
Java™2 Enterprise Edition

J2EE™ Activity Service Specification

JSR095

Specification Lead:

Ian Robinson

IBM Corp.

Technical comments:

Activity service discussion database or

jsr95@hursley.ibm.com

Version 0.1

Expert Group Draft

1.0 Introduction

This document describes the system design and interfaces for the J2EE Activity service. The J2EE Activity service is the realisation, within the J2EE programming model, of the OMG Activity service¹.

The purpose of the Activity service is to provide a middleware framework on which extended unit of work models can be constructed. An extended unit of work model might provide services for a long-running business activity that consists of a number of short-duration ACID transactions. This sort of structuring allows the business activity to acquire and hold resource locks only for the duration of the ACID transaction rather than the entire duration of the long-running activity. In such a model, the failure scenarios may be quite complex, involving the compensation of some or all of the ACID transactions that were committed before the long-running activity failed. The responsibility for providing the appropriate recovery from such a failure may be shared between the application itself, which is the component that understand *what* needs to be compensated, and the extended unit of work service provider, which might provide facilities to register compensating actions.

The Activity service provides a generic middleware framework on which many types of extended transaction, and other unit of work, models can be built.

1.1 Scope

This document and related javadoc describes the architecture of the J2EE Activity service and defines the function and interfaces that must be provided by an implementation of the J2EE Activity service in order to support high-level services constructed on top of this. Such high-level services provide the specific extended transaction model behaviour required by the application component.

Specific high-level services and extended transaction models that use the Activity service are beyond the scope of this specification and should be introduced into J2EE via separate JSRs.

Note that the term *extended transaction model* does not necessarily imply the involvement of any ACID transactions, although it may. Throughout the remainder of this specification, the term *transaction*, if unqualified, will be used to refer to a JTS² transaction which is typically accessed via JTA³ in J2EE.

1. Additional Structuring Mechanisms for the OTS Specification - *OMG document orbos/2000-06-19*

2. Java Transaction Service, V1.0, *Sun Microsystems Inc.*

1.2 Target Audience

The target audience of this specification includes:

- providers of high-level services that offer extended transaction behaviour.
- implementors of application servers and EJB containers.
- implementors of transaction managers, such as a JTS.

1.3 Organization

This document describes the architecture of the Activity service as it relates to the J2EE server environment. The different roles of the components of the service are described, particularly with respect to higher-level services that are built on top of the Activity service. Specific Activity service interfaces are described in general terms in this document and in more detail in the accompanying javadoc package.

1.4 J2EE Activity Service Expert Group

The J2EE Activity service expert group includes the following:

Ian Robinson, ian_robinson@uk.ibm.com

Tony Storey, tony_storey@uk.ibm.com

Tom Freund, tjfreund@uk.ibm.com

IBM (UK) Laboratories
Hursley
Winchester
Hants SO21 2JN
UK

Dave DeCaprio, Daved@alum.mit.edu

i2
5 Cambridge Center
Cambridge, MA 02142

Orest Halustchak, orest.halstchak@autodesk.com

Autodesk Inc.
99 Bank St., Suite 301
Ottawa, ON, K1P 6B9

Canada

Ram Jeyaraman, Ram.Jeyaraman@eng.sun.com
Sun Microsystems Inc,
901 San Antonio Road,
Palo Alto, California 94303,
U.S.A.

Mark Little, Mark@arunja.com
Bluestone Arjuna Labs,
Churchill House,
12 Mosley Street,
Newcastle upon Tyne,
NE1 1DE,
England.

Ramesh Loganathan, rameshl@pramati.com
Pramati Technologies
301 White House
Begumpet
Hyderabad-500016
India

Eric Newcomer, eric.newcomer@iona.com
IONA Technologies
200 West Street
Waltham, MA 02451 USA

Phil Quinlan, philip.quinlan@bankofamerica.com
Bank of America,
2001 Clayton Road,
Concord, CA, 6420-2405,
USA

Satish Viswanathan, satish@iplanet.com
iPlanet
4850, Network Circle
Santa Clara, CA
USA

Martin West, Martin.west@spirit-soft.com
Spiritsoft
1 Royal Exchange Avenue
Threadneedle Street
London EC3V 3LT

Mehmet C. Eliyesil, jsr000095@silverstream.com
SilverStream Software, Inc.
Application Server Division
Two Federal St.
Billerica, MA 01821

1.5 Acknowledgements

Alex Mulholland, of IBM, has provided invaluable technical contributions to this specification.

