ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

IKS Information and Communication Systems Research Group

Model-Driven Service Composition with JOpera

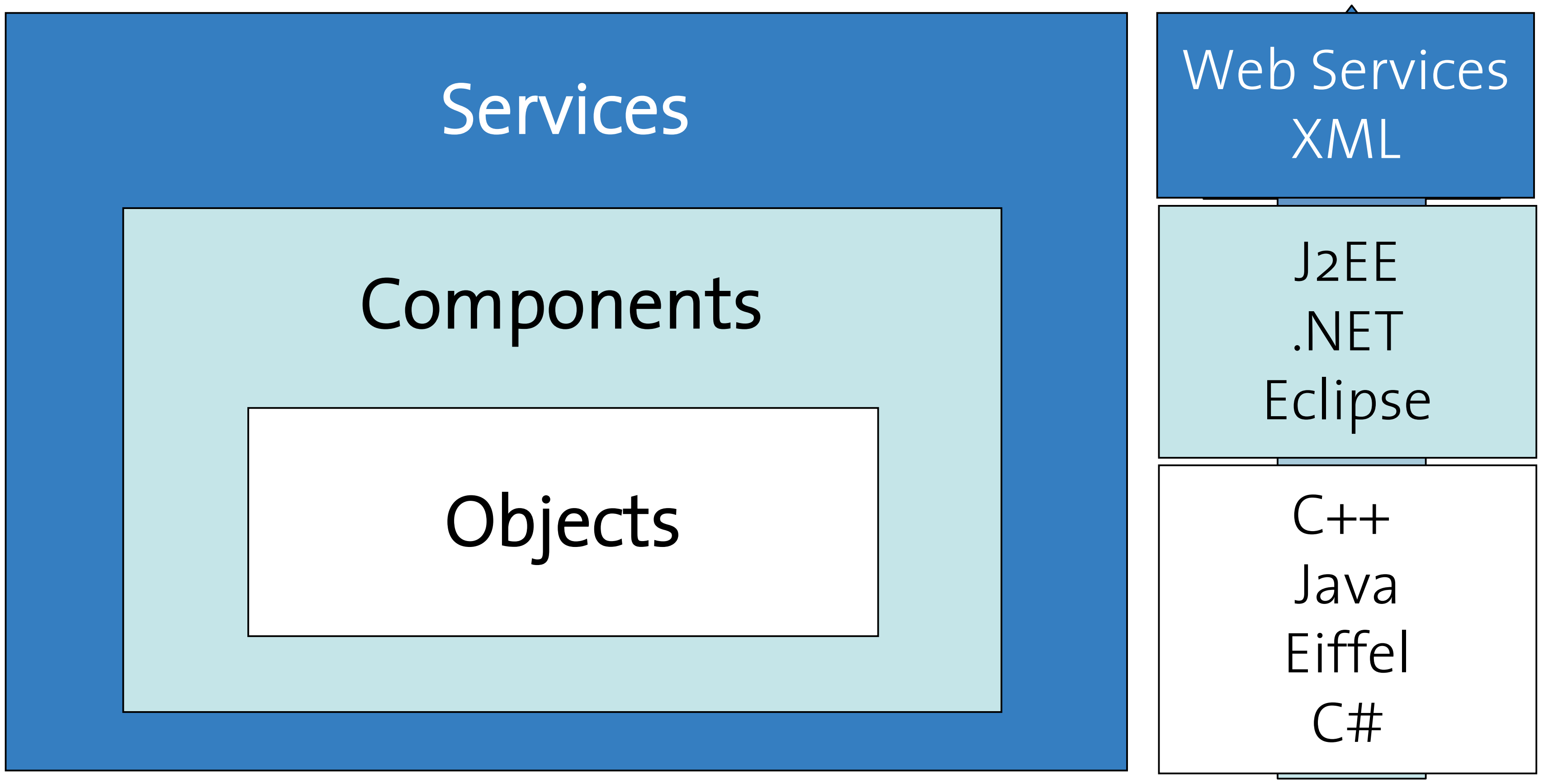
Cesare Pautasso
Department of Computer Science, ETH Zurich, Switzerland
pautasso@inf.ethz.ch – **www.jopera.org**

inf | Informatik Computer Scienc
Jopera
Process Support for Web Services

JOpera is kindly supported by:

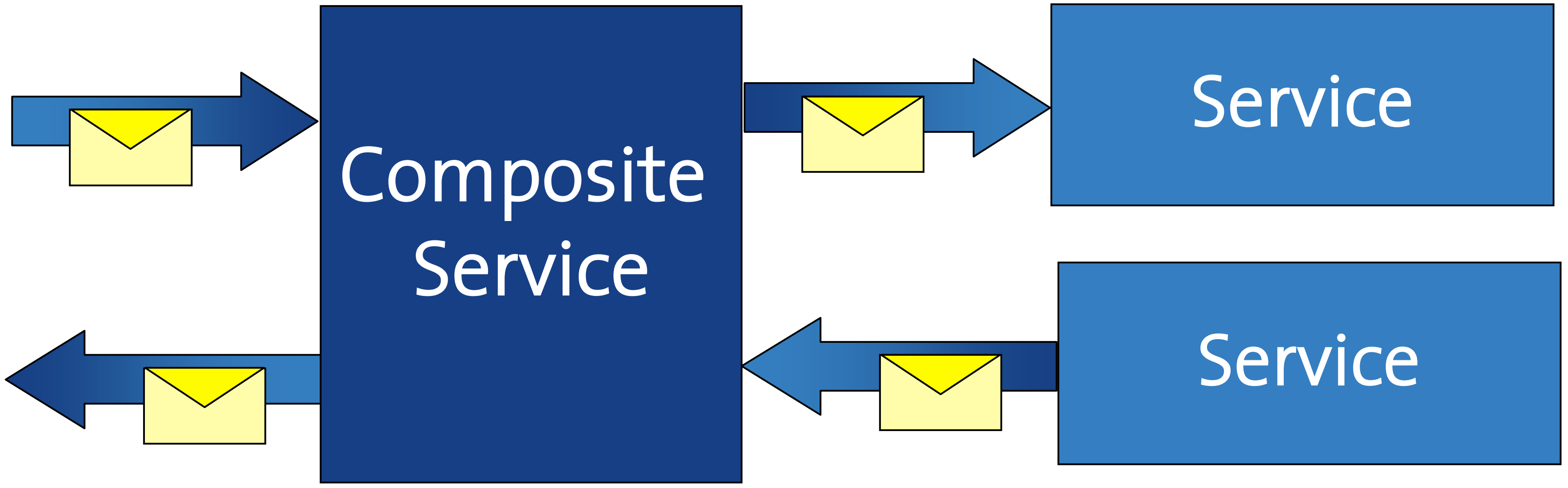
- ETH Zurich
  - **IKS Group**, Prof. Gustavo Alonso
- European Union
  - **ADAPT** - Middleware Technologies for Adaptive and Composable Distributed Components (finished 2005)
  - **SODIUM** - Service Oriented Development in a Unified Framework (until 2007)
  - **AEOLUS** Project - Algorithmic Principles for Building Efficient Overlay Computers (until 2009)
- Hasler Stiftung
  - DICS Project: **Dependable Computing in Virtual Laboratories** (finished 2005)

New Abstractions for Application Integration



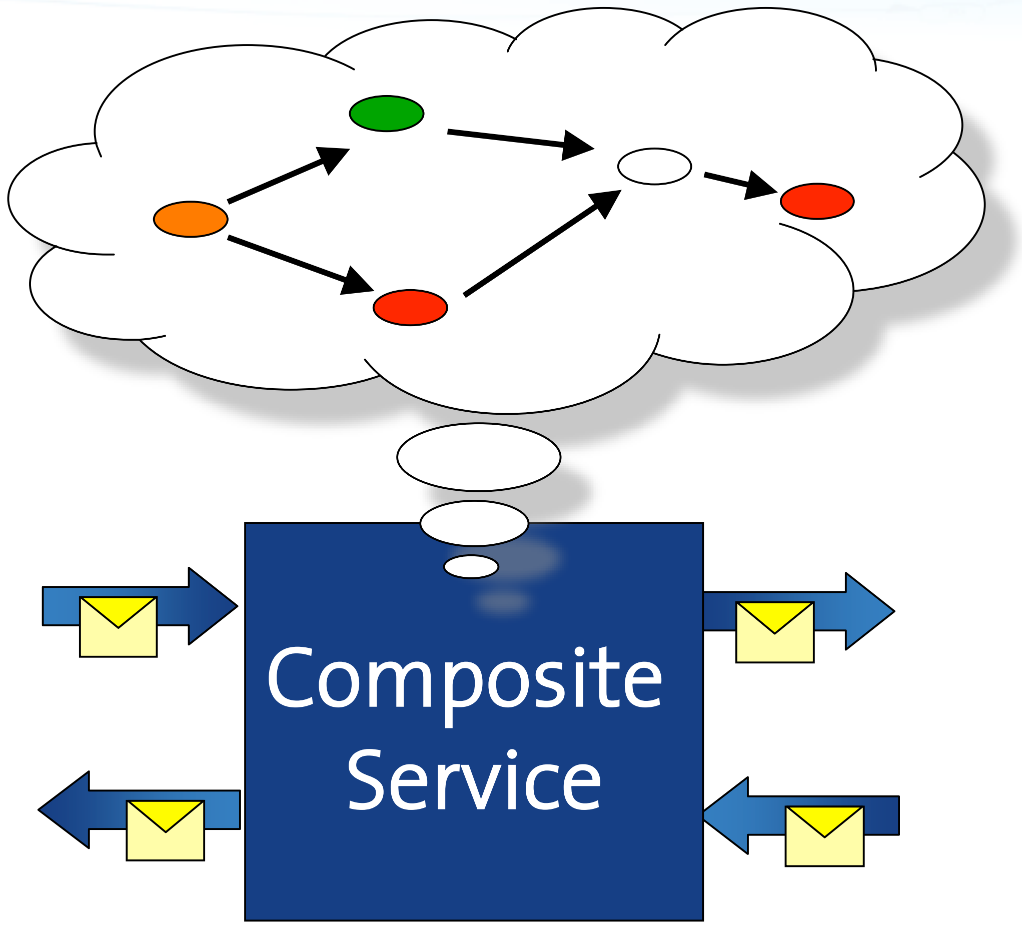
The Problem of Service Composition

- How to build an application by reusing existing components delivered as a service?
- How to script the exchange of messages between a set of services?

