The CORBA Activity Service Framework for Supporting Extended Transactions

I. Houston[1], M. C. Little[2,3], I. Robinson[1],
S. K. Shrivastava[3] and S. M. Wheater[2,3]

[1]IBM Hursley Laboratories, Hursley, UK
[2]HP-Arjuna Laboratories, Newcastle-Upon-Tyne,UK
[3]Department of Computing Science, Newcastle University, Newcastle-Upon-Tyne, UK

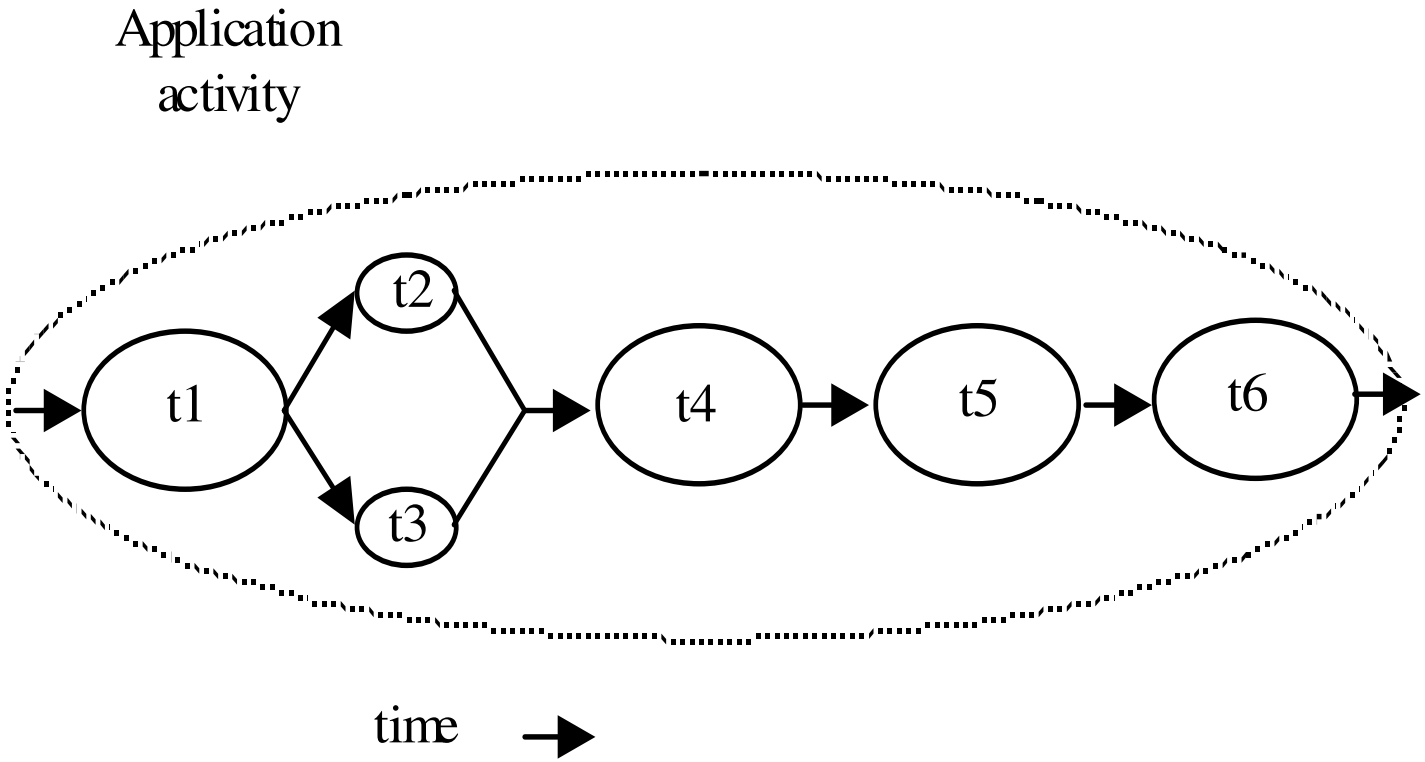
Problem statement

- Transactions imply all ACID properties
- Good for “short” durations
  - Application specific
- Long-running transactions may impose constraints
  - Hours, days, months, …
  - Retain resources for duration of transaction

Structuring transactions

- Could structure transactional applications from short-duration transactions
  - Release locks early
  - Resulting application may still be required to appear to have “ACID” properties
    - May require application specific means to restore consistency
    - Compensation may not be possible
- A transactional workflow system could be used to script the composition of these transactions
  - CORBA workflow standard

Example of structured transactions



Extended transaction models

- There are a number of such models
  - Nested transactions
  - Sagas
  - Compensations
  - Epsilon Serialisability
  - Versioning Schemes
  - Nested top-level transactions
  - Open-nested transactions
  - Glued transactions
  - Coloured actions

Which model to use?

