WS-CAF in a Nutshell

Mark Little, Arjuna Technologies Ltd

What is WS-CAF?

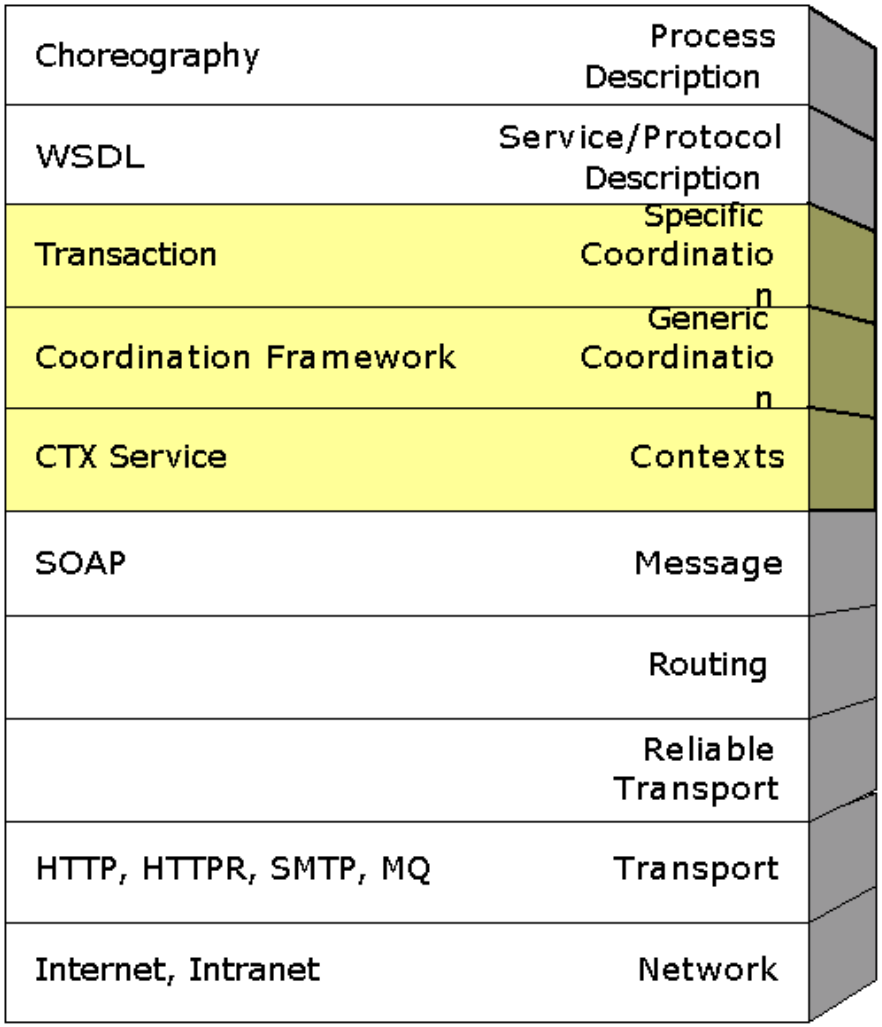
- Collection of 3 specifications
  - Designed to be used independently or together
- 18+ months of effort
- Based on implementation and user feedback
  - Extended transactions
    - OMG Activity Service
  - OASIS BTP experiences
  - Web services
    - Define an architecture

What are the specifications?

- WS-Context
  - Context service(!)
- WS-Coordination Framework
  - Framework for pluggable coordination protocols
- WS-Transaction Management
  - Three transaction models for Web services
    - Interoperability with existing implementations is important
  - Live document
    - Updated with new models as and when required

Hierarchy

arjuna
middleware for reliability



Interaction patterns

- All specifications mandate XML but not binding
  - Can use request/response or one-ways
- One-way messages are the default
  - Best suited to the asynchronous, loosely-coupled nature of Web services
    - Tolerate faults (outages)
    - Allow for redirections

Addressing

- Used own “home-brew” version of addressing
  - Basically URI and additional information
- Better to use something that is a standard
  - Continue to use own in interim

What is a context?

arjuna
middleware for reliability

- A way of doing correlation of messages
- For example
  - Web browser cookie
  - Single-session sign-on
  - Transactions
  - Database session identifier

WS-Context

arjuna
middleware for reliability

- Context and “life-cycle” service
  - Most fundamental aspect of WS architecture
- Defines notion of an activity
  - Unit of work
    - Precise definition left up to higher level services/users
  - Basic context associated with activity
- Context Service maintains context for each activity
  - May be co-located or separate service

Goals

arjuna
middleware for reliability

- To provide a basic context service for Web services
  - Lots of different services (and specifications!) need one
    - Better to standardise early!
- Provide ability to do correlation at a minimum
  - No augmentation of framework required to use it

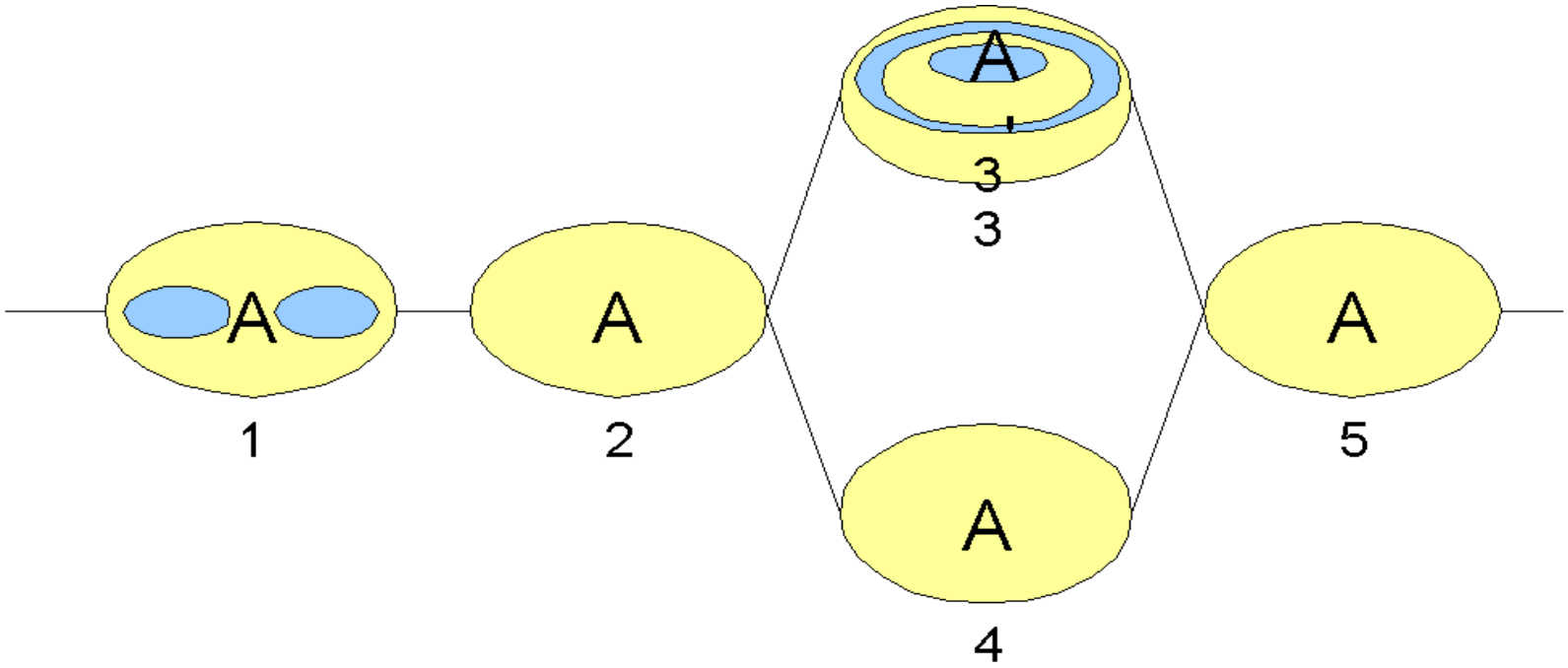
What are activities?

arjuna
middleware for reliability

- Scoping for arbitrary units of work
  - Created, made to run and then terminated
    - Two termination states
      - Success
      - Failure
  - Boundary of a context
- Can do anything in an activity
  - use the default context however services want
- Default context is essentially a UID (URI)
  - Just for correlation

Activity example

arjuna
middleware for reliability



Context specifics

- Context is a first-class element
  - URI which may represent a web resource
- Basic context contains
  - Unique activity id for each activity
  - Timeout period (lifetime of activity)
- May be augmented:
  - Dynamically as remote invocations progress
  - Before application invocation occurs
    - By calling Activity Lifecycle Services (ALS)

Requirements for a context service

- Service definition for start and end of activity
  - Context bound to activity
- Enable services to receive contexts
  - Propagation by reference or value
    - Contexts should be first-class entities (services)
- Enable services to augment contexts
  - Coordination, security, transactions, …