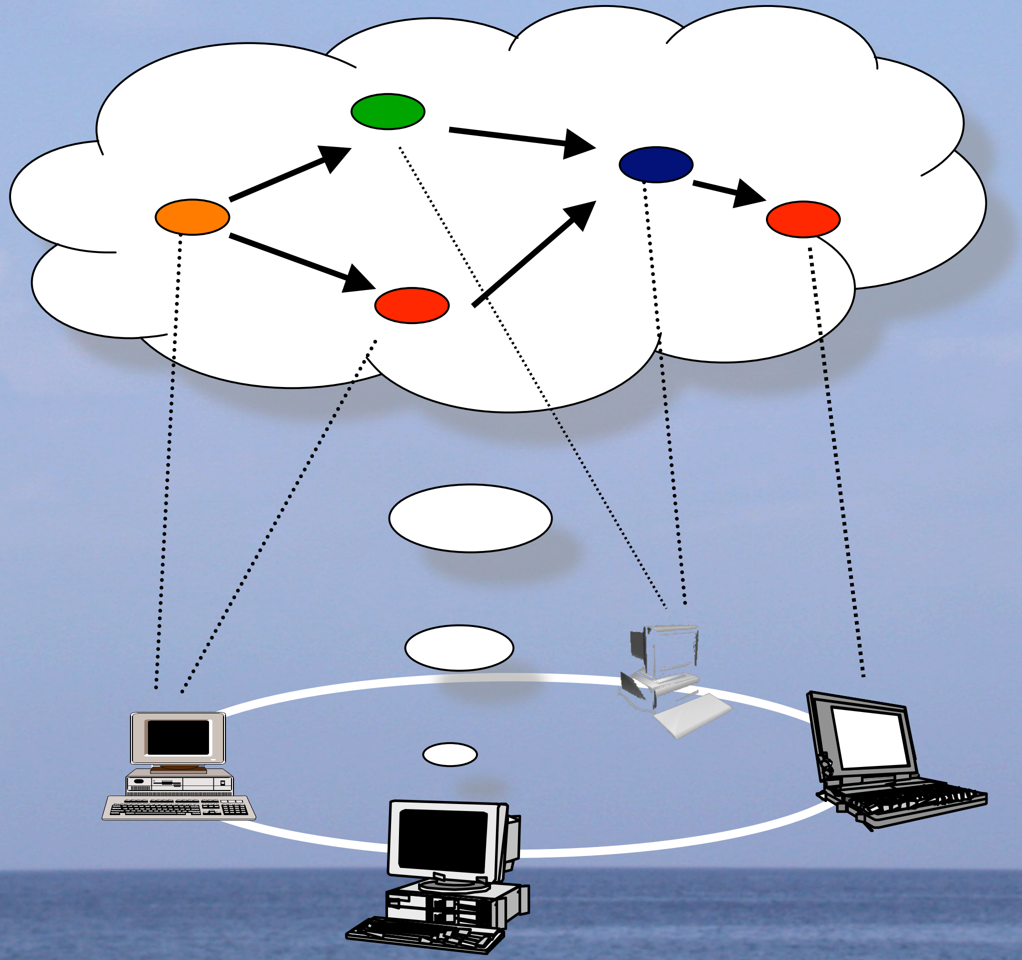


The Model is the Code

- How to model a composition?
- How to execute such a model?
- What kind of services can be composed?



How to model a Service Composition with JOpera?



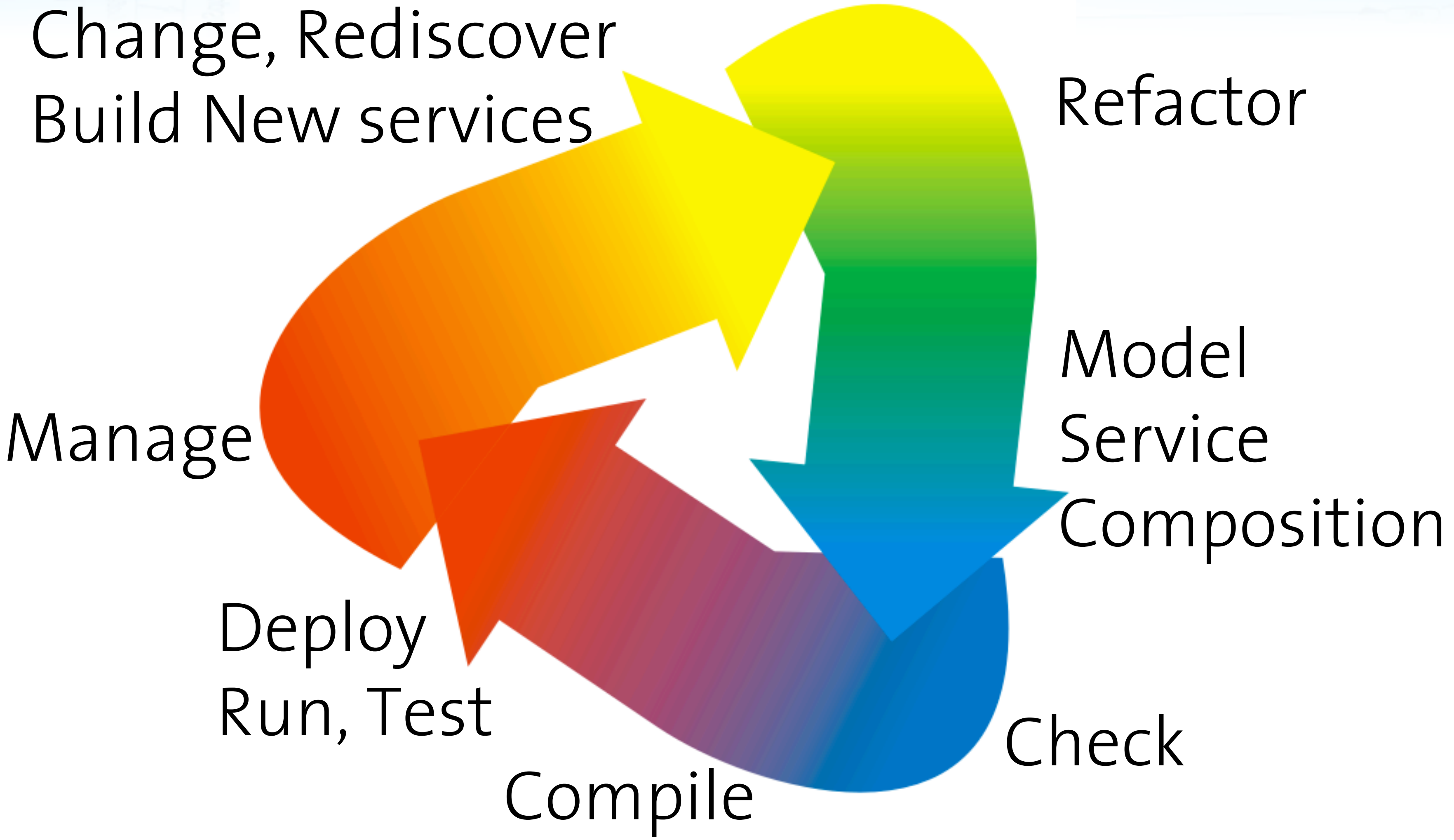
Bottom-up Composition

4. Share and Publish it as Web Service
3. Run, Test, and Debug the execution **within the same modeling environment**
2. Build a composition using a drag, drop and connect **modeling** environment
1. Select component services from a **library**
   - Lookup in a UDDI registry
   - Import from external WSDL
   - Search the standard JOpera library

Top-down Composition

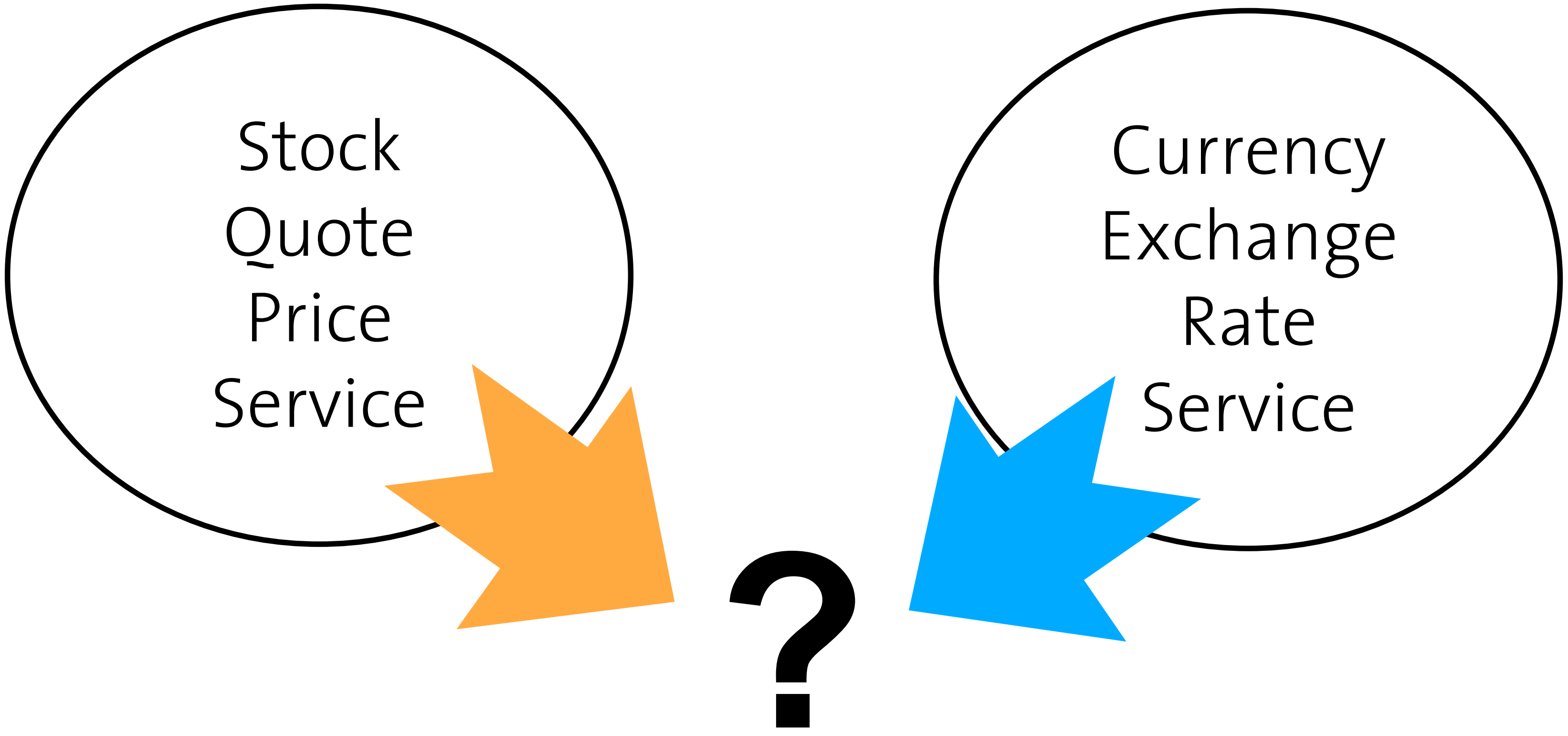
1. Define a **goal** and Draw a *skeleton of the composition* that satisfies it
2. Refine it and **Bind** services into it:
   - Search for existing matching services
   - Build missing services (if necessary)
   - Add required data transformations
3. Run, Test, and Debug the execution **within the same modeling environment**
4. Share and Publish it as Web Service

