Composition

Service-Oriented Architecture

Jeremy Gibbons

Contents

1 Composition of services

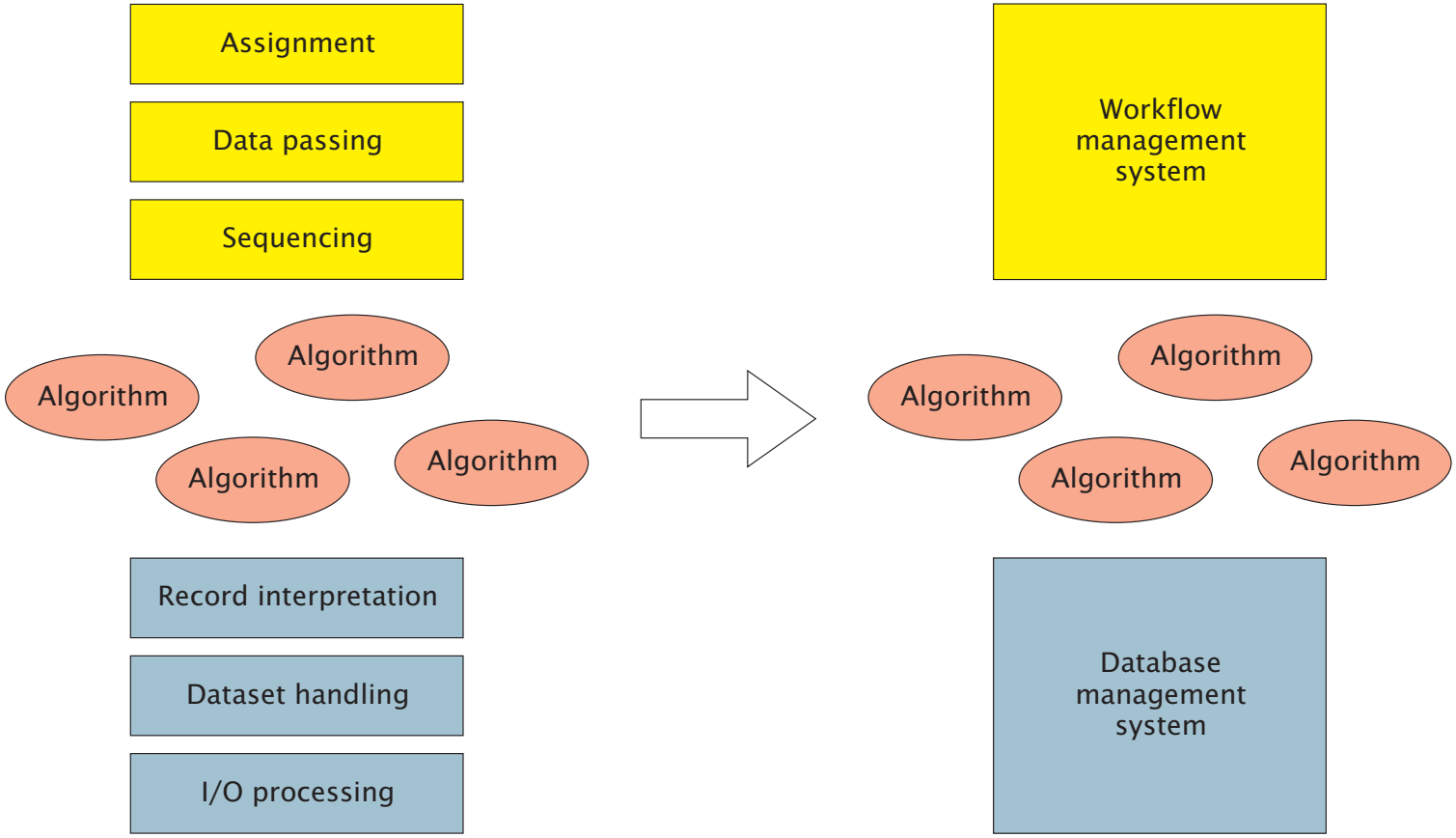
- services provide *platform- and language-independent access* to *software components*
- but these components are *isolated*: they need to be *assembled* into *service-oriented architectures*
- ideally, they should be recursively *composable* to form composite services in their own right
- *workflow* languages for scripting or ‘glue’ between individual services
- beyond mere *business protocol specifications* like RosettaNet, which are essentially paper specifications so can’t be automated and won’t scale
- BPMN, WSCI, WSFL, XLANG, BPEL...

1.1 Workflow

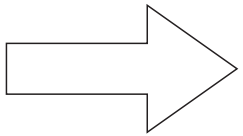
- *individual* services can be represented in WSDL
- but how should they be *combined*?
- textual documentation can explain the protocol; traditional programming language can implement it — but then the business process is *embedded in code*
- the process has value in its own right, and should be recorded separately as a *workflow*
- can then be stored, manipulated, and modified independently of individual activities
- ‘don’t let your business’s widget-making knowledge walk out of the door each evening in the chief widget-maker’s head’ (Virdell)
- agility is the key here (c.f. Sessions on MiG-15 vs F-86)

1.2 Removal of dependencies (Leymann and Roller)

DBMS provides independence from data *representation*;
workflow provides independence from control or data *flow*.



1.3 An architectural analogy



exogenous connectivity

1.4 Historical precedents: EAI, WfMS

- *enterprise application integration* (EAI)
- resolving heterogeneity, typically via asynchronous *message brokers*
- *workflow management systems* (WfMS): automating interactions
- origins in *office automation*: admin processes
- *production workflows*: from information between people to integration of systems
- often associated with *business process re-engineering*: assessment, analysis, modelling, definition, implementation
- service composition = EAI + WfMS

1.5 Sequencing

- control-dependencies between activities:
  predecessor must finish before successor can start
- data-dependencies between activities:
  output from predecessor is (transformed into) input to successor
- conditional constructions: internal or external choice
- time-out constructs

1.6 Exception handling

- consequences of exceptional conditions
- recovery sequences
- must return system to consistent state, wherever exception occurred
- at least as important as the ‘normal’ cases
- typically expands model by factor of 4 or 5!

