SCA Service Component Architecture

ACID Transaction Policy in SCA

SCA Draft Version 0.51, September 25 2007

Technical Contacts:

Dave Booz	IBM Corporation
Mike Edwards	IBM Corporation
Mike Kanaley	TIBCO Software, Inc
Ashok Malhotra	Oracle
Eric Newcomer	Iona Technologies plc.
Sanjay Patil	SAP AG
Ian Robinson	IBM Corporation (Editor)
Michael Rowley	BEA Systems Inc.
Mark Little	Red Hat Inc.

This document is a temporary repository for information pertaining to SCA transaction intents. This information will ultimately be ”promoted” to go alongside SCA Security and Reliability Policy in the SCA Policy Framework specification [2].

Copyright Notice

License

Status of this Document

This specification may change before final release and you are cautioned against relying on the content of this specification. The authors are currently soliciting your contributions and suggestions. Licenses are available for the purposes of feedback and (optionally) for implementation.

Table of Contents

1 Overview

SCA recognizes that the presence or absence of infrastructure for ACID transaction coordination has a direct effect on how business logic is coded. In the absence of ACID transactions, developers must provide logic that coordinates the outcome, compensates for failures, etc. In the presence of ACID transactions, the underlying infrastructure is responsible for ensuring the ACID nature of all interactions. SCA provides declarative mechanisms for describing the transactional environment required by the business logic.

Components that use a synchronous interaction style can be part of a single, distributed ACID transaction within which all transaction resources are coordinated to either atomically commit or rollback. The transmission or receipt of oneway messages can, depending on the transport binding, be coordinated as part of an ACID transaction as illustrated in the *OneWay Invocations* section below. Well-known, higher-level patterns such as store-and-forward queuing can be accomplished by composing transacted one-way messages with reliable-messaging qualities of service.

This document describes the set of abstract policy intents – both implementation intents and interaction intents – that can be used to describe the requirements on a concrete service component and binding respectively.

1.1 Common Transaction Patterns

In the absence of any transaction policies there is no explicit transactional behavior defined for the SCA service component or the interactions in which it is involved and the transactional behavior is environment-specific. An SCA runtime provider may choose to define an out of band default transactional behavior that applies in the absence of any transaction policies.

Environment-specific default transactional behavior may be overridden by specifying transactional intents described in the document. The most common transaction patterns can be summarized as follows:

Managed, shared global transaction **pattern** – the service always runs in a global transaction context regardless of whether the requester runs under a global transaction. If the requester does run under a transaction, the service runs under the same transaction. Any outbound, synchronous request-response messages will – unless explicitly directed otherwise – propagate the service's transaction context. This pattern offers the highest degree of data integrity by ensuring that any transactional updates are committed atomically

Managed, local transaction **pattern** – the service always runs in a managed local transaction context regardless of whether the requester runs under a transaction. Any outbound messages will not propagate any transaction context. This pattern is recommended for services that wish the SCA runtime to demarcate any resource manager local transactions and do not require the overhead of atomicity.

The use of transaction policies to specify these patterns is illustrated later in Table 2.

1.2 Summary of SCA transaction policies

This specification defines implementation and interaction policies that relate to transactional QoS in components and their interactions. The SCA transaction policies are specified as intents which represent the transaction quality of service behavior offered by specific component implementations or bindings.

SCA transaction policy can be specified either in the SCDL or annotatively in the implementation code. Language-specific annotations are described in the respective language binding specifications, for example the SCA Java Common Annotations and APIs specification [3].

This specification defines the following implementation transaction policies:

- managedTransaction – Describes the service component's transactional environment.
- transactedOneWay and immediateOneWay – two mutually exclusive intents that describe whether the SCA runtime will process OneWay messages immediately or will enqueue (from a client perspective) and dequeue (from a service perspective) a OneWay message as part of a global transaction.

This specification also defines the following interaction transaction policies:

- propagatesTransaction and suspendsTransaction – two mutually exclusive intents that describe whether the SCA runtime propagates any transaction context to a service or reference on a synchronous invocation. Note that transaction context MUST NOT be propagated on OneWay messages.

