



# Metadata Support for Transactional Web Services

Maciej Machulak  
Jonathan Halliday  
Mark Little

Middleware for Web Services Workshop  
Annapolis, Maryland, USA  
October 2007

# Agenda

- Introduction
  - Web Services and Transactions
- Motivation for metadata support
- Business Activity Framework
  - API
  - Design
  - Implementation
- Future work
- Summary
- Questions and Discussion

# Web Services and Transactions

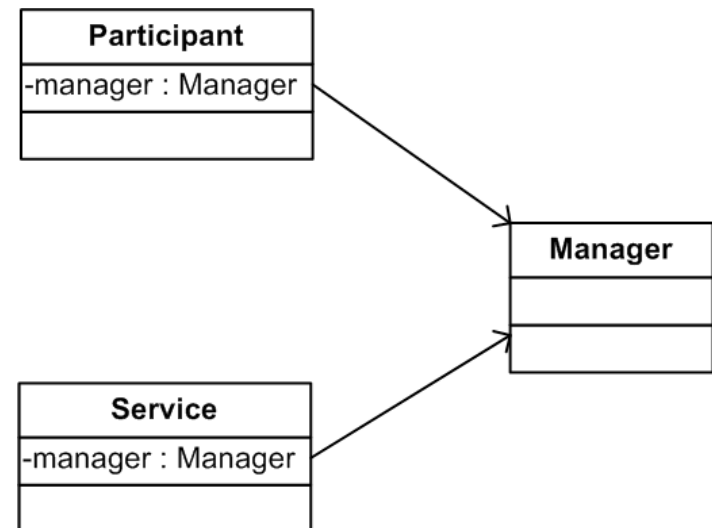
- Web Service environment encourages development of applications that would integrate different businesses
- Such applications rely on complex Business-to-Business interactions that involve many parties, span many different organisations, and last for hours, days or even months
- Web Services-based transactions differ from traditional transactions in that they execute over long periods; isolation levels of such transactions need to be relaxed
- Atomic (ACID) transactions by themselves are not adequate for structuring long-lived applications
- Web Services-based transactions are based on **extended transaction models**

# WS-Business Activity

- Defines extended transaction model – **Business Activity (BA)**
- BA designed specifically for **long-duration interactions**, where exclusively locking resources is impossible or impractical (B2B)
- Services are requested to do some work, and where those services have the ability to undo any work, they inform the BA about it
- If the Business Activity is to cancel its work then services execute their undo behaviour (**compensation** – forward recovery)
- Consistency is maintained by compensation actions (those must be explicitly provided by business programmers)
- Consensus within a single Business Activity is achieved through **agreement protocol**

# Problem statement

- Development of transaction-aware Web Services with accordance to Business Activity model is quite complex and requires substantial programming effort.
- Developers need to provide necessary transaction-related code required to expose Web Services as Business Activity tasks
- This results in transaction-related code being mixed with the business logic
- Compensation actions are application-specific – need to be provided explicitly by developers
- Most commonly application has to be developed according to a transaction-related design pattern: core business logic, participant, manager



# Objectives

- Ease the task of writing transaction-aware Web Services with accordance to the Business Activity extended transaction model
- Provide **separation of concerns** and release programmer's from the necessity of developing any transaction-related code
- Programmers should be able to declaratively specify all transactional requirements of their Web Services (using **metadata** – e.g. **annotations**)
- All necessary transaction processing should be provided basing upon those requirements in a **transparent way** and should be applied at **runtime**

# Business Activity Framework

- Support for transaction-aware Web Services participating in Business Activity type extended transactions
- Transactional requirements specified using **annotation-based API**
  - Web Services can be wired up with their compensation actions with few simple declarative statements
  - the application code preserves its elegance and simplicity (business logic is separated from any transaction-related code)
- Framework provides implementation of the "manager" and "participant" – developer is only concerned with the business logic
- Framework provides **transparent transaction management**
  - automated execution of compensation actions as and when necessary
  - all mechanisms are applied at runtime
- **Flexible management of data** required by compensation actions

