WS-Coordination and WS-Transactions

Transactions

- Distributed system
  - Reliability problems
  - Subject to independent failure of any of its components
  - Decentralization allows:
    - Parts of the system fail
    - Other parts remain functioning
    - Possibility of abnormal behaviors

ACID Properties

- ***Atomicity:*** The transaction completes successfully (commits) or if it fails (aborts) all of its effects are undone (rolled back)
- ***Consistency:*** Transactions produce consistent results and preserve application specific invariants
- ***Isolation:*** Intermediate states produced while a transaction is executing are not visible to other transactions
  - appear to execute serially
  - achieved by locking resources
- ***Durability:*** The effects of a committed transaction are never lost (except by a catastrophic failure)

Atomic Transactions

- can be terminated in two ways:
  - *Committed*
    - *all changes made within it are made durable*
  - *Aborted* (rolled back)
    - all changes made during the lifetime of the transaction are undone

Web Environment

- Web Service activities form a unit of work, but ACID properties are not always appropriate since Web is loosely coupled and activities are frequently long-running
  - Communication delays
  - Server loading (and hence response time) is unpredictable
  - Locks might not be retained for duration of activity
  - Servers are controlled by different organizations
  - Modules might not trust each other
  - Failures more likely

Why Atomic transactions may be too strong

- Suitable for short lived applications
- Long-lived transactions may reduce the concurrency
  - By holding onto resources for a long time
  - If it aborts
    - Much valuable work already performed will be undone
- Pure ACID transactions are not suitable for Web Services

A Suit of Specifications

- As a response to these needs in July 2002, BEA, IBM, and Microsoft released a trio of specifications designed to support business transactions over Web services:
  - BPEL4WS (Business Process Execution Language for Web Services),
  - WS-C (WS-Coordination) and
  - WS-TX (WS-Transaction)

Coordination

- Refers to the mechanism used by the components of an activity to reach common agreement:
  - How are components identified?
  - How are exceptional situations to be handled?
    - System related failures (e.g., crash, communication failure)
    - Application related exceptions (e.g., unanticipated messages)
  - What constitutes successful termination?

WS-Coordination

- WS-Coordination defines a framework for providing protocols that coordinate the actions of distributed applications
  - Provides a generic framework for coordination protocols to be plugged in
- Provides only context management - it allows contexts to be created and activities to be registered with those contexts.