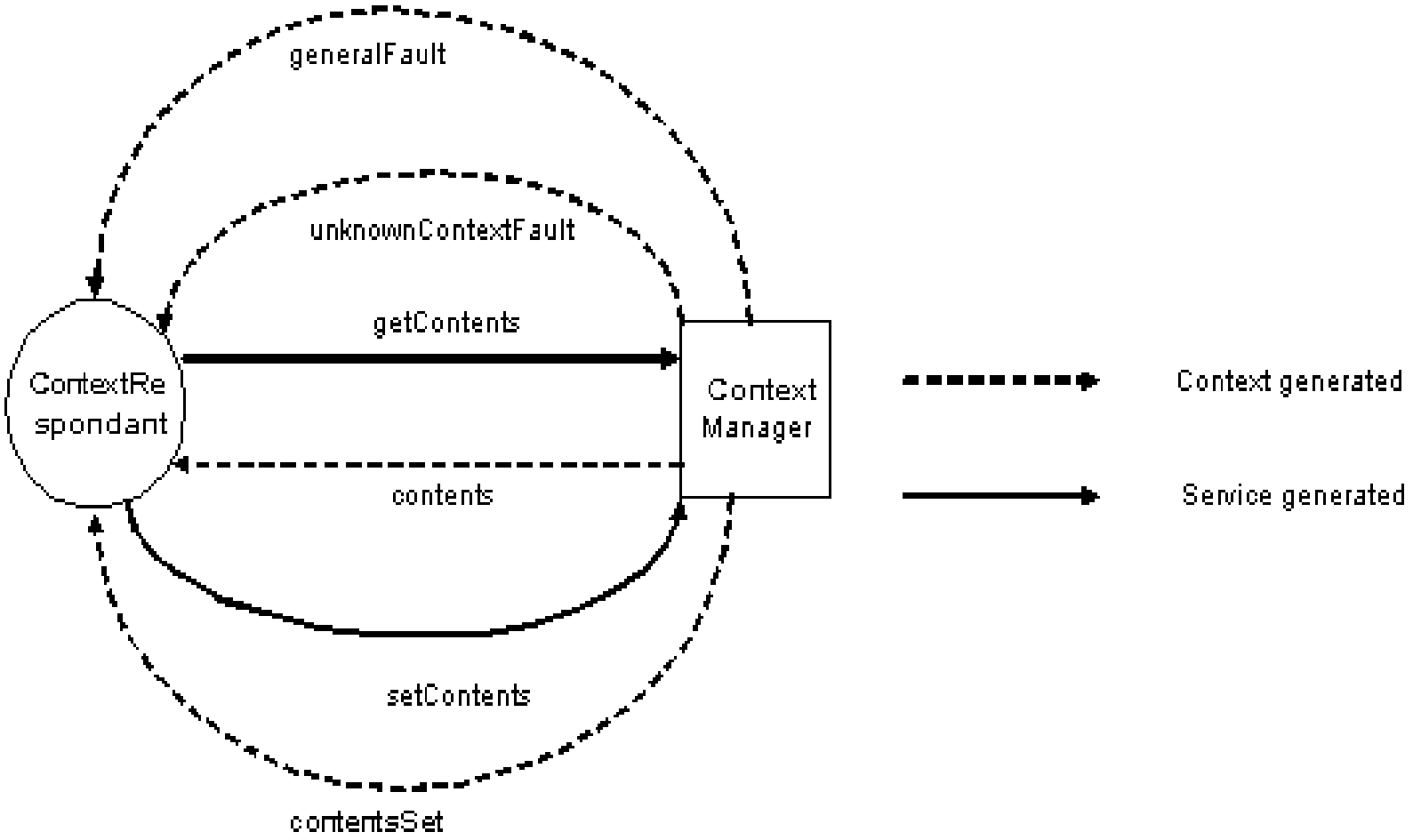
Propagation

arjuna
middleware for reliability

- Context information propagated in SOAP header
  - Minimally defined by a URI
    - In which case this is a service reference
- Context definer specifies whether it must be understood by receiver
  - MustPropagate

Context manager

arjuna
middleware for reliability

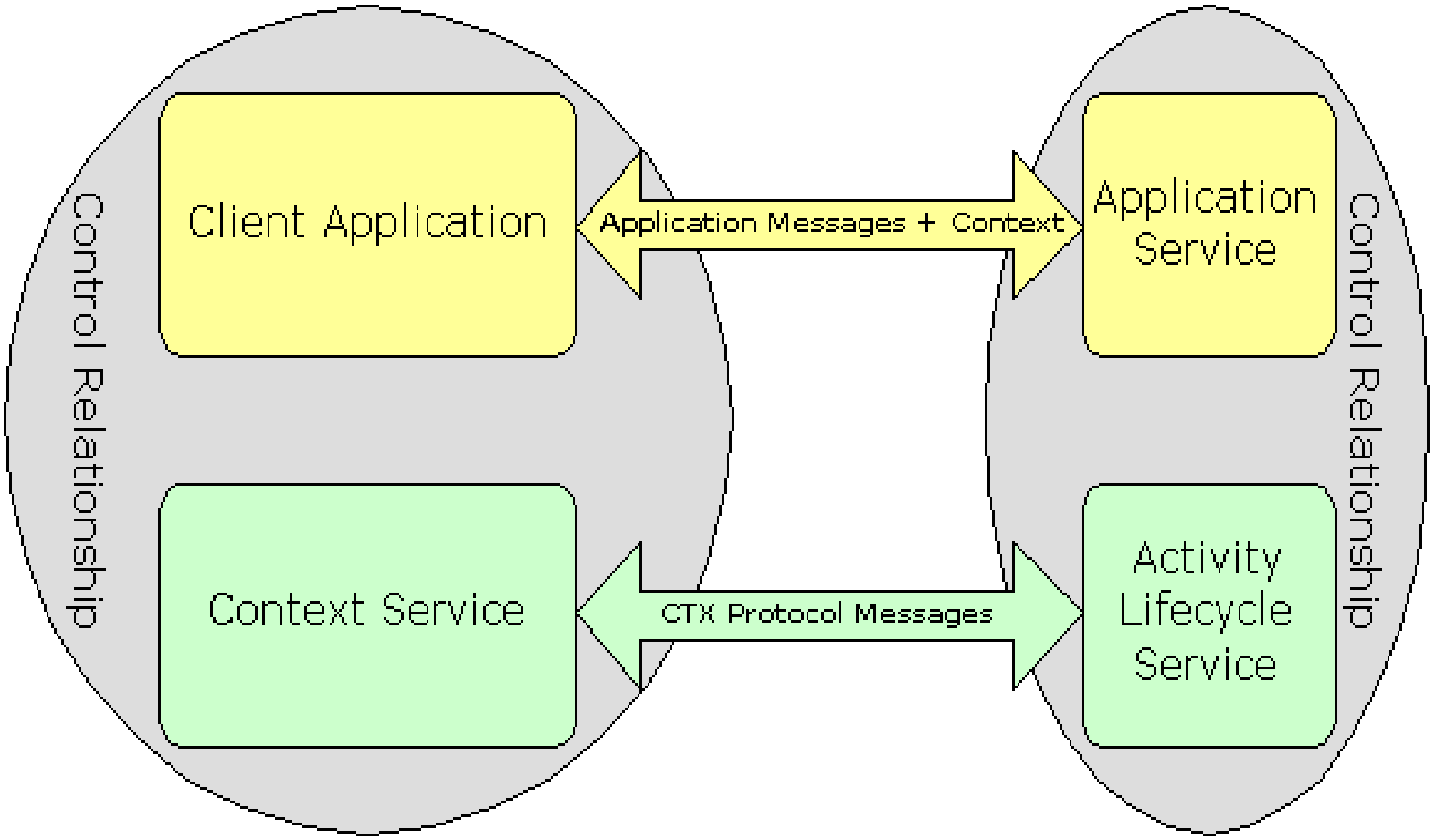


Categories of Services

- Application Service (AS)
  - Sends and receives contexts
- Activity Lifecycle Service (ALS)
  - Context augmentation Service
    - Used to modify the context
      - E.g., security service
  - Trigger actions when activity starts/ends
  - One Service, but perhaps two?