1.7 Orchestration and choreography

- *orchestration*
  - describes procedure
  - instructs participants globally
  - imperative; centralized
  - typically deterministic: ‘must’
- *choreography*
  - describes protocol
  - constraints on interaction, but participants act locally
  - declarative; no ‘current state’
  - usually non-deterministic: ‘may’
- by analogy: orchestra has conductor, ballet does not

1.8 Compensation

- business processes are usually *long-running transactions*
- want to maintain ACID properties (atomicity, consistency, isolation, durability)
- . . . but cannot afford to use locking to achieve these
- therefore might have to compensate (as much as possible) for cancelled transactions
- compensations usually assume forward transaction completed successfully (no ‘partial compensations’)

1.9 Correlation

- business process typically involves some *conversation* between components
- therefore the same instance of a component should be involved throughout the process (at least logically)
- in the OO world, this is handled by *object references*
- but components should maintain no persistent state
- so subsequent messages in a workflow should be *correlated*
- ‘customer reference number’ or ‘case number’

1.10 Polarity

- WSDL operations are of four kinds:

  one-way: port receives `<input>` messages only

  request-response: port receives `<input>` then sends `<output>`

  notification: port sends `<output>` message only

  solicit-response: port sends `<output>` then receives `<input>`

- one-way and request-response operations are *incoming*, solicit-response and notification operations *outgoing*
- workflow specification will match up ports; obviously, matched ports must have operations of opposing *polarities*
- WSDL does not restrict port types to aggregating operations of a common polarity; XLANG spec argues that *allowing a mixture of incoming and outgoing operation types in a port type is highly undesirable*

1.11 Workflow management

- if workflow model is sufficiently *precise and complete*, then *workflow engine* can execute process description automatically
- workflow model dictates ordering, dataflow, compensation, etc
- well-established R&D area (*Workflow Management Coalition* (`http://www.wfmc.org/`) since early 1990s; OMG *Workflow Management Facility Specification* since 1996, version 1.2 in 2000)
- eg *megaprogramming*: module assembly (Wiederhold, 1992)
- now being assimilated into SOA worldview

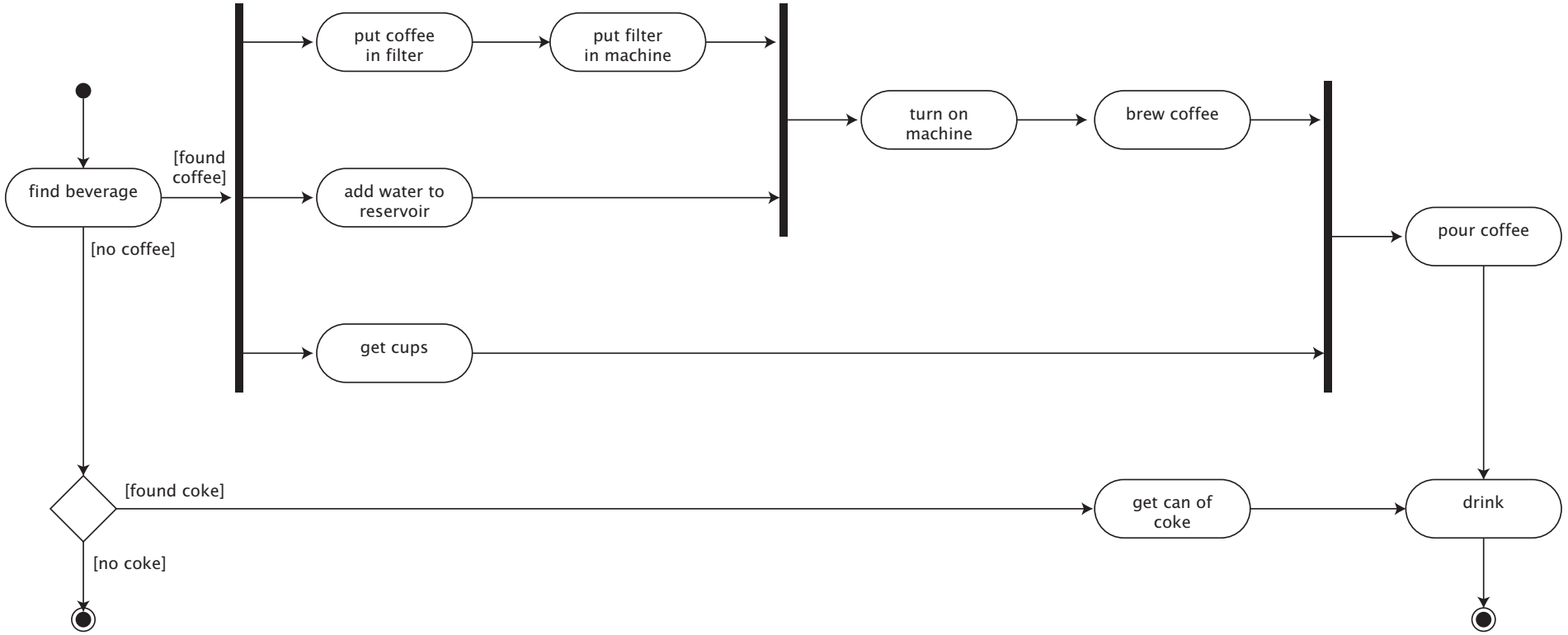
1.12 WS-related standards for workflow

- *Web Services Flow Language* (WSFL) (IBM; 2001)
- *XLANG* (Microsoft; 2001)
- *Web Services Choreography Interface* (WSCI) (BEA, Intalio, SAP, Sun; 2002)
- combined into *Business Process Execution Language for Web Services* (BPEL4WS, or just BPEL) (IBM, Microsoft, BEA, SAP, Siebel; 2003), to be submitted to OASIS
- various ad-hoc and vendor-specific graphical notations; OMG standard *Business Process Modeling Notation* (BPMN) an alternative

2 UML activity diagrams for recording workflow

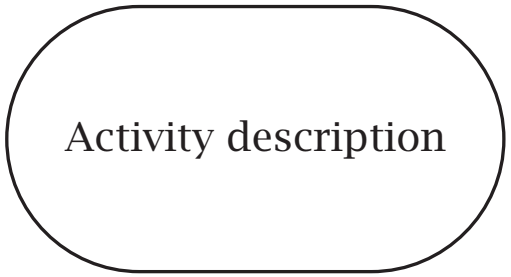
- activity diagrams illustrate *time-* or *data-dependencies* between *activities*
- based on Petri nets
- some similarities with flowcharts, but are also well-suited to presenting *parallel* sequences of activities
- mainly used in UML to document complex operations
- . . . but also useful for expressing *business workflows*
- approach based on Fowler, *UML Distilled*
- (there's now a draft UML profile for BPEL)

