet-core</artifactId>
                <version>${version}</version>
        </dependency>
```

2. SCHEDULER actuator requires a database. Create the Cibet database tables from the sql scripts in cibet-core.jar!sql

3. Next we need the persistence unit on the client side. As this is an SE application we must use unit name CibetLocal and transaction-type=RESOURCE_LOCAL. Create a persistence.xml with the following content in the client application (not in Tutorial4.war)

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation=http://java.sun.com/xml/ns/persistence
        http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd version="2.0">

  <persistence-unit name="CibetLocal" transaction-type="RESOURCE_LOCAL">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <class>com.logitags.cibet.actuator.archive.Archive</class>
    <class>com.logitags.cibet.actuator.common.Controllable</class>
    <class>com.logitags.cibet.core.EventResult</class>
    <class>com.logitags.cibet.resource.ResourceParameter</class>
    <class>com.logitags.cibet.resource.Resource</class>
    <class>com.logitags.cibet.sensor.ejb.EjbResource</class>
    <class>com.logitags.cibet.sensor.http.HttpRequestResource</class>
    <class>com.logitags.cibet.sensor.jdbc.driver.JdbcResource</class>
    <class>com.logitags.cibet.sensor.jpa.JpaResource</class>
    <class>com.logitags.cibet.sensor.jpa.JpaQueryResource</class>
    <class>com.logitags.cibet.sensor.pojo.MethodResource</class>

    <properties>
      <property name="javax.persistence.jdbc.dialect"
                value="org.hibernate.dialect.DerbyDialect"/>
      <property name="javax.persistence.jdbc.driver"
                value="org.apache.derby.jdbc.ClientDriver"/>
      <property name="javax.persistence.jdbc.url"
                value="jdbc:derby://localhost:1527/cibettest"/>
      <property name="javax.persistence.jdbc.user" value="APP"/>
      <property name="javax.persistence.jdbc.password" value="x"/>
      <property name="hibernate.format_sql" value="true" />
    </properties>
  </persistence-unit>

</persistence>
```

4. Again we need the Cibet configuration. Create a cibet-config.xml file with the following configuration in the classpath of the client application (not in Tutorial4.war!WEB-INF/classes).
   The SCHEDULER actuator starts a job that checks regularly for scheduled business cases. We tell the actuator to start this job 10 seconds after current time. The setpoint defines that invocations of the writeStringNoIntercept(String) method of interface SimpleRemoteEjb should be run through the SCHEDULER actuator. Note, that the target is the interface here, not the implementation because the client knows only the interface.

```
<?xml version="1.0" encoding="UTF-8"?>
<cibet xmlns="http://www.logitags.com" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.logitags.com
       http://www.logitags.com/cibet/cibet-config_1.3.xsd">

  <actuator name=”CLIENT-SCHEDULER”>
    <class>com.logitags.cibet.actuator.scheduler.SchedulerActuator</class>
    <properties>
      <timerStart>+10</timerStart>
    </properties>
  </actuator>

  <setpoint id="setpoint-7">
    <controls>
      <event>INVOKE</event>
      <target>com.logitags.cibet.tutorial.SimpleRemoteEjb</target>
      <method>writeStringNoIntercept(String)</method>
    </controls>
    <actuator name="CLIENT-SCHEDULER"/>
  </setpoint>

</cibet>
```

5. Before calling the EJB, the context must be started and the scheduled date and a user must be set into the Cibet context. The context can be started either by annotating the client method with @CibetContext annotation or by code. If started by code don’t forget to end it with Context.end().

```
Context.start();
Context.requestScope().setScheduledDate(Calendar.SECOND, 3);
Context.sessionScope().setUser("Pittiplatsch");
```

6. The EJB-CLIENT sensor is applied by using the CibetRemoteContextFactory in the InitialContext. Call the EJB with the EJB-CLIENT sensor with the following InitialContext:

```
Properties properties = new Properties();
properties.put(javax.naming.Context.INITIAL_CONTEXT_FACTORY,
               CibetRemoteContextFactory.class.getName());
properties.put(CibetRemoteContext.NATIVE_INITIAL_CONTEXT_FACTORY,
               "org.jboss.naming.remote.client.InitialContextFactory");
properties.setProperty(javax.naming.Context.PROVIDER_URL, "remote://localhost:4447");
properties.setProperty("jboss.naming.client.ejb.context", "true");
Context initialContext = new InitialContext(properties);
```