# Business Activity Framework – API

- Clean and intuitive annotation-based API
  - keeps with the current trend in enterprise environments – the use of annotations for declaratively configuring both Web Services and transactional behaviour is widespread in both Java and .NET
  - @BASService, @BAAgreement, @BACompensatedBy, @BACompensationManagement, @BAParam, @BAResult
- Leverages programmer familiarity with EJB3 and JSR-181 – reduces the learning curve for new users of the framework
- All annotations have reasonable default values
  - in simplest case the programmer may mark a Web Service method with @BASService and @BAAgreement annotations
- Additional operations can be used if highly flexibly data management is required
  - put(id,Object), get(id) – programmers use those to store arbitrary data, which may be required during compensation

# BA Framework – API – Example

- Compensation Manager

```
@BACompensationManagement  
BACompensationManager cm;
```

- Transactional Web Service

```
@WebMethod  
@BAService(BAServiceType.MODIFY)  
@BAAgreement(BAAgreementType.PARTICIPANT_COMPLETION)  
@BACompensatedBy(value="cancelRoom", type=DataMatch.CUSTOM)  
@BAResult("reservationNumber")  
public Integer bookRoom(@BAParam("uid") Integer userID,  
                        Integer roomNumber)  
{  
    ... // business logic  
    cm.put("refundValue", 5);  
    ... // business logic  
    return reservationNumber;  
}
```

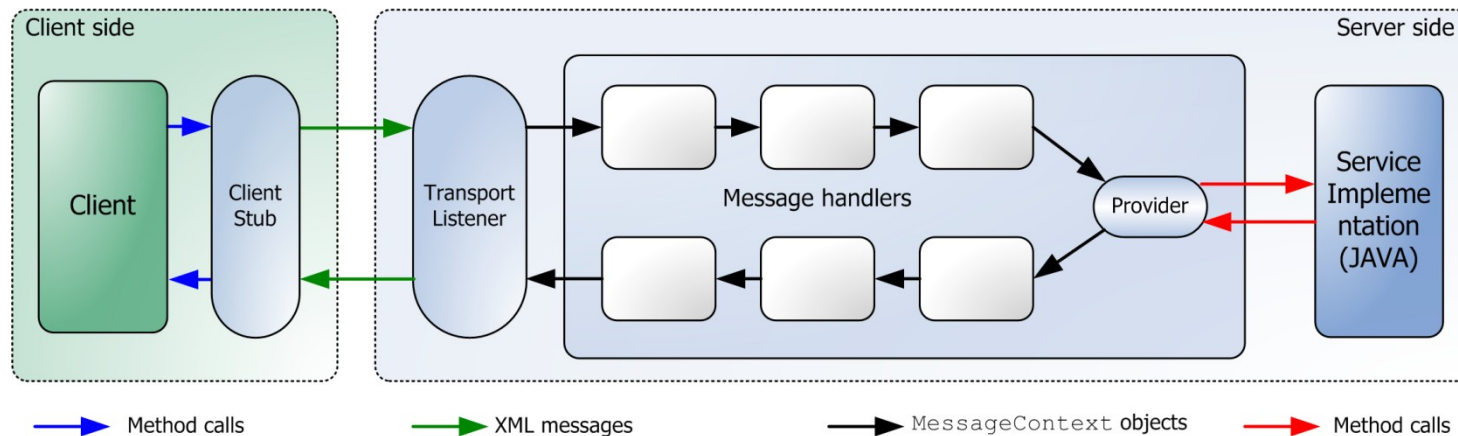
## BA Framework – API – Example (2)

- Compensation action

```
@WebMethod
public void cancelRoom(@BAParam("uid") Integer uid,
    @BAParam("reservationNumber") Integer reservationNumber)
{
    ... // business logic
    Integer refund = (Integer) cm.get("refundValue");
    if (refund != null)
    {
        ... // business logic
    }
    ... // business logic
}
```

