Transactional Web Services Programmer's API

Document version: 0.2.1 September 03, 2007
BA Framework version: 0.2.1 September 03, 2007

Maciej Machulak
m.p.machulak@ncl.ac.uk / mmachulak@redhat.com
University of Newcastle upon Tyne
JBoss, a division of Red Hat

Table of Contents

Disclaimer

Business Activity framework is an on-going research project and is still under heavy development. The current version of framework is intended for tests only and should not be used in production environments. Information about future releases, changelog, discovered bugs, new functionality and all other information directly related to the framework can be found on the framework's website. Questions and remarks concerning the framework can be sent directly to the framework's author or posted to the JBoss Transactions Developer Forum.

1 Introduction

This document describes the programmer's API of the Business Activity Framework. It provides information about metadata that can be used on Web Services so that they can participate in Business Activities. It also describes operations that the programmer may use to increase the flexibility of data management in business applications. Last section of this document presents two examples of Web Services that make use of designed set of metadata and additional operations to participate in Business Activities.

1.1 Acknowledgements

Jonathan Halliday (jhalliday@redhat.com) has provided a valuable technical input into development of the Business Activity Framework.

1.2 Documentation

All documentation directly related to the Business Activity framework can be found on the website of this framework [1]. More information concerning transactional Web Services can be found in [3] and [4]. Questions, requests and comments are more than welcome and can be directed to the author of the framework or posted to the JBoss Transactions Developer Forum [8].

2 Business Activity framework

At this moment writing transaction-aware Web Services with accordance to the Business Activity model [4] is relatively problematic and requires a lot of support from business programmers. Developers need to manually wire up original services with their compensation actions and must ensure the correctness of those links (e.g. in terms of data that needs to be shared). Moreover, it is necessary to follow transaction-related patterns where a developer must code not only the business logic but the *participant* as well. The latter one is an entity, which is capable of responding to transaction protocol messages (such as complete, compensate or close messages). Typically, a third component is also developed. This component usually links participants with the business logic and is often referred to as the *manager*.

The proposed annotation-based Java framework significantly facilitates development of transaction-aware Web Services and releases programmers from mixing transaction-related code with business logic of their applications. Developers do not need to write custom participants and managers and do not have to concern themselves with any of the low level details of transaction processing mechanisms. They may use simple annotations to enrich Web Services with support for participation in Business Activities. With a few declarative statements, users may configure the relationship between units of work and their completion or compensation tasks. The framework manages all necessary aspects of the execution of those tasks, ensuring a reliable and consistent transaction outcome.

Annotations have intuitive and reasonable default values to release the developer from the necessity of specifying all required annotation member values explicitly. Those annotations are sufficient in most scenarios, where Web Services need to be exposed as Business Activity tasks. To extend capabilities

of the framework and cover all needs of business developers, the framework additionally provides a fully flexible data management mechanism for reliably storing arbitrary data. Such data can be stored during original Web Service invocation and is transparently made available during completion or compensation actions. It can be used to either correctly complete the work or restore the prior state. Such flexible data management is provided with the use of annotations and additional operations.

2.1 Programmer's API

The API of the Business Activity framework consists of metadata (Java annotations) and two additional operations. The annotation-based part of the API has been designed following the conventions proposed by the JSR-181 specification [5] for exposing EJB components as Web Services. It consists of the following annotations:

- `@BAService` – marks a class as the one that contains method exposed as Business Activity tasks; specifies common completion-related and compensation-related information for a group of services (class);
- `@BAMethod` – specifies the service in terms of its state management and the agreement protocol the service wants to participate in;
- `@BACompletedBy` – specifies the completion action and the type of completion for a single service (method);
- `@BACompensatedBy` – specifies the compensation action and the type of compensation for a single service (method);
- `@BAParam` – annotates the service's parameter so that it can be processed by the compensation mechanism and used when executing compensation action;
- `@BAResult` – annotates the service's return value to be processed by the compensation mechanism and used when executing compensation action;
- `@BADataManagement` – annotates the data manager object to enable transparent dependency injection.

