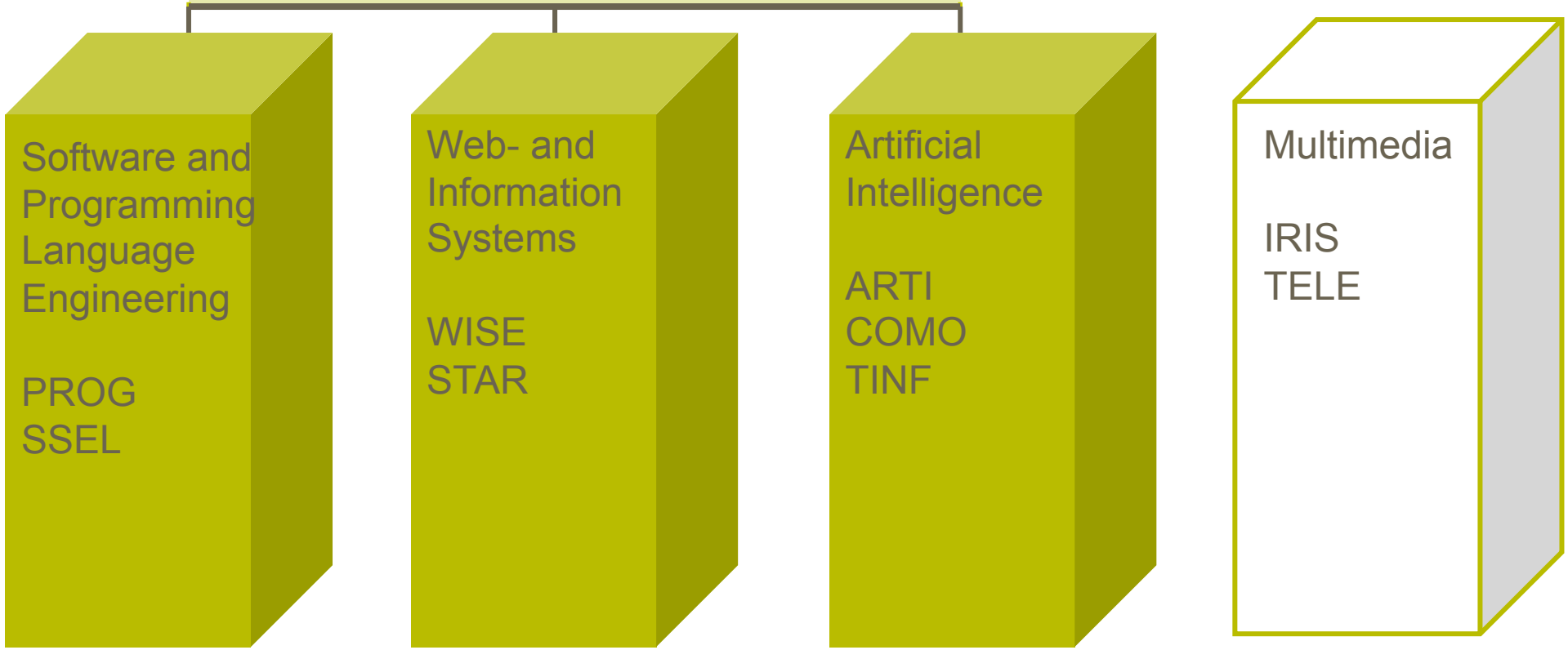
Vrije Universiteit Brussel

Department of Computer Science @VUB

Vrije Universiteit Brussel

Department of Computer Science

80+ Researchers
- 10 professors
- 15 post-doc's
- 55 pre-doc's



Vrije Universiteit Brussel

System and Software Engineering & Programming Technology

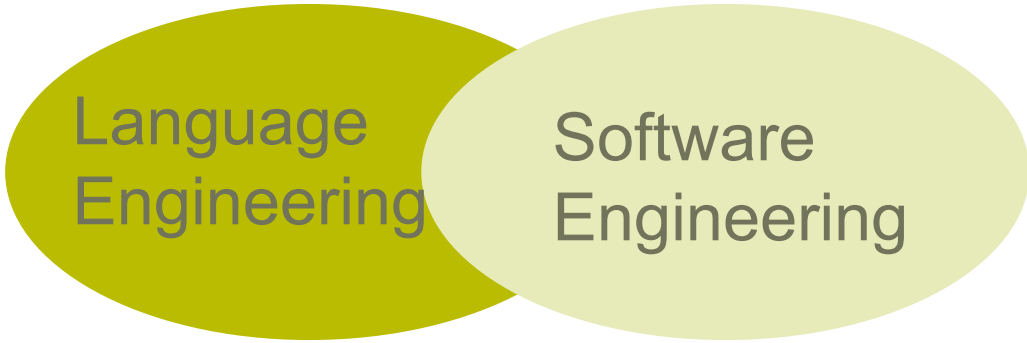
Labs

Prof. Dr. Theo D'Hondt
Prof. Dr. Viviane Jonckers
Prof. Dr. Wolfgang De Meuter

Vrije Universiteit Brussel

PROG & SSEL Labs

30+ Researchers
- 3 professors
- 8 post-doc's
- 23 pre-doc's



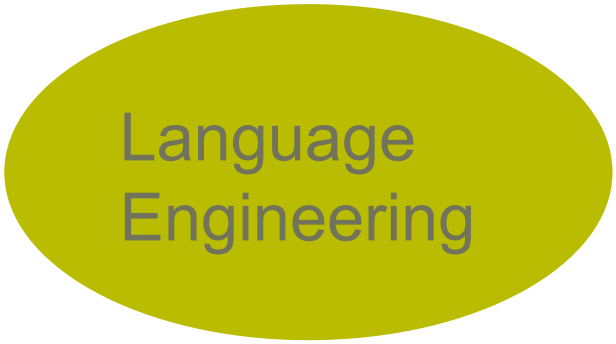
Prof. Dr. Theo D'Hondt
http://prog.vub.ac.be

Prof. Dr. Viviane Jonckers
http://ssel.vub.ac.be

Prof. Dr. Wolfgang De Meuter
http://prog.vub.ac.be

Programming Language Engineering

- **Aspect-Oriented Programming**
  - Rich pointcut languages
  - AOP instantiations (components, workflows, ...)
- **Declarative Meta Programming**
- **Context- & Ambient-Oriented Programming**
- **Concurrent& Multicore Programming**
- **Quantum Programming**
- **Implementation technology**
  - Virtual machines, memory management, language interpreters, ...