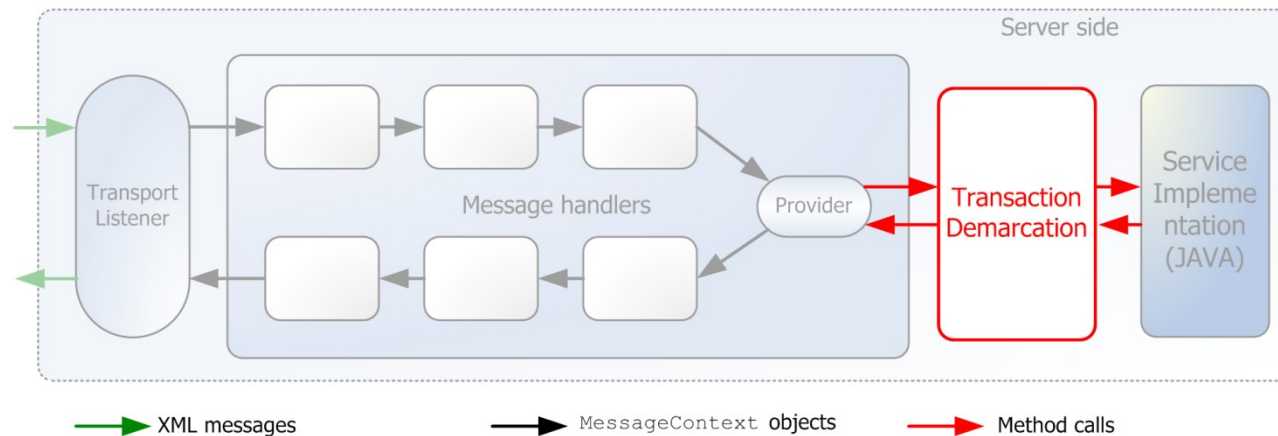
# Business Activity Framework – design

- Framework intercepts calls to transactional Web Services, such calls are subject to transaction processing mechanisms
- Calls are intercepted after being mapped from SOAP messages to actual calls on objects (AOP-based call interception)
- Transaction processing is based on requirements specified by the Web Service



# Business Activity Framework – design

- Framework intercepts calls to transactional Web Services, such calls are subject to transaction processing mechanisms
- Calls are intercepted after being mapped from SOAP messages to actual calls on objects (AOP-based call interception)
- Transaction processing is based on requirements specified by the Web Service