1.3 Global and local transactions

This specification describes "managed transactions" in terms of either "global" or "local" transactions. The "managed" aspect of managed transactions refers to the transaction environment provided by the SCA runtime for the business component. Business components may interact with other business components and with resource managers. The managed transaction environment defines the transactional context under which such interactions occur.

1.3.1 Global transactions

From an SCA perspective, a global transaction is a unit of work scope within which transactional work is atomic. If multiple transactional resource managers are accessed under a global transaction then the transactional work is coordinated to either atomically commit or rollback regardless using a 2PC protocol. A global transaction can be propagated on synchronous invocations between components – depending on the interaction intents described in this specification - such that multiple, remote service providers can execute distributed requests under the same global transaction.

1.3.2 Local transactions

From a resource manager perspective a resource manager local transaction (RMLT) is simply the absence of a global transaction. But from an SCA persective iti is not enough to simply declare that a piece of business logic runs without a global transaction context. Business logic may need to access transactional resource managers without the presence of a global transaction. The business logic developer still needs to know the expected semantic of making one or more calls to one or more resource managers, and needs to know when and/or how the resource managers local transactions will be committed.. The term *local transaction containment* (LTC) is used to describe the SCA environment where there is no global transaction. The boundaries of an LTC are scoped to a remotable service provider method and are not propagated on invocations between components. Unlike the resources in a global transaction, RMLTs coordinated within a LTC may fail independently.

The two most common patterns for components using resource managers outside a global transaction are:

- The application desires each interaction with a resource manager to commit after every interaction. This is the default behavior provided by the **noManagedTransaction** policy (defined below in Transaction implementation policy) in the absence of explicit use of RMLT verbs by the application.

- The application desires each interaction with a resource manager to be part of an extended local transaction that is committed at the end of the method. This behavior is specified by the **managedTransaction.local** policy (defined below in Transaction implementation policy).

While an application may use interfaces provided by the resource adapter to explicitly demarcate resource manager local transactions (RMLT), this is a generally undesirable burden on applications which typically prefer all transaction considerations to be managed by the SCA runtime. In addition, once an application codes to a resource manager local transaction interface, it may never be redeployed with a different transaction environment since local transaction interfaces may not be used in the presence of a global transaction. This specification defines intents to support both these common patterns in order to provide portability for applications regardless of whether they run under a global transaction or not.

1.4 Transaction implementation policy

1.4.1 Managed and non-managed transactions

The mutually exclusive ***managedTransaction*** and ***noManagedTransaction*** intents describe the transactional environment required by a service component or composite.. SCA provides transaction environments that are managed by the SCA runtime in order to remove the burden of coding transaction APIs directly into the business logic. The ***managedTransaction*** and ***noManagedTransaction*** intents can be attached to the sca:composite or sca:componentType XML elements.

The mutually exclusive ***managedTransaction*** and ***noManagedTransaction*** intents are defined as follows:

- **managedTransaction** – There must be a managed transaction environment in order to run this component. The specific type of managedTransaction required is not constrained. The valid qualifiers for this intent are mutually exclusive and are defined as:
- **managedTransaction.global** – There must be an atomic transaction in order to run this component. The SCA runtime must ensure that a global transaction is present before dispatching any method on the component. The SCA runtime uses any transaction propagated from the client or else begins and completes a new transaction. See the ***propagatesTransaction*** intent below for more details.
- **managedTransaction.local** – The component cannot tolerate running as part of a global transaction, and will therefore run within a local transaction containment (LTC) that is started and ended by the SCA runtime. Any global transaction context that is propagated to the hosting SCA runtime must not be visible to the target component. Any interaction under this policy with a resource manager is performed in an extended resource manager local transaction (RMLT). Upon successful completion of the invoked service method, any RMLTs are implicitly requested to commit by the SCA runtime. Note that, unlike the resources in a global transaction, RMLTs so coordinated in a LTC may fail independently. If the invoked service method completes with an exception then any RMLTs are implicitly rolled back by the SCA runtime. Local transactions cannot be propagated outbound across remotable interfaces.
- **noManagedTransaction** – The component runs without a managed transaction, under neither a global transaction nor an LTC. A transaction that is propagated to the hosting SCA runtime MUST NOT be joined by the hosting runtime on behalf of this component. When interacting with a resource manager under this policy, the application (and not the SCA runtime) is responsible for controlling any resource manager local transaction boundaries, using resource-provider specific interfaces (for example a Java implementation accessing a JDBC provider must choose whether a Connection should be set to autoCommit(true) or else

must call the Connection commit or rollback method). SCA defines no APIs for interacting with resource managers.