- One size does not fit all!
- Business domains will impose different requirements on implementers
  - Essentially construct domain-specific models
    - E.g., realtime or telecoms
- The range and requirements for such extended models are not yet known

What is the “activity service”?

- Additional Structuring mechanisms for the OTS
  - OMG adopted specification (orbos/2000-06-19)
    - IBM, University or Newcastle (Bluestone Arjuna Labs), IONA, Inria, Alcatel, Vertel/Expersoft, Bank of America
- Main work provided by IBM and Newcastle University
  - 12+ months to develop and guide through OMG process
- Inria and Bank of America provided an example of use

Basic assumptions

- Models share a basic underlying assumption of “event signalling”
  - Specific model maps event into its “domain”
    - e.g., “prepare”, “rollback”
- OTS transactions may be used as building blocks
  - Transactions may be ignored
- Context propagation
  - Format of context may be different for each model

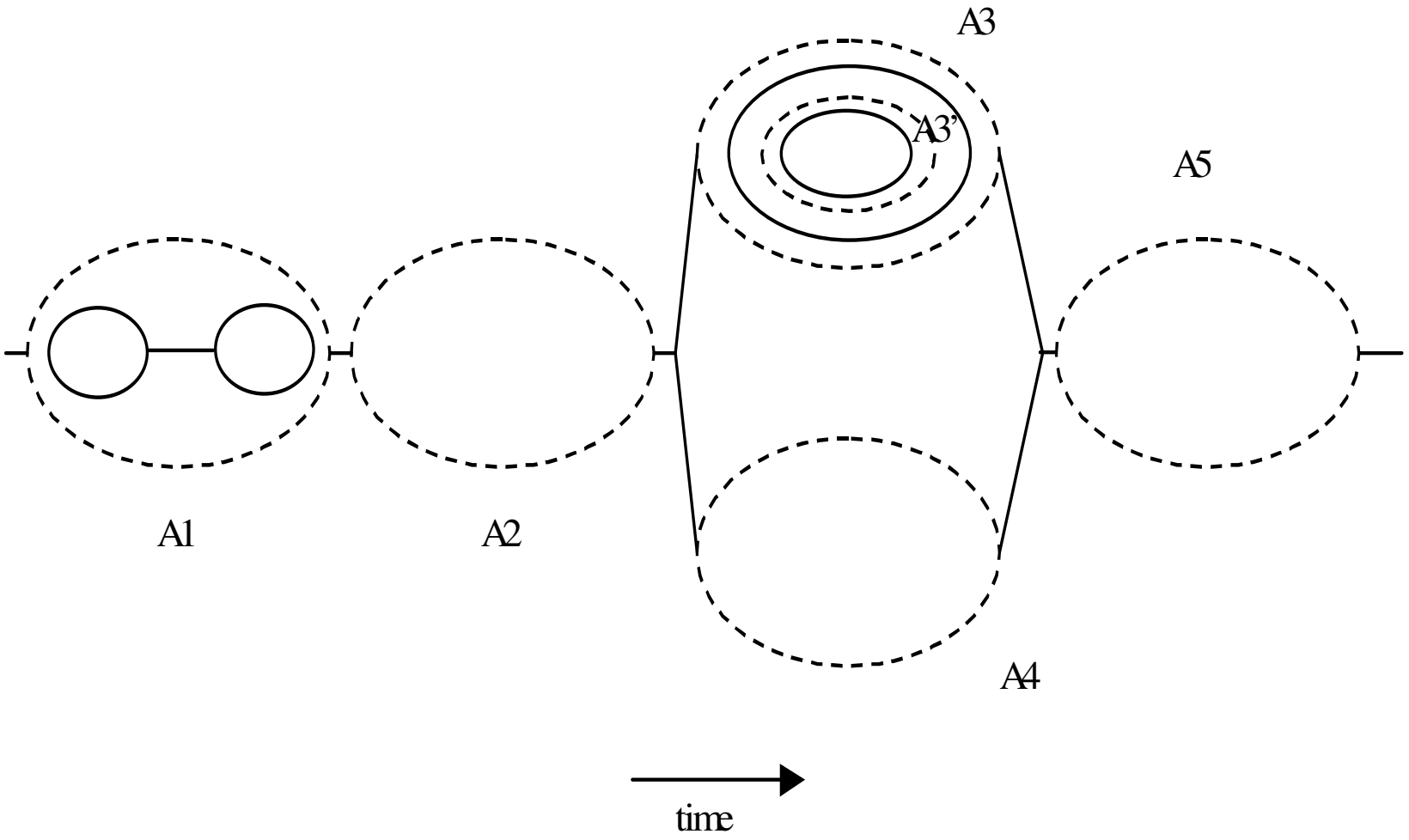
Activity Service Framework

- Defines a generic protocol engine
  - Support basic infrastructure for many extended models
- Pluggable coordination and control engine
- Activity service interfaces not typically for application programmers
  - Too low-level
  - Requires high-level API on top (High Level Service)