WS-Coordination

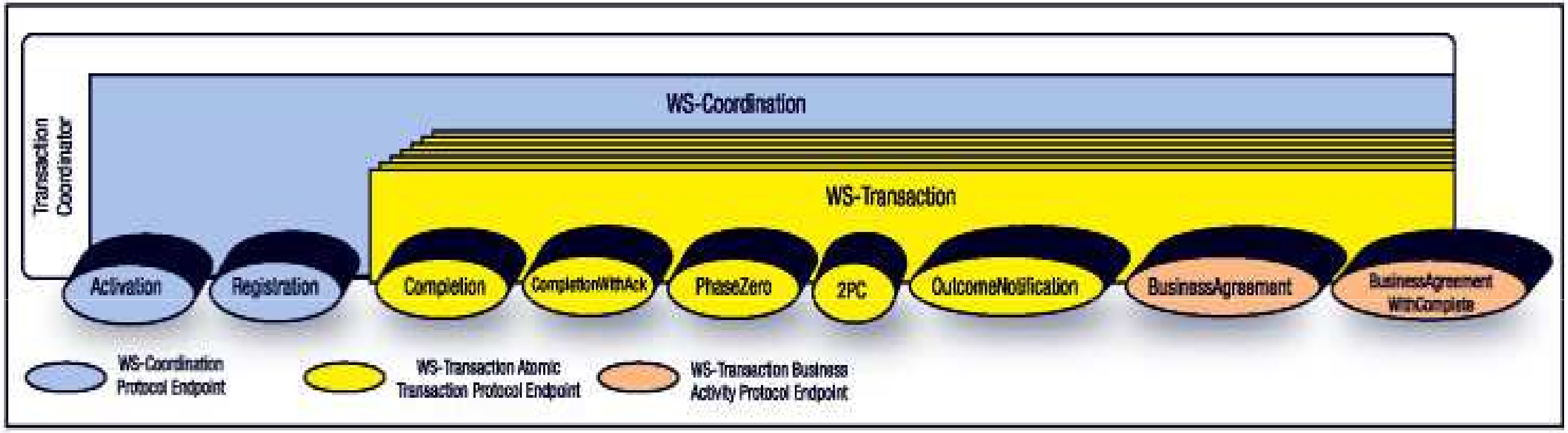

Transaction Coordinator
WS-Coordination
WS-Transaction
Activation
Registration
Completion
CompletionWithAck
PhaseZero
2PC
OutcomeNotification
BusinessAgreement
BusinessAgreement WithComplete
WS-Coordination Protocol Endpoint
WS-Transaction Atomic Transaction Protocol Endpoint
WS-Transaction Business Activity Protocol Endpoint

FIGURE 1 | WS-Coordination and WS-Transaction

Overview

- A coordinator supports three services:
  - **Activation Service:** part of **WS-Coordination**
    - Create a new activity with a particular CoordinationType: AtomicTransaction or BusinessActivity
      - Each type supports several termination protocols; a participant can choose the protocol appropriate to its role in the activity
  - **Registration Service**: part of **WS-Coordination**
    - Allow each participant to register for a protocol within the type
      - Different participants in the same activity might use different protocols
      - Applications register with a **coordinator** to create a **coordination context** that is **carried by all applications** within a given activity
  - **Protocol Service:** supports the execution of the protocol as specified in WS-AtomicTransaction or WS-BusinessActivity

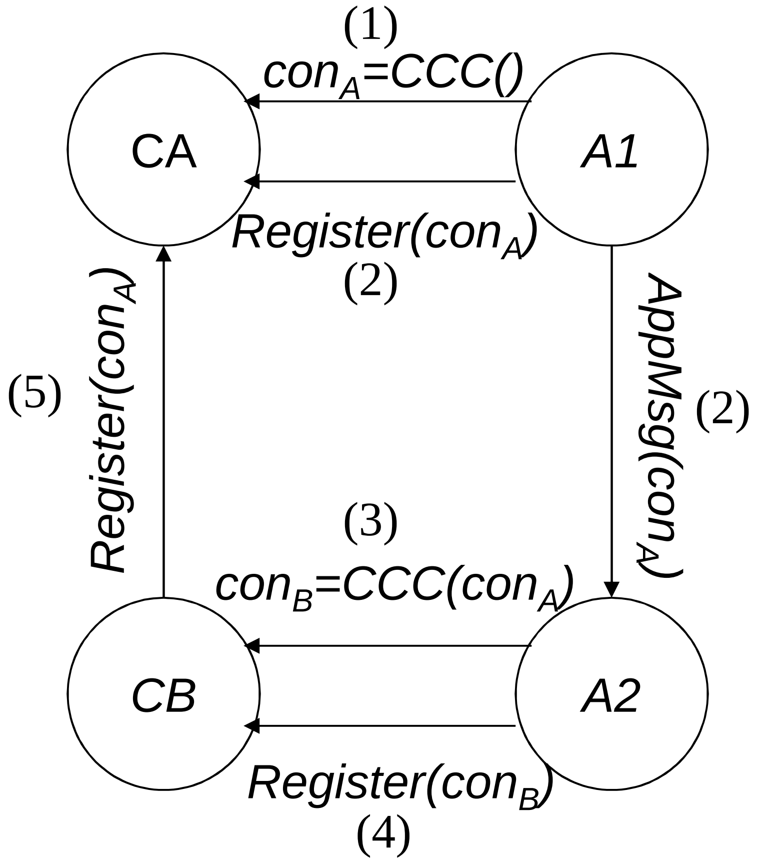
Creating an Activity 1

- *A1* invokes *CreateCoordinationContext()* to get a context, con_A, for a new activity of coordination type *Q* from coordinator *CA*
- *A1* sends an application message containing the context to *A2*
  - When SOAP is used, the context is a header block with *mustUnderstand="true"*
- *A1* and *A2* then use *CA*'s registration service to register for protocols (perhaps different, but supported by *Q)*