Apart from the annotations, the programmer may use `put(id,Object)` and `get(id)` operations, which are invoked on the previously mentioned data manager. Those operations enable storing and retrieving any data, which might be used for either completion or compensation.

A comprehensive description of the Business Activity Framework API is presented in sections 3 and 4 of this document.

To expose a Web Service method as a transaction-aware Web Service, the business programmer must mark it with previously mentioned annotations as shown in the example in section 5. Typically using only annotations is enough. The framework will be able to transparently apply all necessary transaction mechanisms according to the requirements specified by the business programmer. If more flexibility in data management needs to be achieved, the programmer may use additional methods (`put()` and `get()`) to store and retrieve any data that could be potentially used during either completion or compensation. The completion and compensation actions can only have their parameters annotated properly so that middleware mechanisms can match them with any data, which has been stored during the execution of the original service. Additionally, support for numerical identifiers is provided so that completion or compensation actions do not need to have any annotations

specified. Such feature is useful when using third-party services as completion or compensation actions.

2.2 Design

Description of the framework's design and implementation will been presented in the Business Activity Framework Maintenance Guide [2].

2.3 Implementation

Business Activity framework has been built on top of the XTS (XML Transaction Service) component, which is a part of the JBoss Transaction Service [6]. It uses JBoss AOP framework [7] to transparently apply transactional middleware mechanisms into Web Services. Implementation specific parts have been well separated and JBoss AOP can be easily exchanged with any other framework which is capable of intercepting calls to components exposed as Web Services.

3 Transactional Web Services Metadata

The core API of the Business Activity framework is based on metadata – Java annotations. To enrich Web Services with support for their participation in Business Activities, those services can be simply annotated. No additional code has to be written. Most annotations have intuitive and reasonable default values to release the developer from the necessity of specifying all required annotation member values explicitly. Next sections of this document present designed annotations in more details.

3.1 Annotation: @BAService

Qualified name: `org.jboss.txbridge.ba.annotation.BAService`

Annotation type: **optional**

Target element: **class**

3.2.1 Description

This annotation marks a class, which has methods that are exposed as transaction-aware Web Services. It specifies common completion-related and compensation-related information that will be used by all those methods. Single methods may override those values using members of the `@BACompletedBy` and `@BACompensatedBy` annotation.

Member-value	Meaning	Default
serviceClass	Class that contains the method specified as the completion or compensation task.	Empty.
ejbInterface	Interface of the EJB that contains the completion or compensation method.	Empty.

jndiName	JNDI name of the EJB that contains the completion or compensation method.	Empty string.
providerURL	URL of the JNDI service at which the EJB should be looked up.	Empty string.
wsdl *(*Test implementation*)*	URL of the WSDL, which describes the completion or compensation action.	Empty string.
namespace *(*Test implementation*)*	Target namespace used by the completion or compensation service.	Empty string.
serviceName *(*Test implementation*)*	Name of the completion or compensation service.	Empty string.

3.2.2 Definitions

Annotation definition:

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface BAService
{
    public Class[] serviceClass() default {};
    public Class[] ejbInterface() default {};
    public String jndiName() default "";
    public String providerURL() default "";
    public String wsdl() default "";
    public String namespace() default "";
    public String serviceName() default "";
}
```

3.2.3 Example

Java source code:

```
@WebService(name="Hotel")
@BAService(serviceClass=HotelBAImpl.class,
       ejbInterface=HotelBA.class,
       jndiName="bademo/HotelBAImpl/remote")
public class HotelImpl implements Hotel
{
    // ...
}
```

3.2 Annotation: @BAMethod

Qualified name: `org.jboss.txbridge.ba.annotation.BAMethod`

Annotation type: **required**

Target element: **method**

3.2.1 Description

This annotation is used to mark a method that is exposed as a Web Service so that it can be recognised as the one that needs to participate in a Business Activity. It specifies the agreement protocol the service is willing to participate in. This agreement protocol is the equivalent of a commit protocol and enables different services to reach a consensus in a certain business activity. The WS-BusinessActivity specification [4] defines 2 protocols – Business Agreement with Participant Completion and Business Agreement with Coordinator Completion. This annotation also defines the service in terms of its state management (whether it changes any data or is read-only). If the service is read-only then it does not require compensation action to be specified – the participant associated with such service simply exits the Business Activity after it is able to inform the coordinator that all the work has been completed. If the service modifies any data (or simply provides information which is transaction-dependent) then it has to have a compensation action specified.

Member-value	Meaning	Default
agreement	Specifies the agreement protocol the service wants to participate in. This member value takes elements of the AgreementType enum.	AgreementType.PARTICIPANT_COMPLETION
type	Specifies the service in terms of its state management. This member value takes elements of the MethodType enum.	MethodType.MODIFY

3.2.2 Definitions

Annotation definition:

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface BAMethod
{
    public AgreementType agreement() default AgreementType.PARTICIPANT_COMPLETION;
    public MethodType type() default MethodType.MODIFY;
}
```

`AgreementType` definition:

```
public enum AgreementType
{
    PARTICIPANT_COMPLETION,
    COORDINATOR_COMPLETION
}
```

`MethodType` definition:

```
public enum MethodType
{
    READONLY,
    MODIFY
}
```

3.2.3 Example

Java source code:

```
@WebMethod
@BAMethod(AgreementType.PARTICIPANT_COMPLETION,MethodType.READONLY)
public String getHotelInfo()
{
    // ...
}
```

3.3 Annotation: @BACompletedBy

Qualified name: `org.jboss.txbridge.ba.annotation.BACompletedBy`

Annotation type: **optional**

Target element: **method**

3.3.1 Description

This annotation provides necessary information about the completion action. Member values of this annotation override those specified by the `@BAService` annotation.

Member-value	Meaning	Default
value	Name of the method that is used as a completion action.	None
serviceClass	Class that contains the method specified as the completion task.	Empty.

ejbInterface	Interface of the EJB that contains the completion method.	Empty.
jndiName	JNDI name of the EJB that contains the completion method.	Empty string.
providerURL	URL of the JNDI service at which the EJB that contains the completion method should be looked up.	Empty string.
wsdl *(*Test implementation*)*	URL of the WSDL, which describes the completion action.	Empty string.
namespace *(*Test implementation*)*	Target namespace used by the completion service.	Empty string.
serviceName *(*Test implementation*)*	Name of the completion service.	Empty string.
mode	Mode of the completion – whether it is done locally, on an EJB or is remote. This member value takes elements of ExecutionMode enum.	ExecutionMode.EJB
match	Required type of parameters match for the completion action. By default original method must provide enough data to fill in argument list for the compensation action.	ParameterMatch.STRICT
single	If true than completion action for this method will be invoked only once for each transaction (using data provided by the last execution of this method).	false
order	*NOT YET IMPLEMENTED*	ExecutionOrder.NORMAL

type	Type of the data match. Specifies what data needs to be remembered by the data management mechanism. This member takes elements of the DataMatch enum.	DataMatch.RETURN_VALUE

3.2.2 Definitions

Annotation definition:

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface BACompletedBy
{
    public String value();
    public Class[] serviceClass() default {};
    public Class[] ejbInterface() default {};
    public String jndiName() default "";
    public String providerURL() default "";
    public String wsdl() default "";
    public String namespace() default "";
    public String serviceName() default "";
    public boolean single() default false;
    public ParameterMatch match() default ParameterMatch.STRICT;
    public ExecutionOrder order() default ExecutionOrder.NORMAL;
    public DataMatch type() default DataMatch.RETURN_VALUE;
    public ExecutionMode mode() default ExecutionMode.EJB;
}
```

`ExecutionMode` definition:

```
public enum ExecutionMode
{
    EJB,
    POJO,
    DII
}
```

`ParameterMatch` definition:

```
public enum ParameterMatch
{
    STRICT,
    ALLOW_NULL
}
```

`ExecutionOrder` definition:

```
public enum ExecutionOrder
{
    NORMAL,
    FIRST,
    LAST
}
```

`DataMatch` definition:

```
public enum DataMatch
{
    RETURN_VALUE,
    PARAMETERS_MATCH,
    CUSTOM
}
```

3.3.3 Example

Java source code:

```
@WebMethod
@BAMethod(AgreementType.COORDINATOR_COMPLETION,MethodType.MODIFY)
@BACompletedBy(value="checkout",type=DataMatch.CUSTOM)
public Integer bookRoom(String user,String pass,Integer roomNumber)
{
    // ...
}
```

3.4 Annotation: @BACompensatedBy

Qualified name: `org.jboss.txbridge.ba.annotation.BACompensatedBy`

Annotation type: **optional**

Target element: **method**

3.4.1 Description

This annotation provides necessary information about the compensation action. Member values of this annotation override those specified by the `@BACompensation` annotation.

Member-value	Meaning	Default
value	Name of the method that is used for compensation.	None

serviceClass	Class that contains the method specified as the compensation task.	Empty.
ejbInterface	Interface of the EJB that contains the compensation method.	Empty.
jndiName	JNDI name of the EJB that contains the compensation method.	Empty string.
providerURL	URL of the JNDI service at which the EJB that contains the compensation method should be looked up.	Empty string.
wsdl *(*Test implementation*)*	URL of the WSDL, which describes the compensation action.	Empty string.
namespace *(*Test implementation*)*	Target namespace used by the compensation service.	Empty string.
serviceName *(*Test implementation*)*	Name of the compensation service.	Empty string.
mode	Mode of the compensation – whether it is done locally, on an EJB or is remote. This member value takes elements of ExecutionMode enum.	ExecutionMode.EJB
match	Required type of parameters match for the compensation method. By default original method must provide enough data to fill in argument list for the compensation action.	ParameterMatch.STRICT
single	If true than compensation action for this method will be invoked only once for each transaction (using data provided by the last execution of this method).	false

order	*NOT YET IMPLEMENTED*	ExecutionOrder.NORMAL
type	Type of the compensation. Specifies what data needs to be remembered by the data management mechanism. This member value takes elements of the DataMatch enum.	DataMatch.RETURN_VALUE

3.4.2 Definitions

Annotation definition:

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface BACompensatedBy
{
    public String value();
    public Class[] serviceClass() default {};
    public Class[] ejbInterface() default {};
    public String jndiName() default "";
    public String providerURL() default "";
    public String wsdl() default "";
    public String namespace() default "";
    public String serviceName() default "";
    public boolean single() default false;
    public ParameterMatch match() default ParameterMatch.STRICT;
    public ExecutionOrder order() default ExecutionOrder.NORMAL;
    public DataMatch type() default DataMatch.RETURN_VALUE;
    public ExecutionMode mode() default ExecutionMode.EJB;
}
```

`ExecutionMode` definition:

```
public enum ExecutionMode
{
    EJB,
    POJO,
    DII
}
```

`ParameterMatch` definition:

```
public enum ParameterMatch
{
    STRICT,
    ALLOW_NULL
}
```

`ExecutionOrder` definition:

```
public enum ExecutionOrder
{
    NORMAL,
    FIRST,
    LAST
}
```

`DataMatch` definition:

```
public enum DataMatch
{
    RETURN_VALUE,
    PARAMETERS_MATCH,
    CUSTOM
}
```

3.4.3 Example

Java source code:

```
@WebMethod
@BAMethod(AgreementType.PARTICIPANT_COMPLETION,MethodType.MODIFY)
@BACompensatedBy(value="cancelRoom",type=DataMatch.CUSTOM)
public Integer bookRoom(String user,String pass,Integer roomNumber)
{
    // ...
}
```

3.5 Annotation: @BAParam

Qualified name: `org.jboss.txbridge.ba.annotation.BAParam`

Annotation type: **optional**

Target element: **parameter**

3.5.1 Description

This annotation marks a parameter so that it is stored by the framework data management mechanisms and can be used during completion or compensation (either as argument for the service or as additional object, which might be retrieved within the completion/compensation method). This annotation should be used if data match type of any service is DataMatch.CUSTOM.