2.0 *Notes about this draft*

Note to Reviewers: *This is the 0.1 initial draft of the J2EE Activity service and is lacking in detail and completeness, both of which will be addressed in future revisions. This draft should be read in conjunction with the javadoc and is provided in its present state to provoke discussion.*

3.0 Overview

The OMG Activity service document¹ describes how an application activity, A_0 , may be split into many different, coordinated, short-duration activities which together form a logical long-running business transaction. This is illustrated below in Figure 1. A_1 and A_2 are Activities (represented by broken ellipses) containing JTA transactions (represented by solid ellipses) T_1 and T_2 ; A_3 and A_4 do not use JTA transactions at all. In this example A_2 and A_3 are executed concurrently after A_1 .

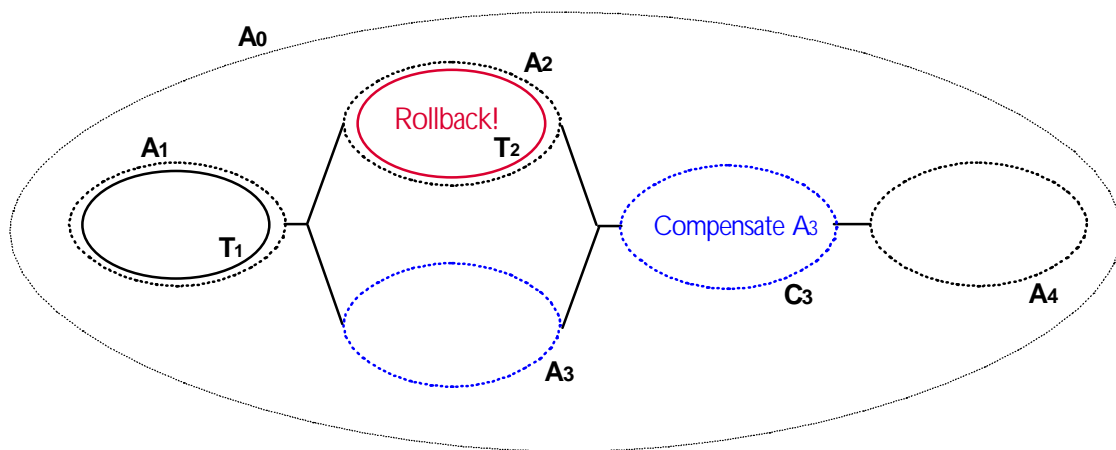


FIGURE 1 A long-running application activity

The reason for structuring the application activity as a *logical long-running transaction* rather than as a single top-level transactions is to prevent certain acquired resources from being held for the entire duration of the application. It is assumed that the application's implementers have segmented the transactional activities within the application into smaller transactional and non-transactional activities, each transaction being responsible for acquiring (and releasing) only those resources it requires. However, if failures and concurrent access occur during the lifetime of these activities then the behaviour of the entire *logical long-running business transaction* may not possess ACID properties. Therefore, some form of (application specific) compensation may be required to attempt to return the state of the system to (application specific) consistency. For example, in Figure 1 assume that the JTA transaction T_2 encapsulated by A_2 has failed (rolls back). Further assume that the application can continue to make forward progress, but in order to do so must now undo some state changes made in A_1 and A_3 . Since T_2 is a transaction, its state changes will be undone automatically by the JTS, so no form of compensation is required for it. Work performed under T_1 has been committed, however, and work performed under A_3 is non-transactional and in this example both need to be compensated. Therefore, new activity C_3 is started as a compensation activity that will attempt to undo state changes performed by A_1 and A_3 . Once C_3 is complete, forward progress continues with A_4 .

Much research on structuring applications out of transactions has been influenced by the ideas of spheres of control⁴. There are several ways in which some or all of the application requirements outlined above could be met. However, it is unrealistic to believe that a single high-level model approach to extended transactions is likely to be sufficient for all (or even the majority of) applications. Therefore, the Activity service provides a low-level infrastructure to support the coordination and control of abstract Activities that are given concrete meaning by the high-level services that are implemented on top of the Activity service. These Activities may be transactional, they may use weaker forms of serializability, or they may not be transactional at all; the important point is that we are only concerned with their control and coordination, leaving the semantics of such Activities to the high-level services.

The OMG Activity service specifications describes how a variety of unit of work models may be applied over the Activity service, including OTS strict two-phase commit transactions, nested transactions, as well as a variety of different kinds of transactional behaviour including long-running transactions similar to Sagas with Compensation, Open Nested Transactions and Workflows.