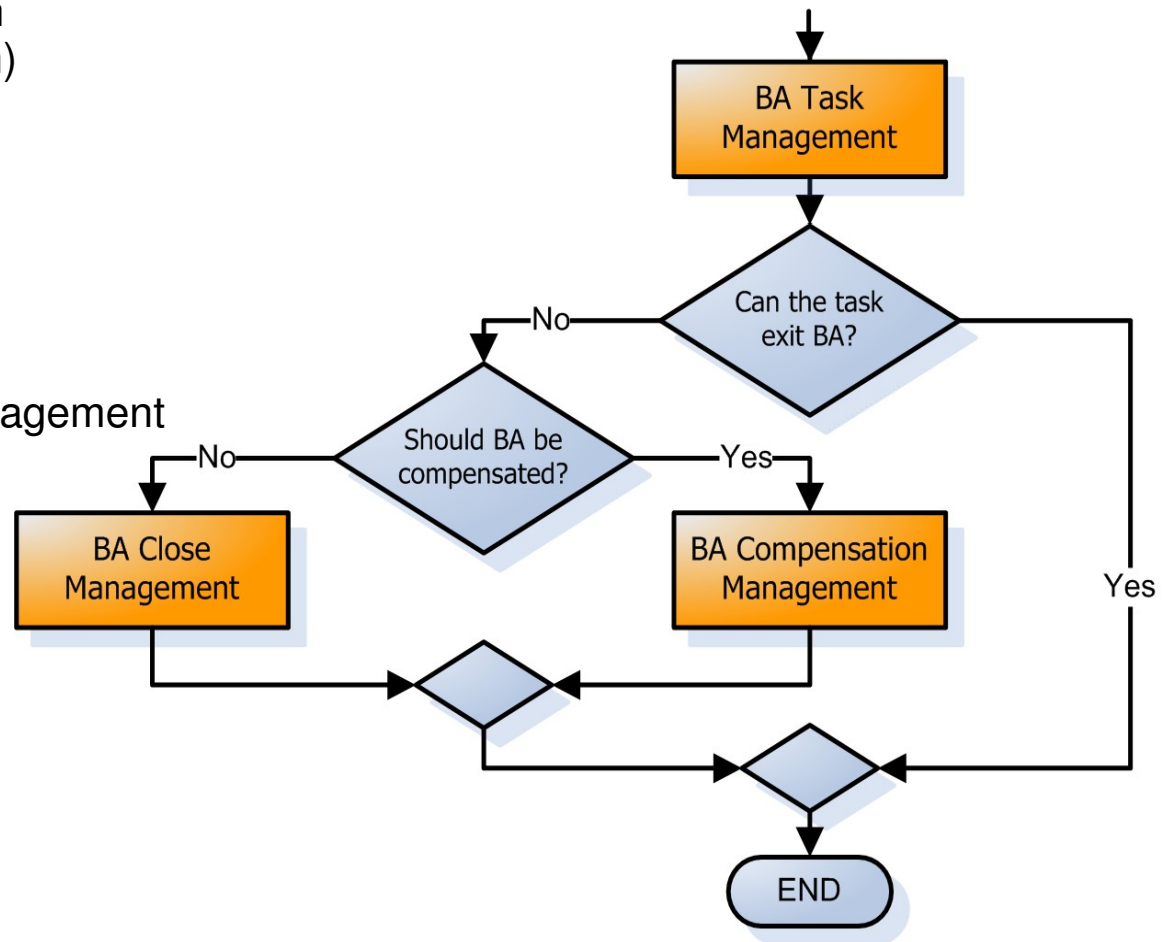


# Business Activity Framework – design

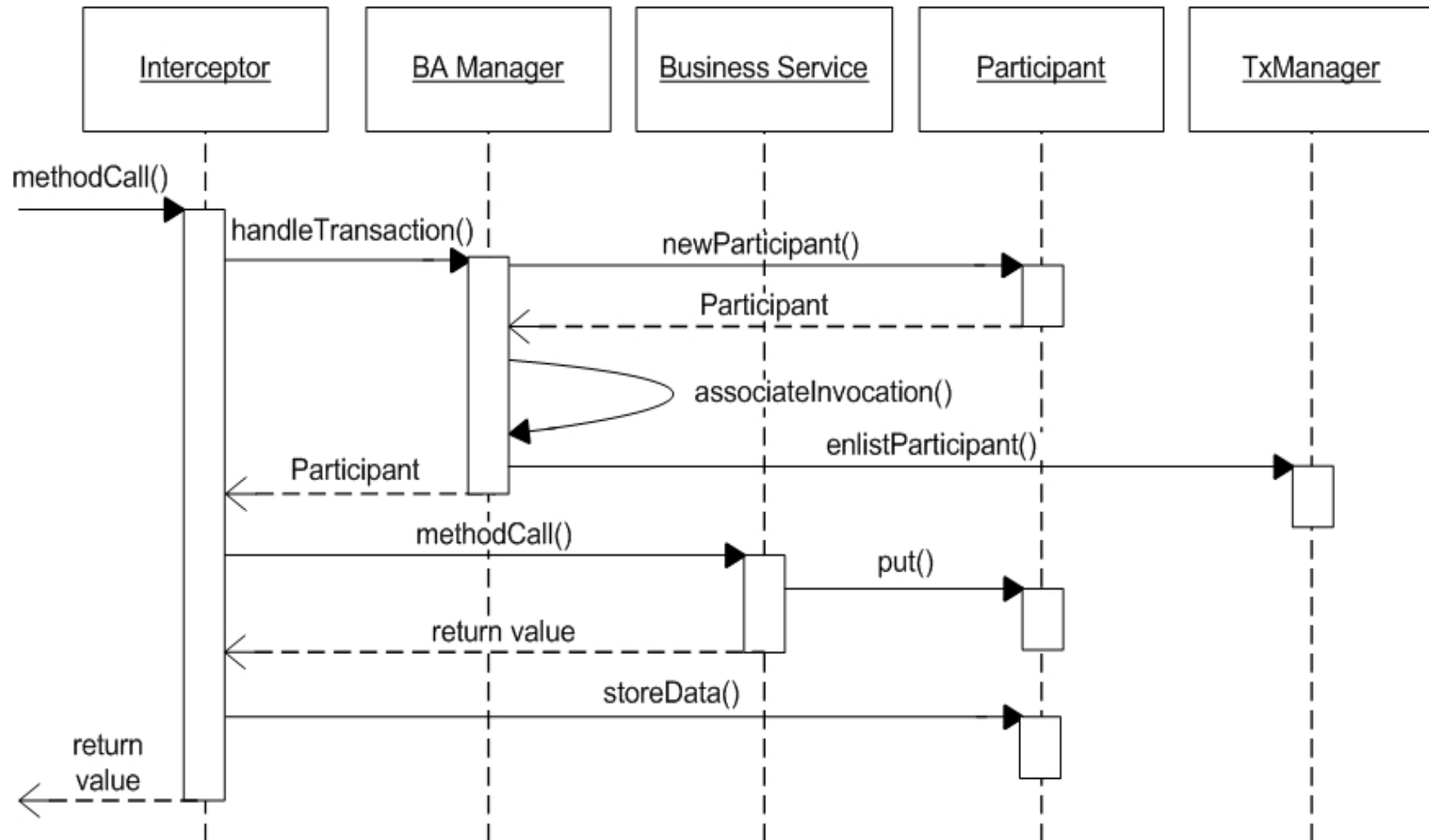
- Each intercepted call to a transactional Web Service is uniquely identified
- Calls are associated with participants – those participants are automatically enlisted to play role in a transaction
- Each participant stores data related to Web Service calls – such data may be later used during compensation
- Participants are responsible for responding to transaction control messages (e.g. *close()*, *compensate()*)
- Framework manages execution of compensation actions as and when necessary