Software Engineering

- **Aspect-Oriented Software Development**
  - Aspect mining, pointcut fragility, aspect interaction
  - (Visual) IDE
- **Model Driven Engineering**
  - Model consistency checking
  - Model refinement, extraction, refactoring
- **Service Oriented Software Development**
  - Workflow languages
  - Service discovery, selection, deployment, management
- **Knowledge-Intensive Software**
  - Explicit and active use of domain knowledge (dom implementation and business domain)
  - Static and dynamic program analysis

Artefact-driven Research

http://prog.vub.ac.be - http://ssel.vub.ac.be

CDDToolkit

ContextL

Linglets

Jasco

PlatformKit

KALA

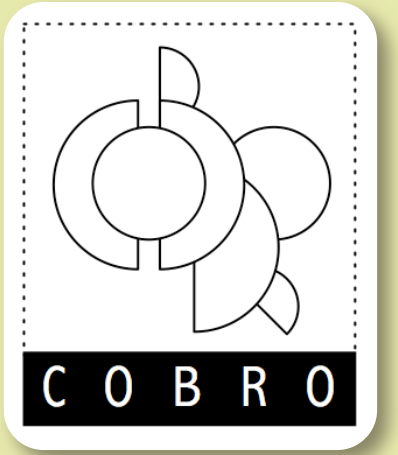
Unify

WSML

PacoSuite

CARMA

FuseJ



Behave

SCE

Padus

SelfSync

Project-driven Research

http://prog.vub.ac.be - http://ssel.vub.ac.be

AspectLab

AOSD-Europe

DyBrowSE

CoDAMoS

WIT-CASE

ORION

VariBru

IWT/FWO grants

Caramelos

MoVES

CrypTask

Stadium

Rococo

Safe-Is

MoVES

Vrije Universiteit Brussel

Software Languages Lab

@VUB

Uniform Modularization of Workflow Concerns using *Unify*

Niels Joncheere, Dirk Deridder, Ragnhild Van Der Straeten, and Viviane Jonckers

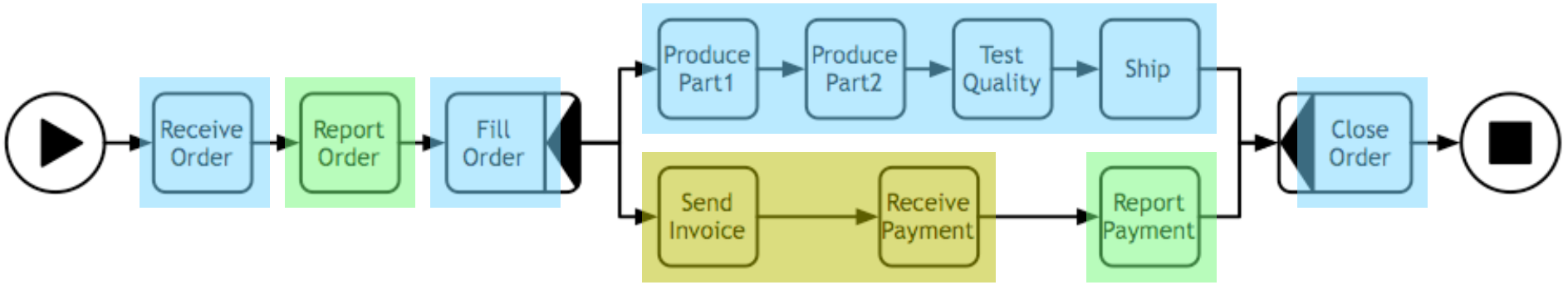
System and Software Engineering Lab (SSEL)

Vrije Universiteit Brussel

Pleinlaan 2, 1050 Brussels, Belgium

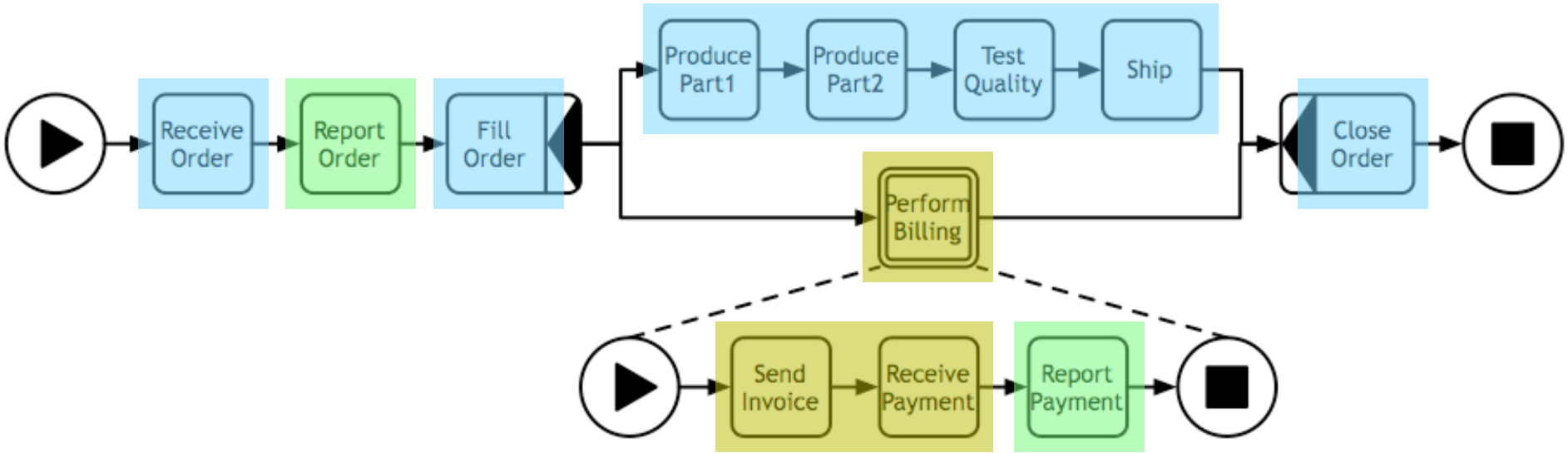
Separation of concerns in workflows - motivation