Iterative Composition

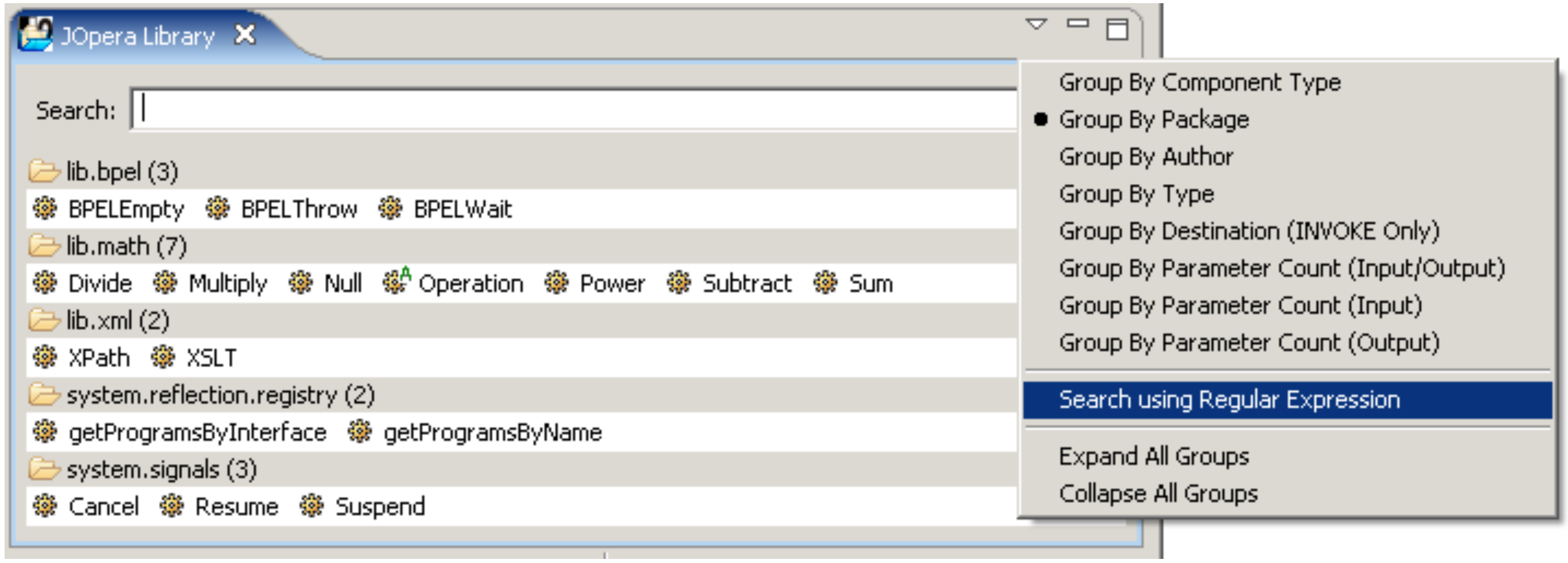


Demo 1

- Stock Quote Currency Conversion



Service Library


JOpera Library

Search:

lib.bpel (3)

BPELEmpty BPELThrow BPELWait

lib.math (7)

Divide Multiply Null Operation Power Subtract Sum

lib.xml (2)

XPath XSLT

system.reflection.registry (2)

getProgramsByInterface getProgramsByName

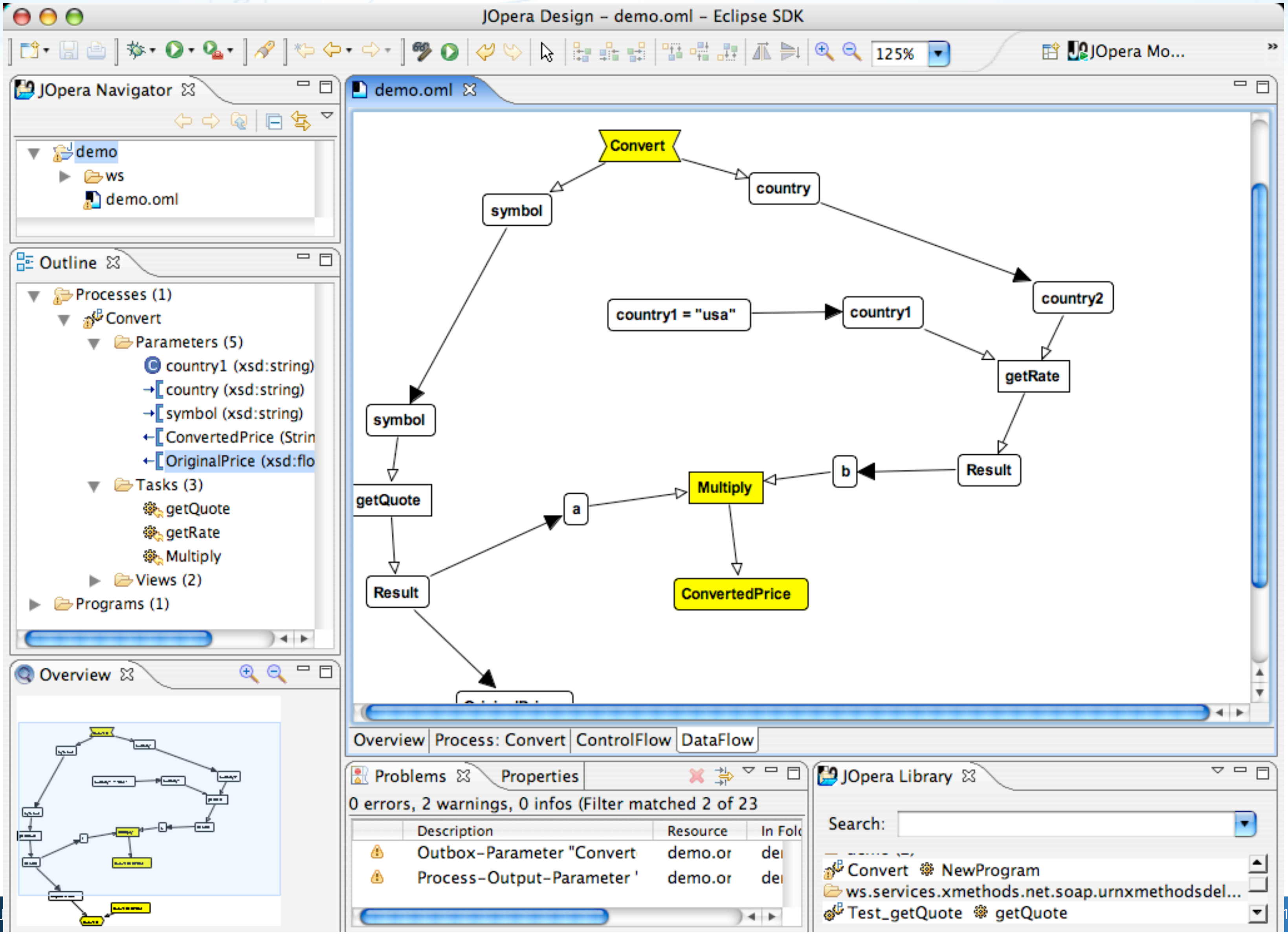
system.signals (3)

Cancel Resume Suspend

Group By Component Type
● Group By Package
Group By Author
Group By Type
Group By Destination (INVOKE Only)
Group By Parameter Count (Input/Output)
Group By Parameter Count (Input)
Group By Parameter Count (Output)
Search using Regular Expression
Expand All Groups
Collapse All Groups

1. Search services as you type (also with regex)
2. Group services by different (orthogonal) criteria

Drag, Drop and Connect



JOpera Design – demo.oml – Eclipse SDK

JOpera Navigator

- demo
  - ws
  - demo.oml

Outline

- Processes (1)
  - Convert
    - Parameters (5)
      - country1 (xsd:string)
      - country (xsd:string)
      - symbol (xsd:string)
      - ConvertedPrice (Strin
      - OriginalPrice (xsd:flo
    - Tasks (3)
      - getQuote
      - getRate
      - Multiply
    - Views (2)
- Programs (1)