- **(absent)** – The absence of an implementation intents leads to runtime-specific behavior. A runtime that supports global transaction coordination may choose to provide a default behavior that is the managed, shared global transaction pattern but is not required to do so.

1.4.2 OneWay Invocations

When a client uses a reference and sends a OneWay message then any client transaction context is not propagated. However, the OneWay invocation on the reference may, itself, be *transacted*. Similarly, from a service perspective, any received OneWay message cannot propagate a transaction context but the delivery of the OneWay message may be *transacted*. A *transacted* OneWay message is a one-way message that - because of the capability of the service or reference binding - can be enqueued (from a client perspective) or dequeued (from a service perspective) as part of a global transaction. SCA defines two mutually exclusive implementation intents, **transactedOneWay** and **immediateOneWay**, that determine whether OneWay messages must be transacted or delivered immediately. Either of these intents may be attached to the sca:service or sca:reference elements but a deployment error will occur if both intents are attached to the same element. Either of these intents may be attached to the sca:component element, indicating that the intent applies to any service or reference element children. The intents are defined as follows:

- **transactedOneWay** – When applied to a reference indicates that any OneWay invocation messages MUST be transacted as part of a client global transaction. If the client is not configured to run under a global transaction or if the binding does not support transactional message sending, then a deployment error occurs. When applied to a service indicates that any OneWay invocation message MUST be received from the transport binding in a transacted fashion, under the target service's global transaction. The receipt of the message from the binding is not committed until the service transaction commits; if the service transaction is rolled back the the message remains available for receipt under a different service transaction. If the service is not configured to run under a global transaction or if the binding does not support transactional message receipt, then a deployment error occurs.
- **immediateOneWay** – When applied to a reference indicates that any OneWay invocation messages is sent immediately regardless of any client transaction. When applied to a service indicates that any OneWay invocation is received immediately regardless of any target service transaction. The outcome of any transaction under which an immediateOneWay message is processed has no effect on the processing (sending or receipt) of that message.

The absence of either intent leads to runtime-specific behavior. The SCA runtime may send or receive a OneWay message immediately or as part of any sender/receiver transaction. The results of combining this intent and the ***managedTransaction*** implementation policy of the component sending or receiving the transacted OneWay invocation are summarized below in Table 1.

transacted/immediate intent	**managedTransaction (client or service implementation intent)**	**Results**
transactedOneWay	managedTransaction.global	OneWay interaction (either client message enqueue or target service dequeue) is committed as part of the global

		transaction.
transactedOneWay	managedTransaction.local or noManagedTransaction	This is an "incompatible deployment" Error
immediateOneWay	Any value of managedTransaction	The OneWay interaction occurs immediately and is not transacted.
<absent>	Any value of managedTransaction	Runtime-specific behavior. The SCA runtime may send or receive a OneWay message immediately or as part of any sender/receiver transaction.

Table 1 Transacted OneWay interaction intent

[**Note:** The SCA Assembly specification [1] will need to specify the semantics of oneway sends. For example, can a oneway send result in a synchronous Runtime exception related to protocol error that occurs during the send?]

1.5 Transaction interaction policies

The mutually exclusive ***propagatesTransaction*** and ***suspendsTransaction*** intents may be attached either to an interface (e.g. Java annotation or WSDL attribute) or explicitly to an sca:service and sca:reference XML element to describe how any client transaction context will be made available and used by the target service component. Section 1.5.1 considers how these intents apply to service elements and Section 1.5.2 considers how these intents apply to reference elements.

