

**ETH**

Eidgenössische Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zurich



# Challenges and Opportunities for formal specifications in Service Oriented Architectures

**Gustavo Alonso**

Systems Group

Department of Computer Science

Swiss Federal Institute of Technology (ETH Zurich)





**Enactment Engines**  
Business Processes

**Transactional Interaction**

**Enterprise Service Bus**

Process Re-engineering

**BPMN**

**BPFE**

Workflow

Message Queues



# Overview

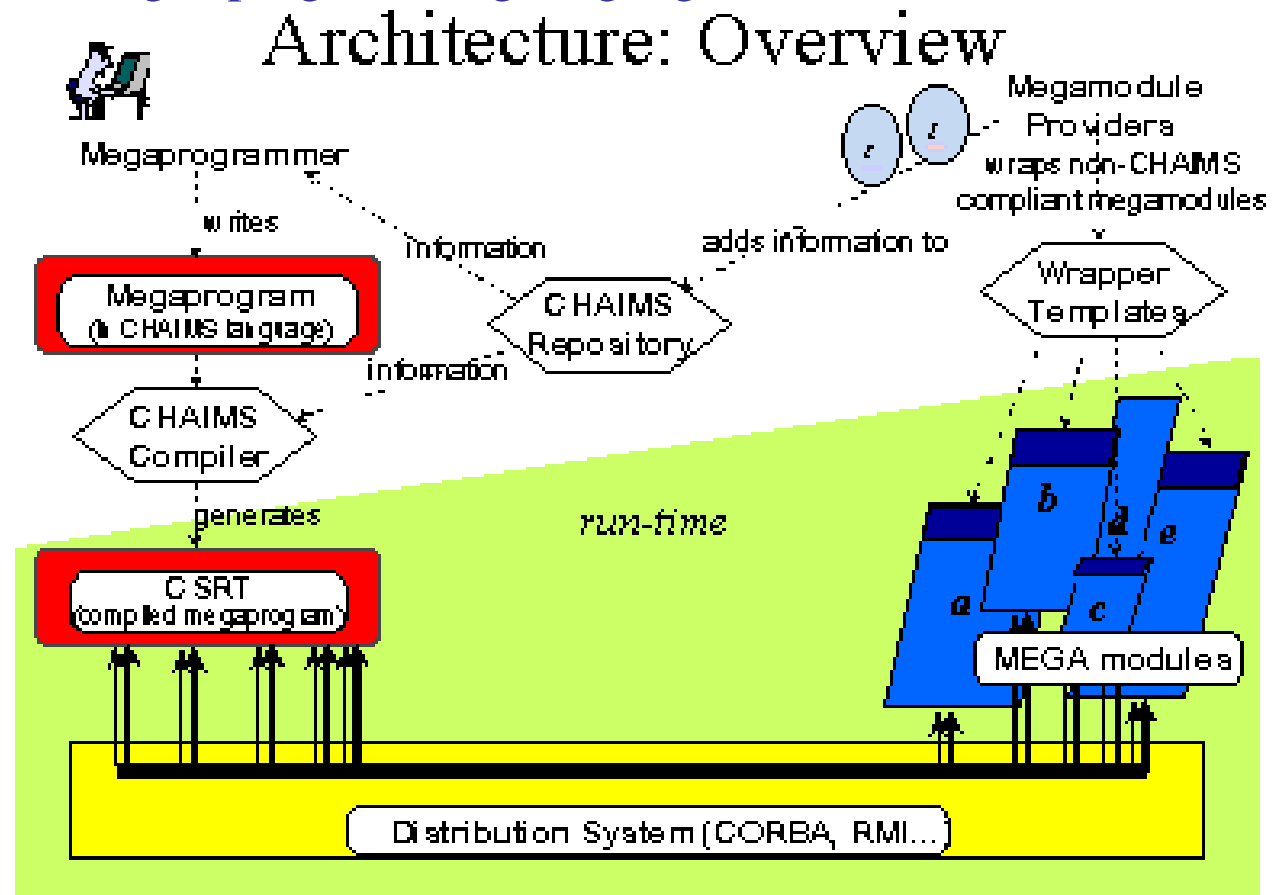
- Workflows and business processes
- Service Oriented Architectures
- What to do next?

# Workflows and business processes

# Processes

**Megaprogramming** (Large-scale Interoperation and Composition): a move from coding as the focus of programming to a focus on composition based on a very-high level (mega-)programming language for software module composition.