# BA Framework – transaction management

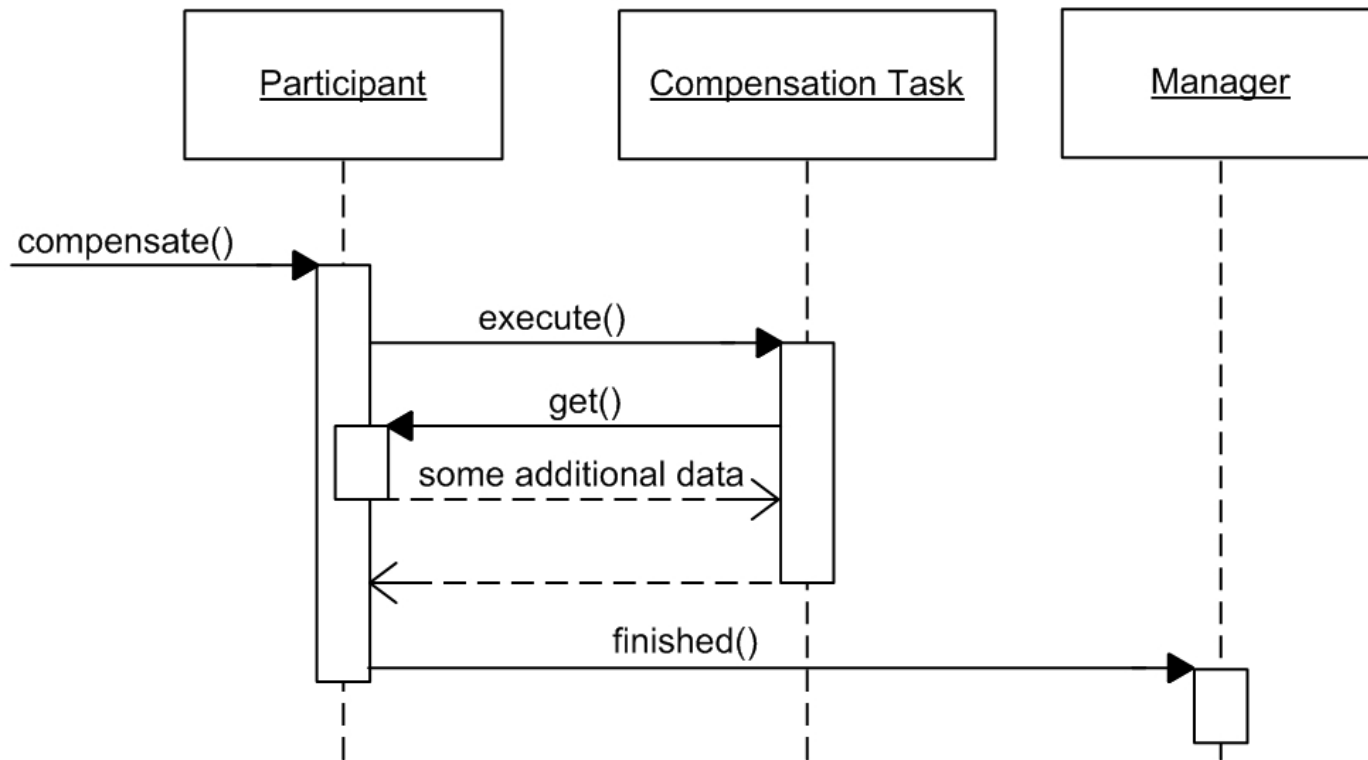
- Framework manages tasks
  - task: original invocation (+ compensation action)
- 3-step processing:
  - BA Task Management
  - BA Close Management
  - BA Compensation Management