- Workflows address several **concerns**, e.g. order processing, reporting, billing
- A single, monolithic module is hard to comprehend, maintain, or reuse



Separation of concerns in workflows - AOP to the rescue

- Workflows can be decomposed into sub-workflows
  - For concerns that align with this decomposition a sub-workflow construct in the language is called for
  - For concerns that end up scattered across the workflow (e.g. crosscutting concerns) an AOP style solution is needed



Separation of concerns in workflows - Early work

- AO4BPEL (Charfi & Mezini 2004) present a first AOP extension for BPEL
  - Extra functionality can be added before/after/ around each activity
  - Xpath is used as pointcut language
  - Advice is expresses in BPEL
  - A modified BPEL engine is needed
- Courbis&Finkelstein (2005) also propose an extention to BPEL
  - Advice is expressed in JAVA

PADUS

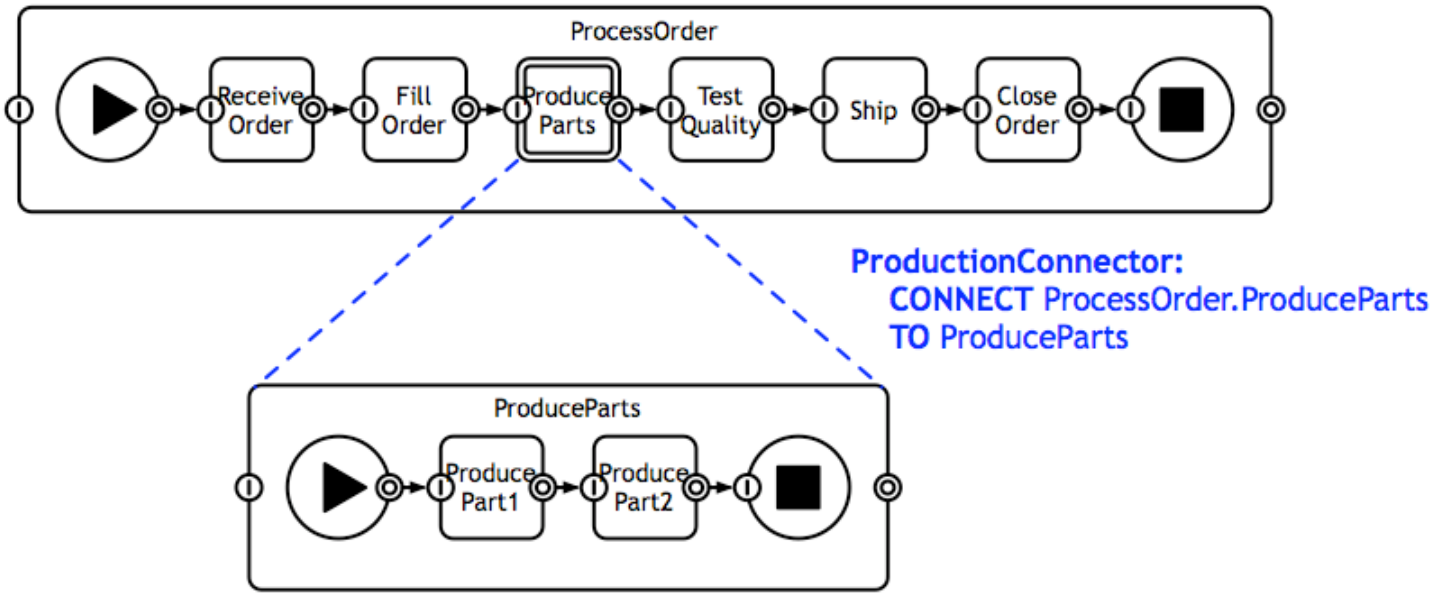
- Our proposal for an extension of BPEL developed in the context of a telecom service-delivery platform project (2006)
  - Rich join point model with Prolog as a pointcut language
  - Introduces next to **before**, **after** and **around** advice also **in** advice to add new behaviour to existing elements (e.g. add a branch to a split)
  - Introduces an explicit deployment construct to specify aspect instantiation to a concrete process
  - Aspects are statically woven, a regular BPEL engine can run the application

Unify

- Unify supports *uniform* modularization of workflow concerns:
  - every concern, regular or crosscutting, is modeled using a sub-workflow
  - Sub-workflows are connected by explicit connector constructs
- Unify is a generic framework, it targets a range of concrete workflow languages
- (Paper and presentation use YAWL concrete syntax for the examples)