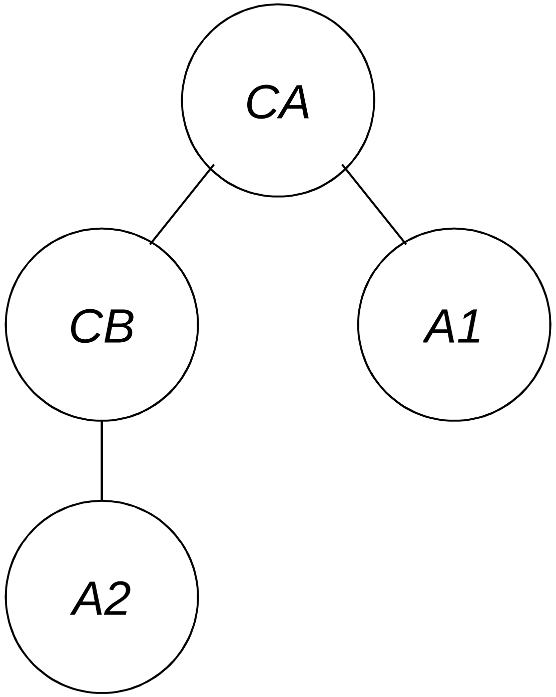
Creating an Activity 2

- *A2* might want to use a different coordinator
  - Reasons: performance, trust
- Protocol:
  - *A2* invokes *CreateCoordinationContext()* at coordinator *CB*, passing con_A as a parameter
  - *CB* creates a new context, con_B, with the same identifier and CoordinationType, returns it to *A2*
  - *A1* registers for a protocol with *CA*
  - *A2* registers for a protocol with *CB* and *CB* registers for the same protocol with *CA*

Creating an Activity



Message exchange



Protocol tree

Registration Service

- Participant explicitly registers for a particular protocol supported by the CoordinationType.
  - Contrasts with two-phase commit for distributed transactions where registration is automatic when server is invoked
  - Participant can register several times in order to participate in activity termination in several ways.

WS-Transaction

- WS-T proposes two distinct models:
  - **Atomic transaction (AT) Model** is used to coordinate activities having a short duration and executed within limited trust domains
    - addresses **"fine-grained" transactions**
  - **Business Activity (BA) Model** is used to coordinate activities that are long in duration and desire to apply business logic to handle business exceptions
    - addresses **“coarse-grained" transactions**
    - BA defines a protocol based on compensation used to achieve distributed consensus on whether the results of a long-running message exchange should be made persistent.

Atomic Transactions

- Similar to traditional ACID transactions
- Services enroll transaction-aware resources
  - Databases
  - Message queues

The Players

- Completion Initiator
  - Signals coordinator to complete a transaction
    - Can request commit or rollback
- Coordinator
  - Responsible for coordinating a single outcome
  - Drives 2PC with participants
    - Phase 1: Ensure all participants are prepared
    - Phase 2: Notify participants of outcome
- Participants
  - can vote to abort
  - Can vote “prepared to commit”
    - Must honor coordinator’s commit decision

WS-AtomicTransaction

- A coordination type that implements transactional atomicity using two-phase commit
  - Intended for tightly-coupled, short-lived activities within an organizational structure
  - Systems should trust each other
    - They must be responsive since locks are held until protocol completes
    - A single system can (perhaps maliciously) abort the entire transaction
- Protocols supported
  - **Completion**: initiates termination, final result returned (no durable resources)
    - *commit*, *rollback*, *committed*, and *aborted* messages
  - Two-Phase Commit (with presumed abort)
    - **Volatile 2PC** (no durable resources)
    - **Durable2P**C (durable resources)
    - *prepare*, *vote*, *commit*, *rollback*, *readOnly* (for cohorts that need not participate in phase 2), *aborted*, *committed* (done), and *replay* (for cohorts dealing with failure) messages
- A participant might register for both completion and 2PC