Activities

- Activity is an entity that does some work
  - Represented by an ActivityCoordinator
    - Coordination can occur at arbitrary points in lifetime
  - Activities can be nested
  - May use transactions during parts of its lifetime
  - In one of three states
    - Success, Fail, or FailOnly
- Contexts flow between address spaces
- Recoverable
  - Tree is recreated upon recovery

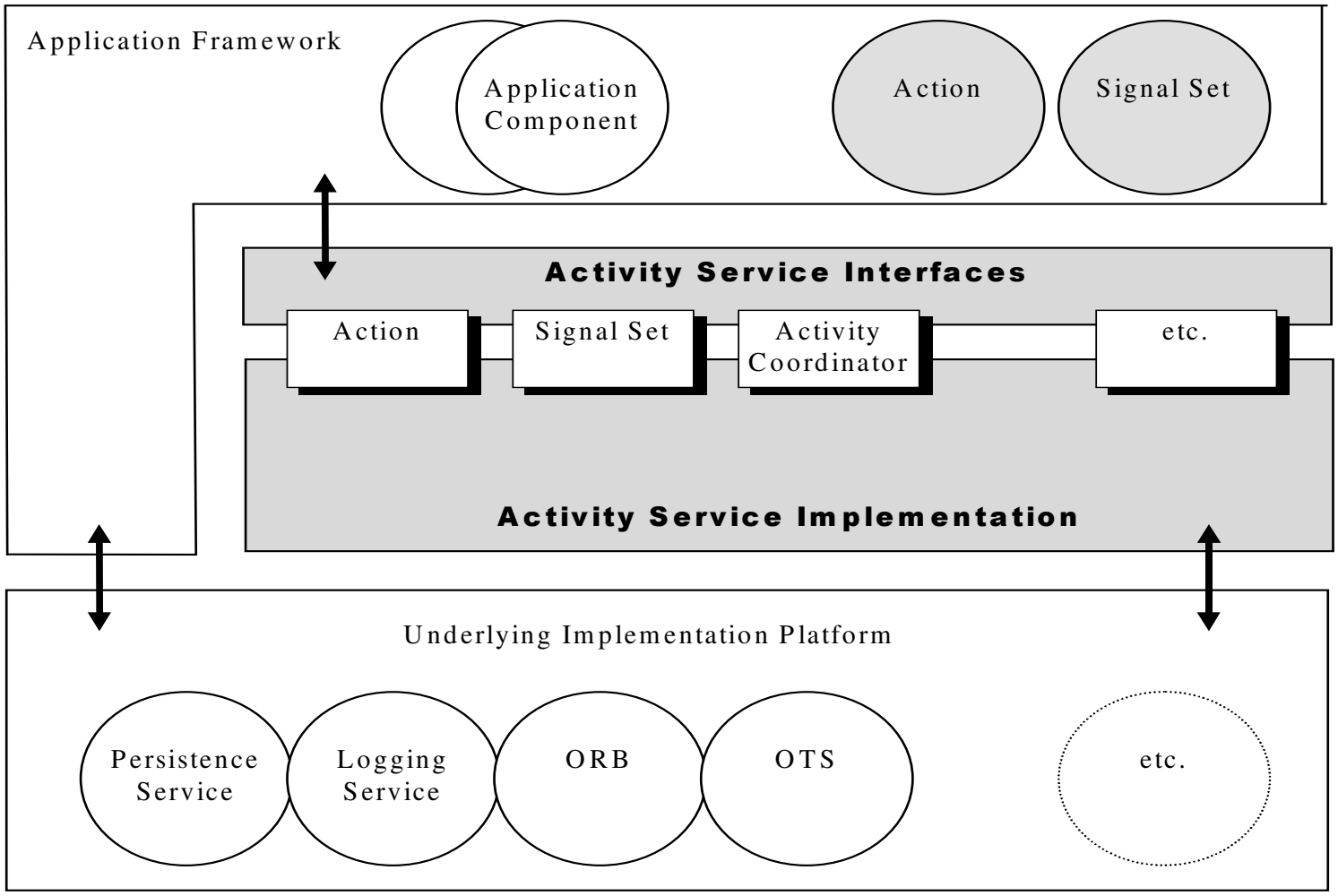
Activities and transactions