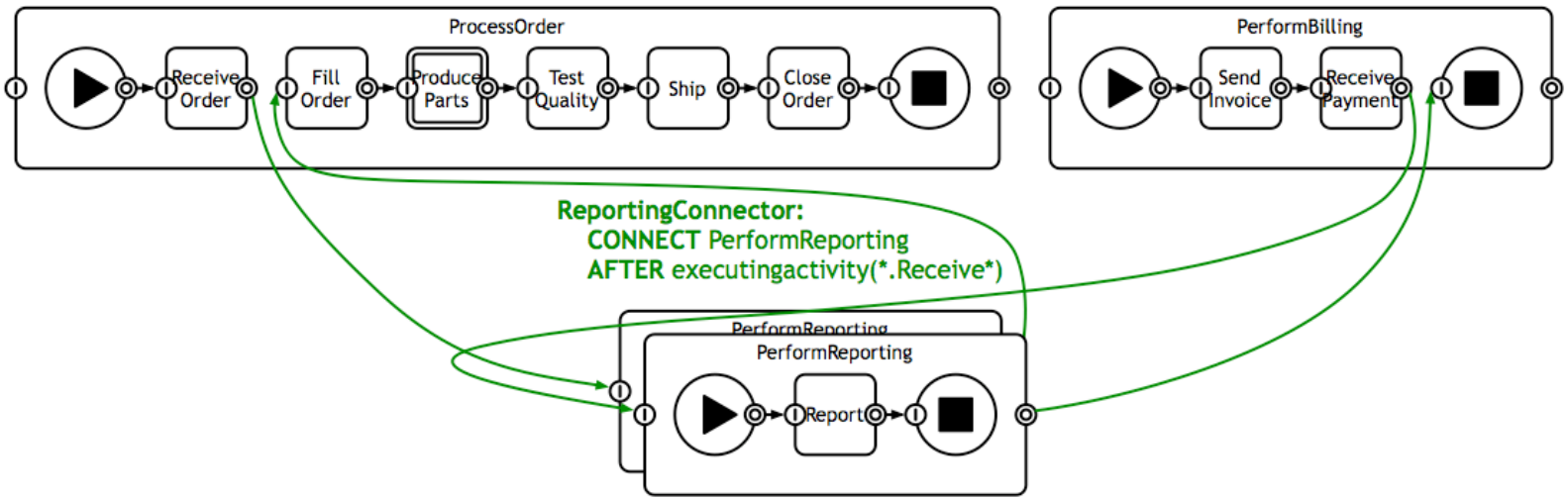
Activity connectors

- Specify regular interactions, correspond to the traditional sub-workflow mechanism
- One concern explicitly 'calls' another concern, the concrete link is specified by the connector in order to decouple both concerns



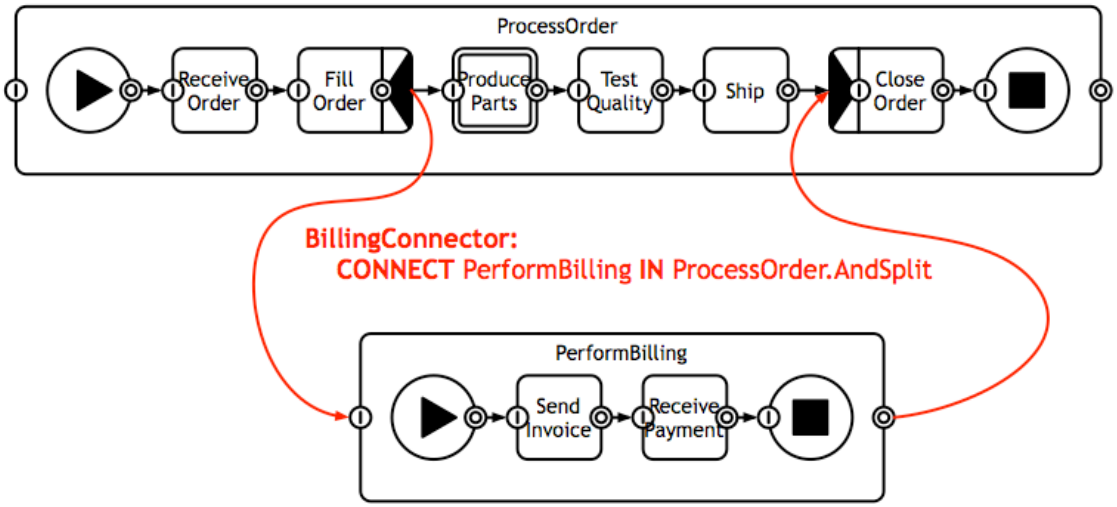
Inversion of control connectors

- Specify aspect-oriented interactions
- One concern is added to another concern, without this other concern being aware of it
- **Before**, **after**, and **replace** connectors correspond to the classic advice types



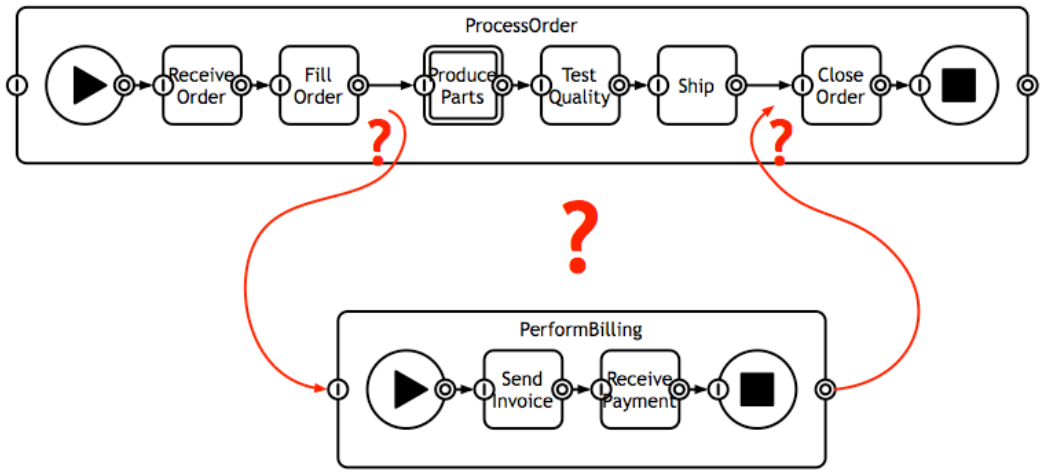
Inversion of control connectors - IN advice