Example

arjuna
middleware for reliability



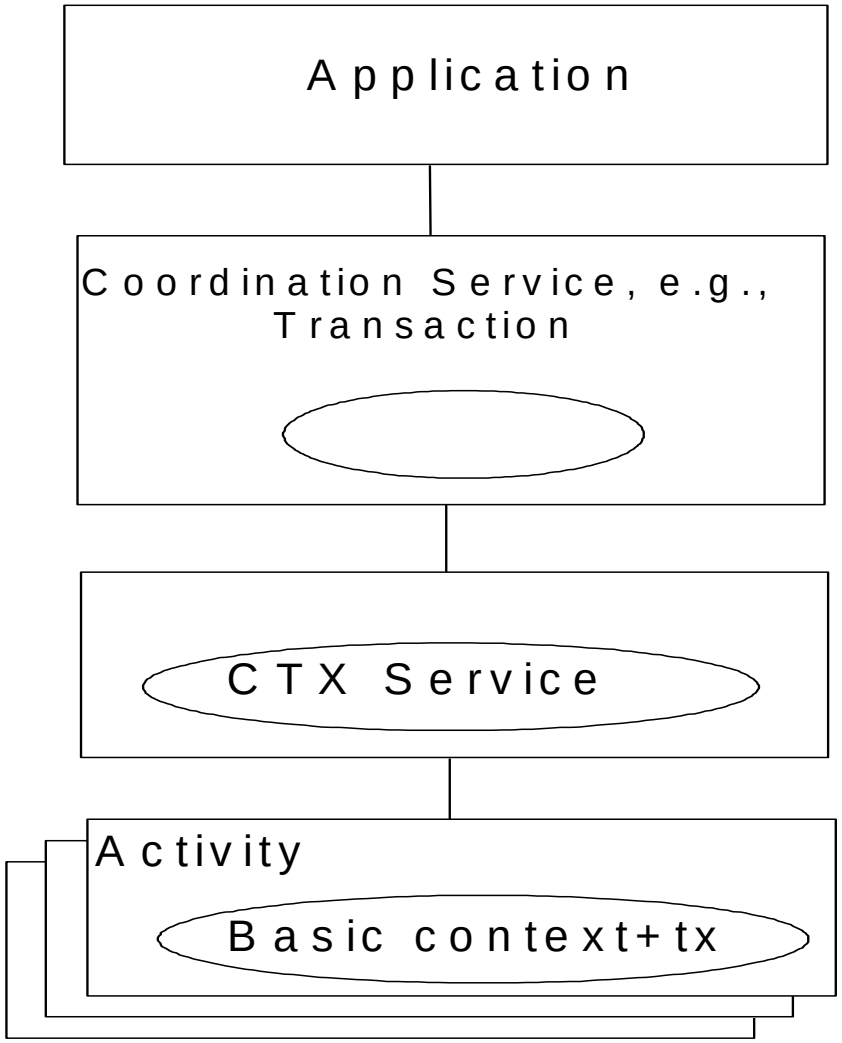
ALS

arjuna
middleware for reliability

- Services may register to participate in lifecycle of activity
  - Informed when begun and when ending
- Are given a chance to augment context before application invocation
  - Not called prior to each invocation, only on lifecycle “events”
- Example
  - Coordination
  - Security

ALS example

arjuna
middleware for reliability



Context schema


```
<xs:complexType name="ContextType">
  <xs:sequence>
    <xs:element name="context-identifier" type="xs:anyURI"/>
    <xs:element name="activity-service" type="xs:anyURI" minOccurs="0"/>
    <xs:element name="type" type="xs:anyURI" minOccurs="0"/>
    <xs:element name="activity-list" minOccurs="0">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="service" type="xs:anyURI" minOccurs="0"
            maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="mustUnderstand" type="xs:boolean" use="optional"
          default="false"/>
        <xs:attribute name="mustPropagate" type="xs:boolean" use="optional"
          default="false"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="child-contexts" minOccurs="0">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="child-context" type="tns:ContextType"
            maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="timeout" type="xs:int" use="optional"/>
</xs:complexType>
```

Contains

- Context identifier (URI)
- *Context service URI*
  - activity-service (rename)
- *An identifier that indicates the type of the activity*
- *An list of the services currently participating in the activity*
  - participating-services
- *A list of child activities*
  - child-contexts
- *A timeout value*
- *Extensibility element*
  - ALS information

Summary

arjuna
middleware for reliability

- Context is a building block
- Basic context should be useful in its own right
  - Correlation
- Should work with existing contexts and services
  - WS-BPEL
  - Security

WS-CF

arjuna
middleware for reliability

- Coordination is more fundamental than transactions
  - Security
  - Replication
  - Transactions
  - Caching
  - Process-flow

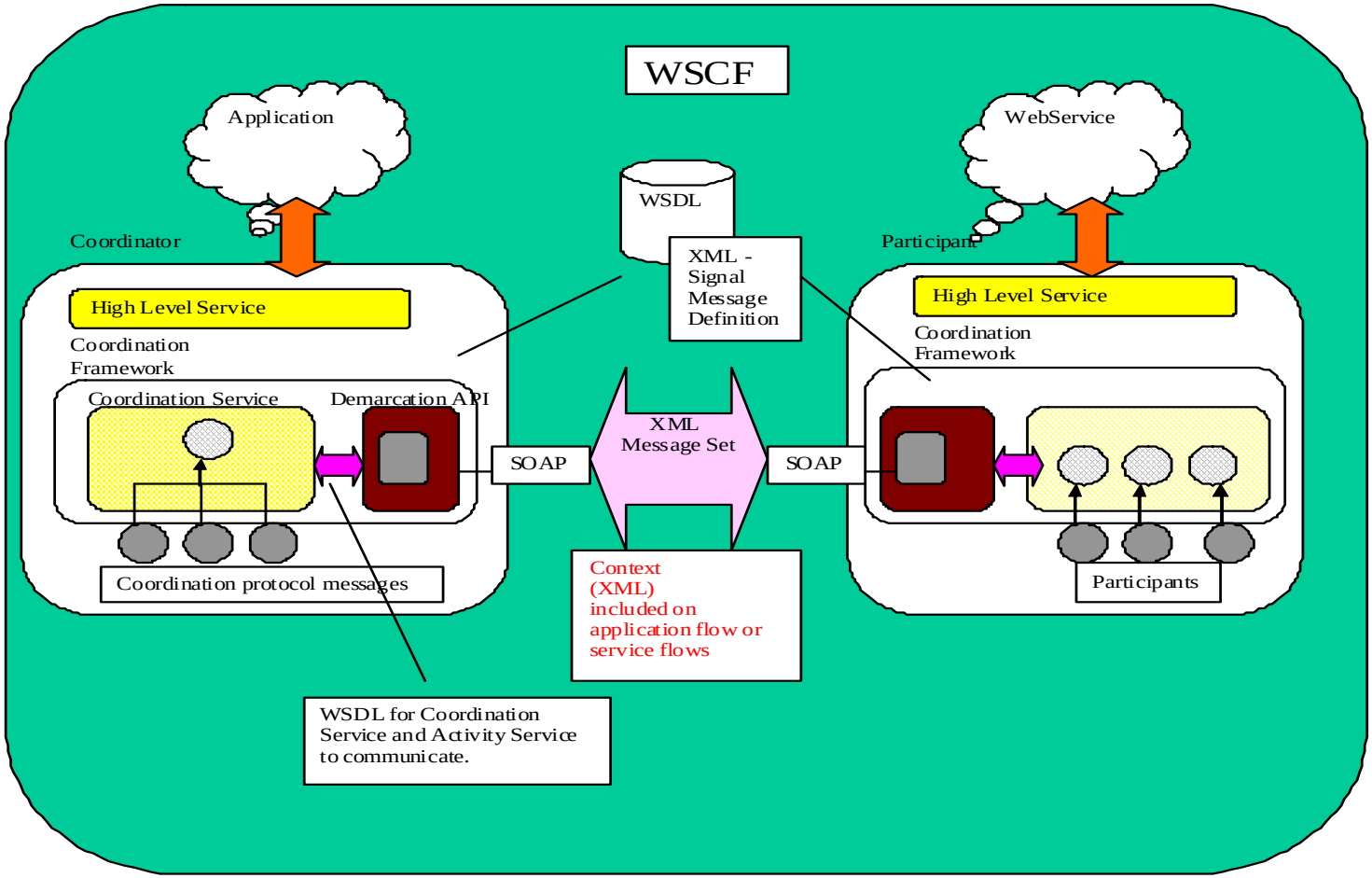
Goals

arjuna
middleware for reliability

- Provide a general framework for coordination protocols
  - Existing implementations to be plugged in
  - New implementations can be supported
    - Defines coordinator and participant relationships
- Work with WS-Context
  - Define an appropriate ALS
  - Augment context
- Scope of activity becomes scope of coordination boundary

Architecture

arjuna
middleware for reliability



Coordination protocol

- Defined by the message interactions between coordinator and participant
- WS-CF is protocol neutral
- Protocols uniquely identified
  - URI
  - Something else has to manage/impose the semantic meaning of a specific protocol identifier

What does it define?