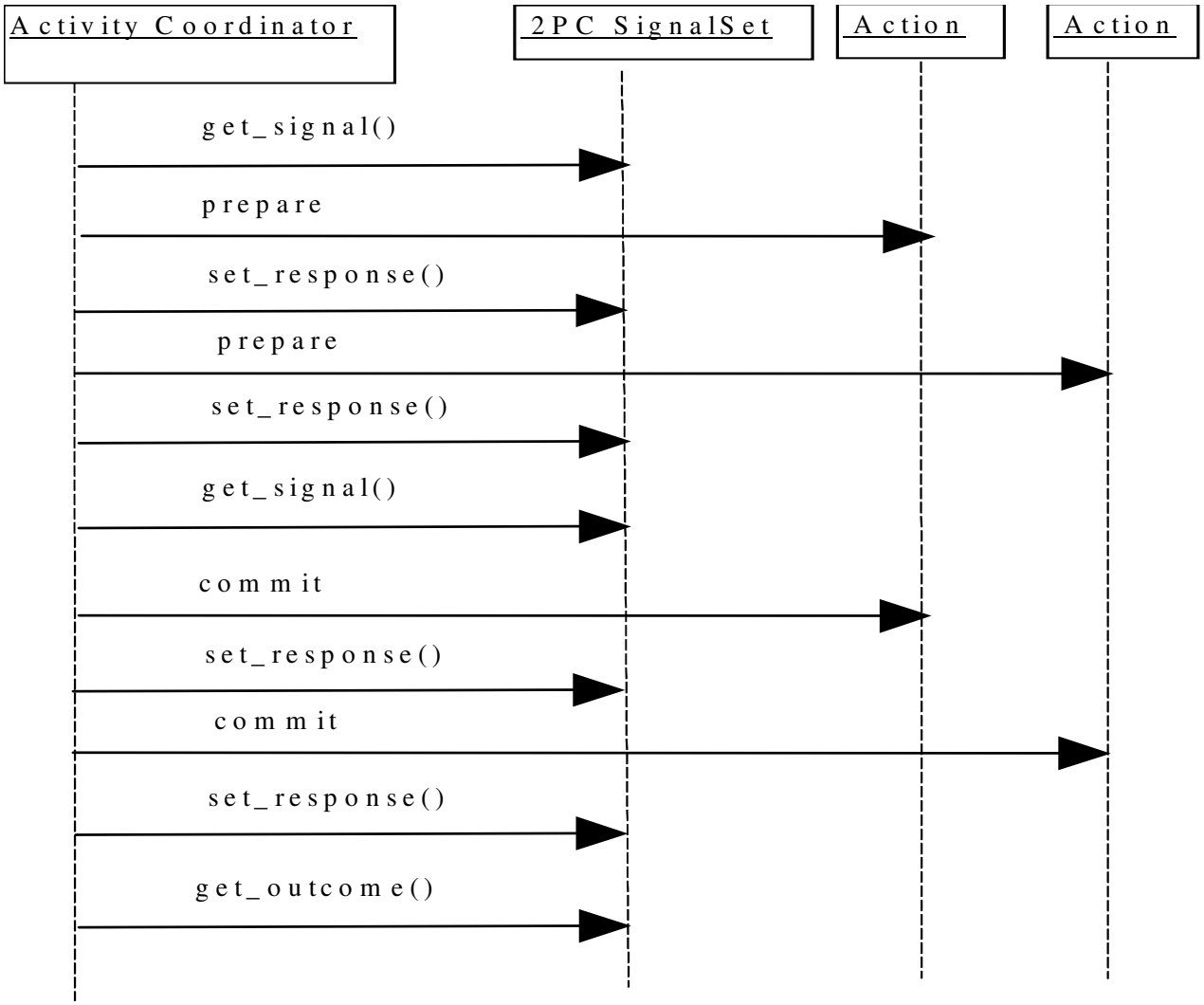
Actions and signals

- Send *Signal* between entities (*Actions*)
  - Some Signals are pre-defined
  - Most are defined by the extended model and dealt with at that level
  - Different delivery guarantees possible
    - E.g., at most once
- Signal factory (*SignalSet*) is plugable and adaptable
  - Responsible for interpreting results of Signal processing
- Infrastructure does not know how to interpret Signals or responses (*Outcomes*) to them