AT: Example

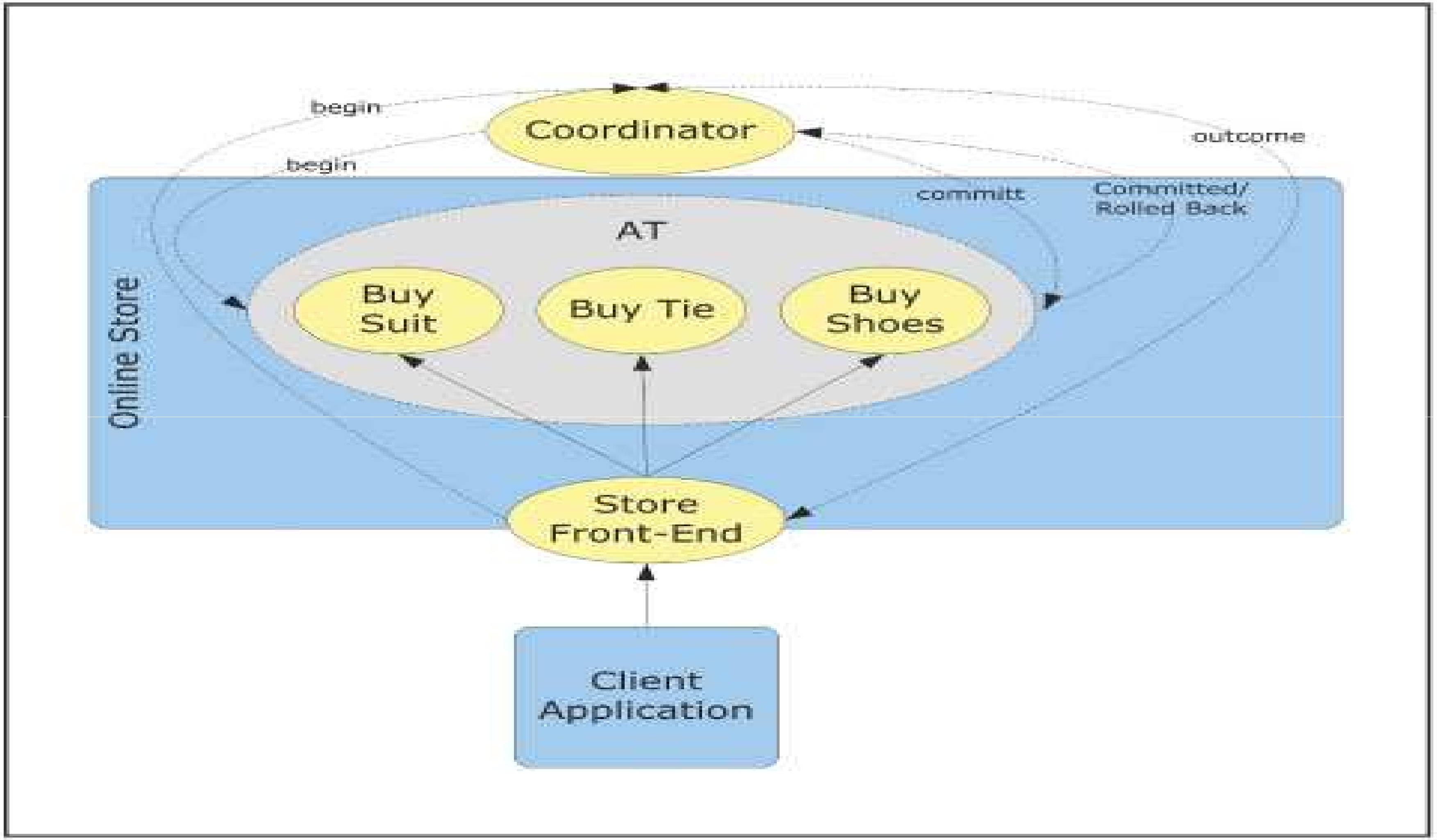


FIGURE 6 | **Using an AT to ensure all-or-nothing semantics for buying formalwear**

Business Activity

- Short running atomic transactions can be part of a long running business transaction
- The actions of the embedded atomic transactions are committed and made visible before the long running business transaction completes
- In the event of the long running business transaction failing, the effects of such atomic transactions need to be compensated for.