(Gio Wiederhold)

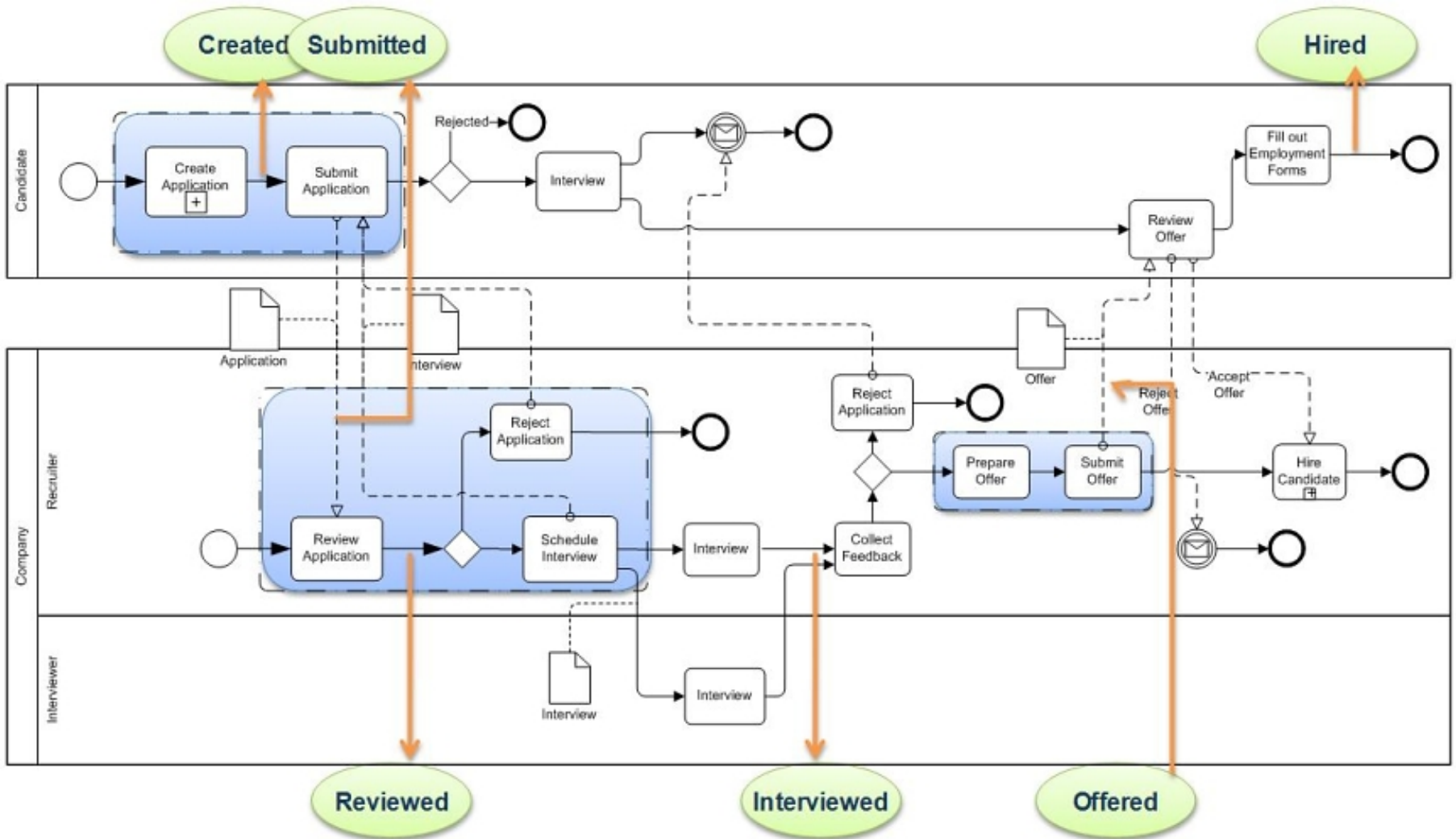


## A quote

**BPEL is geared towards *programming in the large*, which supports the logic of business processes. These business processes are self-contained applications that use Web services as activities that implement business functions. BPEL does not try to be a general-purpose programming language. Instead, it is assumed that BPEL will be combined with other languages which are used to implement business functions (*programming in the small*)**

**BPELJ: BPEL for Java technology (IBM, BEA)**

# Business processes



# Formal models come in ...

**Add formalism and a basic model behind the processes:**

- Petri nets
- Pi calculus
- Transactional logic

**And the result is a very attractive programming platform:**

- High level, highly intuitive, graphical
- Independent of the underlying execution and systems
- Suitable to formal treatment
- Composable, reusable, etc.

**Hiding all the nasty details of real systems:**

- Communication, distribution, exceptions, data management, ...



# Questions to the audience

Is it “mega-programming” or “mega-specification”?

Is it BPEL or BPMN?

It cannot be both – in practice it is neither



# A great idea – A bad implementation

The workflow paradigm failed in the 90's because:

- It did not solve the real problem => system integration
- The high level abstraction had a very high price tag => integration
- The technology was simply not there

# The problem with formalism

- One size does not fit all
  - Human activities vs. automated workflows
    - Not at all the same: Different engines, different constraints
    - Humans cannot be modeled and are not deterministic
    - Either it can be formalized (and the humans removed) or it can't be formalized and formal treatment is irrelevant
  - Top-down design or bottom-up design
    - Bottom-up does not work formally (a random composition of Mealy machines do not result in a regular language)
    - Top-down is impossible in practice
  - Business versus IT, representation vs implementation
    - A process is not centralized
    - Composition of services, each with its own automaton

# Things are better now

## Business Processes are back on the front page:

- several standardization efforts
- a flood of new tools
- great interest in the topic
- lots of new expectations