Problems | Properties

0 errors, 2 warnings, 0 infos (Filter matched 2 of 23

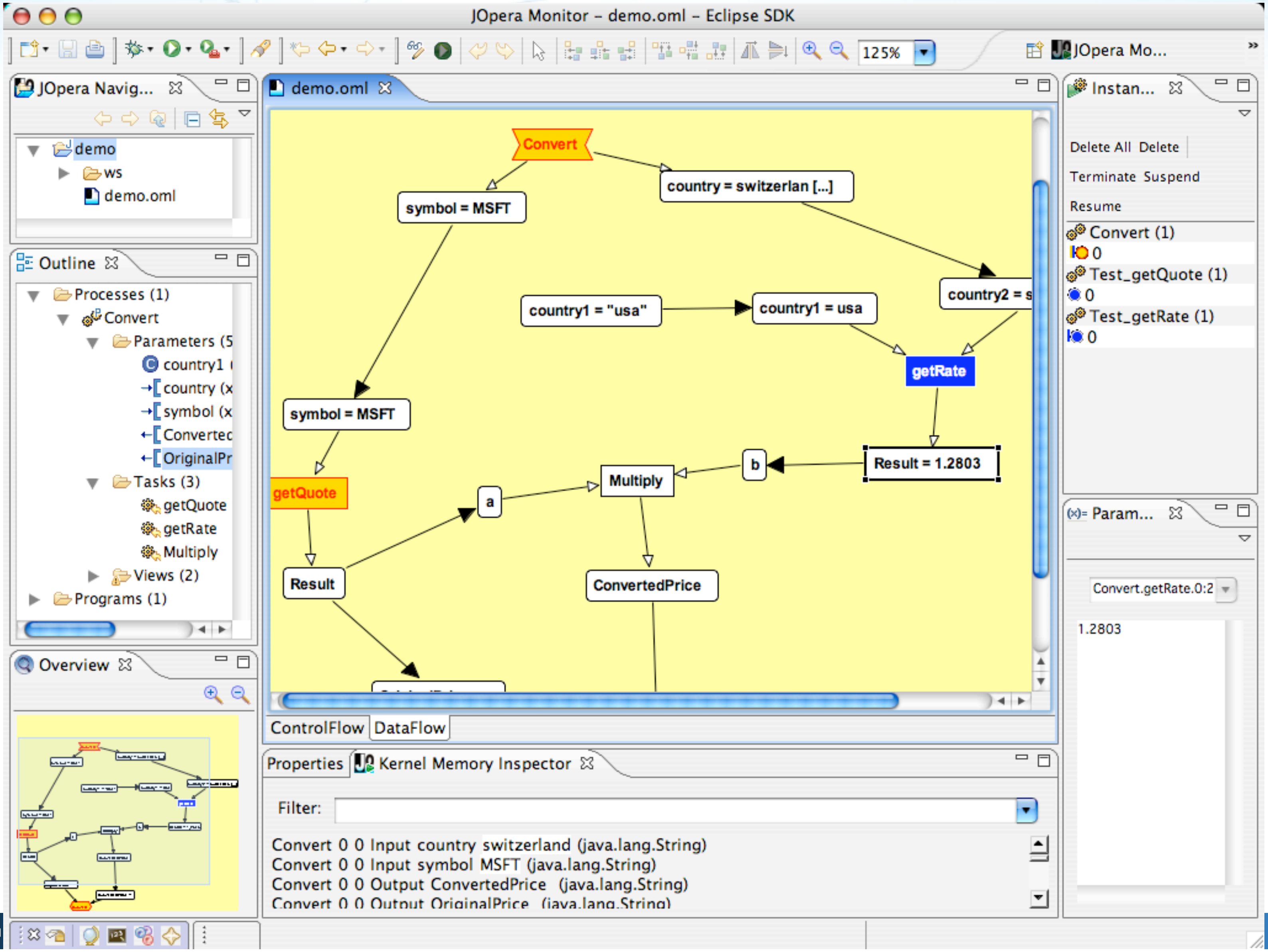
	Description	Resource	In Fold
⚠	Outbox-Parameter "Convert	demo.or	de
⚠	Process-Output-Parameter '	demo.or	de

JOpera Library

Search:

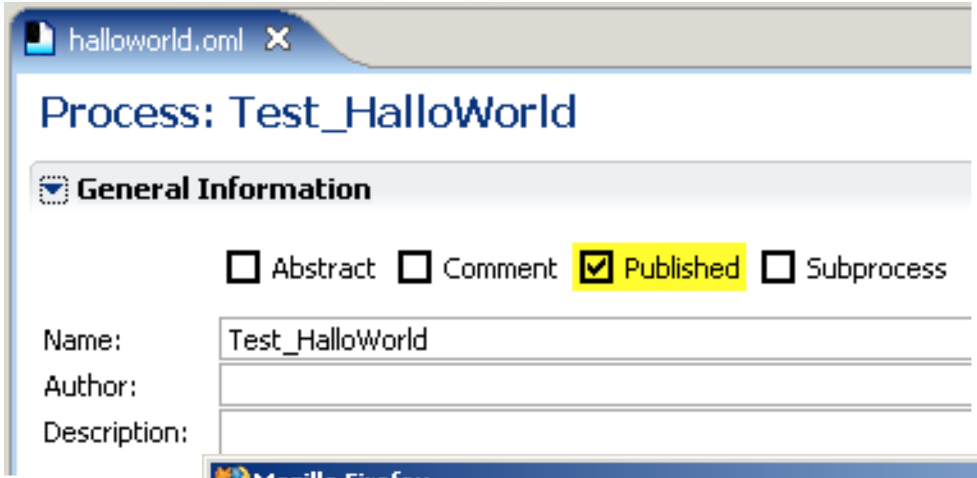
- Convert NewProgram
- ws.services.xmethods.net.soap.urnxmethodsdel...
- Test_getQuote getQuote

Run, Monitor, Steer and Debug



```
Convert 0 0 Input country switzerland (java.lang.String)
Convert 0 0 Input symbol MSFT (java.lang.String)
Convert 0 0 Output ConvertedPrice  (java.lang.String)
Convert 0 0 Output OriginalPrice  (java.lang.String)
```

Publish as a Web/Grid service

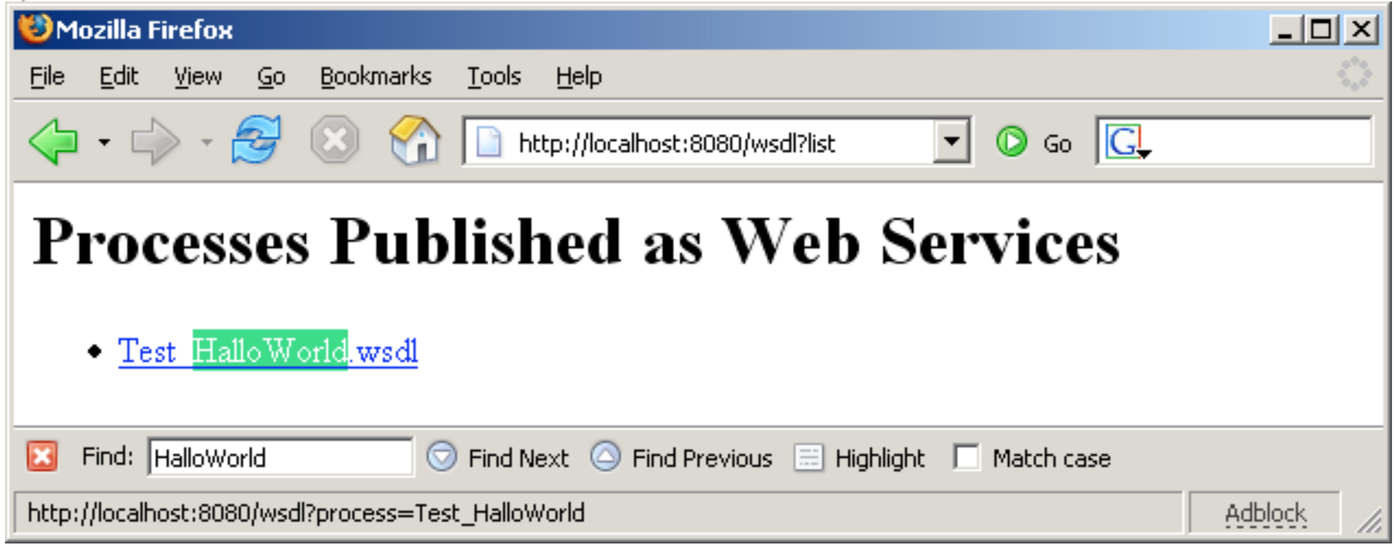

halloworld.oml

Process: Test_HalloWorld

General Information

[ ] Abstract [ ] Comment [✓ Published] [ ] Subprocess

Name:	Test_HalloWorld
Author:	
Description:	