BA: Example

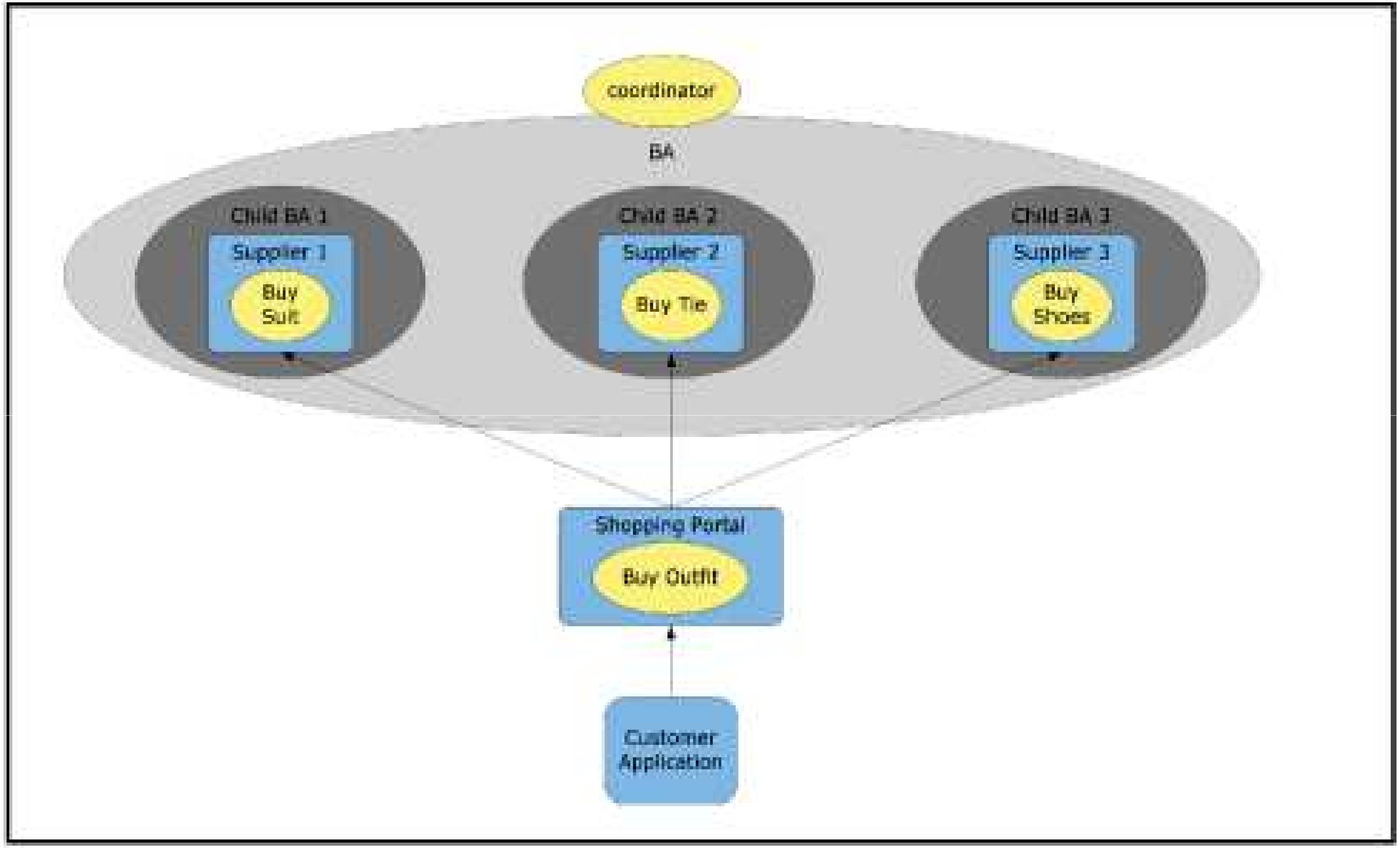


FIGURE 1 | **Using a business activity to support compensating transactions across enterprise boundaries**