- **in** connectors correspond to Padus's in advice type, and allow inserting a concern as an extra branch to an existing split



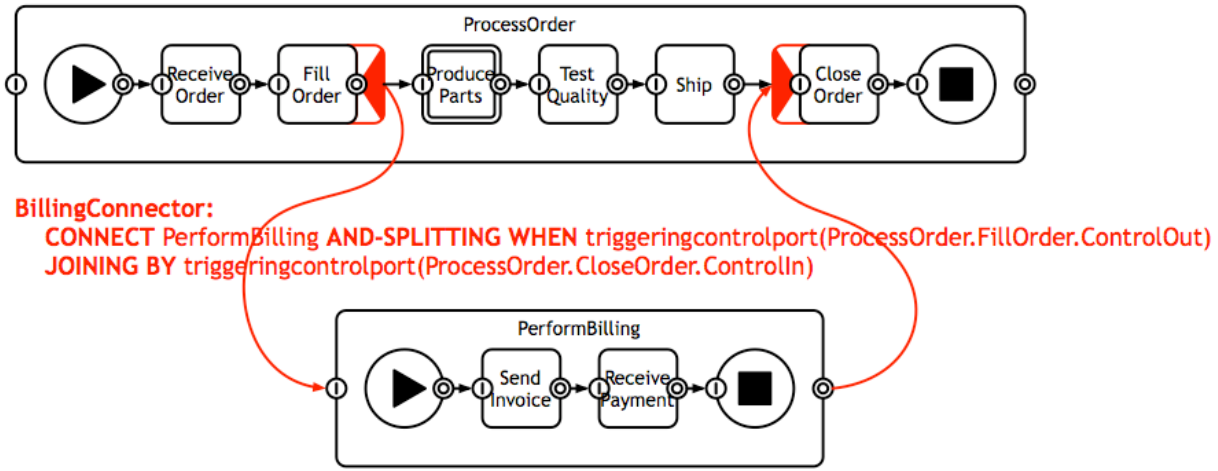
The FREE advice type - motivation

- For example, no existing approach supports executing an advice in parallel with a certain part of a workflow if there is not already a split present

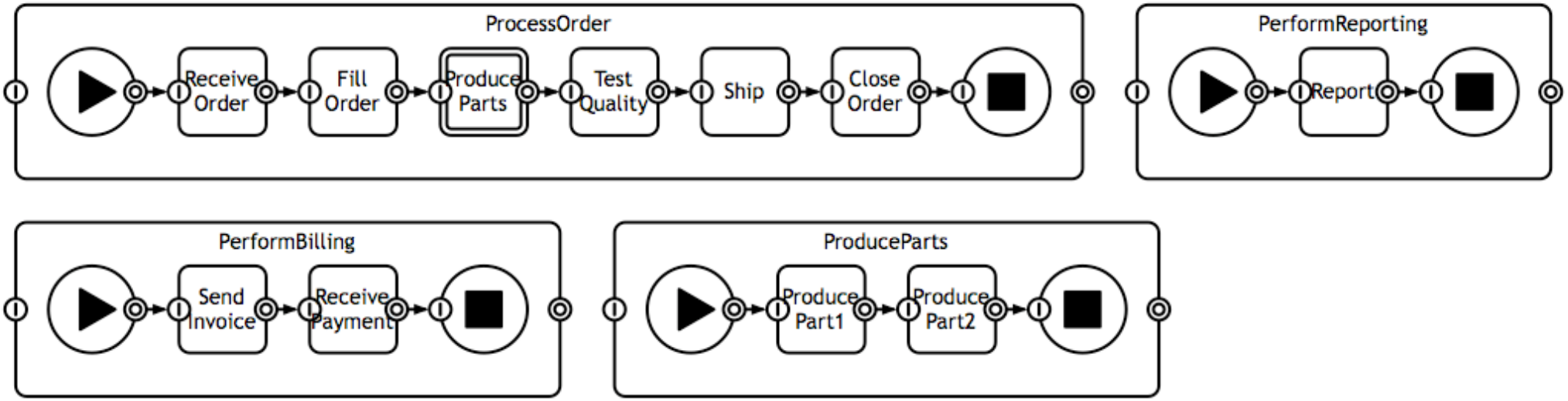


The FREE advice type

- The **free** advice type allows splitting a concern's control flow into another concern at any point of its execution, and joining at any other point of its execution
- Repeated use of the free advice may show certain patterns in the way it is used, which can lead to the creation of new advice types