http://localhost:8080/wsdl?list

Processes Published as Web Services

- Test_HalloWorld.wsdl

Find: HalloWorld

http://localhost:8080/wsdl?process=Test_HalloWorld

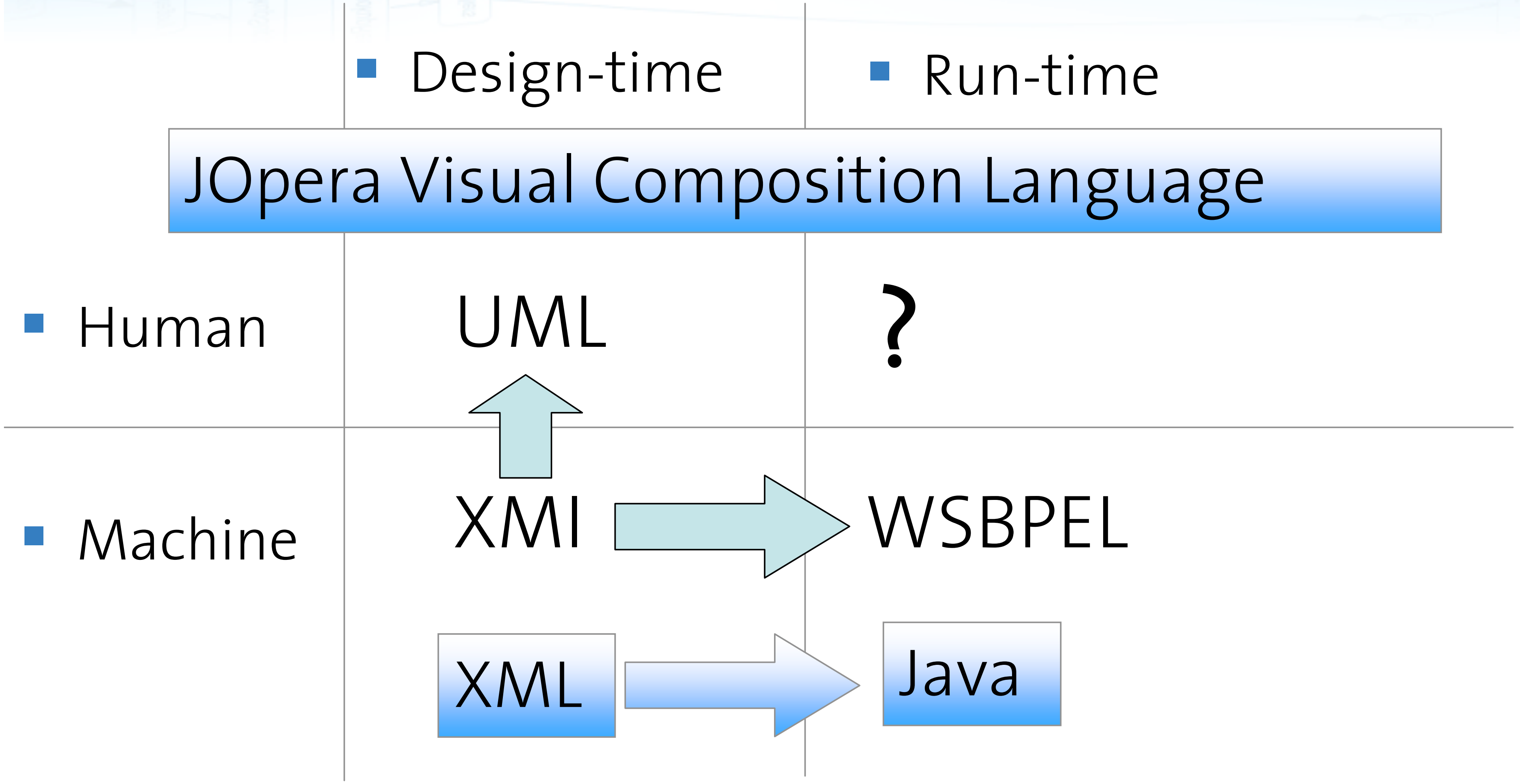
With one mouse click!

[e-Science2005]

Modeling Service Compositions

- What are good abstractions for modeling service composition?
  - Structure (UML, Architectural Description Languages)
  - Behavior (BPM, Activity Diagrams, Business Rules)
- What about the syntax?
  - Visual, Textual (XML), or both
- What about the semantics?
  - Formal, Verifiable, and Executable

Modeling Service Compositions



Model Transformation in JOpera

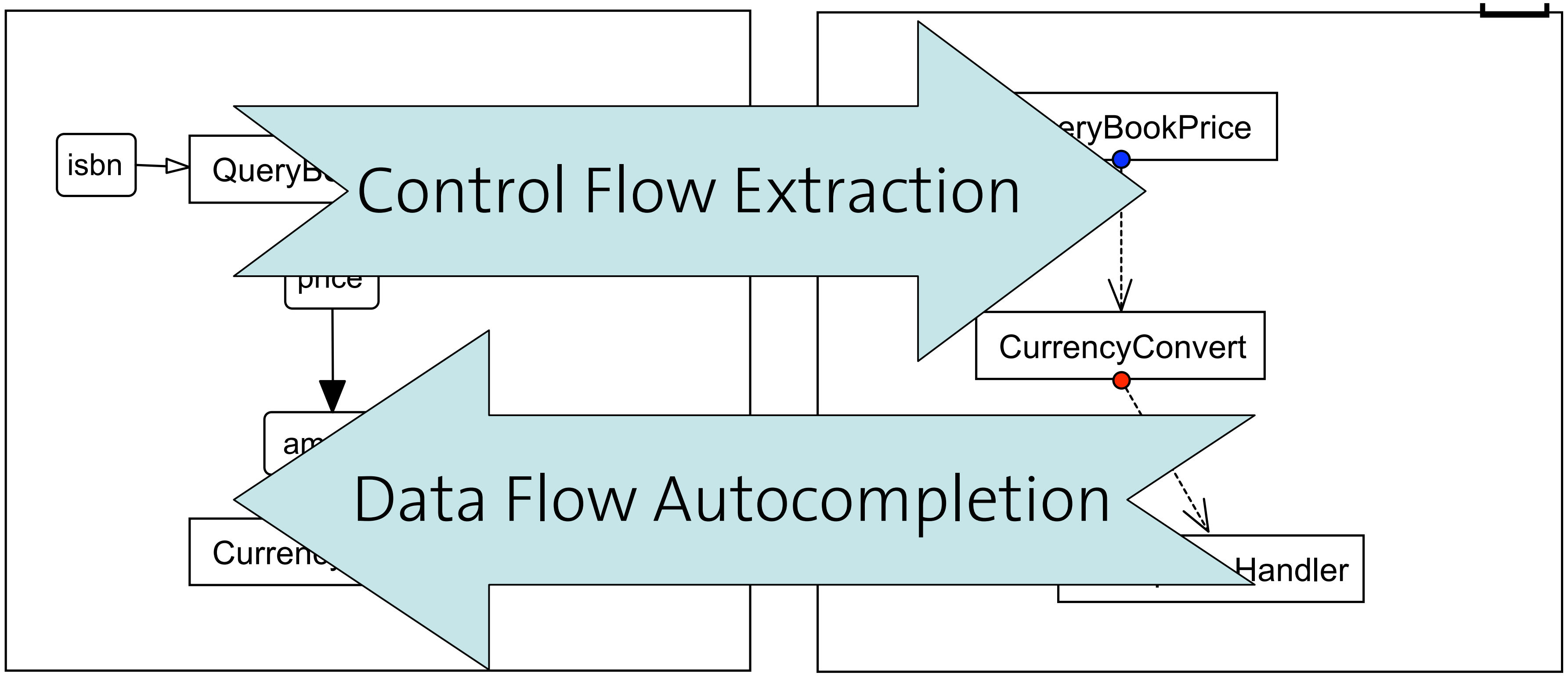
- What are good abstractions for modeling a service composition? **It depends**

▪ End user	JOpera Visual Composition Language
▪ Developer	Graphs and Dependency Rules
▪ Storage	XML (OML)
▪ Compiler	Intermediate Representation (FSM)
▪ Execution	Java Bytecode

JOpera Visual Composition Language Overview

- Services are composed using processes, which define their interactions using two graphs:
  - Data Flow
  - Control Flow

[HCC2003]

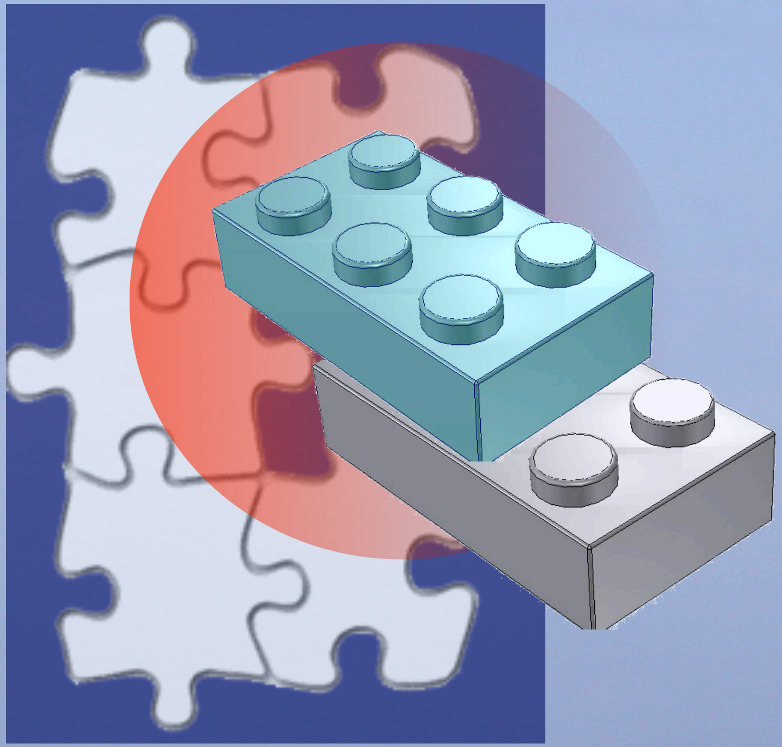


JOpera Visual Composition Language Features

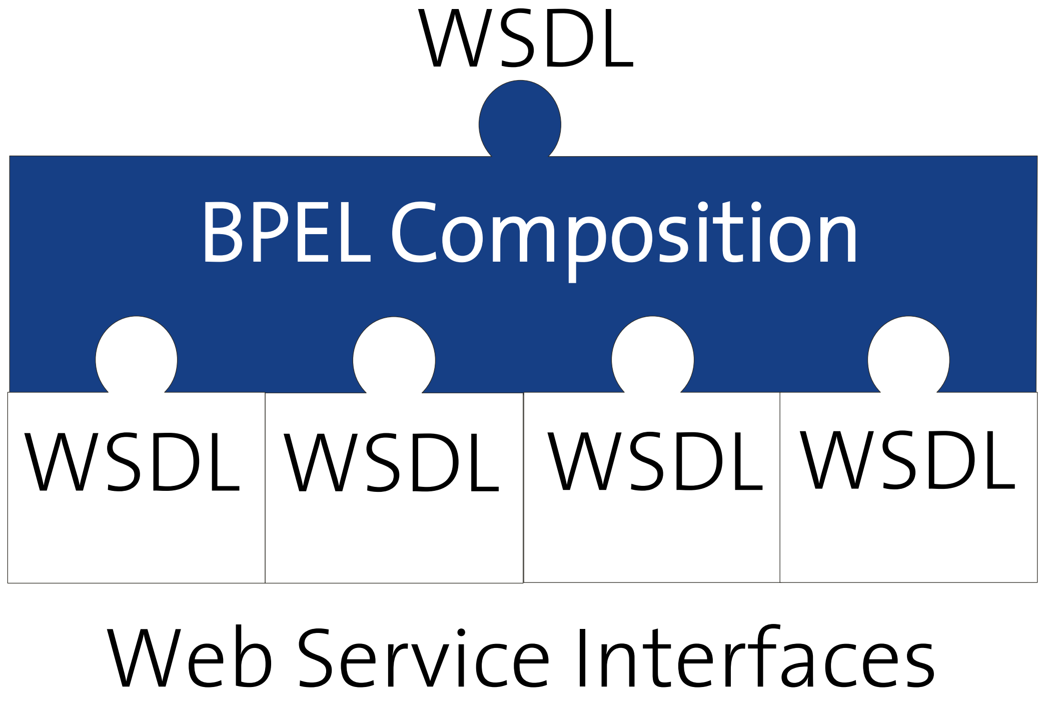
- Processes model generic service composition
  - **Data flow** as the primary representation
  - Explicit **control flow** (branch, synchronization, exception handling, loops, pipeline, workflow patterns)
- **SubProcesses**: Modularity, Nesting and Recursion
- **First order functions**
  - Map (parallel/sequential/discriminator) and Reduce
- **Reflection** (introspection)
  - Dynamic late binding
  - Quality of Service monitoring

[JVLC2005]

What kinds of Services can you compose with JOpera?