1.5.1 Handling Inbound Transaction Context

The mutually exclusive ***propagatesTransaction*** and ***suspendsTransaction*** intents may be attached to an sca:service XML element to describe how a propagated transaction context should be handled by the SCA runtime, prior to dispatching a service component. If the service requester is running within a transaction and the service interaction policy is to propagate that transaction, then the primary business effects of the provider's operation are coordinated as part of the client's transaction – if the client rolls back its transaction, then work associated with the provider's operation will also be rolled back. This allows clients to know that no compensation business logic is necessary since transaction rollback can be used.

These intents specify a contract that MUST be implemented by the SCA runtime. This aspect of a service component is most likely captured during application design. Either the ***propagatesTransaction*** or ***suspendsTransaction*** intent can be attached to sca:service elements and their children but a deployment error will occur if both intents are specified. The intents are defined as follows:

- **propagatesTransaction** – The SCA runtime MUST ensure that the service is dispatched under any propagated (client) transaction.
- **suspendsTransaction** – The SCA runtime MUST ensure that the service is NOT dispatched under any propagated (client) transaction.

The absence of either interaction intent leads to runtime-specific behavior; the client is unable to determine from transaction intents whether its transaction will be joined.

Transaction context is never propagated on OneWay messages. The SCA runtime ignores ***propagatesTransaction*** for OneWay methods.

These intents are independent from the implementation's ***managedTransaction*** intent and provides no information about the implementation's transaction environment.

The combination of these service interaction policies and the ***managedTransaction*** implementation policy of the containing component completely describes the transactional behavior of an invoked service, as summarized in Table 2.

service interaction intent	managedTransaction (component implementation intent)	Results
propagatesTransaction	managedTransaction.global	Component runs in propagated transaction if present, otherwise a new global transaction. This combination is used for the **managed, shared global transaction** pattern described in Common Transaction Patterns.
propagatesTransaction	managedTransaction.local or noManagedTransaction	This is an "incompatible deployment" Error
suspendsTransaction	managedTransaction.global	Component runs in a new global transaction
suspendsTransaction	managedTransaction.local	Component runs in a managed local transaction containment. This combination is used for the **managed, local transaction** pattern described in Common Transaction Patterns. This is the default behavior for a runtime that does not support global transactions.
suspendsTransaction	noManagedTransaction	Component is responsible for managing its own local transactional resources.

Table 2 Combining service transaction intents

Note - the absence of either interaction or implementation intents leads to runtime-specific behavior. A runtime that supports global transaction coordination may choose to provide a default behavior that is the managed, shared global transaction pattern.

In the case where the **propagatesTransaction** intent conflicts with the component's ***managedTransaction.local*** intent, an appropriate error message must be issued at deployment. SCA tooling may also detect the error earlier in the development process.

1.5.2 Handling Outbound Transaction Context

The mutually exclusive ***propagatesTransaction*** and ***suspendsTransaction*** intents may also be attached to an sca:reference XML element to describe whether any client transaction context should be propagated to a target service when a synchronous interaction occurs through the reference. These intents specify a contract that MUST be implemented by the SCA runtime. This aspect of a service component is most likely captured during application design. Either the ***propagatesTransaction*** or ***suspendsTransaction*** intent can be attached to sca:service

elements and their children but a deployment error will occur if both intents are specified. The intents are defined as defined in Section 1.5.1. When used as a reference interaction intent, the meaning of the qualifiers is as follows:

- **propagatesTransaction** – any transaction context under which the client runs will be propagated when the reference is used for a request-response interaction. To satisfy policy framework *compatible wire* rules, the reference binding MUST be capable of propagating a transaction context. The reference should be wired to a service that provides this intent and thus will join a client's transaction. The reference consumer can then be designed to rely on the work of the target service being included in the caller's transaction.
- **suspendsTransaction** – any transaction context under which the client runs will not be propagated when the reference is used. The reference consumer can use this intent to ensure that the work of the target service is not included in the caller's transaction. .

The absence of either interaction intent leads to runtime-specific behavior. The SCA runtime may or may not propagate any client transaction context to the referenced service, depending on the SCA runtime capability.