An Activity is a unit of (distributed) work that may or may not be transactional. During its lifetime an Activity may have transactional and non-transactional periods. Every entity including other Activities can be parts of an Activity, although an Activity need not be composed of other activities. An Activity is characterized by an application-demarcated context under which a distributed application executes. This context is implicitly propagated with all requests made in the scope of the Activity and defines the unit of work scope under which any part of an application executes.

An Activity is created, made to run, and then completed to produce an *Outcome*. Demarcation signals of any kind are communicated to any registered entities (*Actions*) through *Signals* which are produced by *SignalSets*. Actions allow an Activity to be independent of the specific work it is required to do in response to broadcasting a Signals. For example, if a JTS were to be implemented as a HLS on top of the Activity service, the `org.omg.CosTransactions.Resources` would be registered as Actions with an interest in a two-phase-commit SignalSet which produced *prepare*, *commit*, *rollback*, *commit_one_phase* and *forget* Signals.

The purpose of the J2EE Activity service specification is to define the roles and responsibilities of the components of such a service implementation in a J2EE server environment and, where appropriate, the J2EE client environment. In particular this specification defines the interfaces and behaviour of an Activity service such that vendors may implement higher-level services that use these interfaces to provide the desired extended transaction, or other unit of work, models.

4. C. T. Davies, "Data processing spheres of control", *IBM Systems Journal*, **17**(2), 1978, pp. 179-198.

5. Jim Gray, Andres Reuter, "Transaction Processing", *Morgan Kaufmann Publishers Inc.*, 1993

4.0 J2EE Activity Service Architecture

The architecture for a high-level service providing an extended unit of work model and using the facilities of the Activity service is shown in Figure 2.

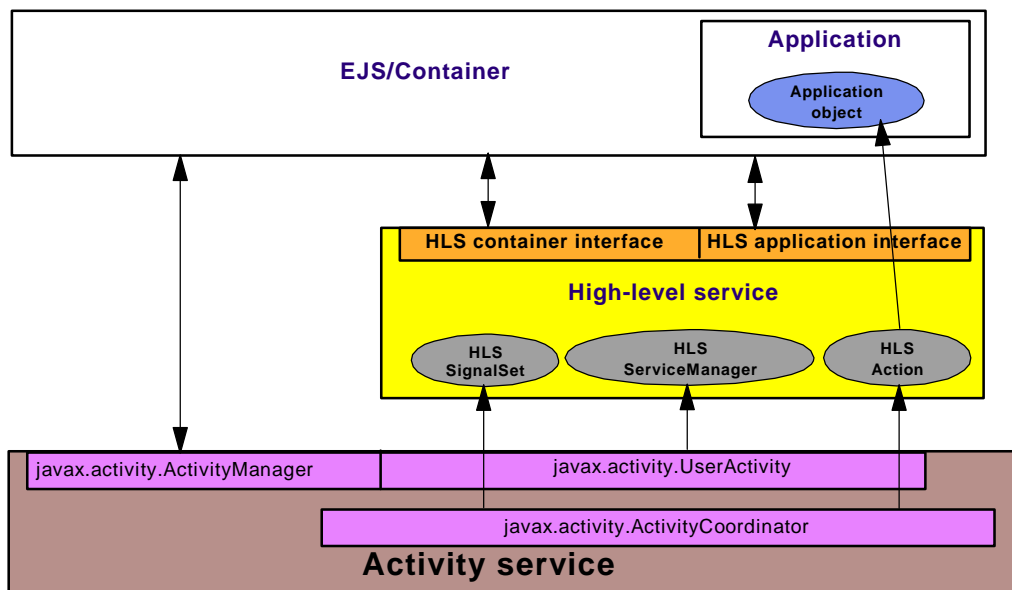



FIGURE 2 Activity and high-level service architecture

Responsibility may be apportioned across three main components in this architecture as follows:

High-level service (HLS) -- this defines the behaviour of a specific extended unit of work model and offers interfaces to the application that uses it, as well as the application server and container. The HLS uses the function provided by the Activity service to manage its distributed context and relationships between this context and any JTS context. It also uses the Activity service as the means by which signals pertaining to the HLS are distributed to participants in an HLS unit of work. In particular, the HLS provides implementations of the `javax.activity.Action`, `javax.activity.SignalSet` and `javax.activity.ServiceManager` interfaces.