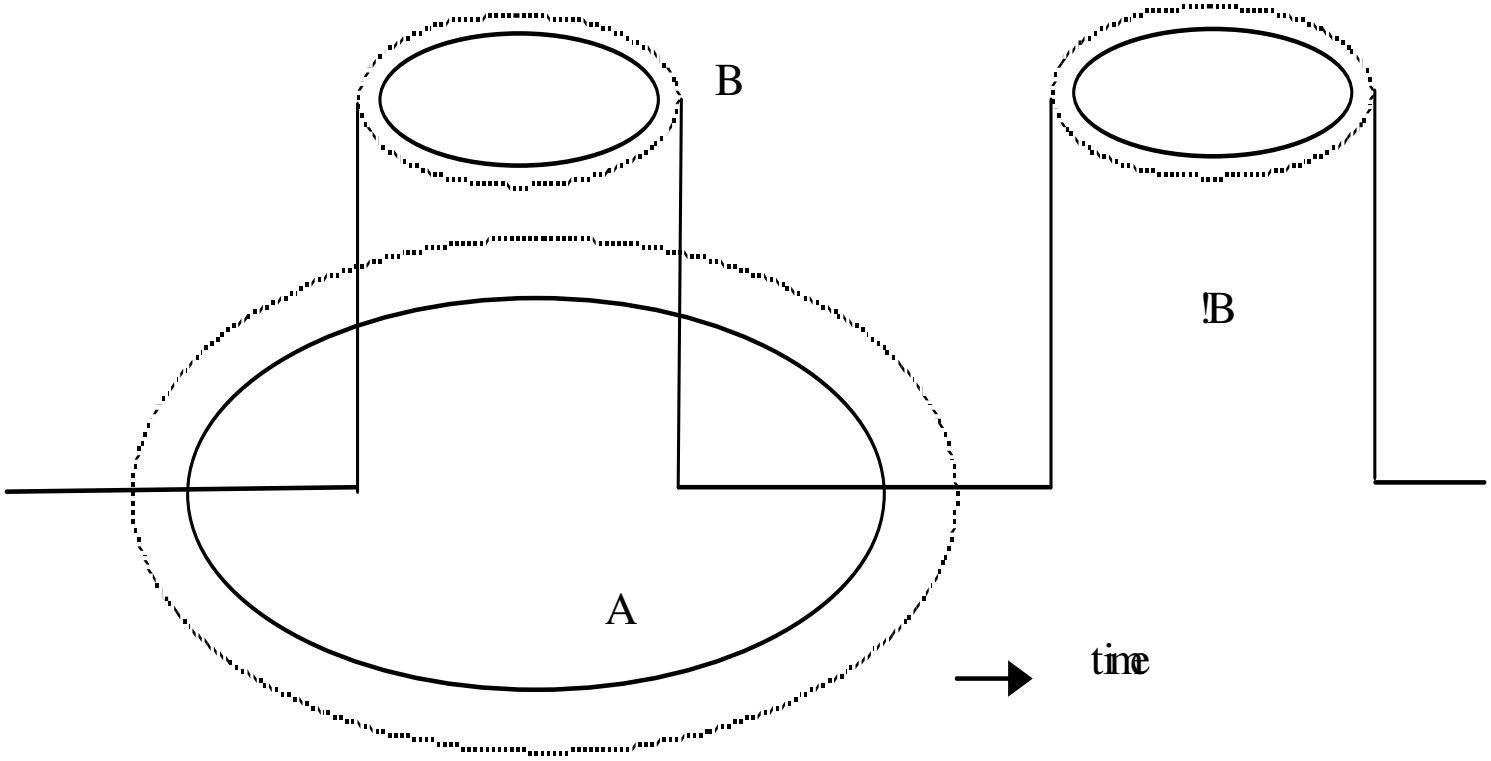
Architecture



Two Phase Completion



Nested top-level transactions with compensations



Compensation Example

- Each enclosing activity has a CompletionSignalSet
  - Success Signal
    - Completes successfully, without dependencies
  - Failure Signal
    - Completes abnormally
  - Propagate Signal
    - Completes successfully, but has dependencies
      - Encodes identity of Activity to be registered with
- Enlisted with B is an Action for !B
- !B Action enlisted with A if B sends Propagate Signal
  - Fired on receipt of Failure Signal from A completion

Conclusions and future work

- ACID transactions are insufficient for structuring long-lived applications
  - Several different extended transaction models exist
    - Address different problem domains
    - “One size does not fit all”
      - But do we really have to re-implement the wheel each time?
- OMG Activity Service standardises middleware support for these models
  - Useful paradigm allowing programmers to concentrate on event dispatch
- Work is going on into incorporating this into J2EE