2.1 Example activity diagram (after UML spec)



2.2 Activities

- an atomic action, eg execution of an operation
- with duration, and with possible interruption points
- denoted by an oval: straight top and bottom, round ends

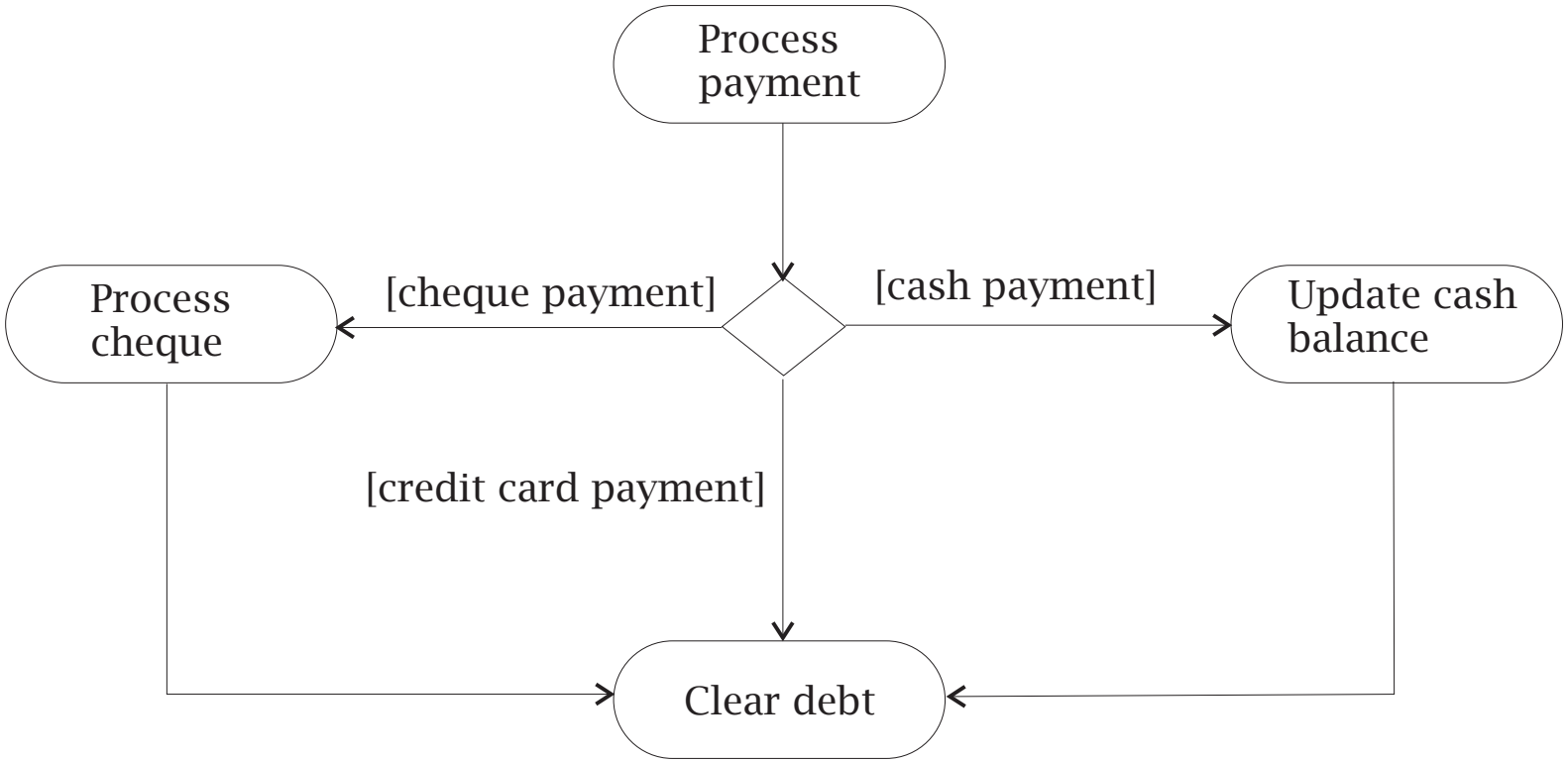


2.3 Transitions

- movement from one activity to another
- normally occurs when all the actions of an activity have been completed
- may occur when an event triggers the exit from a particular activity
- indicated by an arrow between activities

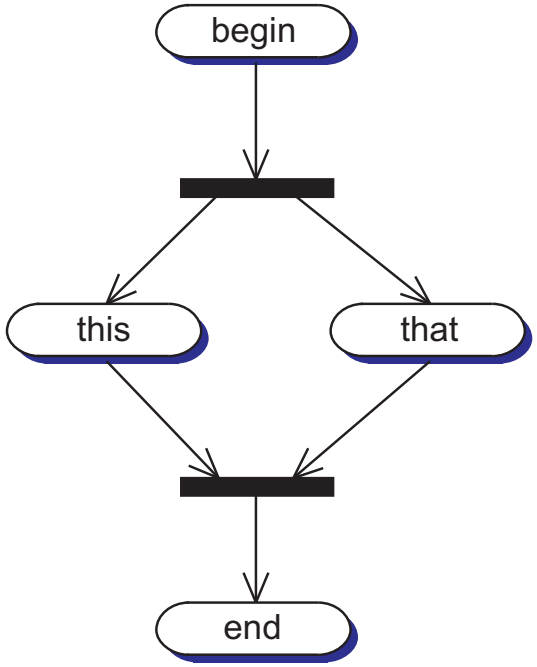
2.4 Decision points

- a point where the exit transition from an activity can branch in alternative directions depending on a condition



2.5 Synchronization bars

- split a transition into multiple paths, and combine multiple paths into a single transition
- *fork* vertices have a single incoming transition
- *join* vertices have a single outgoing transition
- in general, synchronization bar may have multiple incoming and outgoing transitions
- forks and joins should match up ('token balance')