Member-value	Meaning	Default
value	Unique identifier of the method's argument that is to be stored. The identifier must be unique at a single method's scope.	None

3.5.2 Definitions

Annotation definition:

```
@Target(ElementType.PARAMETER)
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface BAParam
{
    String value();
}
```

3.5.3 Example

Java source code:

```
@WebMethod
@BAMethod(AgreementType.PARTICIPANT_COMPLETION,MethodType.MODIFY)
@BACompensatedBy(value="cancelMeal",type=DataMatch.CUSTOM)
public MealOrder orderMeal(@BAParam("user")Integer userId, Integer mealId)
{
    // ...
}
```

Compensation action:

```
@WebMethod
public Integer cancelMeal(@BAParam("user")Integer userId)
{
    // ...
}
```

3.6 Annotation: @BAResult

Qualified name: `org.jboss.txbridge.ba.annotation.BAResult`

Annotation type: **optional**

Target element: **method**

3.6.1 Description

This annotation marks a method so that its return value is remembered by the framework data management mechanisms and can be used during completion or compensation (either as argument for the method or as additional object, which might be retrieved within the completion/compensation method). This annotation should be used if data match type of any service is DataMatch.CUSTOM.

Member-value	Meaning	Default
value	Unique identifier of the method's return value that is to be remembered. The identifier must be unique at a single method's scope.	None

3.6.2 Definitions

Annotation definition:

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface BAResult
{
    String value();
}
```

3.6.3 Example

Java source code:

```
@WebMethod
@BAMethod(AgreementType.PARTICIPANT_COMPLETION,MethodType.MODIFY)
@BACompensatedBy(value="cancelTicket",type=DataMatch.CUSTOM)
@BAResult("orderID")
public Integer bookTicket(@BAParam("user")Integer userId, Integer movieId)
{
    // ...
}
```

Compensation action:

```
@WebMethod
public Integer cancelTicket(@BAParam("user")Integer userId,
                                                @BAParam("orderID") Integer orderId)
{
    // ...
}
```

3.7 Annotation: @BADataManagement

Qualified name: `org.jboss.txbridge.ba.annotation.BADataManagement`

Annotation type: **optional**

Target element: **field**

3.7.1 Description

This marker annotation is used to tag the `DataManager` object so that it can be transparently injected during runtime by the framework mechanisms. `DataManager` is used for storing and retrieving arbitrary data, which should be shared between the original service and its completion or compensation actions.

3.7.2 Definitions

Annotation definition:

```
@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface BADataManagement
{
}
```

3.7.3 Example

Java source code:

```
public class SomeClass
{
    @BADataManagement
    private DataManager dm;

    // ...
}
```

4 Data Manager Methods

The Business Activity Framework supports storing arbitrary data, which is used by the original service and might be potentially useful when invoking completion or compensation actions (either as arguments of those services or within the business logic). Storing data from inside business methods can be performed with the use of a `DataManager` component. This component also enables retrieving data in the corresponding completion or compensation actions. `DataManager` is automatically injected during runtime by the framework mechanisms.

Qualified name: `org.jboss.txbridge.ba.datamgmt.DataManager`

Usage: **optional**

DataManager interface definition:

```
public interface DataManager
{
    public void put(Object objectId, Object object);
    public Object get(Object objectId);
}
```

Usage example:

```
public class HotelBAImpl implements HotelBA
{

    @BADataManagement
    private DataManager dm;

    @WebMethod
    @BAMethod(AgreementType.PARTICIPANT_COMPLETION,MethodType.MODIFY)
    @BACompensatedBy(value="cancelMeal",type=DataMatch.CUSTOM)
    @BAResult("id")
    public Integer orderMeal(@BAParam("user")Integer userId, Integer mealId)
    {
        // ...
        dm.put("refund",refundValue);
        // ...
        return orderId;
    }

    @WebMethod
    public void cancelMeal(@BAParam("user")Integer userId,@BAParam("id")Integer orderId)
    {
        // ...
        Integer refundValue = (Integer) dm.get("refund");
        if (refundValue != null)
        {
            // ...
        }
        // ...
    }