Complete Example

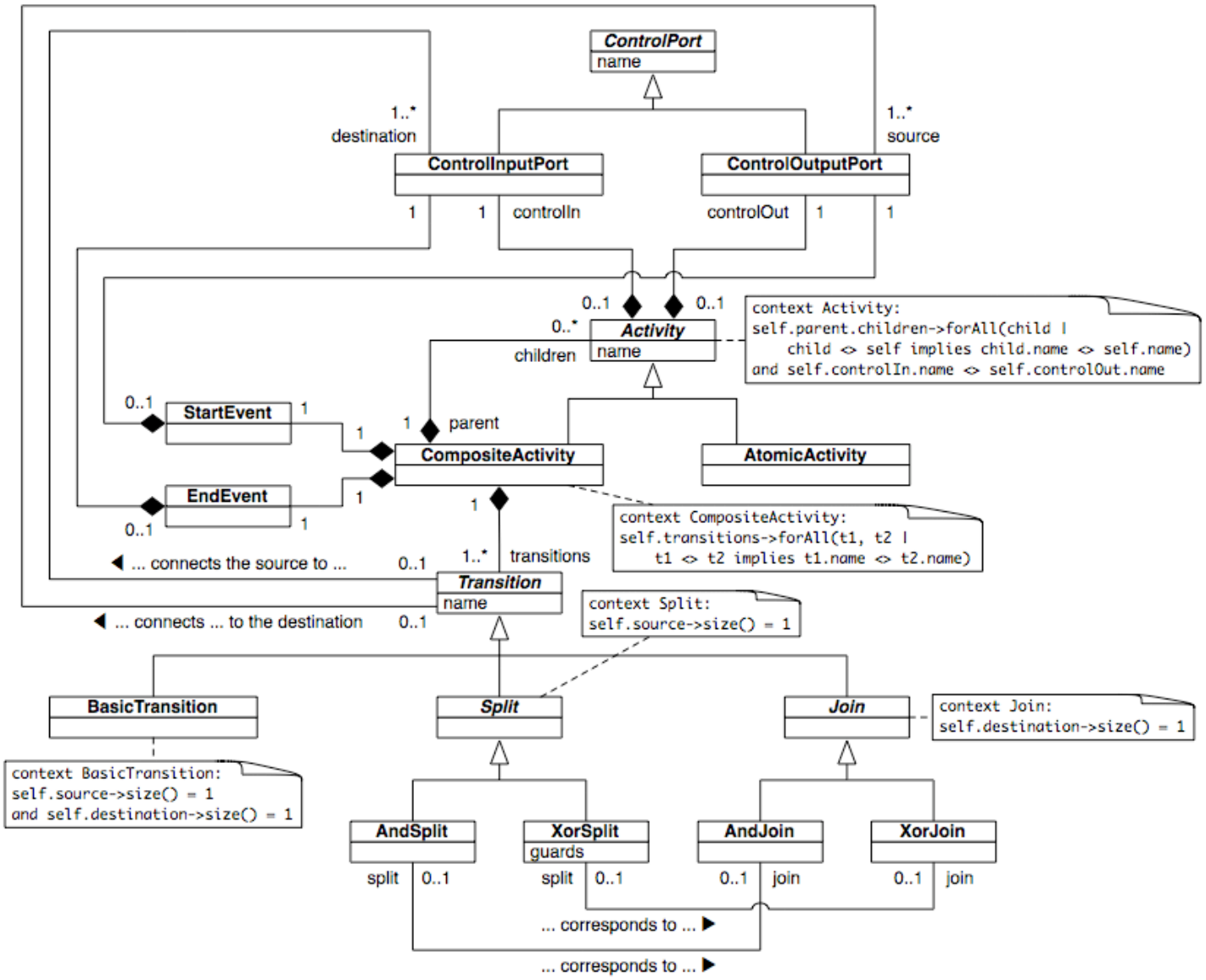


ProductionConnector: CONNECT ProcessOrder.ProduceParts **TO** ProduceParts

BillingConnector:
CONNECT PerformBilling **AND-SPLITTING WHEN** triggeringcontrolport(ProcessOrder.FillOrder.ControlOut)
JOINING BY triggeringcontrolport(ProcessOrder.CloseOrder.ControlIn)

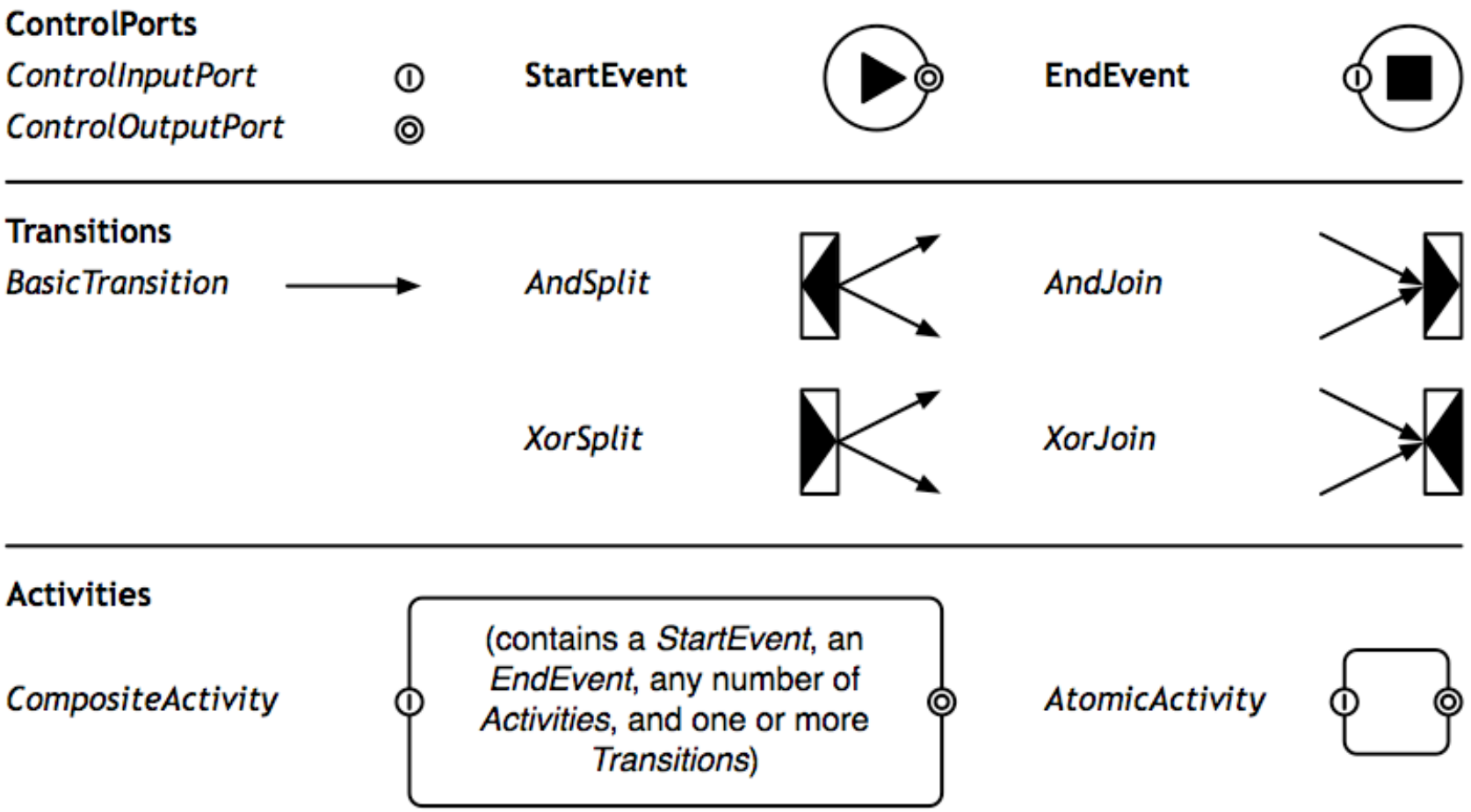
ReportingConnector: CONNECT PerformReporting **AFTER** executingactivity(*.Receive*)