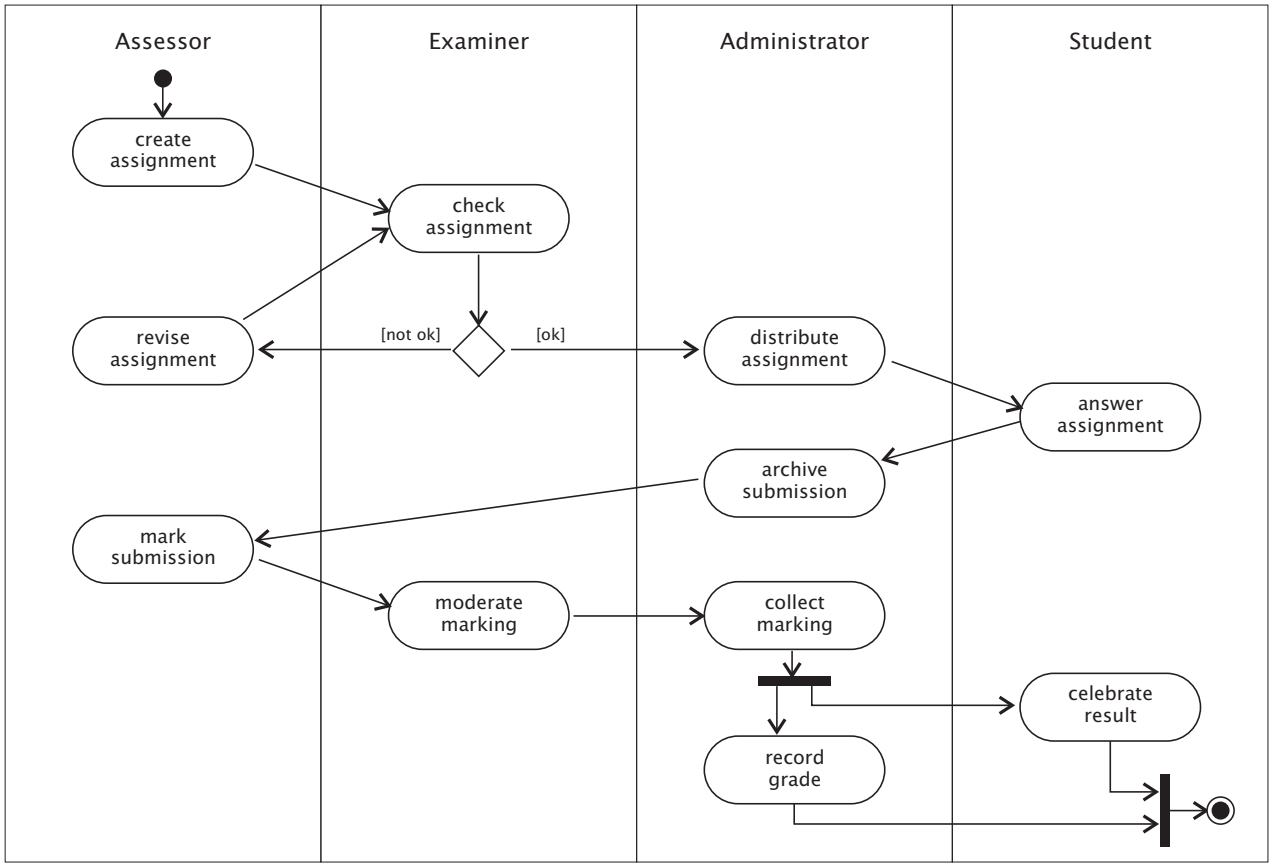
2.6 Start and end states

- *start state* is the entry point to a flow
- only one start state allowed

- end states indicate the exit point from a flow
- multiple end states are allowed

2.7 Swimlanes

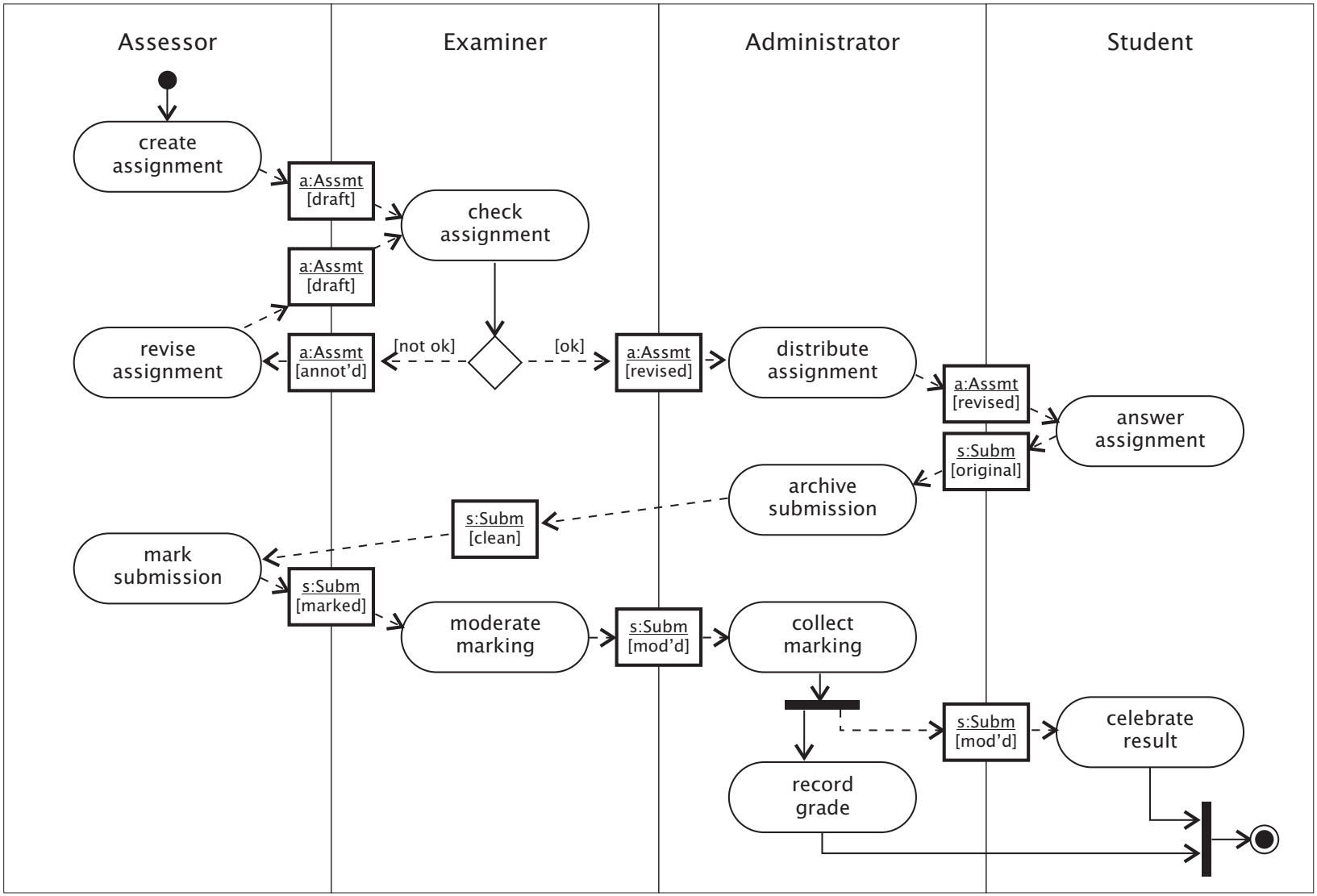
- partition an activity diagram into the responsibilities of different entities