Effect:
The method writeStringNoIntercept(String param) logs the param as INFO message and echoes it back as return value. When the EJB is called like

```
Context.start();
Context.requestScope().setScheduledDate(Calendar.SECOND, 3);
Context.sessionScope().setUser("Pittiplatsch");

String lookupName =
       "Tutorial4/SimpleRemoteEjbImpl!com.logitags.cibet.tutorial.SimpleRemoteEjb";
SimpleRemoteEjb ejb = (SimpleRemoteEjb) initialContext.lookup(lookupName);
String answer = ejb.writeStringNoIntercept("Hello Ejb scheduled by client");
Assert.assertEquals(null, answer);
Context.end();
```

the method call is not executed but scheduled on the client side. The client scheduler runs some seconds later and executes the method. This can be checked in the server log file where the message 'Hello Ejb' appears and in table CIB_CONTROLLABLE, where the executionDate is set, the executionstatus set to EXECUTED and the result column contains 'Hello Ejb scheduled by client'. Please note that the Cibet context must be started and ended on the client side

5.3Scheduling a RESOURCE_LOCAL Insert of a JPA Entity

Business Case:
We have a business entity Person which has a 1:n relationship to entity Address. A Servlet is responsible for persisting Persons. The Insert of a Person should not be done instantly but scheduled for a later time. The application is using non JTA datasources.

Application: Tutorial5.war

Test: com.logitags.cibet.tutorial.Tutorial5.schedule3()

Sensor: JPA

Actuator: SCHEDULER

Solution:

1. Include following dependency to your web application:

```
<dependency>
       <groupId>com.logitags</groupId>
       <artifactId>cibet-jpa</artifactId>
       <version>${version}</version>
</dependency>
```

2. SCHEDULER actuator requires a database. Create the Cibet database tables from the sql scripts in cibet-core.jar!sql

3. Next we need some persistence units. This time we don't use JTA. The Cibet persistence unit must be named CibetLocal instead of Cibet and transaction type must be RESOURCE_LOCAL:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation=http://java.sun.com/xml/ns/persistence
        http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd version="2.0">

        <persistence-unit name="CibetLocal" transaction-type="RESOURCE_LOCAL">
                <provider>org.hibernate.ejb.HibernatePersistence</provider>
                <non-jta-data-source>java:/MyDatasource</non-jta-data-source>
                <class>com.logitags.cibet.actuator.archive.Archive</class>
                <class>com.logitags.cibet.actuator.common. Controllable</class>
                <class>com.logitags.cibet.core.EventResult</class>
                <class>com.logitags.cibet.resource.ResourceParameter</class>
                <class>com.logitags.cibet.resource.Resource</class>
                <class>com.logitags.cibet.sensor.ejb.EjbResource</class>
                <class>com.logitags.cibet.sensor.http.HttpRequestResource</class>
                <class>com.logitags.cibet.sensor.jdbc.driver.JdbcResource</class>
                <class>com.logitags.cibet.sensor.jpa.JpaResource</class>
                <class>com.logitags.cibet.sensor.jpa.JpaQueryResource</class>
                <class>com.logitags.cibet.sensor.pojo.MethodResource</class>

                <properties>
                  <property name="hibernate.format_sql" value="true" />
                  <property name="hibernate.transaction.jta.platform" value=
                            "org.hibernate.service.jta.platform.internal.JBossAppServerJtaPlatform" />
                </properties>
        </persistence-unit>

        <persistence-unit name="APPL-UNIT" transaction-type="RESOURCE_LOCAL">
                <provider>com.logitags.cibet.sensor.jpa.Provider</provider>
                <non-jta-data-source>java:/MyDatasource </non-jta-data-source>
                <class>com.logitags.cibet.tutorial.Person</class>
                <class>com.logitags.cibet.tutorial.Address</class>
                <properties>
                  <property name="com.logitags.cibet.persistence.provider" value=
                            "org.hibernate.ejb.HibernatePersistence" />
                  <property name="hibernate.transaction.jta.platform" value=
                            "org.hibernate.service.jta.platform.internal.JBossAppServerJtaPlatform" />
                </properties>
        </persistence-unit>