# BA Task Management

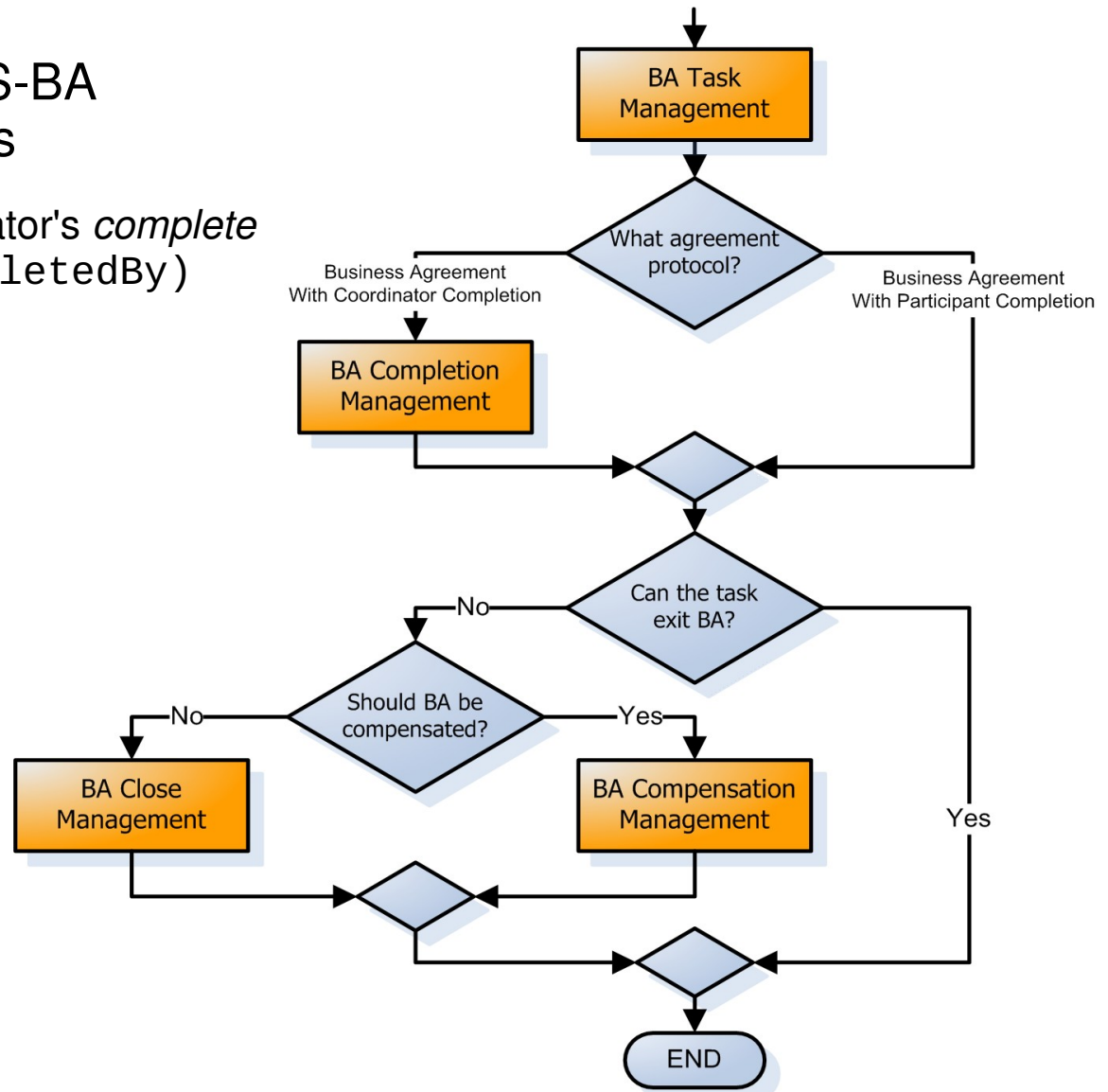


# Compensation Management



# Business Activity Framework 0.2.1.GA

- Support for both WS-BA agreement protocols
  - Support for coordinator's *complete* message (@BACompletedBy)



# BA Framework - Implementation

- Built on top of the JBoss XML Transaction Service component of the JBoss Transaction Service
- Call interception makes use of the JBoss AOP framework
- Java Reflection API – dynamic execution of services

# Future work

- Full support for transaction messages
  - support for coordinator's *close* and *cancel* messages (@BAClosedBy and @BACanceledBy annotations)
- Dynamic Execution of Web Services ✓
  - completion and compensation actions accessed as Web Services
- Exposing existing components as Transactional Web Services
  - transactional-requirements specified in XML configuration file
- Completion and Compensation chains
  - programmers specify multiple possible completion/compensation actions – framework executes only a single, currently available one
- Support for crash recovery
  - logging registered completion and compensation actions; making managed data serializable

# Summary

## ■ Business Activity Framework

- provides full support for transaction management of Web Services exposed as Business Activity tasks (**separation of concerns, declaratively specified transactional requirements, transparency**)
- Released as an open-source project under GNU General Public License 2.0;  
<http://labs.jboss.com/jbosstm/baframework> (official website)  
<http://jira.jboss.com/jira/browse/JBTM> (bug tracking)
- Three public releases: 0.1.0GA, 0.2.0GA and 0.2.1GA: framework, demo application, documentation (*Transactional Web Services Programmer's API*)
- Framework scheduled to be included in the core JBoss Transaction Service component; annotations to be reused with other technologies (EJBs?)
- Framework's API to be used for the JSR-156 (JAXTX – Java API for XML Transactions)
- Interest shown by the community – e.g. „*JBoss Business Activity Framework - for transactional web services*” by Bill Burke, available at [http://www.theserverside.com/news/thread.tss?thread\\_id=46557](http://www.theserverside.com/news/thread.tss?thread_id=46557)

# Questions and Discussion





# Thank You

Maciej Machulak  
mmachulak@redhat.com

Jonathan Halliday  
jonathan.halliday@redhat.com

Mark Little  
mlittle@redhat.com