BA: Example

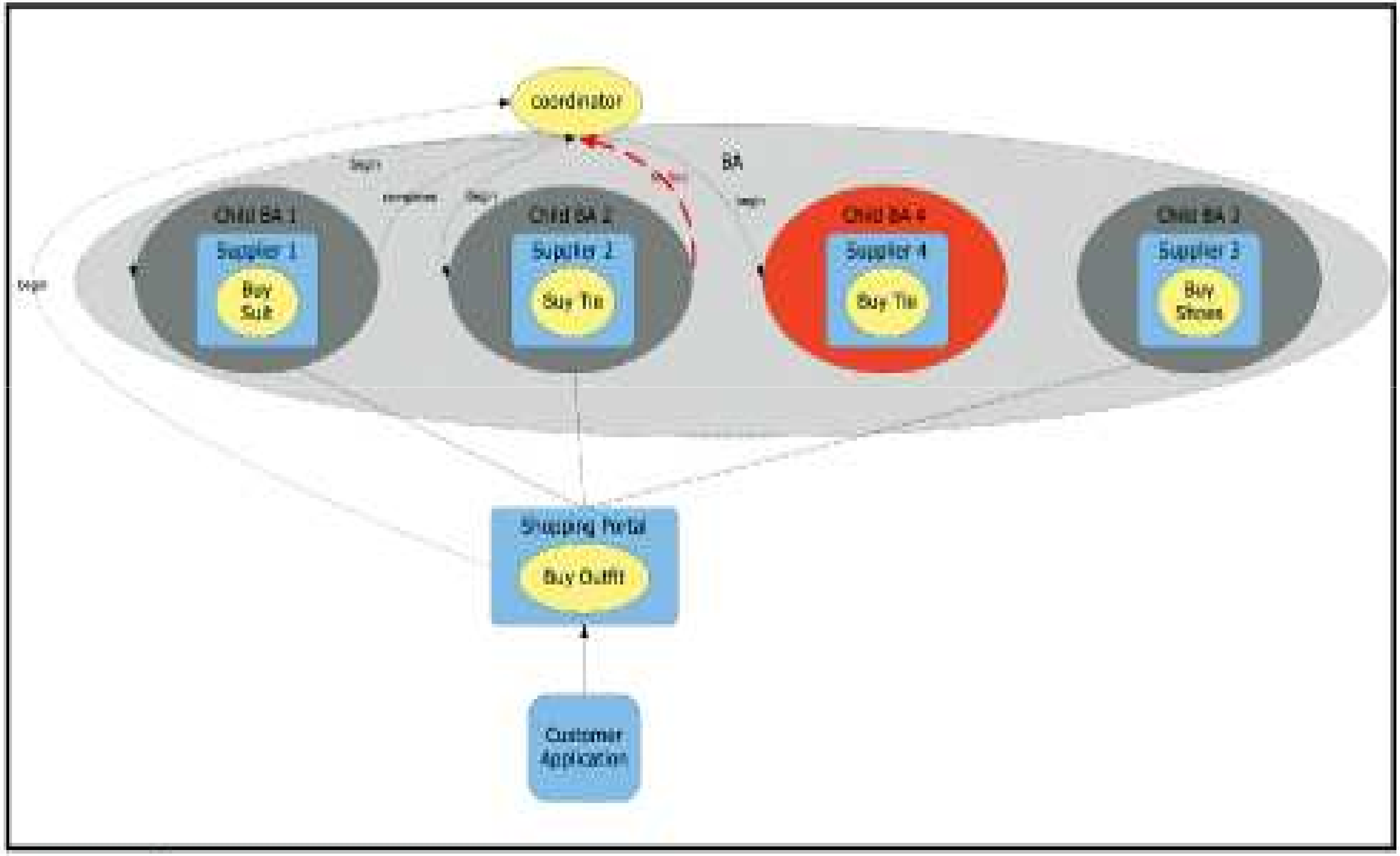


FIGURE 2 | Error handling and forward recovery

WS-BusinessActivity

- WS-BusinessActivity coordination type supports two protocols
  - BusinessAgreementWithParticipantCompletion (BAWPC)
    - Participant registering for this protocol can initiate termination
  - BusinessAgreementWithCoordinatorCompletion (BAWCC)
    - Participant registering for this protocol expects coordinator to tell it when to terminate

WS-BusinessActivity

- Components of same business activity might register for different protocols depending on who is responsible for determining completion
  - Root application might register for BAWPC, other participants for BAWCC