</persistence>
```

The CibetLocal persistence unit is used internally by Cibet to persist its entities.

The APPL-UNIT persistence unit is the application's PU to persist entities Person and Address. We apply the JPA sensor here by using the Cibet provider. The original provider is set in property *com.logitags.cibet.persistence.provider*.

4. Finally, we need the Cibet configuration. Create a cibet-config.xml file in the classpath, for example in Tutorial5.war!WEB-INF/classes and/or add the following snippet. The SCHEDULER actuator starts a job that checks regularly for scheduled business cases. We define a new scheduler actuator that starts this job 10 seconds after current time. We must tell the actuator to use the APPL-UNIT persistence unit when executing scheduled persistence operations.
   The setpoint defines that persisting of Person and Address entities should be postponed in case a scheduled date is set in the Cibet context. As we use the same cibet-config.xml for all Tutorials we add an invoker control to schedule only if a Person is persisted within method TutorialServlet5.persist():

```
<?xml version="1.0" encoding="UTF-8"?>
<cibet xmlns="http://www.logitags.com" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.logitags.com
       http://www.logitags.com/cibet/cibet-config_1.3.xsd">

  <actuator name=”LOCAL-SCHEDULER”>
```

```
    <class>com.logitags.cibet.actuator.scheduler.SchedulerActuator</class>
    <properties>
      <timerStart>+10</timerStart>
      <persistenceUnit>APPL-UNIT</persistenceUnit>
    </properties>
  </actuator>

  <setpoint id="setpoint-8">
    <controls>
      <event>INSERT </event>
      <target>com.logitags.cibet.tutorial.Person,
              com.logitags.cibet.tutorial.Address</target>
      <invoker>com.logitags.cibet.tutorial.TutorialServlet5.persist()</invoker>
    </controls>
    <actuator name="LOCAL-SCHEDULER"/>
  </setpoint>

</cibet>
```

5. As we don't use the CibetContextFilter in this example we have to start the Cibet context, set the scheduled date and a user before persisting the Person in TutorialServlet5. This time we start the context with the @CibetContext annotation. Note that the method must be public:

```
@CibetContext
public void persist() {
  Context.requestScope().setScheduledDate(Calendar.SECOND, 3);
  Context.requestScope().setRemark(«This is scheduled»);
  Context.sessionScope().setUser("Mausi");
```

Effect:

When a Person entity is persisted like

```
EntityManager em = Persistence.createEntityManagerFactory("APPL-UNIT").createEntityManager();
em.getTransaction().begin();
em.persist(person);
em.getTransaction().commit();
```

the person is not persisted instantly but a few seconds later. This can be checked in the database and in the log files of the server and the test class.

6. Security

6.1 Setting Spring Security permissions on a four-eyes controlled business case

Business Case:
We have a 1:n relationship between entities Person and Address. Persisting new Person entities should be controlled by four eyes dual control. The persisting user must have permission to persist Persons and the releasing user must have permission to release. The application is secured by Spring Security.

Application: Tutorial1.war

Test: com.logitags.cibet.tutorial.Tutorial1.spring1()

Sensor: JPA

Actuator: SPRINGSECURITY, FOUR_EYES

Solution:

1. Include following dependencies to your web application:

```
        <dependency>
                <groupId>com.logitags</groupId>
                <artifactId>cibet-jpa</artifactId>
                <version>${version}</version>
        </dependency>