What kind of services can you compose with WS-BPEL?



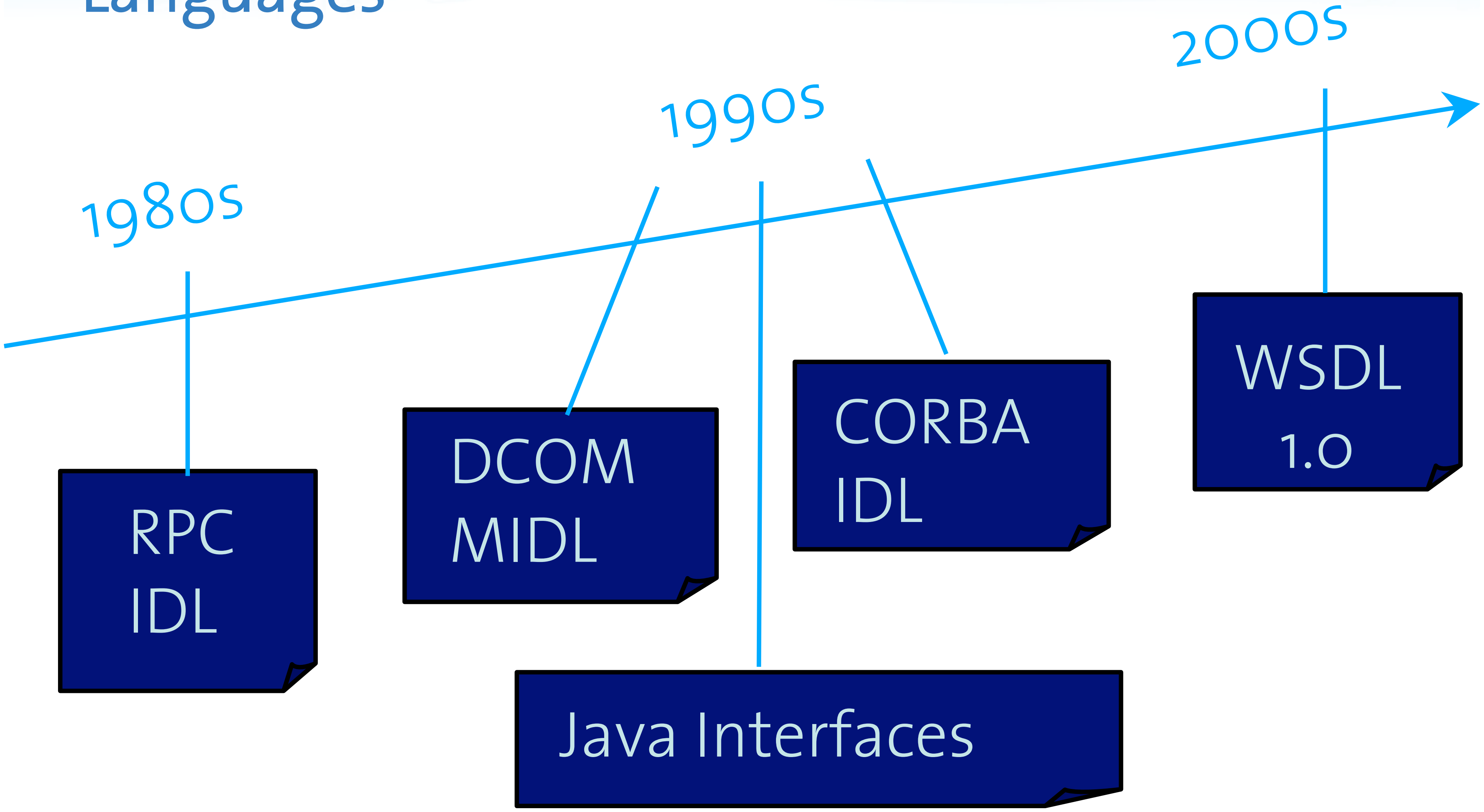
Assumption:
Web Services (SOAP/WSDL) are the only kind of services to be composed

Problem:
extensions to the BPEL standard are needed to support code snippets (**BPELJ**) and human tasks (**BPEL4PEOPLE**)

Problems of composing *only* Web Services

- Web Services are **coarse-grained**
- All existing heterogeneous systems must be **wrapped** as a Web Service
  - Wrapping imposes both a performance penalty and additional development & maintenance costs
- The **adapter/mediator** between mismatching Web services must also be a Web service
- Offline testing difficult
- Web services standards are not stable

A Brief History of Interface Description Languages



Generalizing service composition

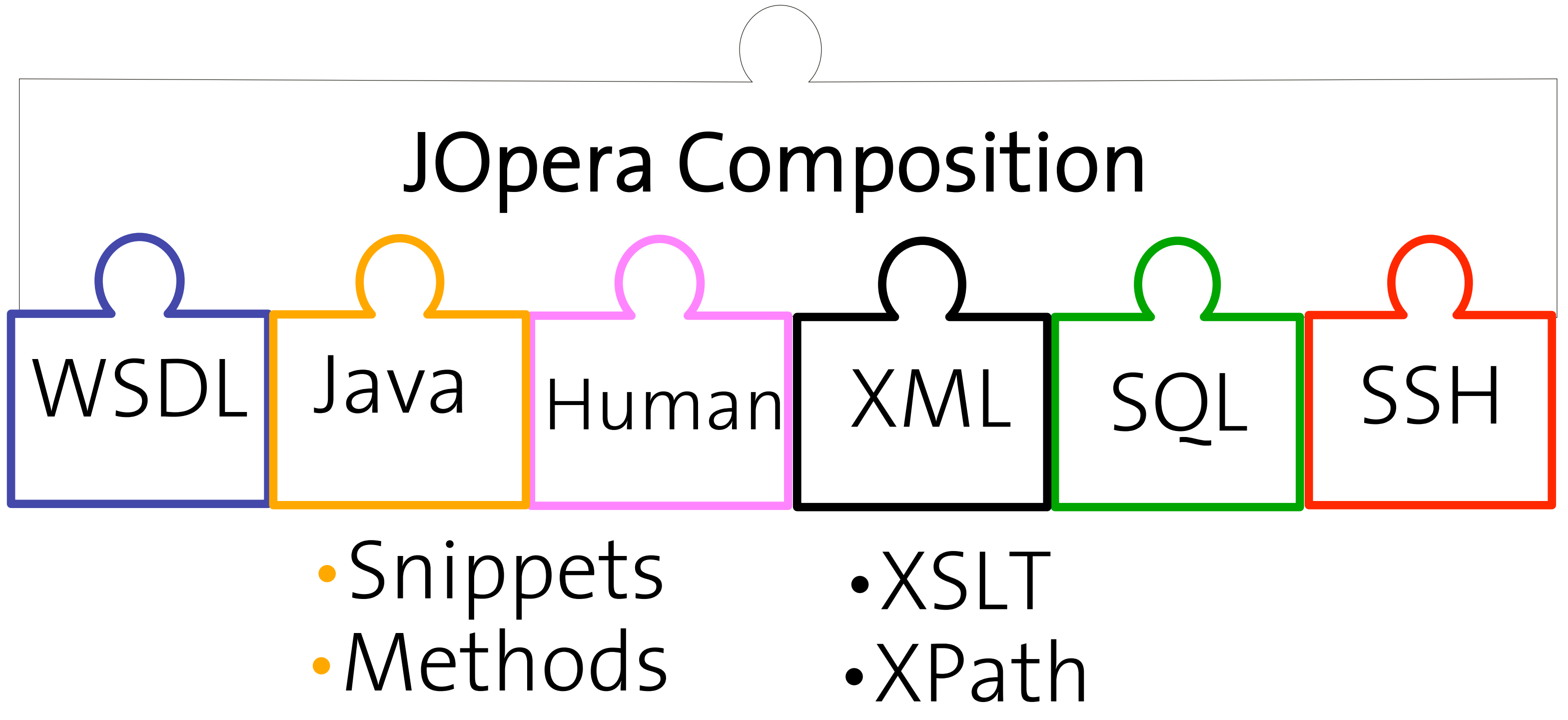
- How to design a language independent of the kinds of services to be composed?