- The structure of the augmented context
  - Coordination service *must* register with Activity Service as an ALS
- Coordinator and participants
  - Basic interaction protocol
  - Not mandated
    - Can plug in other implementations
- Must be able to leverage existing infrastructure

Mandatory

arjuna
middleware for reliability

- The Coordination ALS
  - So that the context can be augmented
    - Registration service for participants
    - Implementation specific data
- Context service informs ALS at start/end of activity
  - Coordinator-to-participant protocol is not defined at all

Context type

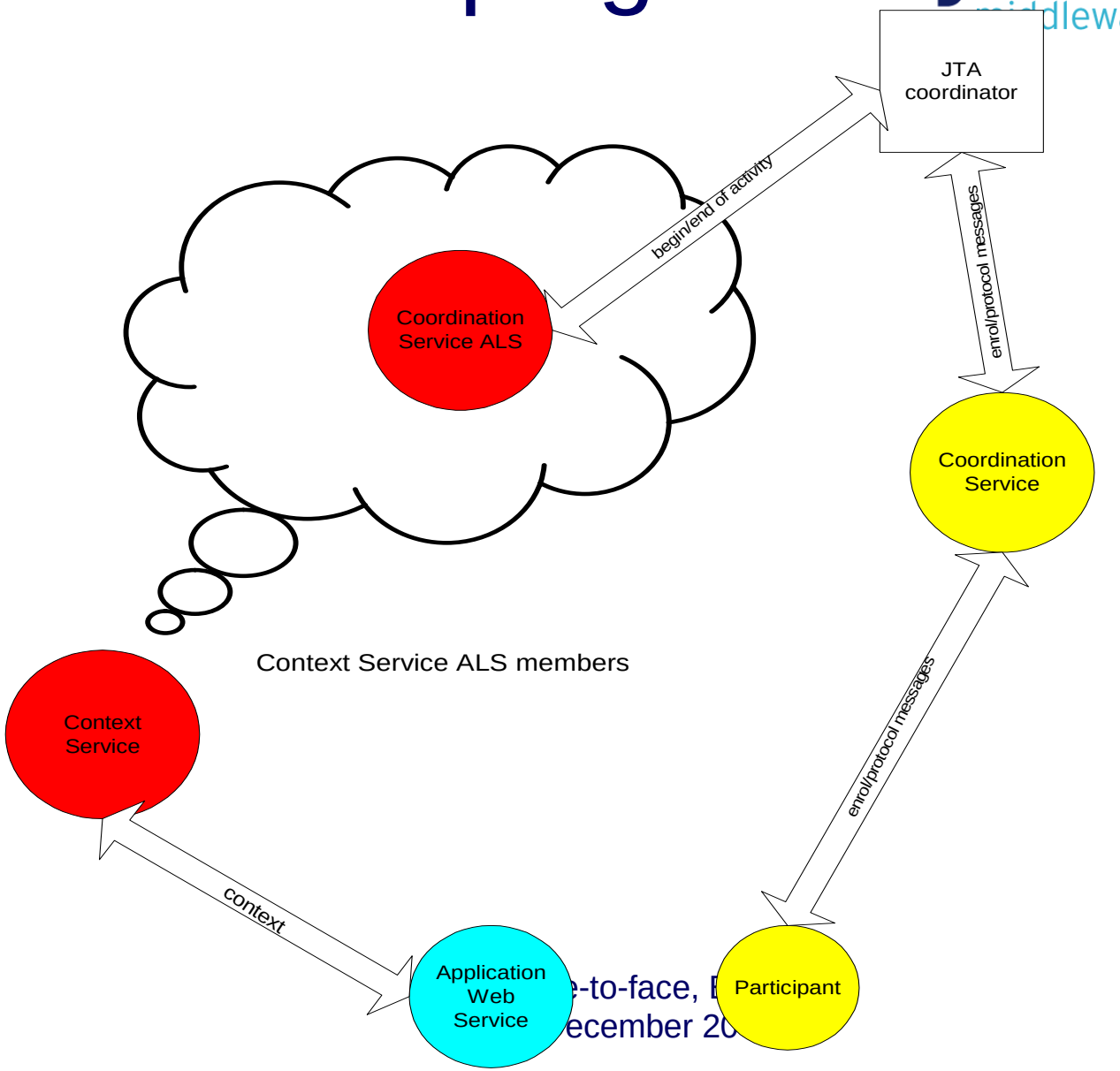

```
<xs:complexType name="ContextType">
  <xs:complexContent>
    <xs:extension base="wsctx:ContextType">
      <xs:sequence>
      <xs:element name="protocol-reference" type="tns:ProtocolReferenceType"/>
      <xs:element name="coordinator-reference" type="tns:CoordinatorReferenceType"
        maxOccurs="unbounded"/>
      <xs:any namespace="##any" processContents="lax" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Optional

arjuna
middleware for reliability

- The Coordinator Service
  - Provides a participant registration endpoint
  - Coordination status and identity
  - Can be driven at arbitrary points during activity
- Participant
  - The operations performed as part of coordination sequence processing
    - Works in terms of AssertionTypes
  - status

WS-Context plug-in



AssertionType

- “Base class” for all coordinator-to-participant message interactions
  - Requests and responses
- All protocol specific messages enhance this type
- One service (participant or coordinator) can accept multiple protocols
  - Bridging

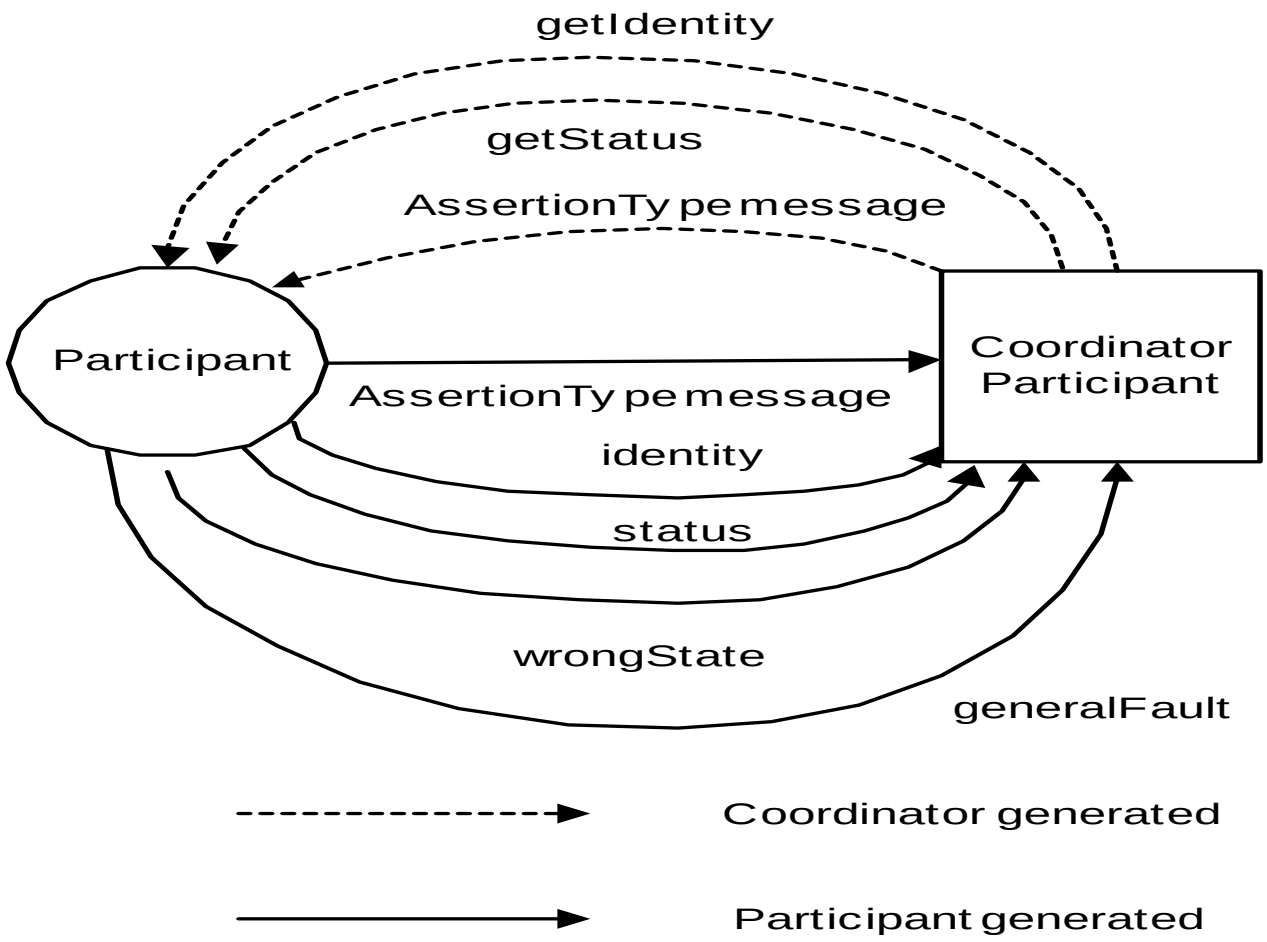
Operation basics

- Participant accepts
  - *getStatus*, *AssertionType* and *getIdentity*
  - AssertionType is the key to extensibility
    - All protocol messages are AssertionTypes
    - Think of it as a pure-virtual base class/interface
  - The coordinator and participant agree on the format of AssertionTypes
    - Out of scope for WS-CF

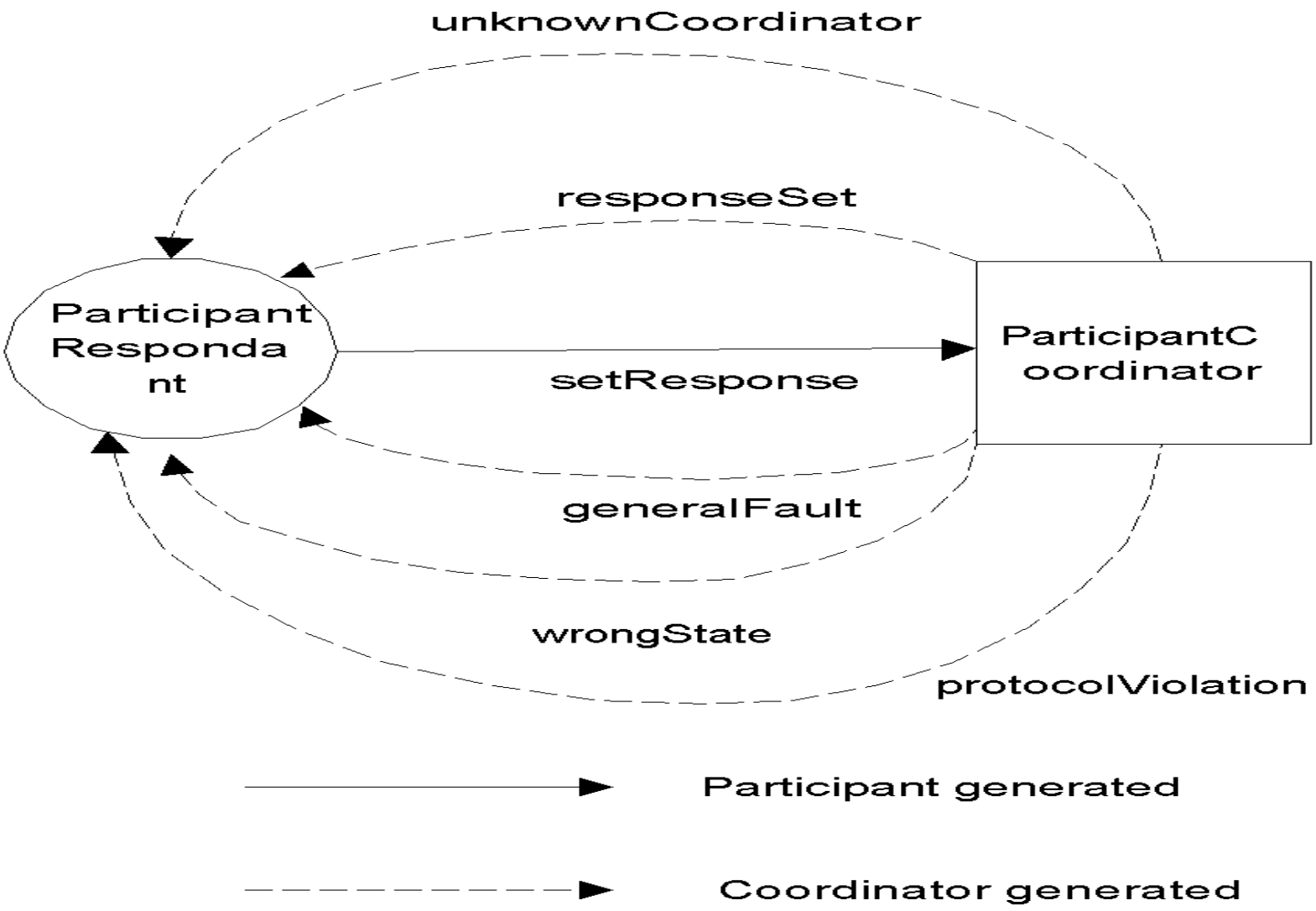
Operation basics

- The Coordinator accepts
  - *status*, *AssertionType*, *identity*, *wrongState* and *generalFault*
  - Participant can make asynchronous calls to coordinator (*setResponse*)
  - *addParticipant/removeParticipant* and a few others

Coordinator-to-participant