        <dependency>
                <groupId>com.logitags</groupId>
                <artifactId>cibet-springsecurity</artifactId>
                <version>${version}</version>
        </dependency>
```

2. The SPRINGSECURITY actuator doesn't need a database but as we are also applying FOUR_EYES actuator a database is required. Create the Cibet database tables from the sql scripts in cibet-core.jar!sql

3. Next we need some persistence units:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation=http://java.sun.com/xml/ns/persistence
        http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd version="2.0">

        <persistence-unit name="Cibet" transaction-type="JTA">
                <provider>org.hibernate.ejb.HibernatePersistence</provider>
                <jta-data-source>java:/MyDatasource</jta-data-source>
                <class>com.logitags.cibet.actuator.archive.Archive</class>
                <class>com.logitags.cibet.actuator.common.Controllable</class>
                <class>com.logitags.cibet.core.EventResult</class>
                <class>com.logitags.cibet.resource.ResourceParameter</class>
                <class>com.logitags.cibet.resource.Resource</class>
                <class>com.logitags.cibet.sensor.ejb.EjbResource</class>
                <class>com.logitags.cibet.sensor.http.HttpRequestResource</class>
                <class>com.logitags.cibet.sensor.jdbc.driver.JdbcResource</class>
                <class>com.logitags.cibet.sensor.jpa.JpaResource</class>
                <class>com.logitags.cibet.sensor.jpa.JpaQueryResource</class>
                <class>com.logitags.cibet.sensor.pojo.MethodResource</class>

                <properties>
                  <property name="hibernate.format_sql" value="true" />
                  <property name="hibernate.transaction.jta.platform" value=
                        "org.hibernate.service.jta.platform.internal.JBossAppServerJtaPlatform" />
                </properties>
        </persistence-unit>

        <persistence-unit name="APPL-UNIT" transaction-type="JTA">
                <provider>com.logitags.cibet.sensor.jpa.Provider</provider>
                <jta-data-source>java:/MyDatasource </jta-data-source>
                <class>com.logitags.cibet.tutorial.Person</class>
                <class>com.logitags.cibet.tutorial.Address</class>
                <properties>
                  <property name="com.logitags.cibet.persistence.provider" value=
                        "org.hibernate.ejb.HibernatePersistence" />
                  <property name="hibernate.transaction.jta.platform" value=
                        "org.hibernate.service.jta.platform.internal.JBossAppServerJtaPlatform" />
                </properties>
        </persistence-unit>
```

```
</persistence>
```

The Cibet persistence unit is used internally by Cibet to persist its entities.

The APPL-UNIT persistence unit is the application's PU to persist entities Person and Address. Here we set the JPA sensor: The provider must be the Cibet provider and the original provider is set in property *com.logitags.cibet.persistence.provider*.

4. Make the persistence unit available in JNDI. Add to your web.xml:

```
   <persistence-unit-ref>
      <persistence-unit-ref-name>java:comp/env/Cibet</persistence-unit-ref-name>
      <persistence-unit-name>Cibet</persistence-unit-name>
   </persistence-unit-ref>
```

5. Add the Cibet context filter to web.xml. It starts and ends the Cibet context and synchronizes it with the http session. Add also the necessary Spring configuration in web.xml, Spring config file location, Spring filter and ContextLoaderListener.

```
        <filter>
                <filter-name>cibetContextFilter</filter-name>
                <filter-class>com.logitags.cibet.context.CibetContextFilter</filter-class>
        </filter>

        <filter-mapping>
                <filter-name>cibetContextFilter</filter-name>
                <url-pattern>/*</url-pattern>
        </filter-mapping>
```

6. In your Spring configuration file add

```
<context:component-scan base-package="com.logitags.cibet"/>

<aop:config proxy-target-class="true"/>

<sec:http auto-config='true' use-expressions="false"/>

<sec:global-method-security
   jsr250-annotations="enabled"
   secured-annotations="enabled"
   pre-post-annotations="enabled" />
```