These intents are independent from the client's ***managedTransaction*** implementation intent. The combination of the interaction intent of a reference and the ***managedTransaction*** implementation policy of the containing component completely describes the transactional behavior of a client's invocation of a service. Table 3 summarizes the results of the combination of either of these interaction intents with the ***managedTransaction*** implementation policy of the containing component.

reference interaction intent	**managedTransaction (client implementation intent)**	**Results**
propagatesTransaction	managedTransaction.global	Target service runs in the client's transaction. This combination is used for the **managed, shared global transaction** pattern described in Common Transaction Patterns.
propagatesTransaction	managedTransaction.local or noManagedTransaction	This is an "incompatible deployment" Error
suspendsTransaction	Any value of managedTransaction	The target service will not run under the same transaction as any client transaction. This combination is used for the **managed, local transaction** pattern described in Common Transaction Patterns.

Table 3 Transaction propagation reference intents

Note - the absence of either interaction or implementation intents leads to runtime-specific behavior. A runtime that supports global transaction coordination may choose to provide a default behavior that is the managed, shared global transaction pattern.

In the case where the **propagatesTransaction** reference intent conflicts with the using component's ***managedTransaction.local*** intent, an appropriate error message must be issued at deployment. SCA tooling may also detect the error earlier in the development process.

Table 4 shows the valid combination of interaction and implementation intents on the client and service that result in a single global transaction being used when a client invokes a service through a reference.

managedTransaction (client implementation intent)	reference interaction intent	service interaction intent	managedTransaction (service implementation intent)
managedTransaction.global	propagatesTransaction	propagatesTransaction	managedTransaction.global

Table 4 Intents for end-to-end transaction propagation

Transaction context is never propagated on OneWay messages. The SCA runtime ignores ***propagatesTransaction*** for OneWay methods.

1.6 Example

The following example shows some of the transaction polices in use for an implementation.

```
<?xml version="1.0" encoding="UTF-8"?>
<componentType xmlns:sca=" http://www.osoa.org/xmlns/sca/1.0"
      requires="managedTransaction.global">

      <implementation.java class="com.acme.TransactionalComponent1"
            requires="managedTransaction.global">

      <service name="Service1" requires="propagatesTransaction">
            <interface />
      </service>

      <reference name="Reference1" requires="transactedOneWay">
            <interface />
      <reference>

      <implementation/>

</componentType>
```

2 Intent Definitions

The SCA Policy Framework specification defines an XML schema for defining abstract intents. The following XML snippet shows the intent definitions for the transaction policy domain.

2.1 Intent.xml snippet

```
<?xml version="1.0" encoding="ASCII"?>

<intents xmlns="http://www.osoa.org/xmlns/sca/1.0" >

  <intent name="managedTransaction" constrains="sca:implementation">
      <description>
            Used to indicate the transaction environment desired by a component
            implementation.
      </description>
  </intent>

  <intent name="managedTransaction.global" constrains="sca:implementation">
      <description>
            Used to indicate that a component implementation requires a managed
            global transaction.
      </description>
  </intent>

  <intent name="managedTransaction.local" constrains="sca:implementation">
      <description>
            Used to indicate that a component implementation requires a managed local
            transaction.
      </description>
  </intent>

  <intent name="noManagedTransaction" constrains="sca:implementation">
      <description>
            Used to indicate that a component implementation will manage its own
            transaction resources.
      </description>
  </intent>


  <intent name="propagatesTransaction" constrains="sca:binding">
      <description>
            Used to indicate that a reference will propagate any client transaction
            or that a service will be dispatched under any received transaction.
      </description>
  </intent>

  <intent name="suspendsTransaction" constrains="sca:binding">
      <description>
            Used to indicate that a reference will not propagate any client
            transaction or that a service will not be dispatched under any received
            transaction.
      </description>
  </intent>
```

```
364
365
366     <intent name="transactedOneWay" constrains="sca:binding">
367         <description>
368               Used to indicate that the component requires the SCA runtime to transact
369               OneWay send of messages as part of any client global transaction or
370               to transact oneWay message receipt as part of any service global
371               transaction.
372         </description>
373     </intent>
374
375     <intent name="immediateOneWay" constrains="sca:binding">
376         <description>
377               Used to indicate that the component requires the SCA runtime to process
378               the sending or receiving of OneWay messages immediately, regardless of
379               any transaction under which the sending/receiving component runs.
380         </description>
381     </intent>
382
383
384   </intents>
```