2.8 Dataflow

- transitions between activities represent *control dependencies*: one activity must complete before another can start
- workflows also have *data dependencies*: one activity produces a result that another requires
- UML activity diagrams allow *object flow* as well as *control flow*
- dependent data is shown as an object icon (rectangle with underlined name and type)
- *dependencies* shown as dashed arrows from generating activity to object, and from object to consuming activity(s)
- same object may occur multiple times in an activity diagram, typically in different *states* (shown in square brackets after object name)

2.9 Example of object flow



3 Business Process Modelling Notation (BPMN)

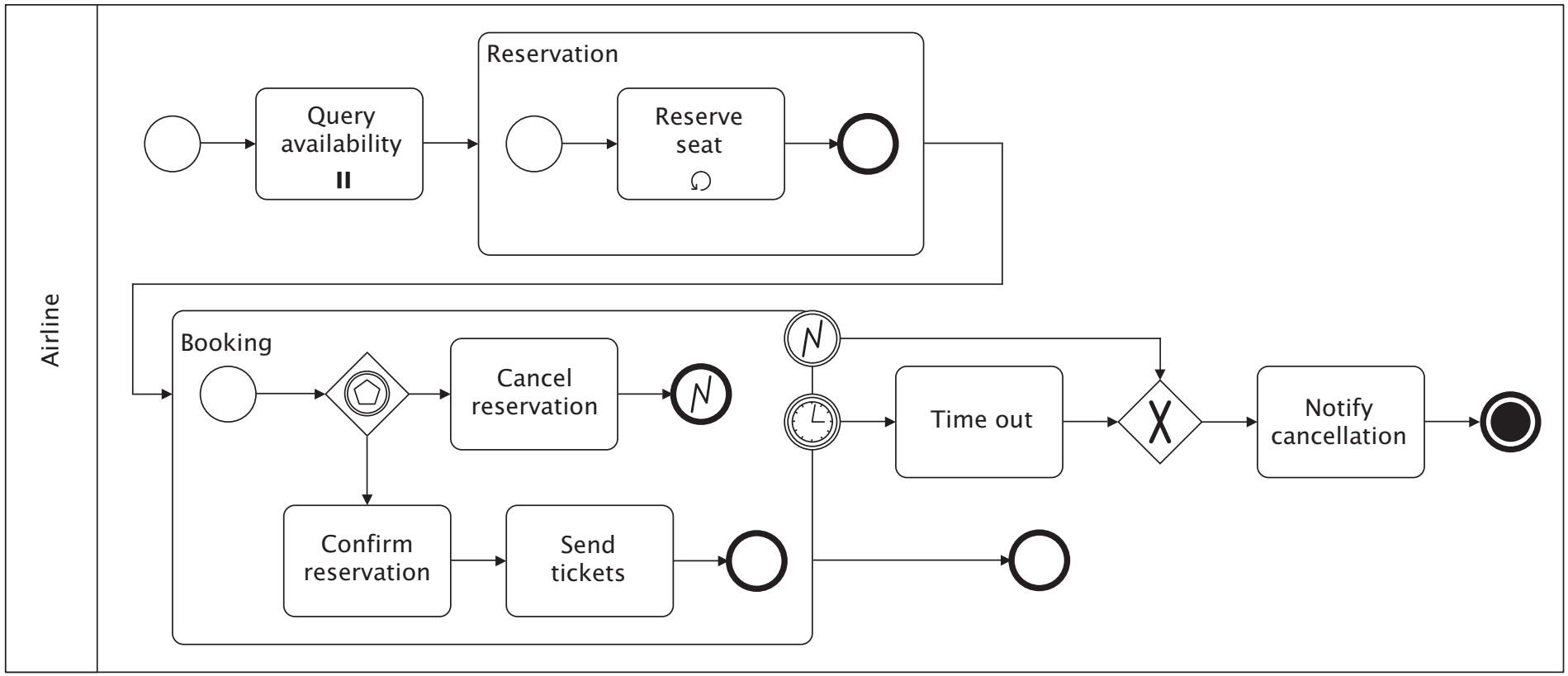
- graphical notation specifically for workflow
- developed by Business Process Management Initiative, now part of OMG
- currently v1.1; working on v2.0 (two different ones!)
- spec also defines mapping to BPEL
- my student Peter Wong translating to CSP

3.1 BPMN constructs

Very briefly:

- flow objects
  - event (start, end)
  - activity
  - gateway (branching)
- connecting objects
  - sequence flow (control)
  - message flow (data)
- swimlanes
- data object

3.2 BPMN example



4 WSCI

- BEA Systems, Intalio, Sun, SAP
- describes the flow of messages exchanged with a web service
- expressed in terms of temporal and logical dependencies between messages
- features sequencing rules, correlation, exception handling and transactions
- allows client to understand how to interact with service, and to anticipate its behaviour
- ‘operating manual’ for a web service
- not an executable workflow language

4.1 Questions answered by WSCI

- can messages be sent and/or received in any order?
- what rules govern the sequencing of messages?
- what is the relationship between incoming and outgoing messages?
- is there a ‘start’ and an ‘end’ to a sequence of messages?
- can a sequence be partially undone?