Participant-to-coordinator



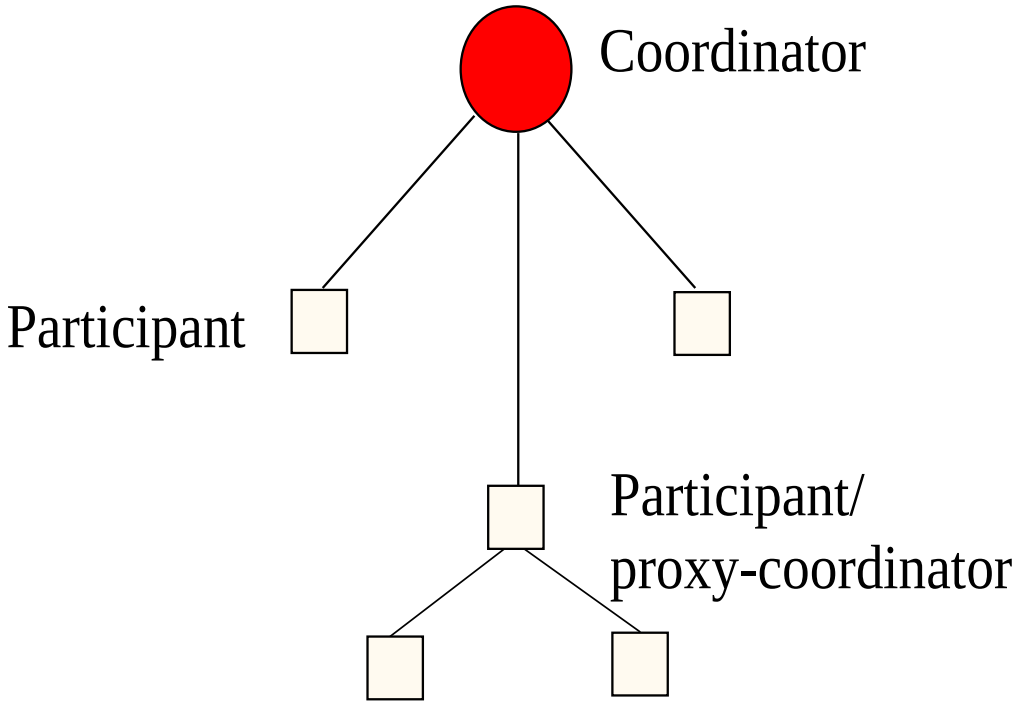
Qualifiers

- Runtime protocol extensibility option
- Typically in enlist/delist
  - For coordinator/participant information
  - E.g., will cancel in 24 hours

Interposition

- Important for security and performance reasons
  - Part of most distributed transaction protocols
- Subordinate coordinator
  - Participant as far as coordinator is concerned
  - Coordinator as far as participant is concerned
- Supported by WS-CF
  - Not mandated

Example



Recovery

arjuna
middleware for reliability

- Distributed application federated into natural recovery (admin) domains
  - Can’t mandate one specific recovery mechanism
    - Very application specific anyway
  - Have to allow administrative domains autonomy
- Therefore, support but not mandate

Recovery support

- RecoveryCoordinator
  - Drives recovery on behalf of participant
  - Participants may not be able to recover at same URI
    - Machine crash, domain migration, …
- Coordinator can replace one endpoint with another to continue protocol

But …

arjuna
middleware for reliability

- Very participant driven
- What if coordinator moves?
  - Out of scope
  - Implementation specific
    - Context could be augmented with alternate endpoints

WS-TXM

- Transactions for Web services
- Builds on WS-CF and WS-Context
- Based on experience of using Web service transactions
  - OASIS BTP
  - Role-your-own
- Intended as a live document
  - New models can be added as required
- Scope of activity becomes scope of transaction

Goals

arjuna
middleware for reliability

- Support range of use cases
- “One-size does not fit all”
  - Therefore a single protocol cannot cope with all requirements
  - All requirements aren’t “two-phase”
    - BT
      - Weighted voting for weaker consistency
    - PMI
      - Three phase

Where does it fit?

- Web Services are as much about interoperability as they are about the Web
- In the short term Web Services transactions will be about interoperability between existing TP systems rather than running transactions over the Web
- Companies aren't going to discard existing investments
  - Need to use what they've paid for!

Defines

- Three transaction models
  - ACID transaction
    - For interoperability and high-cost services where ACID transactions are a requirement
  - Long running action
    - Loosely coupled, long duration work that uses compensations