This component is external to the Activity service and is beyond the scope of this specification.

Application and container -- the application is designed to participate in a specific type of extended transaction model and uses, either directly or through the container, the facilities provided by the HLS to control the units of activity supported by the HLS. If the HLS provided a compensating extended transaction model, for example, in which a long-running transaction is composed of a sequence of ACID transactions that may

need to be compensated following a failure, then the application component would be expected to provide the compensation data that the HLS would drive at the appropriate time. The application component does not call the Activity service directly. This component is external to the Activity service and is beyond the scope of this specification. is not provided by the Activity service 

Activity service -- the Activity service manages the HLS's service context, both with respect to other Activity contexts and with respect to JTS context, ensuring its appropriate implicit propagation with remote requests. It offers a `javax.activity.UserActivity` interface to the HLS to demarcate these contexts and an extended `javax.activity.ActivityManager` interface to the container to allow these contexts to be suspended and resumed. In addition the ActivityService provides a `javax.activity.ActivityCoordinator` through which the Actions(s) and SignalSet(s) of the HLS are registered. During general signal processing and Activity completion, the ActivityCoordinator obtains the HLS signals from the HLS SignalSet and distributes them to HLS Actions that have registered an interest. This specification is concerned solely with this component.

This component division is intended to be illustrative rather than prescriptive. For example, a container may provide the function of a high-level service.

5.0 *Elements of the Activity service*

Note to Reviewers: *At present, the javadoc contains more detail than this section.*

5.1 *UserActivity*

The `javax.activity.UserActivity` interface defines the methods offered to high-level services (HLS) constructed on top of the Activity service framework. These methods control the demarcation scope of an Activity associated with the calling thread of execution.

The specific behaviour of individual HLS's is encapsulated by a `ServiceManager`. An instance of the `UserActivity` interface has a `ServiceManager` registered with it prior to any Activities being begun by that instance.

Each Activity started by a `UserActivity` instance is an Activity instance of the HLS represented by the registered `ServiceManager`. Methods of the `UserActivity` interface that operate on the active Activity context are operating on the HLS Activity instance most recently associated with the calling thread.

5.2 *ServiceManager*

A `ServiceManager` is an entity that is provided by a HLS built on top of the Activity service that defines how the HLS's Activities should be managed. In particular, it defines which `PropertyGroups` the HLS uses, which `ContextGroup` the HLS participates in, and the `CompletionSignalSet` that is used to complete the HLS's Activities.

The Activity service uses the properties of the `ServiceManager` when it creates and operates on Activities specific to that service.

5.3 *ContextGroup*

A `ContextGroup` is identified by each `ServiceManager` to determine which HLS's Activities will be cooperatively managed. All Activity contexts within a particular Context-

Group are strictly nested with respect to one another but are independent of Activity contexts in any other ContextGroup. This provides different HLS's the flexibility to cooperate with or be independent of other HLS's. A ContextGroup is identified by a String name. A HLS that wishes to have its contexts managed independently of any other HLS should specify the package name of the HLS as the name of the ContextGroup it wishes to participate in.

A *default* ContextGroup is provided, identified by an empty String, which must be used by any HLS that wishes to have its Activity contexts managed cooperatively with JTS context. This default ContextGroup provides nesting behaviour compatible with that described in the OMG Activity service.

5.4 ActivityManager

The `javax.activity.ActivityManager` interface extends the `UserActivity` interface by offering methods to manipulate the Activity context. This interface is not available to services built on top of the Activity service but is available to an EJB container.

5.5 ActivityToken

An `ActivityToken` is used to manipulate hierarchies of Activity and transaction contexts via the `suspend()` and `resume(ActivityToken)` operations of the `ActivityManager` interface. `ActivityTokens` are local to the execution process but may be used on any thread within the execution process.

5.6 GlobalId

A `GlobalId` uniquely identifies an Activity across the namespace.

5.7 ActivityCoordinator

The `ActivityCoordinator` is responsible for broadcasting signals to registered Actions. It has no logic to understand the signals or the meaning of Action outcomes, it simply acts as the messenger for the signals.

There is a single ActivityCoordinator instance per Activity context.

5.8 Signal

Signals are events that are broadcast to interested parties as part of a coordinated SignalSet. Each Signal is uniquely identified by a combination of its SignalName and the name of the containing SignalSet.