7. Finally, we need the Cibet configuration. Create a cibet-config.xml file in the classpath, for example in Tutorial1.war!WEB-INF/classes and/or add the following snippet. We must define a SpringSecurity actuator for the persist and one for the release. The first one defines that when applied the user must have role persister while the second one is defined with role releaser. Now we define two setpoints. Setpoint-9 defines that inserting a Person should be set under four eyes control and that the persisting user must have role persister. Note that we add a tenant control. That prevents that setpoint-9 is applied in Tutorial1.archve1 (chapter 3.1). The second setpoint defines that users who release postponed inserts of Person entities must have role releaser. Setpoint-10 extends from setpoint-9 and inherits the controls except <event> which is overridden.

```
<?xml version="1.0" encoding="UTF-8"?>
<cibet xmlns="http://www.logitags.com" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.logitags.com
       http://www.logitags.com/cibet/cibet-config_1.3.xsd">

  <actuator name="SPRINGSECURITY_PERSIST">
    <class>com.logitags.cibet.actuator.springsecurity.SpringSecurityActuator</class>
    <properties>
      <preAuthorize>hasRole("persister")</preAuthorize>
    </properties>
  </actuator>

  <actuator name="SPRINGSECURITY_RELEASE">
    <class>com.logitags.cibet.actuator.springsecurity.SpringSecurityActuator</class>
    <properties>
      <preAuthorize>hasRole("releaser")</preAuthorize>
    </properties>
  </actuator>

  <setpoint id="setpoint-9">
    <controls>
      <event>INSERT</event>
      <target>com.logitags.cibet.tutorial.Person</target>
      <tenant>SpringTest</tenant>
    </controls>
    <actuator name="FOUR_EYES"/>
    <actuator name="SPRINGSECURITY_PERSIST"/>
  </setpoint>

  <setpoint id="setpoint-10" extends="setpoint-9">
    <controls>
      <event>RELEASE</event>
    </controls>
    <actuator name="SPRINGSECURITY_RELEASE"/>
  </setpoint>

</cibet>
```

Effect:
When a Person object is inserted a user must be logged in who has the role ‘persister’. If so, the persisting action is postponed for a dual control release. Please note that an Archive entry is persisted because in the Tutorial1 application the setpoint-1 applies as well.
Now a second user must log in who must be different to the persisting user and must have the role ‘releaser’. He can now release the business case and the Person is inserted into the database.

7. Load Control

7.1Applying Shed Load Control on a URL

Business Case:

A URL must be secured against overloading. We want to restrict the number of parallel executions. The process behind the URL is also vulnerable to memory overloading. Therefore we raise an alarm if memory exceeds a threshold.

Application: Tutorial6.war

Test: com.logitags.cibet.tutorial.Tutorial6.loadcontrol1()

Sensor: HTTP-FILTER

Actuator: LOADCONTROL

Solution:

1. Include following dependency to your web application:

```
<dependency>
	<groupId>com.logitags</groupId>
	<artifactId>cibet-core</artifactId>
	<version>${version}</version>
</dependency>
```

2. The HTTP-FILTER sensor is a servlet filter. Add the CibetFilter to web.xml:

```
<filter>
	<filter-name>cibetFilter</filter-name>
	<filter-class>com.logitags.cibet.sensor.http.CibetFilter</filter-class>
</filter>

<filter-mapping>
	<filter-name>cibetFilter</filter-name>
	<url-pattern>/*</url-pattern>
</filter-mapping>
```

3. Finally, we need the Cibet configuration. Create a cibet-config.xml file in the classpath, for example in Tutorial6.war!WEB-INF/classes and/or add the following snippet. The LOADCONTROL actuator sets the maximum allowed threads to 9 and raises an alarm when memory usage exceeds 85% and memory collection usage exceeds 60% of the available memory. All notifications are sent to LoggingCallback which logs the information to logger LOADCONTROL_ALARM.

```
<?xml version="1.0" encoding="UTF-8"?>
<cibet xmlns="http://www.logitags.com" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.logitags.com  http://www.logitags.com/cibet/cibet-config_1.3.xsd">

   <actuator name="LOADCONTROL">
      <properties>
         <loadControlCallback>com.logitags.jmeter.LoggingCallback</loadControlCallback>
         <threadCountMonitor.status>ON</threadCountMonitor.status>
         <threadCountMonitor.shedThreshold>9</threadCountMonitor.shedThreshold>

         <memoryMonitor.status>ON</memoryMonitor.status>
         <memoryMonitor.collectionUsageAlarmThreshold>60%</memoryMonitor.collectionUsageAlarmThreshold>
         <memoryMonitor.usageAlarmThreshold>85%</memoryMonitor.usageAlarmThreshold>
      </properties>
   </actuator>

   <setpoint id="SP1-servletFilter">
      <controls>
         <target>http://localhost:8788/*</target>
      </controls>
      <actuator name="LOADCONTROL"/>
   </setpoint>

</cibet>
```

Effect:

This tutorial requires Apache JMeter 2.13. Point constant JMETER_HOME in Tutorial6.java to the installation directory.