4.2 Features provided by WSCI

- *message choreography*: ordering rules
- *transaction boundaries and compensation*, allowing participation in distributed transactions
- *exception handling*, reactions to exceptional circumstances
- *thread management*: capability for concurrent conversations, and correlation of these
- *properties and selectors* that govern observable behaviour
- *connections* between consumers and producers
- *dynamic participation*: how target identity is selected

5 WSFL

- IBM's contribution to standardization (Frank Leymann et al.)
- two modes of use:

  flow model describes *usage pattern* of collection of web services

  global model describes distributed *interaction pattern* between collection of web services

- supports recursive composition: each flow model or global model can in turn be represented as a web service, and further composed

5.1 Composition

- one use case for WSFL
- enterprise implements a business process by composing existing web services
- existing base services already provided by other vendors (or by this one)
- coordination of these services described in WSFL
- eg providing a FAÇADE for a complex collection of services
- advert in service repository would attract offers from consumers

5.2 Parametrization

- a second use case for WSFL, a variation on the first
- abstract away from particular implementations of base services; leave *plug links* for later binding
- still use WSFL to describe the coordination
- describes *exports* (services provided) and *imports* (services required)
- appropriate for describing service *brokers*; eg comparative shopping service acts as a broker between shoppers and shops
- advert in service repository would attract two kinds of offers: from producers and consumers

6 XLANG

- Microsoft's contribution, part of BizTalk (EAI for .Net)
- sequential and parallel flow constructs
- internal and external choice
- long running transactions with compensations
- custom correlation of messages
- exception handling
- abstraction from detail: *opaque values*

7 WS-BPEL

- formerly *Business Process Execution Language for Web Services*
- a combination of IBM's WSFL (graph-oriented control flow) and Microsoft's XLANG (structured control flow)
- input also from BEA, Siebel, SAP, Sun
- envisage two usage patterns: for describing *abstract processes* (describing interaction patterns but not data transformations) and *executable processes*
- common core, with extensions for each usage pattern

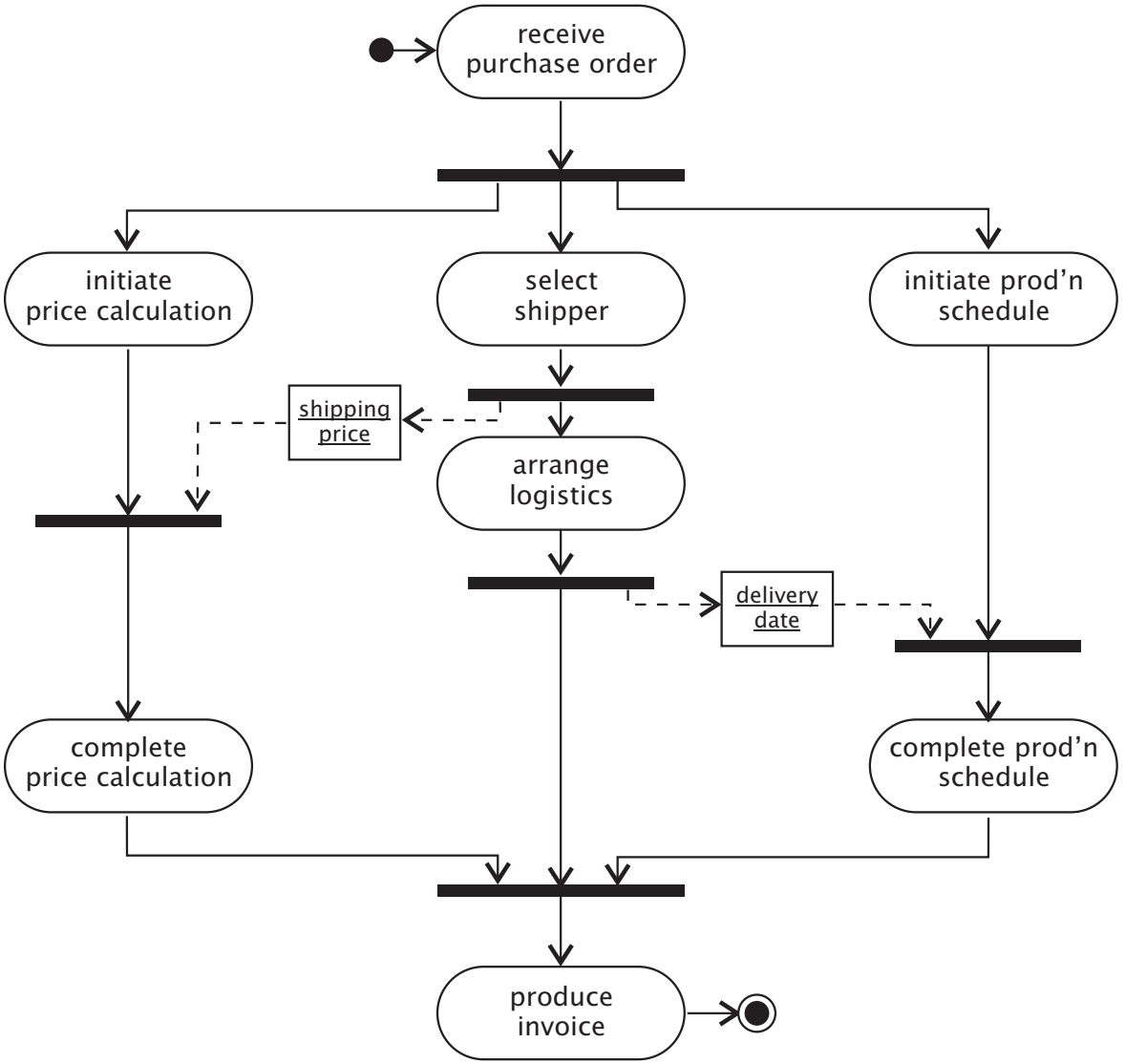
7.1 Executable processes vs business protocols

- *executable business process* captures participants' exact behaviour
- *business protocol* is an abstraction of this: describes message exchanges but not specific data
- business protocol involves *transparent data* (relevant to public aspects) and *opaque data* (relevant only to private back-end systems)
- analogous to network packet's *control fields* and *payload*
- from business protocol point of view, opaque data is nondeterministically chosen
- a contribution from XLANG (which provided only business protocols, but anticipated extensions to executable processes)