5.9 SignalSet

A SignalSet is an entity that is provided by a HLS built on top of the Activity service that produces Signals and understands the responses to those Signals. The SignalSet abstracts from the ActivityCoordinator the knowledge of which Signal should be distributed to the registered Actions based on the state of the Activity and responses to previous Signals. The Activity service itself then needs to provide only a very generic ActivityCoordinator to drive any specific SignalSet. The ActivityCoordinator simply asks a SignalSet for the next Signal and then broadcasts it to each interested Action in turn. The response from each Action is fed back to the SignalSet which has the knowledge of what that result means and which Signal should be sent next.

5.10 CompletionSignalSet

The CompletionSignalSet is a distinguished SignalSet that is used to broadcast completion signals. Each HLS specifies, through its ServiceManager, the identity of its CompletionSignalSet.

Signals from the completionSignalSet are preceded by the preCompletion signal from the predefined Synchronization SignalSet and followed by the postCompletion signals from the Synchronization SignalSet.

5.11 *Predefined SignalSets*

The Activity service provides implementations of two predefined SignalSets.

5.11.1 *Synchronization*

The org.omg.Synchronization SignalSet contains the signals preCompletion and postCompletion, which are sent to Actions with an interest in ChildLifetime under the following circumstances:

preCompletion -- sent prior to distributing signals from the CompletionSignalSet if the CompletionStatus is CompletionStatusSuccess.

postCompletion -- sent after all Signals produced by the CompletionSignalSet have been distributed.

5.11.2 *ChildLifetime*

The org.omg.ChildLifetime SignalSet contains the signals childBegin and childComplete, which are sent to Actions with an interest in ChildLifetime under the following circumstances:

childBegin -- sent when a child Activity context is started. This signal is sent after the child Activity and all its PropertyGroups have been created, when the child Activity context is the active context on the thread.

childComplete -- sent when a child Activity context is completed. This signal is sent after the Synchronization postCompletion signal has been processed.

5.12 *Outcome*

An Outcome is given meaning by a SignalSet when it has finished producing signals. A CompletionSignalSet produces such an Outcome and this is returned on the complete() and completeWithStatus(CompletionStatus) methods of the UserActivity interface.

5.13 Action

An Action is an entity that is registered with an interest in a SignalSets. An Action may only be registered with a single ActivityCoordinator.

An Action is the target object to which a Signal, produced by a SignalSet, is sent during broadcast() and completion operations.

5.14 PropertyGroup

A PropertyGroup is used to provide distributed context, scoped to an Activity, that may be set by an application or a HLS built on top of the Activity service. The format of the distributed context is specific to the PropertyGroup implementation and is neither examined nor understood by the Activity service.

The semantics of the behavioural relationship between PropertyGroups in nested Activities is defined by the specification of each type of PropertyGroup and not by the Activity service. Any number of named PropertyGroup types may be registered with the Activity service. When an Activity is started, an instance of each type of PropertyGroup used by the Activity is created and associated with the Activity. The types of PropertyGroup used by an Activity may be specified through the ServiceManager object registered for a particular HLS.

5.15 PropertyGroupManager

A PropertyGroupManager is an entity that is provided by a PropertyGroup implementation and understands how to create and manipulate a specific type of PropertyGroup. It is registered with the Activity service and is used by the Activity service to create PropertyGroup instances and to manipulate the PropertyGroupContext that is implicitly propagated as part of an Activity context.

For a particular type of PropertyGroup, there must be a PropertyGroupManager registered in each client and server execution environment for which the PropertyGroup will be accessed. If PropertyGroupContext is propagated, as part of an Activity context, to an environment in which there is no appropriate PropertyGroupManager registered, then the PropertyGroupContext is not available within that environment although it may be cached by the Activity service and propagated on to any downstream environment to which the Activity context is further distributed.

5.16 PropertyGroupContext

A PropertyGroupContext object is provided by a PropertyGroup implementation and encapsulates the context data that is implicitly propagated, on behalf of a PropertyGroup, as part of an Activity context. This interface is used by both the Activity service and by PropertyGroupManagers to aid the PropertyGroupManagers marshal and unmarshal the PropertyGroup part of the Activity context at an execution environment boundary when the context needs to be converted to or from the CDR encapsulated form used for remote propagation over RMI/IIOP.

6.0 *Typical Object Interactions*

Note to Reviewers: *To be completed.*

7.0 *Interoperability*

A J2EE Activity service implementation is required to be interoperable across different vendors' ORB boundaries. The format of the interoperable service context is defined by the `CosActivity::ActivityContext` structure in the OMG Activity service specification.