}
```

Following operations can be invoked on the data manager:

- `put(id,Object)`
- `get(id)`

The first one supports storing arbitrary objects with given identifiers. The other one is used in completion/compensation tasks for retrieving objects with given identifiers. Identifiers must be unique at a method's scope. Next sections of this document provide a more detailed description of those methods .

4.1 put()

4.1.1 Description

This method can be used to store any data, which might be required by the completion/compensation action. The completion/compensation method may need this data either as its arguments or may retrieve them from inside its body. Any data that is stored must be given a unique identifier so that it can be retrieved successfully. It is possible to use any objects as identifiers (i.e. String, Integer, etc).

Method signature:

```
public void put(Object objectID, Object object);
```

4.1.2 Example

Java source code:

```
public class HotelBAImpl implements HotelBA
{

    @BADataManagement
    private DataManager dm;

    @WebMethod
    @BAMethod(AgreementType.PARTICIPANT_COMPLETION,MethodType.MODIFY)
    @BACompensatedBy(value="cancelMeal",type=DataMatch.CUSTOM)
    public MealOrder orderMeal(@BAParam("user")Integer userId, Integer mealId)
    {
        // ...
        dm.put("id",orderID);
        dm.put("bonusID",bonusID);
        // ...
        return mealOrder;
    }

    @WebMethod
    public void cancelMeal(@BAParam("user")Integer userId,@BAParam("id")Integer orderId)
    {
        // ...
        Integer bonusID = (Integer) dm.get("bonusID");
        if (bonusID != null)
        {
            // ...
        }
        // ...
    }

}
```

4.2 get()

4.2.1 Description

This method can be used to get any data, which is required by the completion/compensation action. It is necessary for the completion/compensation action to know the identifiers, which were used when storing data by the main method. This method returns null if the data manager does not have data with the given identifier. This can be used to differentiate, whether the service has been executed as a completion/compensation action or as a normal method.

Method signature:

```
public Object get(Object objectID);
```

4.2.2 Example

Java source code:

```
public class HotelBAImpl implements HotelBA
{

    @BADataManagement
    private DataManager dm;

    @WebMethod
    @BAMethod(AgreementType.PARTICIPANT_COMPLETION,MethodType.MODIFY)
    @BACompensatedBy(value="cancelMeal",type=DataMatch.CUSTOM)
    public MealOrder orderMeal(@BAParam("user")Integer userId, Integer mealId)
    {
        // ...
        dm.put("id",orderID);
        dm.put("bonusID",bonusID);
        // ...
        return mealOrder;
    }

    @WebMethod
    public void cancelMeal(@BAParam("user")Integer userId,@BAParam("id")Integer orderId)
    {
        // ...
        Integer bonusID = (Integer) dm.get("bonusID");
        if (bonusID != null)
        {
            // ...
        }
        // ...
    }

}
```

5 Examples

For examples, please refer to the Business Activity Framework Demo Application. Any questions regarding the framework's usage are welcome and can be directed to the author of the framework. Questions can be also posted to the JBoss Transactions Developer Forum [8].

6 References

[1] Machulak M.P. Business Activity Framework. http://labs.jboss.com/jbosstm/baframework

[2] Machulak M.P. Business Activity Framework Maintenance Guide – to be released – September 2007.

[3] Little, M., J. Maron, and G. Pavlik, *Java Transaction Processing: Design and Implementation*, Prentice Hall PTR, New Jersey, USA, 2004.

[4] WS-BusinessActivity Specification, version 1.0. August 2005. http://www.arjuna.com/library/specs/ws-tx/WS-BusinessActivity.pdf.

[5] JSR 181: Web Services Metadata for the Java™ Platform, version 1.0. June 2005. http://jcp.org/en/jsr/ detail?id=181

[6] Halliday J.J. et al. JBoss XML Transaction Service. http://labs.jboss.com/jbosstm/

[7] Khan K. et al. JBoss AOP. http://labs.jboss.com/jbossaop/

[8] JBoss Transactions Developer Forum. http://www.jboss.com/index.html?module=bb&op=viewforum&f=164