Base language meta-model



- ✓Arbitrary workflows
- ✓Composite pattern
- ✓Concerns are modeled using *CompositeActivities*
- ✓Control flow is modeled using *Transitions*
- ✓*ControlPorts* allow intercepting a concern's control flow

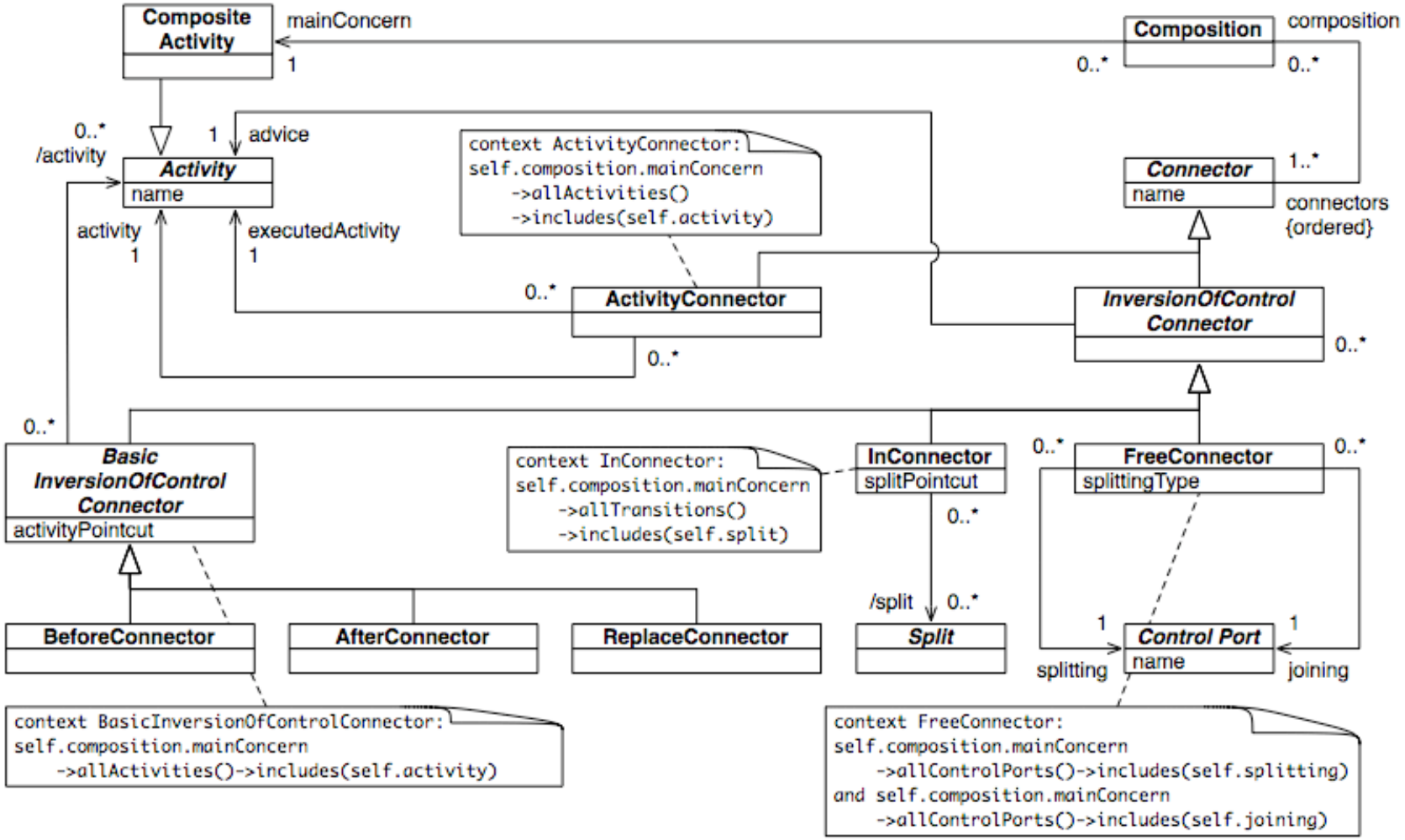
Mapping to YAWL notation



Base language semantics

- The semantics of the Unify base language is defined in terms of Petri nets
- Petri nets are recognized as a good execution model for workflows
- The mapping from workflows to Petri nets is well-known:
  - Activities are mapped to Petri net transitions
  - StartEvents, EndEvents and Transitions are mapped to Petri net places
  - Splits and Joins are mapped to the appropriate combination of Petri net transition(s) and place(s)