  - Business process
    - For treating all steps in an automated business process as part of a single logical transaction

Relationship to WS-CF

- Builds on WS-CF
- All protocols define all optional parts of that specification
  - Use AssertionTypes
- Augment context

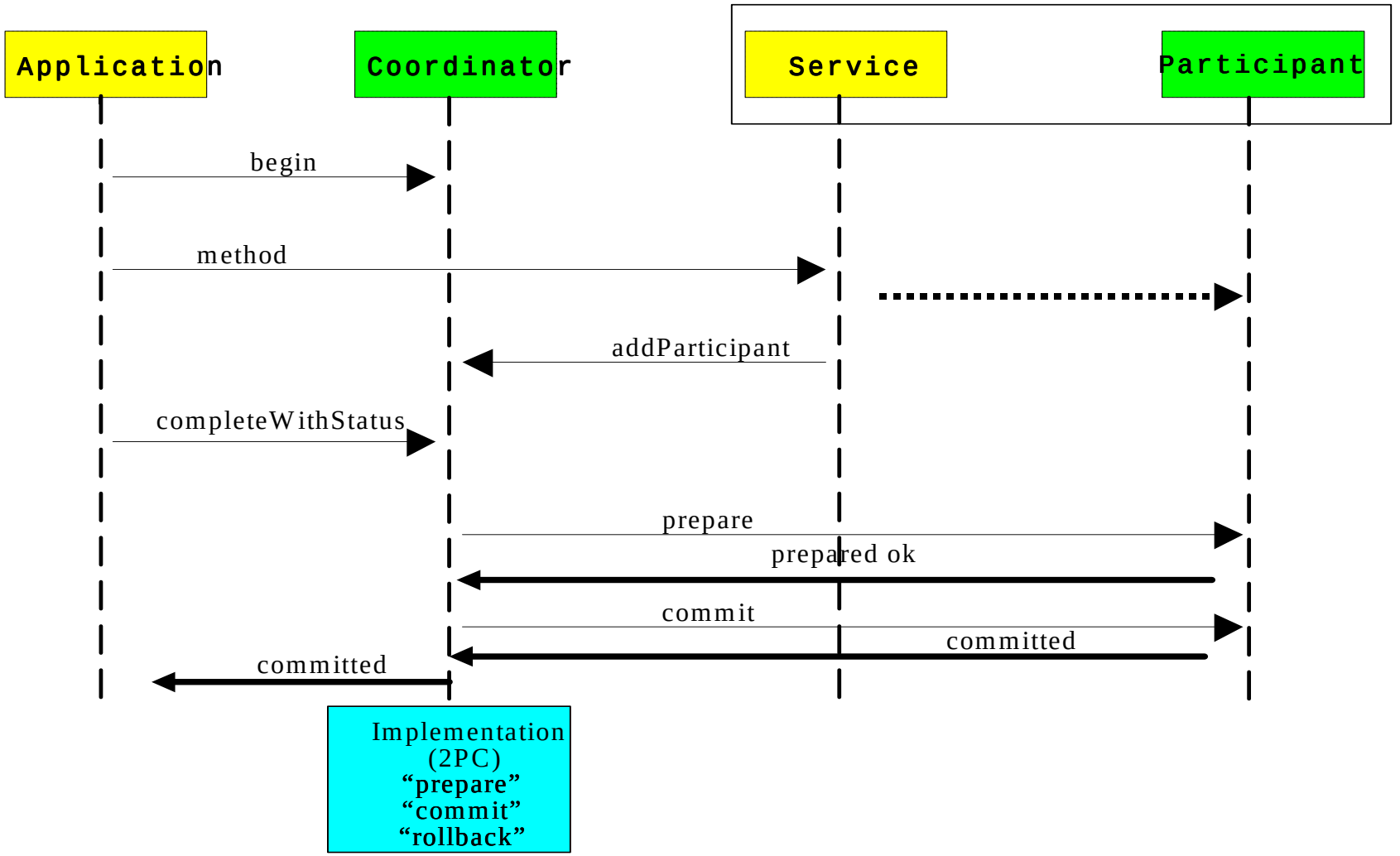
ACID Transaction

- Traditional *ACID transaction* with two sub-protocols (different URIs)
  - Two-phase commit
  - Synchronizations
- Interoperability across different vendor implementations
  - *removeParticipant* illegal
  - *coordinate* cannot be used

Model

- ACID semantics explicitly required
- Presumed rollback
- One-phase optimization
- Read-only optimization
- Heuristics

Example


Application
Coordinator
Service
Participant
begin
method
addParticipant
completeWithStatus
prepare
prepared ok
commit
committed
committed
Implementation
(2PC)
“prepare”
“commit”
“rollback”

2PC protocol messages

- Usual suspects for two-phase commit
  - prepare
    - voteReadonly, voteCommit, voteRollback
    - And heuristics
  - commit
    - Heuristics
  - rollback
    - Heuristics

Synchronization messages

- beforeCompletion
  - Runs before two-phase commit begins
- afterCompletion
  - Runs after two-phase protocol

Long running action model

- Specifically for long duration interactions
- ACID transactions are not appropriate
  - Can't lock resources for the duration
  - No assumed trust relationships
- Compensation actions used
  - Forward work to return the business state to consistency
    - E.g., credit your credit card and give you back interest payments

Relationship to context and coordination

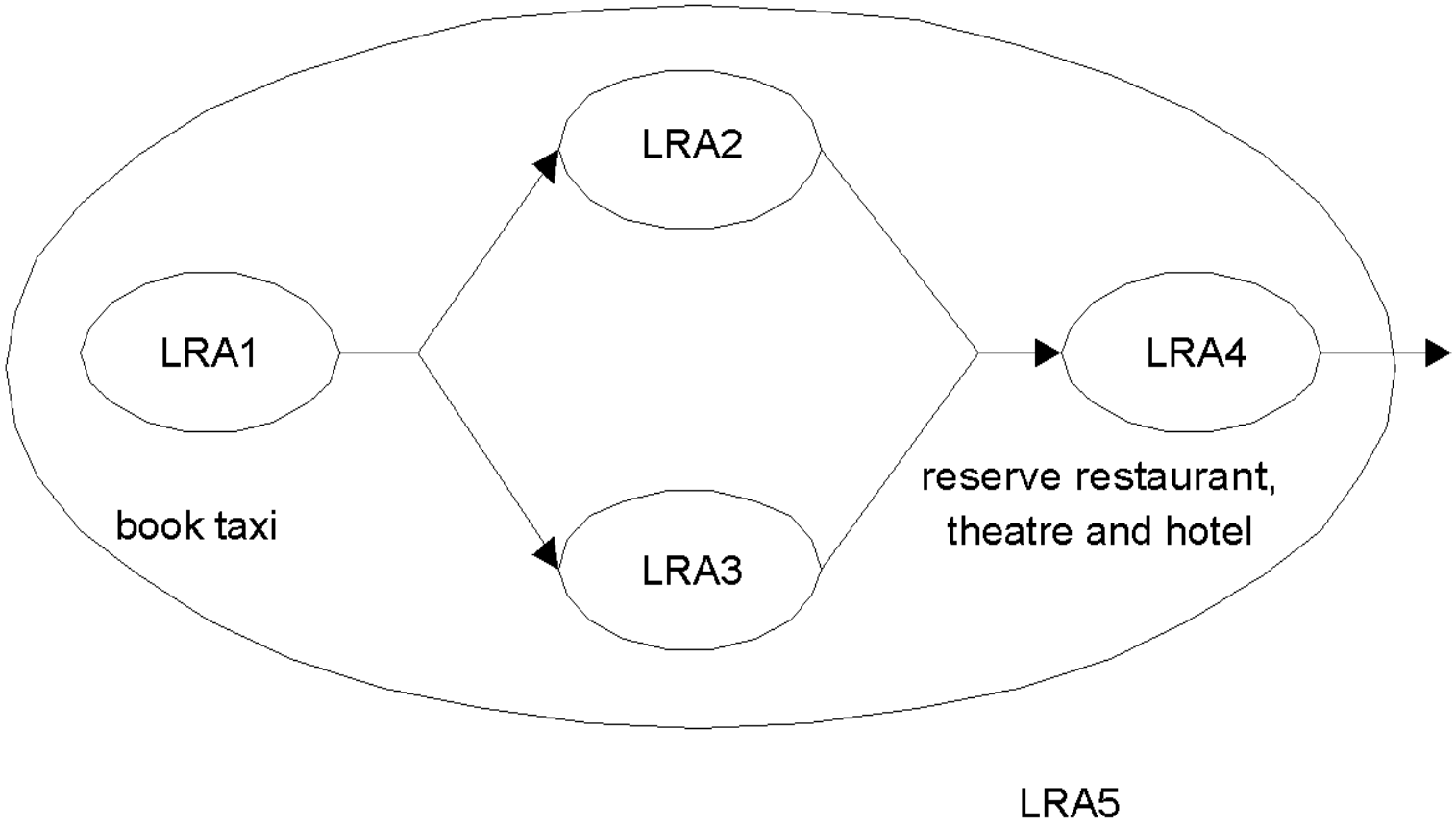
- Activity becomes the scope of business interactions
  - Travel agent, computer construction, etc.
- How services do work is not important
  - Back-end implementation choice
- If work can be compensated then compensation is bound to the activity
  - Non-atomic behaviour
- Activities can be nested

Activities and compensators

- Each activity is a unit of compensatable work
- Work performed must remain compensatable for duration of activity
  - Heuristics again!
  - TimeLimit qualifier
- Nested activities imply nested compensators
  - Could be different compensator from child to parent

Travel agent example

arjuna
middleware for reliability



Context

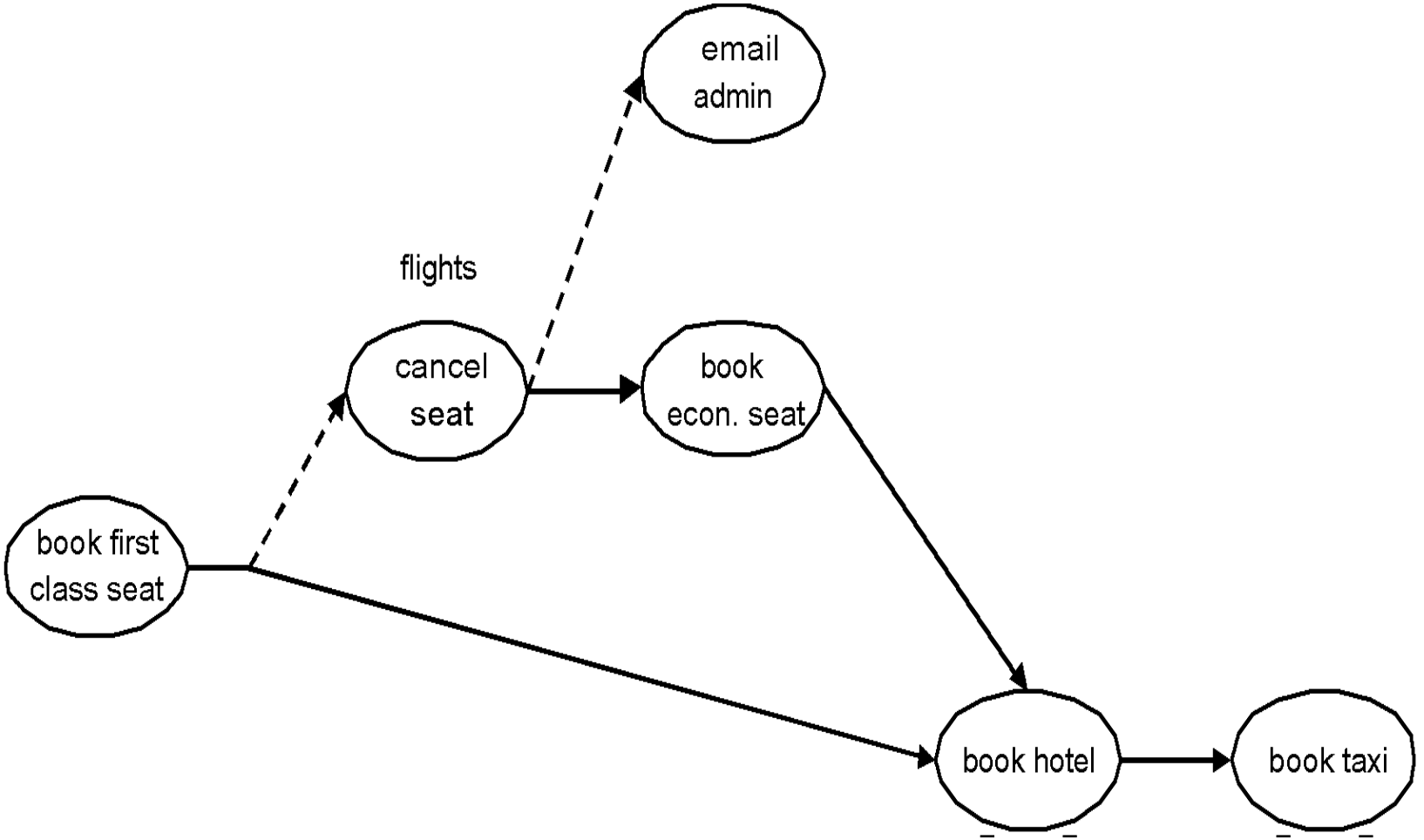