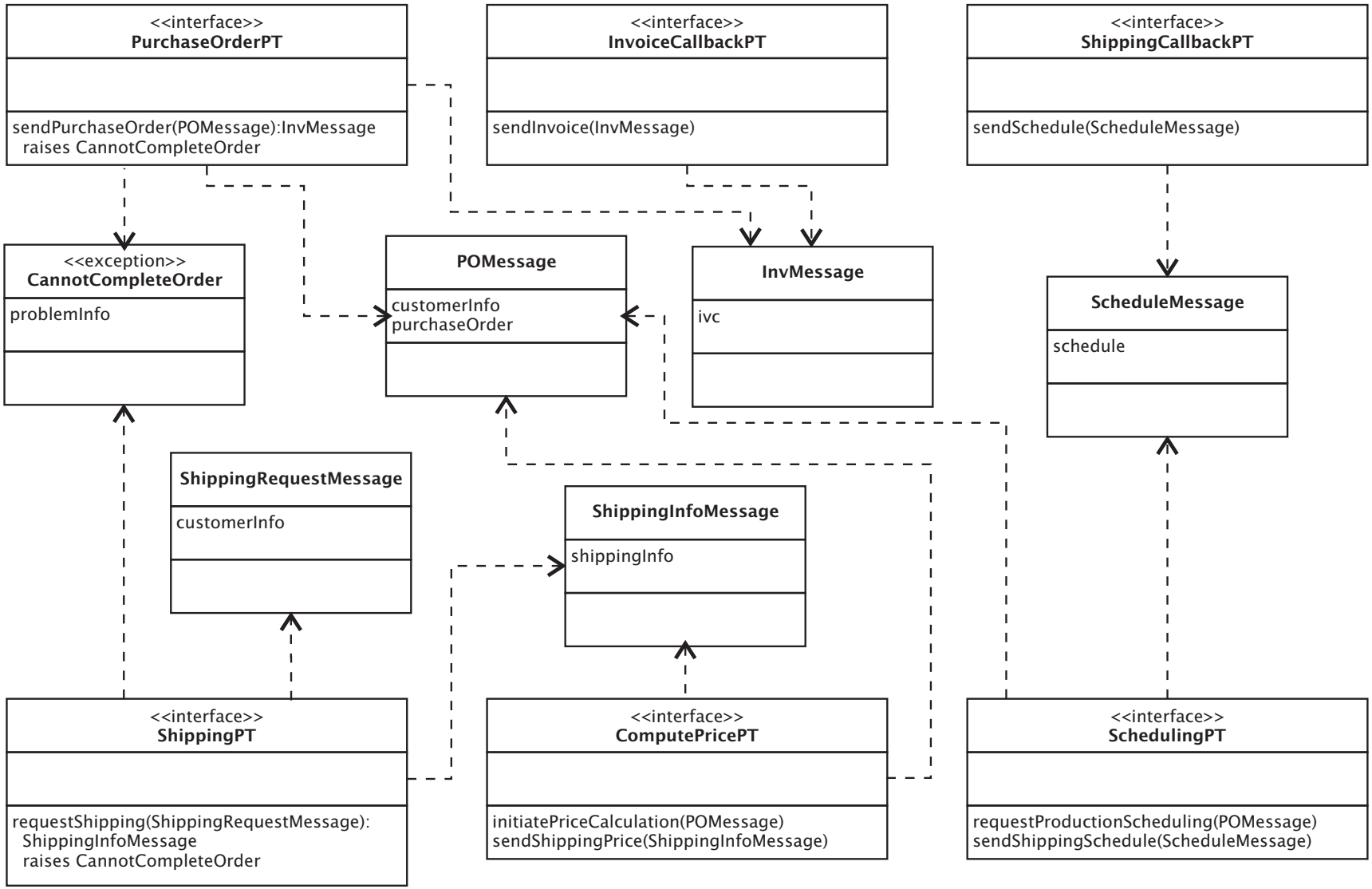
7.2 Simple example

- purchase order system
- system receives purchase order from customer
- price is calculated, shipping arranged, production scheduled (in parallel)
- on completion, invoice is produced
- choice of shipper feeds into price calculation
- shipping date feeds into production schedule
- see appendix for WSDL and BPEL

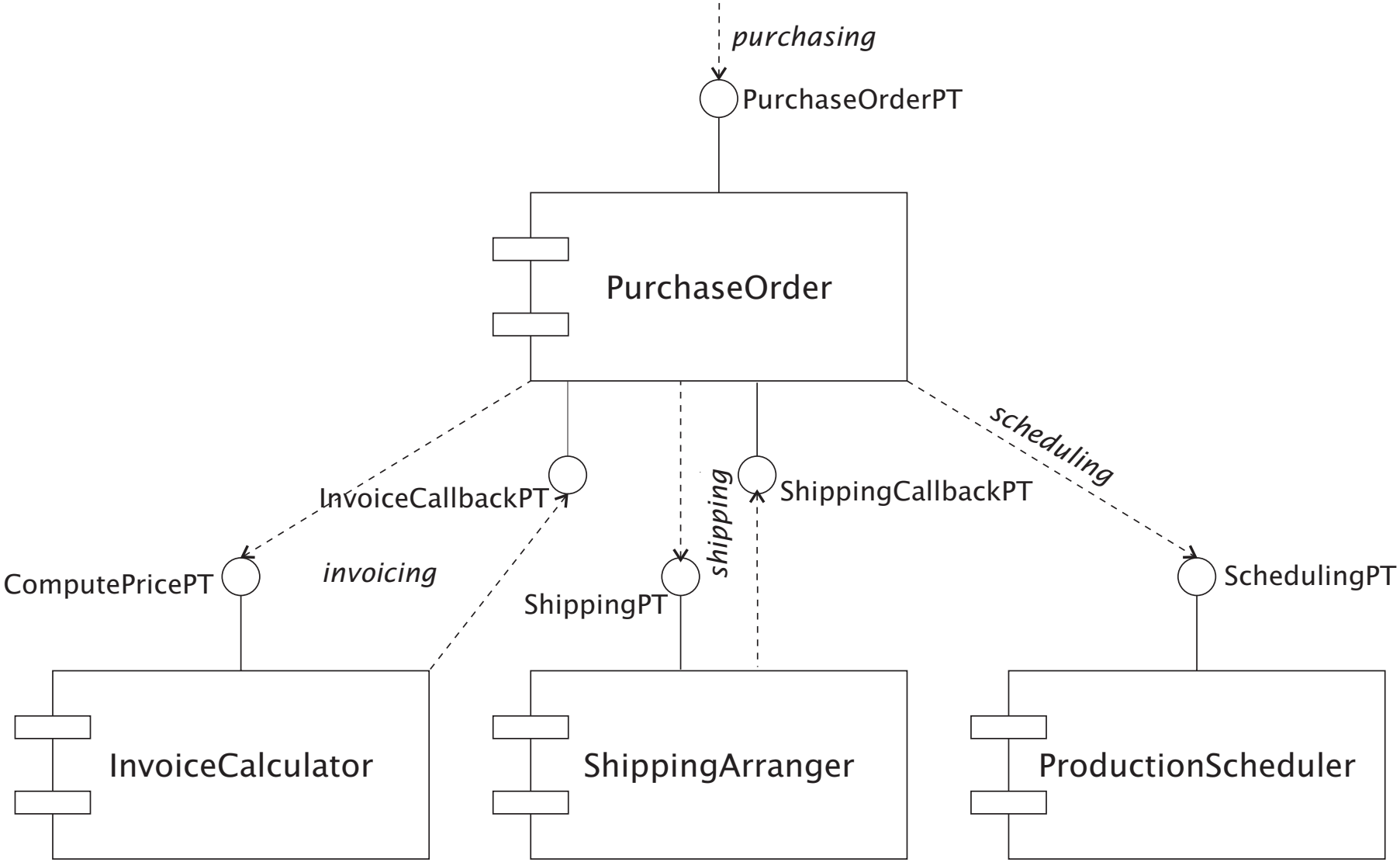
7.3 Abstract activity diagram for purchase orders