1. Separate the description of the process from the description of how to invoke each of its tasks
2. A process should make minimal assumptions about its tasks (i.e., data flow signature)
3. Bind tasks to different invocation mechanisms without affecting the process definition

[VLDB/TES2004]

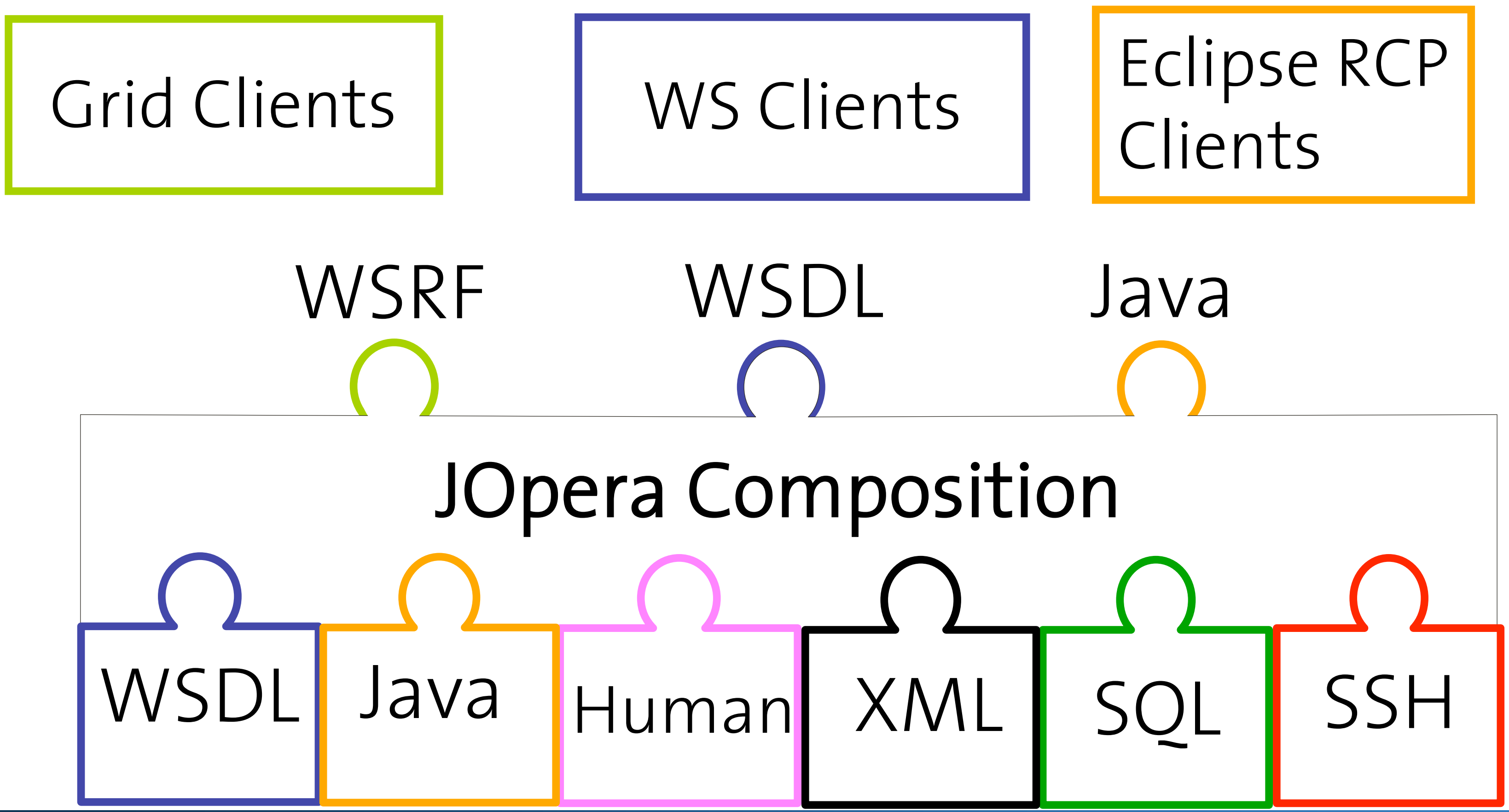
Dealing with heterogeneity in JOpera

- The JOpera composition language does not have to be changed when adding a new kind of service



Publishing a composition with JOpera

- JOpera processes are automatically published to clients using a variety of access protocols

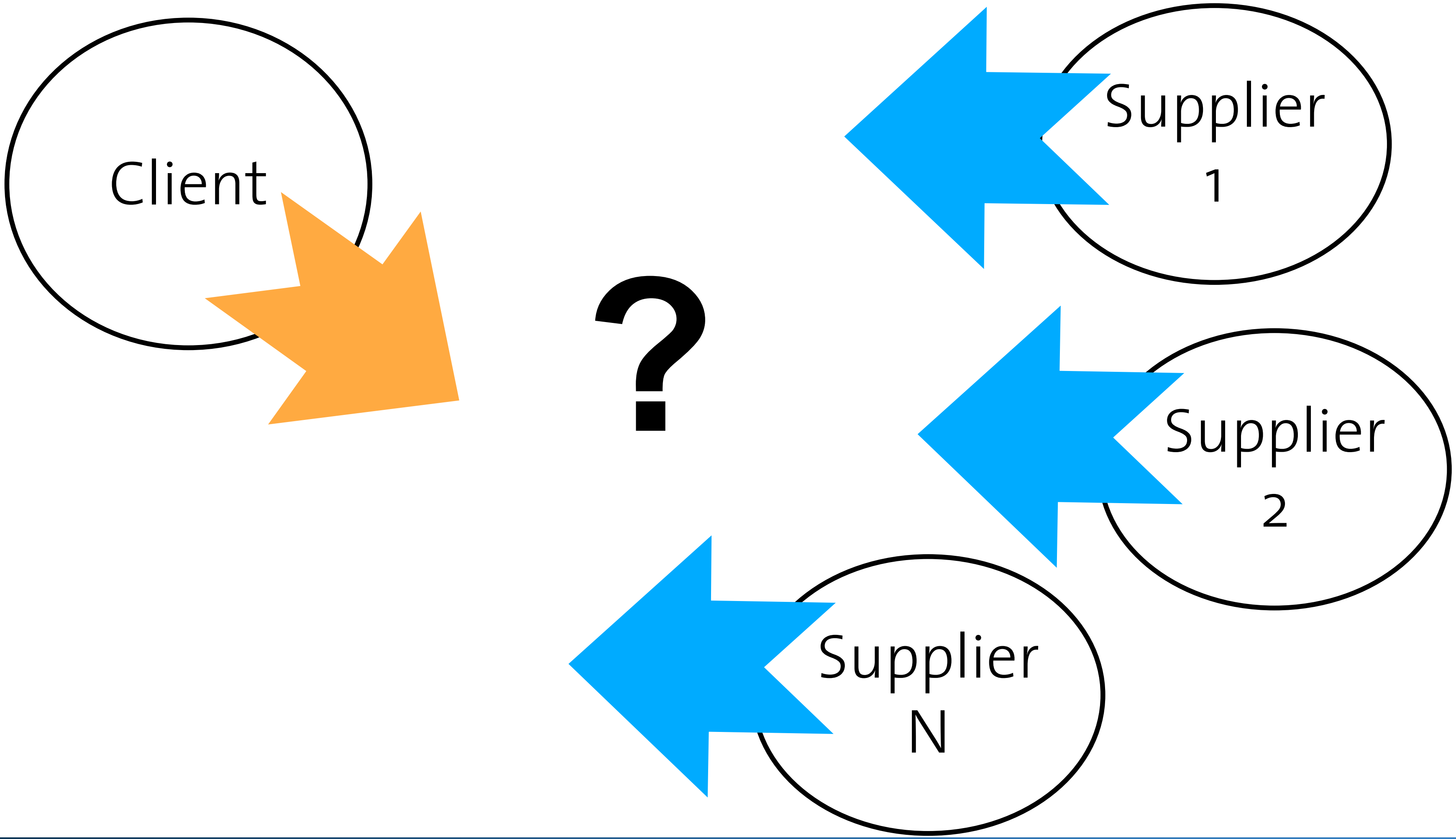


[eScience2005, CCGrid2006]

Executing Service Composition Models with JOpera for Eclipse

Demo 2

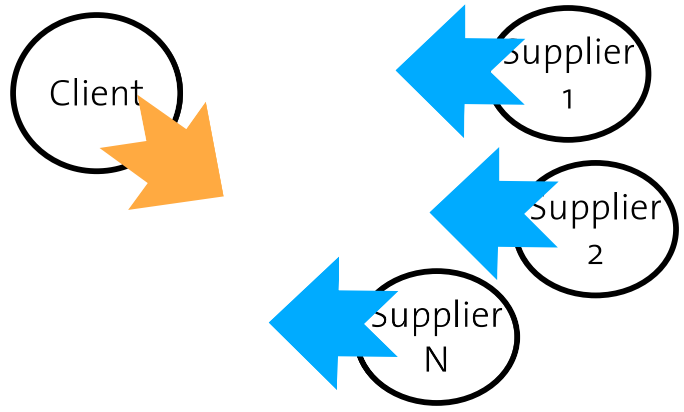
- A variable number of suppliers bid for a client



Demo 2

- A variable number of suppliers bid for a client

1. Clients sends a request for proposal
2. Broker forwards it to matching suppliers
3. Broker gathers bids
4. Broker calls back client with all bids
5. Client chooses
6. Broker notifies suppliers
7. Client gets confirmation