```
<xs:element name="context">
 <xs:complexType>
  <xs:complexContent>
   <xs:extension base="wstxm:ContextType">
    <xs:sequence>
     <xs:element name="lra-id" type="xs:anyURI"/>
     <xs:element name="coordinator-hierarchy">
      <xs:complexType>
       <xs:sequence>
        <xs:element name="coordinator-location" type="xs:anyURI" minOccurs="0"
         maxOccurs="unbounded"/>
       </xs:sequence>
      </xs:complexType>
     </xs:element>
    </xs:sequence>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>
</xs:element>
```

To compensate or not?

- Some services may not be able to compensate
- The user defines whether or not this is important
  - MustUnderstand
- Must be an explicit choice
  - Adverse consequences otherwise

Compensators

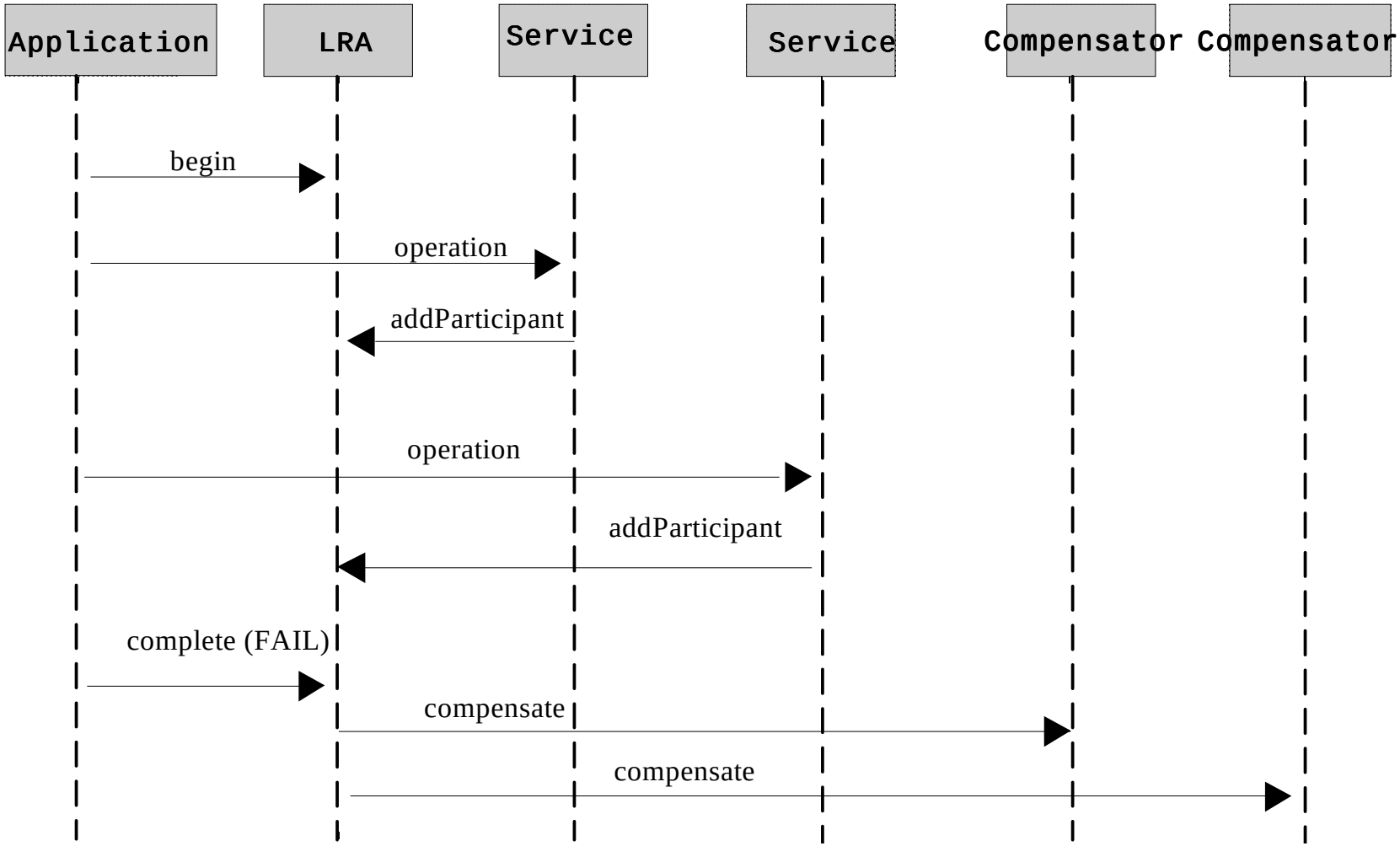
arjuna
middleware for reliability



Requirements on Services

- Log sufficient information to compensate
  - Best effort only
- For failure recovery too

Example


Application
LRA
Service
Service
Compensator
Compensator
begin
operation
addParticipant
operation
addParticipant
complete (FAIL)
compensate
compensate

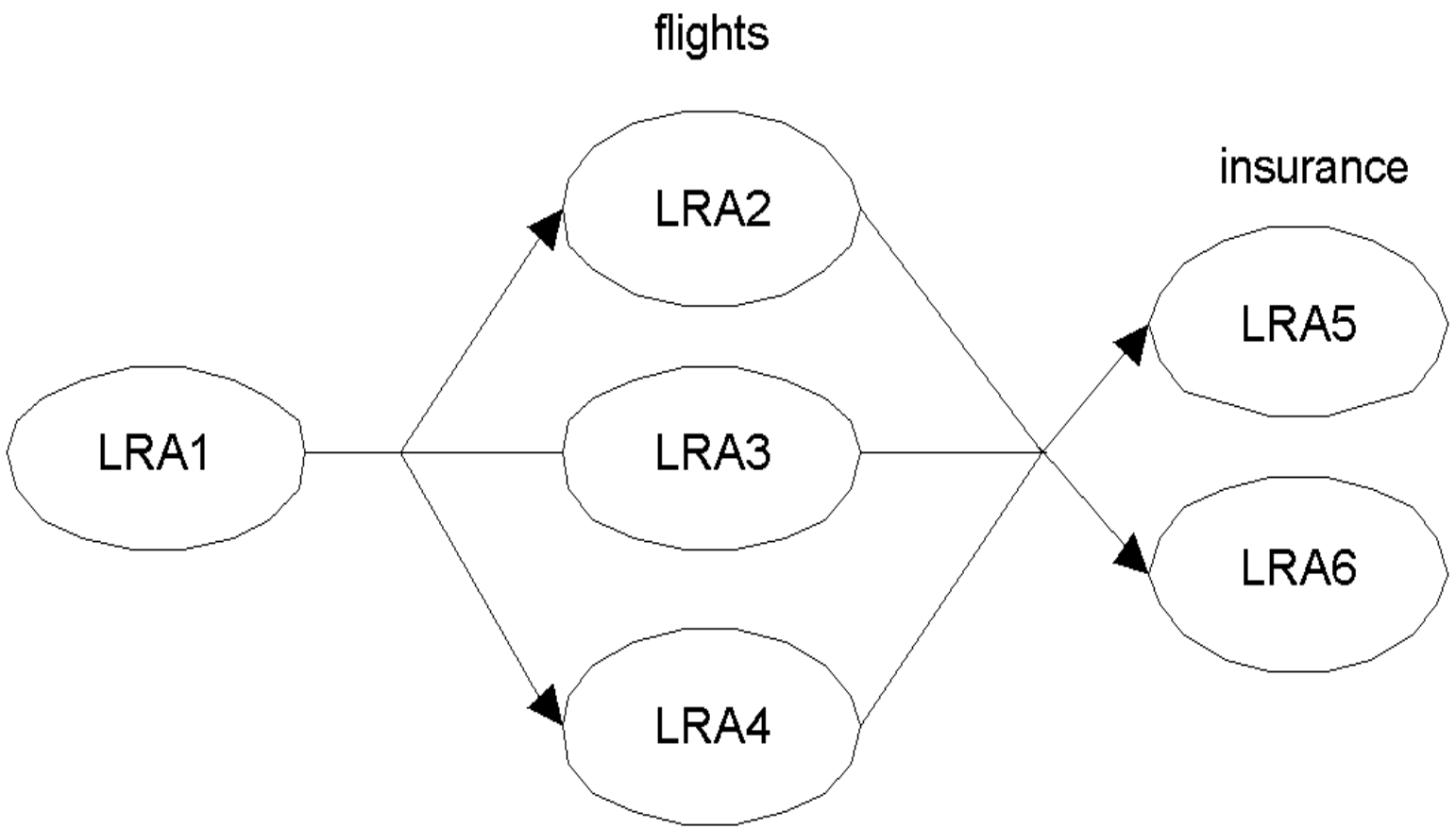
Concurrent LRAs

arjuna
middleware for reliability

- Activities can be concurrent
  - Therefore, LRAs can be too
- Allows selection of work within overall activity
  - E.g., choosing the cheapest flight

Selection of work

arjuna
middleware for reliability



Business process model

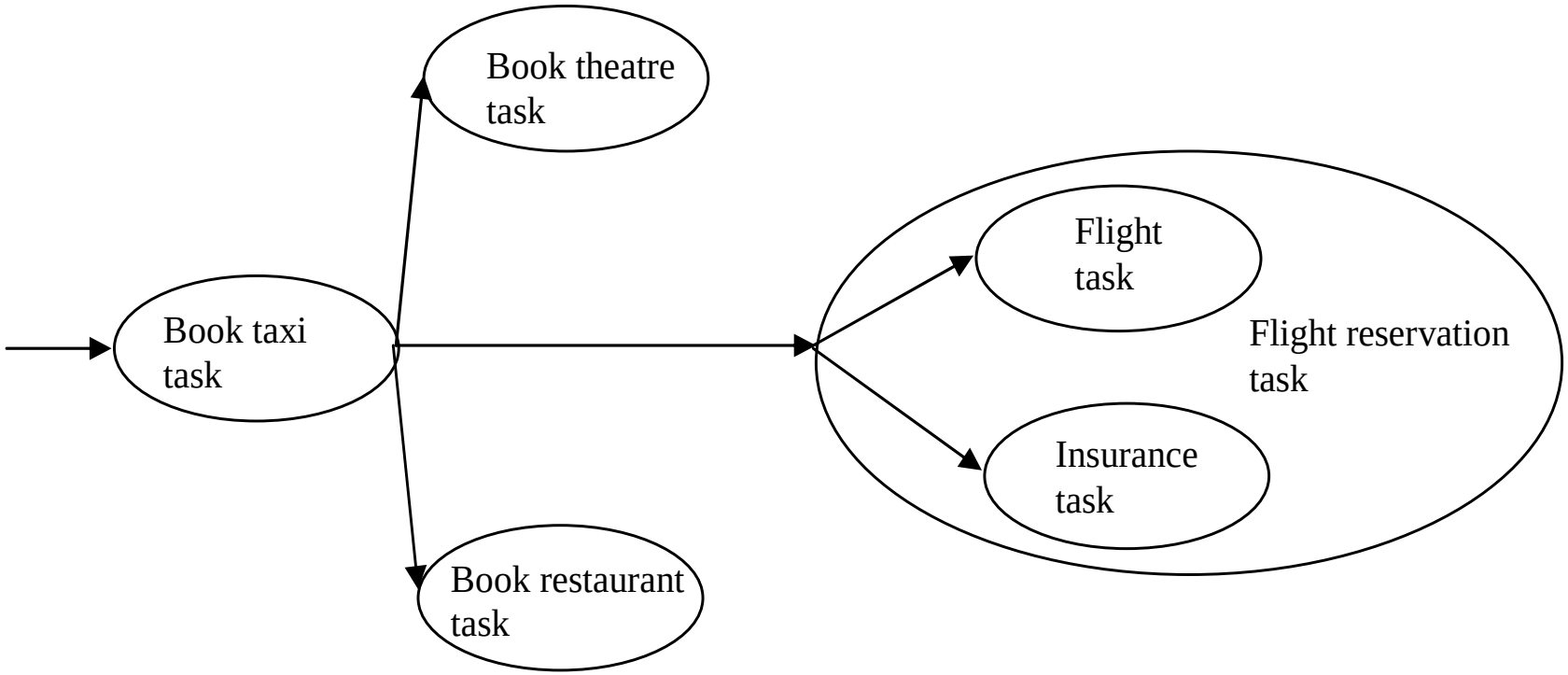
- Aimed at *long running* interactions that span different domains *and* models
  - Workflow
  - Messaging
  - Traditional ACID transactions
- Federated systems that can't/won't expose back-end implementations

Domains

arjuna
middleware for reliability