3 Issues:

- TX-1. This specification defines no intents that can be used to constrain behaviour as follows:

1. there is no reference intent that compels a target service to run under a client transaction
2. there is no service intent that compels a client to propagate a transaction context (a la EJB Mandatory transaction descriptor).

The authors of this spec believe we do not need such intents but wish to be clear that this is something we considered rather than overlooked.

- TX-2. SCA context – this proposal assumes that SCA components access transactional resource managers in some way. This proposal does not indicate how that happens, but supports 1) direct use of a resource manager, 2) abstract a RM as a component, and 3) abstract a RM as a binding. Make it clearer that *how* transaction is established and *how* resources managers are accessed are out of scope.
- TX-3. TODO: converge use of exceptions, faults, return codes in terminology
- TX-4. ISSUE: in managed local tran, cannot commit work and throw an exception (i.e there needs to be greater flexibility than: “if exception rollback else commit”).
- TX-5. Issue: Should the 4 intents for ManagedTransaction all really be qualifiers on a single intent, since there is no meaning for an unqualified “managedTransaction” intent. Perhaps they should be separate intents? As a variation, we could just remove ManagedTransaction.any and replace it with the unqualified ManagedTransaction. In this case the “none” case would still be a separate intent. Thus the intents would be:
  - ManagedTransaction (unqualified means any)
  - ManagedTransaction.global
  - ManagedTransaction.local
  - NoManagedTransaction

  Further discussion :We are trying to express an intent that has 4 distinct values that are mutually exclusive. We have proposed doing it like this:

  <intent name="managedTransaction.global" constrains=...>

  where the "global" part of the intent is one of a set of mutually exclusive values rather than a qualification of "managedTranasction".

  An alternative approach is to simply make these each distinct intents:

  <intent name="managedTransaction_global" constrains=...>

  i.e replace the ‘dot’ qualifier with an underscore or simply camel-case the “qualifier” part of the intent. RESOLVED in this draft.

  TX-6 TO DO. Add section on global trans; don’t assume familiarity with EJB. RESOLVED in this draft.
-
- TX-7. Issue raised by MR on June 26 2007: Do we need this implementation policy or can we remove it? RESOLVED in this draft.
- TX-8. Interaction policies are mutually exclusive and require additional details to be defined:
  (1) the policy f/w needs a syntax to define mtually exclusive intents
  (2) we need to define the behavior of “cascading intents “ i.e can a child element “reverse”

the intent of a parent and, if not, what does that mean for using a top-level element intent as a "default". RESOLVED in this draft.

- TX-9: "wire compatibility" rules only relate the binding to the reference and say nothing about the requirements on the target service. How do we (or should we) try to articulate the requirement for a target service to provide a compatible intent.
- TX-10: Need a mechanism to exclude suspendsTransaction intent from the selection of a binding or service.
- TX-11: Clarify the semantics of transactedOneway. RESOLVED in this draft.
- TX-12: There is no means for the service provider to indicate that it is capable of joining a propogated transaction without requiring the client to propogate a transaction. Note: It is possible for a binding implementation to declare capability (@provides) but not for a service provider. This same problem is noted as issue 251 against the Policy FW spec.

4 References

[1] SCA Assembly Model Specification v1.0
http://www.osoa.org/download/attachments/35/SCA_AssemblyModel_V100.pdf

[2] SCA Policy Framework v1.0
http://www.osoa.org/download/attachments/35/SCA_Policy_Framework_V100.pdf

[3] SCA Java Common Annotations and APIs
http://www.osoa.org/download/attachments/35/SCA_JavaAnnotationsAndAPIs_V100.pdf