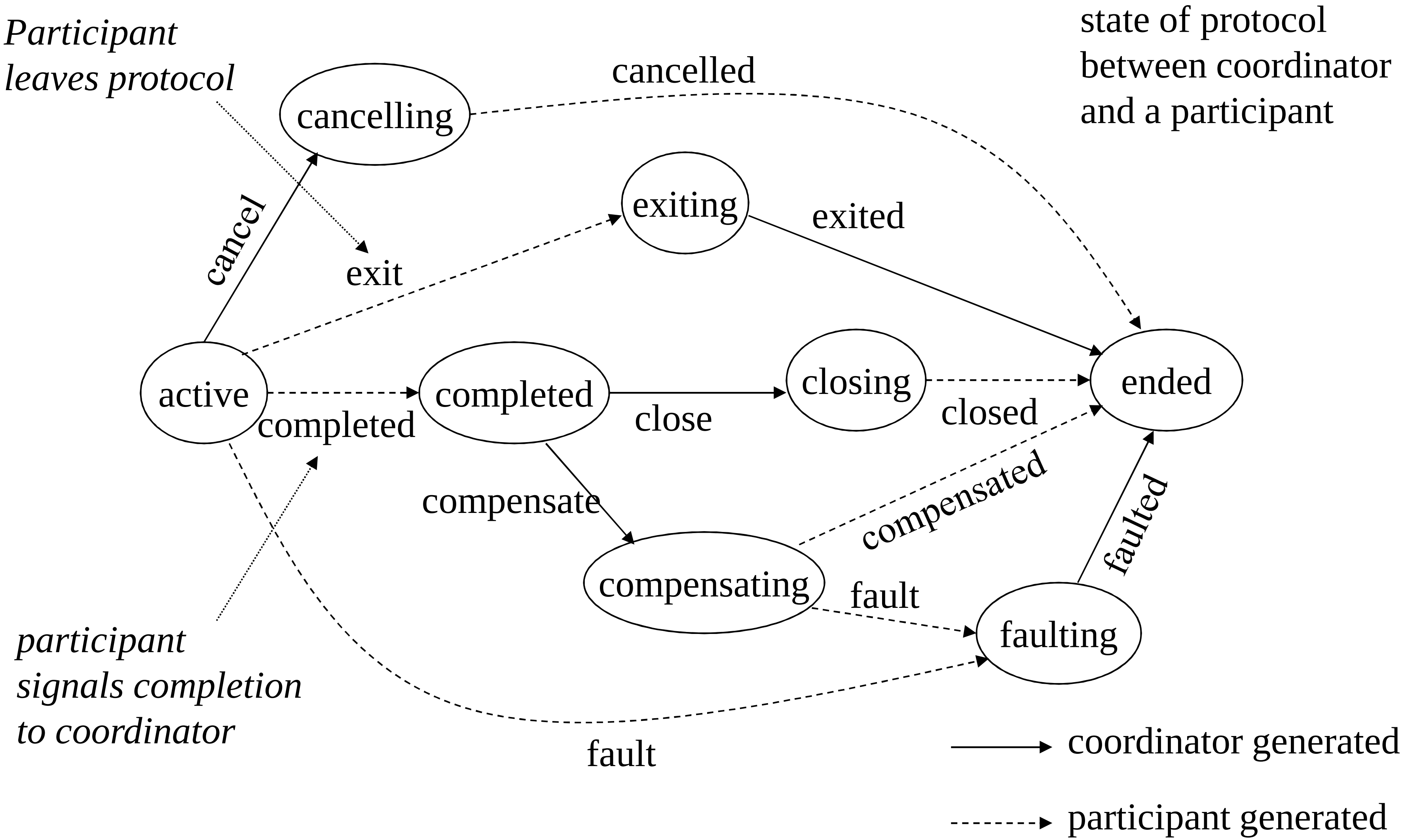
BAWPC Messages

- *Completed* – analagous to a vote message
- *Exit* – participant leaves the protocol
- *Cancel* – participant is forced out of the protocol by the coordinator
- *Close/Compensate* – in completed state coordinator decides on outcome based on coordination logic specified in context
- *Fault* – participant notifies coordinator that it has failed