7.4 Messages and port types



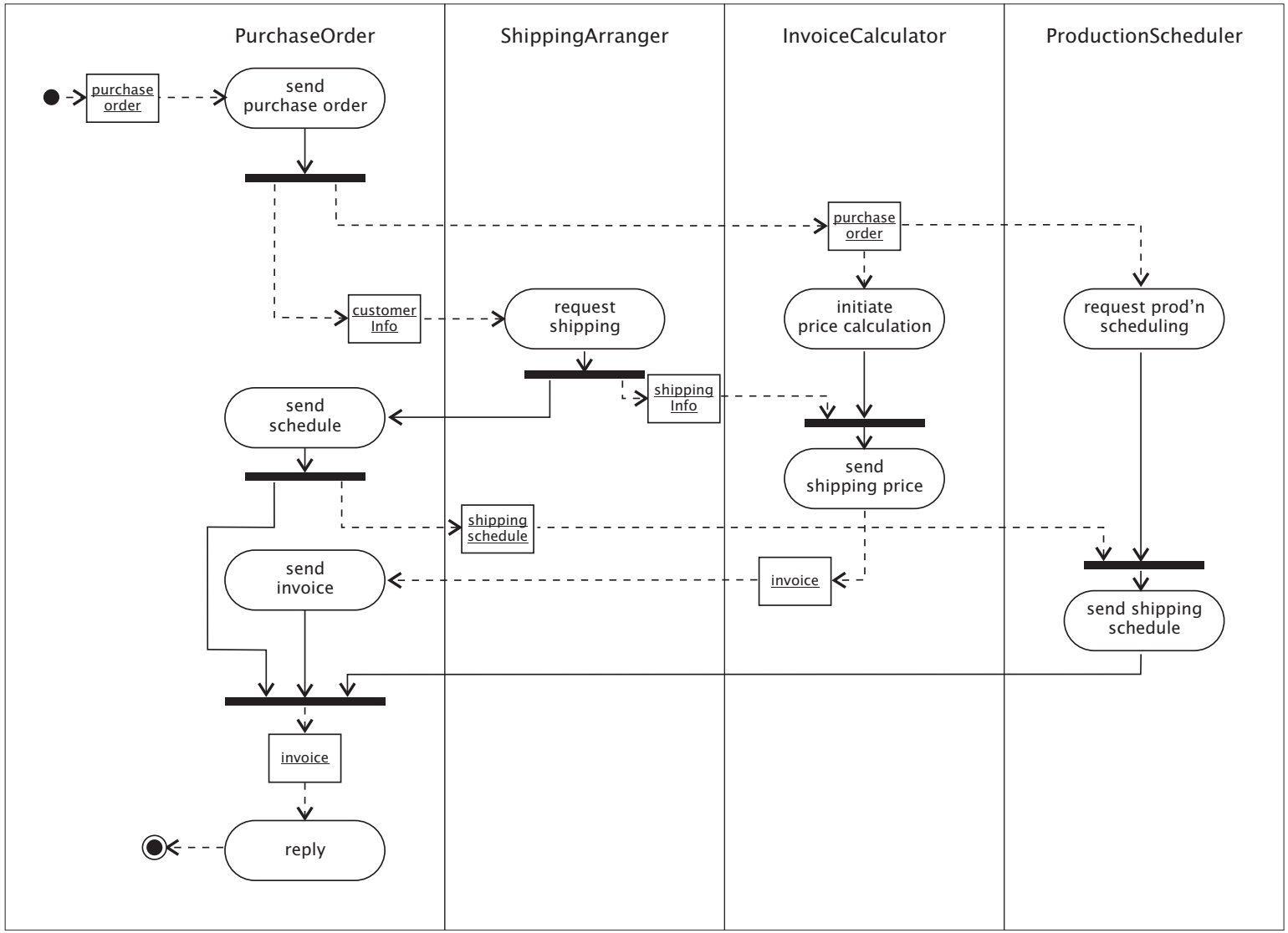
7.5 Partner links



7.6 Notes on partner links

- interactions between business process and external parties
- *partner link type* defines one (for unidirectional links) or two (for bidirectional) *roles* of corresponding *port types*
- (here, no requirements are placed on the customer using the *PurchaseOrder* component, and no requirements are placed on this for using the *ProductionScheduler* component; for an asynchronous service, customer would have to provide a callback interface)
- *partner links* instantiate partner link types, specifying *myRole* (played by this process) and/or *partnerRole* (played by external party)
- bindings of actual partners to external roles are omitted
- would expect opposites to match up, but WS-BPEL supports only incoming

7.7 Coordination



7.8 Notes on coordination

- dataflow links by global *variables*
- *fault handlers* for exceptions (omitted in activity diagram for simplicity)
- `<sequence>` element for sequential ordering
- `<flow>` element for concurrent execution
- `<link>` elements for extra dependencies within `<flow>`
- simple `<assign>`ments shown; more complex ones available
- control constructs (`<switch>`, `<while>`) also available
- `<receive>` for messages consumed by process, and `<reply>` for responses
- `<invoke>` for messages sent to external parties

8 Reflection

- BPEL represented in XML is very ugly; not worth exploring here
- tools should do all the relevant translating for you: UML to BPEL translators exist, for example
- we can use this approach to describe abstract and concrete coordination processes; the concrete coordination process has to run somewhere
- design activity is the crucial piece — UML profiles can help

Index

Contents

Service-Oriented Architecture

Monday	Tuesday	Wednesday	Thursday	Friday
Introduction Components	REST	Composition	Architecture	Engineering
coffee	coffee	coffee	coffee	coffee
Components	REST	Composition	Architecture	Conclusion
lunch	lunch	lunch	lunch	lunch
Web Services	Qualities	Objects	Semantic Web	
tea	tea	tea	tea	
Web Services	Qualities	Objects	Semantic Web	