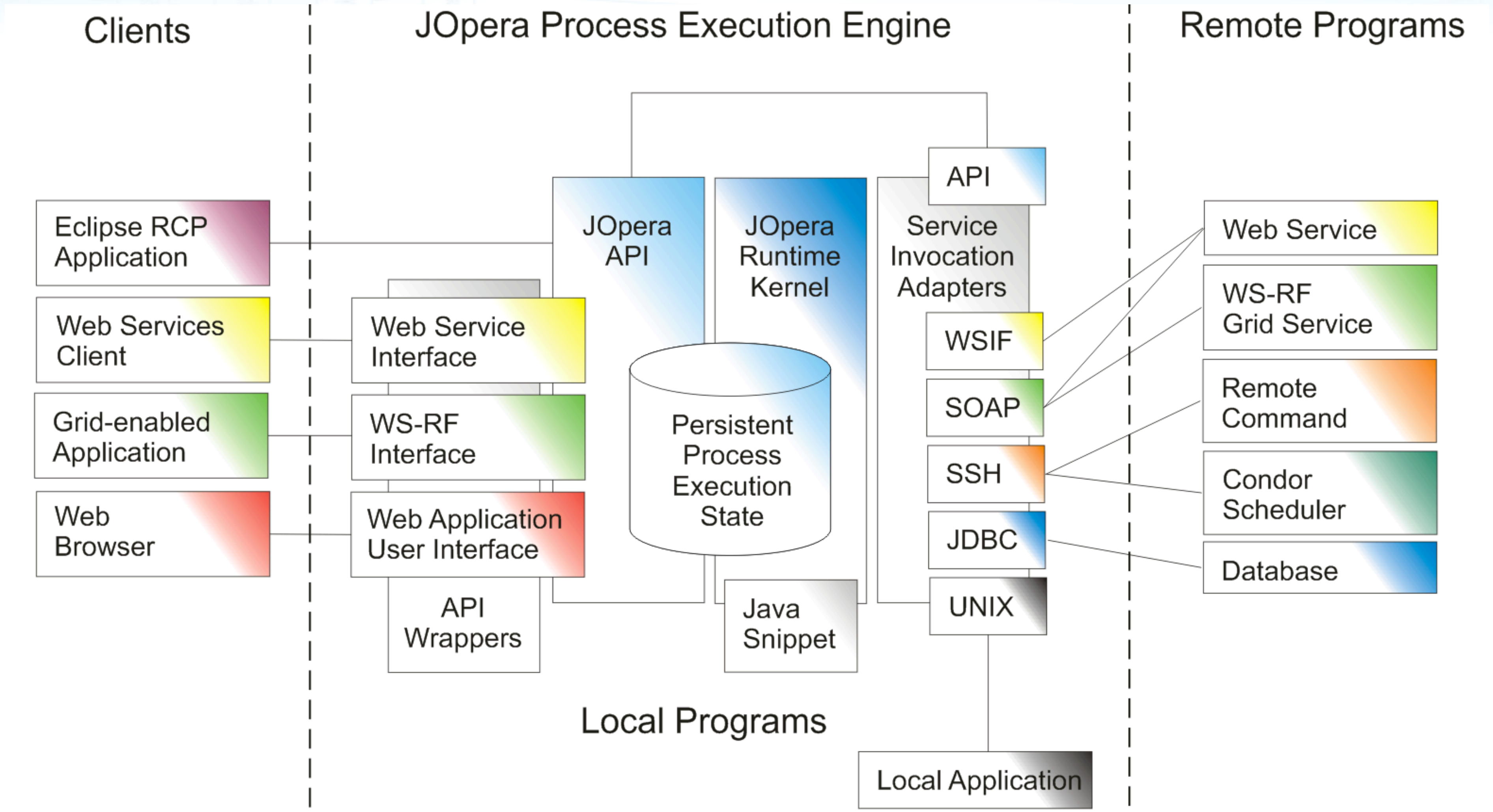


More Demos on Workflow Patterns

- Available when you download JOpera
- (*New, Examples, JOpera Examples, patterns.oml*)
- Quicktime movies also available online:

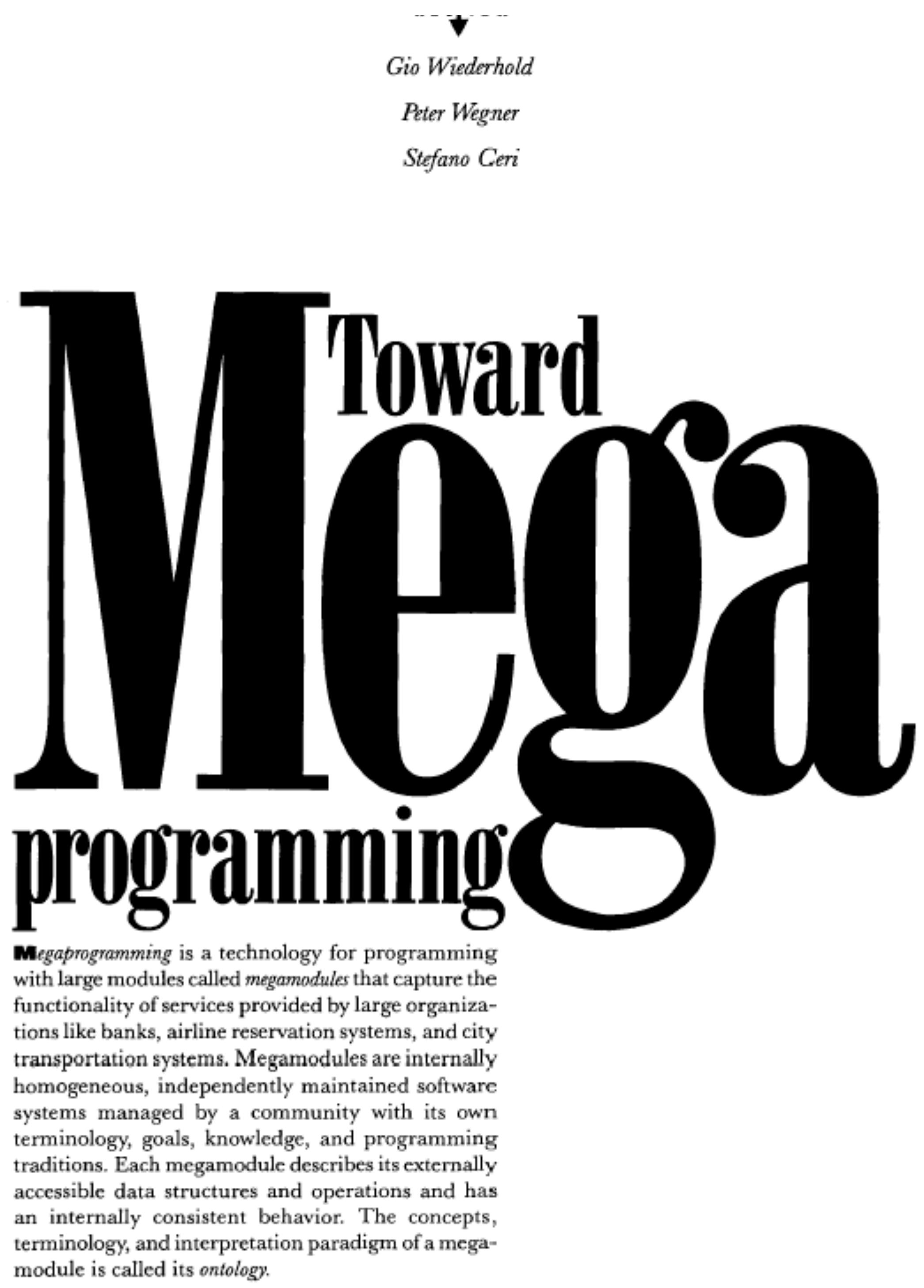
  www.jopera.ethz.ch/docs/patterns

Architecture of JOpera for Eclipse



Conclusion

- **Modeling** service composition behavior
  - Flow-based **composition language** (Visual & XML)
  - Development and Debugging tools for Eclipse
  - Composition not limited to Web services
- **Execution** of the composition models
  - Efficiency (compiled to Java bytecode)
  - Distributed engine (on a cluster of computers)
  - Autonomic platform (self-healing, self-tuning)
  - Extensibility (Eclipse plug-ins to provide custom service publishing and invocation adapters)



Gio Wiederhold

Peter Wegner

Stefano Ceri

Toward Megaprogramming

Megaprogramming is a technology for programming with large modules called *megamodules* that capture the functionality of services provided by large organizations like banks, airline reservation systems, and city transportation systems. Megamodules are internally homogeneous, independently maintained software systems managed by a community with its own terminology, goals, knowledge, and programming traditions. Each megamodule describes its externally accessible data structures and operations and has an internally consistent behavior. The concepts, terminology, and interpretation paradigm of a megamodule is called its *ontology*.

Process Support for Web Services

Gio Wiederhold
Peter Wegner
Stefano Ceri

Communications of the ACM
Volume 35
Issue 11
November 1992
Pages: 89 - 99

References on the language

[WORKS06] Cesare Pautasso, Gustavo Alonso, Parallel Computing Patterns for Grid workflows, In: Proc. of the Workshop in Support for Large Scale Science at HPDC2006, Paris, France, July 2006.

[VL/HCC2005] Cesare Pautasso, **JOpera: an Agile Environment for Web Service Composition with Visual Unit Testing and Refactoring**, In Proceedings of the 2005 IEEE Symposium on Visual Languages and Human Centric Computing (VL/HCC'05), Dallas, TX, September 2005.

[JVLC2005] Cesare Pautasso, Gustavo Alonso **The JOpera Visual Composition Language** Journal of Visual Languages and Computing (JVLC), 16(1-2):119-152, 2005

[VLDB/TES2004] Cesare Pautasso, Gustavo Alonso: **From Web Service Composition to Megaprogramming** In: Proceedings of the 5th VLDB Workshop on Technologies for E-Services (TES-04), Toronto, Canada, August 29-30, 2004.

[HCC2003] Cesare Pautasso, Gustavo Alonso: **Visual Composition of Web Services** In: Proc of the 2003 Symposia on Human Centric Computing Languages and Environments (HCC 2003), Auckland, New Zealand, Oct 2003.

References on the system

[CCGrid2006] Thomas Heinis, Cesare Pautasso, Gustavo Alonso, **Mirroring Resources or Mapping Requests: implementing WS-RF for Grid workflows**, accepted to the 6th IEEE International Symposium on Cluster Computing and the Grid (CCGrid2006), Singapore, May 2006.

[e-SCIENCE2005] Thomas Heinis, Cesare Pautasso, Oliver Deak, Gustavo Alonso, **Publishing Persistent Grid Computations as WS Resources**, accepted to the 1st IEEE International Conference on e-Science and Grid Computing (e-Science 2005), Melbourne, Australia, December 2005.

[ICWS2005] Cesare Pautasso, Thomas Heinis, Gustavo Alonso: **Autonomic Execution of Service Compositions**, In: Proc. of the 3rd International Conference on Web Services (ICWS 2005), Orlando, Florida, July 2005.

[ICAC2005] Thomas Heinis, Cesare Pautasso, Gustavo Alonso: **Design and Evaluation of an Autonomic Workflow Engine**, In: Proc of the 2nd International Conference on Autonomic Computing (ICAC-05), Seattle, Washington, June 2005.

[IJET'04] C. Pautasso, G. Alonso **JOpera: a Toolkit for Efficient Visual Composition of Web Services** International Journal of Electronic Commerce (IJEC), 9(2):107-141, Winter 2004/2005

Cesare Pautasso Thomas Heinis Bioern Bioernstad Andreas Bur Fabian Pichler Patrick Moor Adrian Listyo Patrick Jayet Sandra Brockmann Christoph Schwank
Dennis Rietmann Dominique Schneider Markus Egli Michael Lorenzi Christian Rupp Markus Haller Axel Wathne Antonio Caliano Oliver Deak
Reto Schaeppi Nicholas Born Philip Frey Claus Hagen Win Bausch Gustavo Alonso

Thank you for your feedback:

www.jopera.org

Free Download