Connector meta-model



Connector syntax

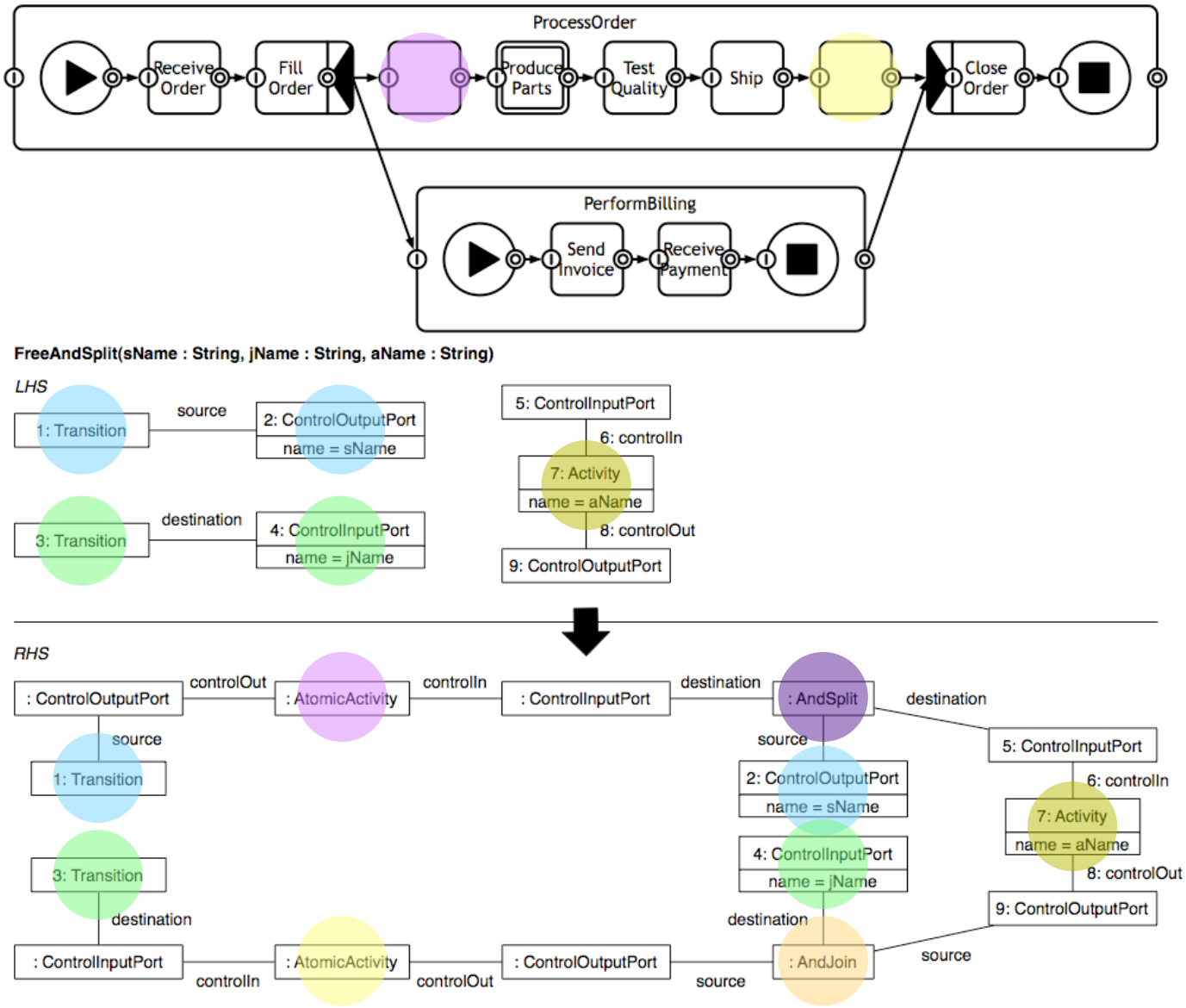
```
<connector> ::= <activity-connector>
              | <inversion-of-control-connector>

<activity-connector> ::= "CONNECT" <activity>
                         "TO" <activity>

<inversion-of-control-connector> ::= "CONNECT" <activity>
                                     <advice>

<advice> ::= "BEFORE" <activity-pointcut>
           | "AFTER" <activity-pointcut>
           | "REPLACING" <activity-pointcut>
           | "IN" <split-pointcut>
           | ("AND-" | "XOR-") "SPLITTING WHEN" <control-port-pointcut>
             "JOINING BY" <control-port-pointcut>
```

Connector semantics



- ✓Concerns are woven on the abstract syntax level
- ✓The weaving is defined using 13 graph transformation rules (2 in paper, all 13 in technical report)

Implementation

- Workflows, connectors and compositions are parsed into Java objects that conform to our meta-model
- Connectors are applied to the main concern in the specified ordering
- Weaving is performed according to the semantics specified by the graph transformations
- The result is transformed into Petri nets and is executed

Future work

- Implement Padus+ a concrete AOP extension of BPEL
- Address data perspective (in context of CAE)
- Address aspect interaction
  - Supporting the developer in specifying a correct ordering of connectors
  - Verification of user-defined constraints

Conclusion

- Unify improves on existing research on the following three points:
  - It supports uniform modularization of both main and crosscutting concerns
  - It provides more expressive advice types than before, after, and around advice types
  - It is designed to be independent of a particular concrete syntax through the use of a base language meta model

Thanks!

Niels Joncheere
System and Software Engineering Lab (SSEL)
Vrije Universiteit Brussel
njonchee@vub.ac.be