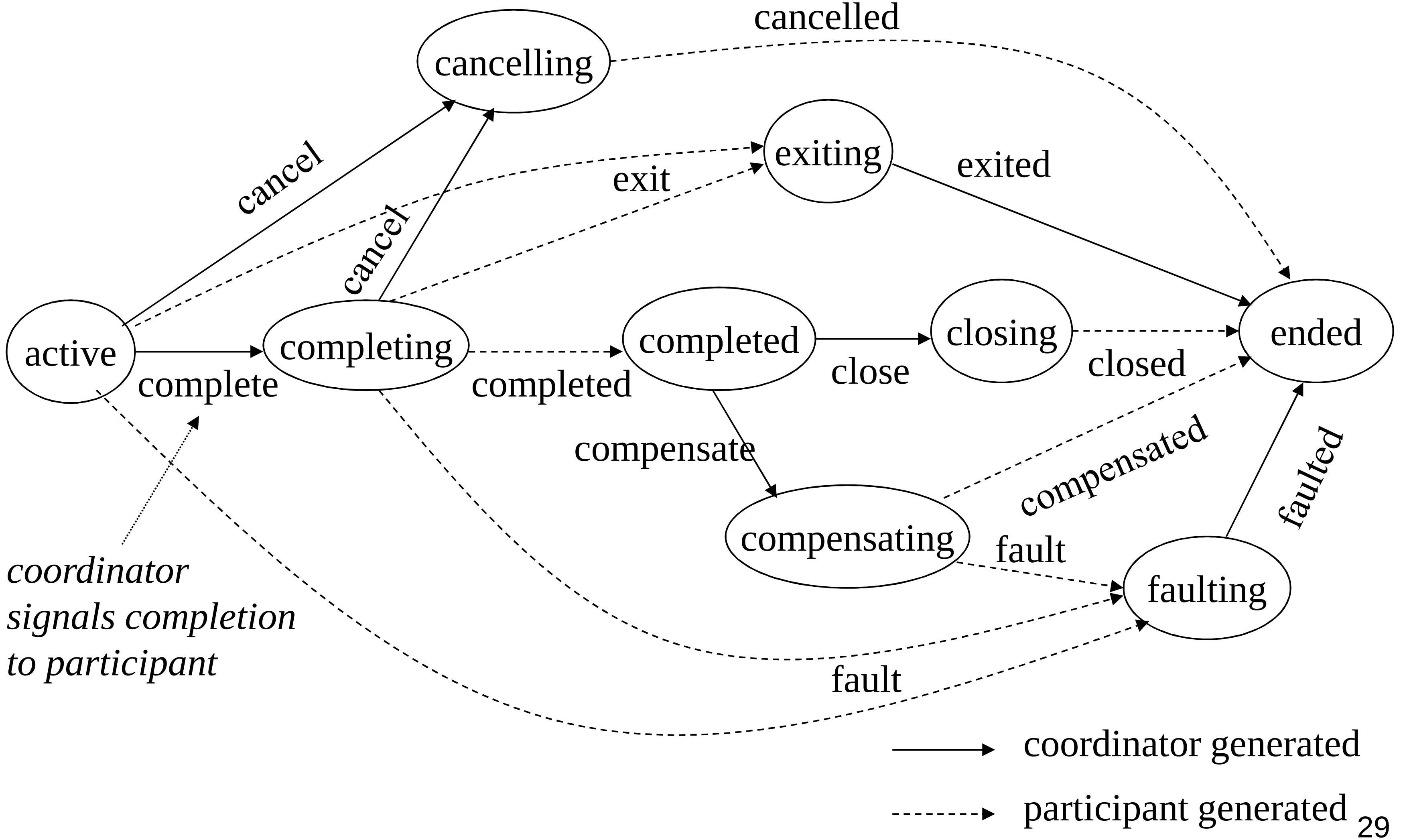
Example

- Buyer sends copies of a purchase order to sellers *A*, *B*, *C*
  - *A* can’t make a quote, responds with *fault* message
  - *B* and *C* make quotes using *completed* message
    - Indicates that *B* and *C* have completed successfully
- Buyer chooses *B*
  - Notifies *B* using *close* message
    - Indicates that *B* has successfully participated in a terminated business activity
  - Notifies *C* using *compensate* message
    - Indicates that *C* should rollback any action it has taken

BAWPC State Diagram



BAWCC State Diagram


cancelled
cancelling
cancel
cancel
exiting
exit
exited
active
complete
completing
completed
completed
close
closing
closed
ended
compensate
compensated
compensating
fault
faulting
faulted
fault
coordinator
signals completion
to participant
coordinator generated
participant generated