When the Tutorial6 test is started JMeter fires for a few minutes http requests. The effect of the LOADCONTROL configuration can be seen for example with jconsole in MBean LoadControlActuator. When the log ouput of logger LOADCONTROL_ALARM is directed to the console or into a file the alarm notification messages can be seen.

8. Locking

8.1Locking a JPA Entity and a Method Call

Business Case:
A long running batch processes Person entities. It is mandatory that nobody selects Person objects during the batch run because state is not guaranteed. It is also not allowed to persist new Person objects.

That means we must define two setpoints for applying the LOCKER actuator, one for controlling SELECT statement of Persons and one for controlling invocation of the persist() method.

Application: Tutorial1.war

Test: com.logitags.cibet.tutorial.Tutorial1.locker1()

Sensor: JPA, ASPECT

Actuator: LOCKER

Solution:

4. Include following dependency to your web application:

```
        <dependency>
                <groupId>com.logitags</groupId>
                <artifactId>cibet-jpa</artifactId>
                <version>${version}</version>
        </dependency>
```

5. LOCKER actuator requires a database. Create the Cibet database tables from the sql scripts in cibet-core.jar!sql

6. Next we need some persistence units:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation=http://java.sun.com/xml/ns/persistence
        http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd version="2.0">

        <persistence-unit name="Cibet" transaction-type="JTA">
                <provider>org.hibernate.ejb.HibernatePersistence</provider>
                <jta-data-source>java:/MyDatasource</jta-data-source>
                <class>com.logitags.cibet.actuator.archive.Archive</class>
                <class>com.logitags.cibet.actuator.common. Controllable</class>
                <class>com.logitags.cibet.core.EventResult</class>
                <class>com.logitags.cibet.resource.ResourceParameter</class>
                <class>com.logitags.cibet.resource.Resource</class>
                <class>com.logitags.cibet.sensor.ejb.EjbResource</class>
```

```
                <class>com.logitags.cibet.sensor.http.HttpRequestResource</class>
                <class>com.logitags.cibet.sensor.jdbc.driver.JdbcResource</class>
                <class>com.logitags.cibet.sensor.jpa.JpaResource</class>
                <class>com.logitags.cibet.sensor.jpa.JpaQueryResource</class>
                <class>com.logitags.cibet.sensor.pojo.MethodResource</class>

                <properties>
                  <property name="hibernate.format_sql" value="true" />
                  <property name="hibernate.transaction.jta.platform" value=
                            "org.hibernate.service.jta.platform.internal.JBossAppServerJtaPlatform" />
                </properties>
        </persistence-unit>

        <persistence-unit name="APPL-UNIT" transaction-type="JTA">
                <provider>com.logitags.cibet.sensor.jpa.Provider</provider>
                <jta-data-source>java:/MyDatasource </jta-data-source>
                <class>com.logitags.cibet.tutorial.Person</class>
                <class>com.logitags.cibet.tutorial.Address</class>
                <properties>
                  <property name="com.logitags.cibet.persistence.provider" value=
                            "org.hibernate.ejb.HibernatePersistence" />
                  <property name="hibernate.transaction.jta.platform" value=
                            "org.hibernate.service.jta.platform.internal.JBossAppServerJtaPlatform" />
                </properties>
        </persistence-unit>

</persistence>
```

The Cibet persistence unit is used by Cibet to persist the lock information in table CIB_CONTROLLABLE.

The APPL-UNIT persistence unit is the application's PU to persist entities Person and Address. Please note that we configured the JPA sensor: the provider is the Cibet provider and the original provider is set in property *com.logitags.cibet.persistence.provider*.

7. Make the Cibet persistence unit available in JNDI. Add to your web.xml:

```
   <persistence-unit-ref>
     <persistence-unit-ref-name>java:comp/env/Cibet</persistence-unit-ref-name>
     <persistence-unit-name>Cibet</persistence-unit-name>
   </persistence-unit-ref>
```

8. Add the Cibet context filter to web.xml. It is necessary to start and end the Cibet context and to synchronize the Cibet context with the http session:

```
        <filter>
                <filter-name>cibetContextFilter</filter-name>
                <filter-class>com.logitags.cibet.context.CibetContextFilter</filter-class>
        </filter>

        <filter-mapping>
                <filter-name>cibetContextFilter</filter-name>
                <url-pattern>/*</url-pattern>
        </filter-mapping>
```

9. For controlling the method call we need the ASPECT sensor: Add @CibetIntercept annotation to method TutorialServlet1.persist(). The method must be public. Alternatively, create a file aop.xml in Tutorial1.war!WEB-INF/classes with the following content:

```
<aspectj>
  <aspects>
    <concrete-aspect name="com.myApp.persist.aspect"
                     extends="com.logitags.cibet.sensor.pojo.CustomAspect">
```

```
      <pointcut name="cibetIntercept" expression="target(com.logitags.cibet.tutorial.TutorialServlet1)
                &amp;amp;&amp;amp; execution(public void persist(*))"/>
    </concrete-aspect>
  </aspects>
</aspectj>
```

10. Finally, we need the Cibet configuration. Create a cibet-config.xml file in the classpath, for example in Tutorial1.war!WEB-INF/classes and/or add the following snippet. We configure the Locker actuator to throw an exception when the business case is locked. Next we define two setpoints. Setpoint-11 prevents that EntityManager.find() methods on Person entities could be called by other users and setpoint-12 defines that TutorialServlet1.persist() method could be called by other users. We set the tenant control to prevent interfering with the other tutorial examples.

```
<?xml version="1.0" encoding="UTF-8"?>
<cibet xmlns="http://www.logitags.com" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.logitags.com
       http://www.logitags.com/cibet/cibet-config_1.3.xsd">

  <actuator name="LOCKER">
    <properties>
      <throwDeniedException>true</throwDeniedException>
    </properties>
  </actuator>

  <setpoint id="setpoint-11">
    <controls>
      <event>SELECT</event>
      <target>com.logitags.cibet.tutorial.Person</target>
      <tenant>LockTest</tenant>
    </controls>
    <actuator name="LOCKER"/>
  </setpoint>

  <setpoint id="setpoint-12">
    <controls>
      <event>INVOKE</event>
      <target>com.logitags.cibet.tutorial.TutorialServlet1</target>
      <method>persist</method>
      <tenant>LockTest</tenant>
    </controls>
    <actuator name="LOCKER"/>
  </setpoint>

</cibet>
```

11. The aspect must be weaved into the classes. This can be done at runtime. Download aspectjweaver and start the application (server) with
    set "JAVA_OPTS=%JAVA_OPTS% -javaagent:path-to\aspectjweaver.jar "

Effect:

When the batch starts (in method Tutorial1.locker1() by calling the batch URL) a user must be set into the context before we lock SELECT event for Person entities and the invoking of Tutorial1.persist() method. The lock is set at the beginning of the batch process:

```
ut.begin();
LockedObject lock1 = Locker.lock(Person.class, ControlEvent.SELECT, null);
LockedObject lock2 = Locker.lock(this.getClass(), "persist", ControlEvent.INVOKE, null);
ut.commit();
```

Please note also that calling Locker.lock() methods must be done within a transaction.

When now a user logs in and tries to execute an entityManager.find() on Persons or tries to call the persist() method a DeniedEjbException is thrown.

Normally, the locked objects would be unlocked at the end of the batch but for the purpose of demonstration we unlock them later in a separate call:

```
ut.begin();
LockedObject lock1 = Context.requestScope().getEntityManager().find(LockedObject.class, lock1Id);
Locker.unlock(lock1, null);

LockedObject lock2 = Context.requestScope().getEntityManager().find(LockedObject.class, lock2Id);
Locker.unlock(lock2, null);
ut.commit();
```

9. Nested Controls

9.1Tracker

Business Case:
A servlet calls an EJB which persists an entity. All Cibet controls must be tracked. The EJB call is secured by SHIRO and the insert of the entity must be archived.

Application: Tutorial7.war

Test: com.logitags.cibet.tutorial.Tutorial7.nested1()

Sensor: HTTP-FILTER, EJB, JPA

Actuator: TRACKER. INFO, SHIRO, ARCHIVE

Solution:
TODO