- All parties reside within *business domains*
  - Recursive structure is allowed
  - May represent a different transaction model
- Business process is split into *business tasks*
  - Execute within domains
  - Compensatable units of work
    - Forward compensation during activity is allowed
      - Keep business process making forward progress

Travel agent



Model

arjuna
middleware for reliability

- Supports synchronous and asynchronous interactions
  - Users can submit work and call back later
  - Or interact synchronously (traditionally)
- Business Process manager
- Optimistic rather than pessimistic
  - Assumes failures are rare and can be handled offline if necessary
- Pragmatic approach

How does it work?

- Each domain is exposed as a subordinate coordinator
  - Responsible for mapping incoming BP requests to domain specific protocol
- Protocol messages
  - checkpoint, confirm, cancel, restart, workStatus

checkpoint/restart

- Application driven
  - E.g., via *coordinate* message
- Checkpoints are created across the domains
  - Uniquely identified
- Domains can then roll back to specific checkpoint

workstatus

- Domain calls back to coordinator to inform it of final status
- Or application can enquire
  - WorkCancelled
  - WorkCompleted
  - WorkProcessing

confirm/cancel

arjuna
middleware for reliability

- BP termination protocol messages
- Map to success/failure of activity
- Because long-running, heuristics may occur
  - Mixed responses from domains
  - Sufficient information to allow administrative handling

Conclusions

- Phew!
- Each specification offers useful functionality
- Some overlap with other proprietary specifications
- Good feedback from developers and potential users