## Technically, it has become easier:

- Web services remove the need for wrappers
- XML standardizes data passing and opens the black box
- Technology has moved ahead (bandwidth, cluster computing, CPU power, storage space, middleware architectures ...)
- Convergence towards a few platforms (Java, .NET) significantly simplifies the target without having to work on a closed system

# Workflow is not the point

**These developments are not related to business processes**

**Problems solved for us not by us:**

- Cost of wrappers = web services
- No way of dealing with data heterogeneity = XML
- Human activities = emphasis today is on automated processes
- Implementation = container technology provides all you need

**Process specifications still confusing and confused**

- BPELJ
- BPEL 4 People
- BPEL vs BPMN
- UML vs. the rest of the world

# Service Oriented Architectures

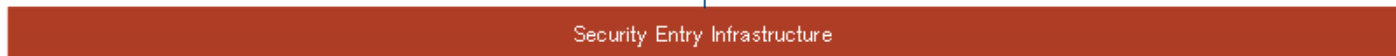




# Why Service Orientation?

**There is no problem in system design that cannot be solved by adding a level of indirection. There is no performance problem that cannot be solved by removing a level of indirection.**

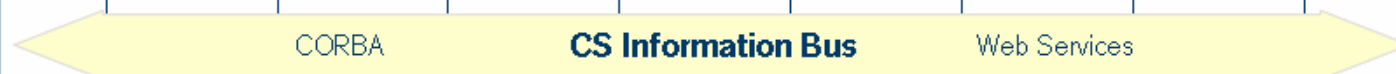
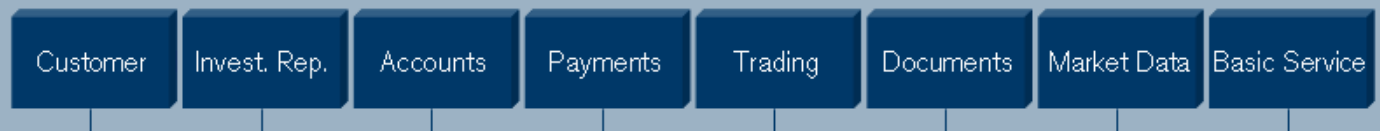
# Reality strikes ...



## Portal Platform

Reference	Currency	Amount	Interest Rate	Start Date	Maturity Date	Account Interest	Value Incl. acc. int. in EUR
LD23445234534	EUR	500,000.00	8.00 %	7.04.04	7.02.05	-34,000.00	500,000.00
LD12345323234	USD	1,000,000.00	8.00 %	7.04.04	7.02.05	-79,872.22	937,191.00

## Multi Channel Platform



**Service Oriented Architecture**  
 Governance, Architecture, Platforms,  
 Building Blocks, Cost, Initiatives

**Platform Security**  
 Design and Development Platform  
 IT Operations and Platform Management  
 Hardware / System Software

# Services: integration not programming

The key issue in enterprise computing today is integration:

- Enterprise Application Integration
- Transactional and Message Oriented Middleware
- Enterprise Software Bus
- Data integration
- Enterprise-wide Architectures

**Programming is to integration what  
laying bricks is to architectural design**



# What SOA is about

## Conventional software engineering cannot address:

- Non-functional properties: performance, reliability, security, ...
- Large scale composition
- Document and message based interaction
- Distributed systems architecture
- Run time properties and contracts
- Protocols, data formats

**SOA and web services bring to the fore the dependencies of existing programming languages on (single CPU) hardware concepts**



# Two contemporary examples

## XML

- Programming language variables
- Semi-structured Documents
- Procedural interfaces
- Document based interfaces
- Behavioral interfaces
- Variable assignment
- Declarative queries

**Impedance mismatch**

## Asynchronous Data Streams

- Procedural control flow
- Event based control flow
- Language based distribution
- Platform based distribution
- Behavioral interfaces
- Sequential programs
- Highly parallel and concurrent

**Model mismatch**

# Two sides of the coin

PROCESSES

A diagram consisting of a yellow oval with a black outline, centered on a horizontal dashed line. The word "INTEGRATION" is written in bold black capital letters inside the oval.

INTEGRATION

OO LANGUAGES

# Two sides of the coin

PROCESSES



INTEGRATION

HIC SUNT DRACONES

# Proof by enumeration:



## Workflow evolves ...

- It is constantly repositioned as the high level programming abstraction for whatever model is in fashion at the moment:
  - 80's Paperless office = networking (e-mail)
  - 90's Workflow engines = middleware platforms
  - 00's Business Processes = service oriented architectures

**Workflow and business processes do not contribute to the evolution of integration abstractions, they ignore them except to make their case**



# Two sides of the coin

HERE THERE BE DRAGONS



OO LANGUAGES

**The post-modern programmer: there is no perfect OO language, there is no universal solution to all software development problems**

Steve Cook: *Object Technology – A Grand Narrative?*  
ECOOP'06

Initiatives already in place:  
XL, XQueryP, ...

They are the mirror image of business processes

# Ample literature on the topic



# What to do next?



# The beautiful garden = Semantic Web

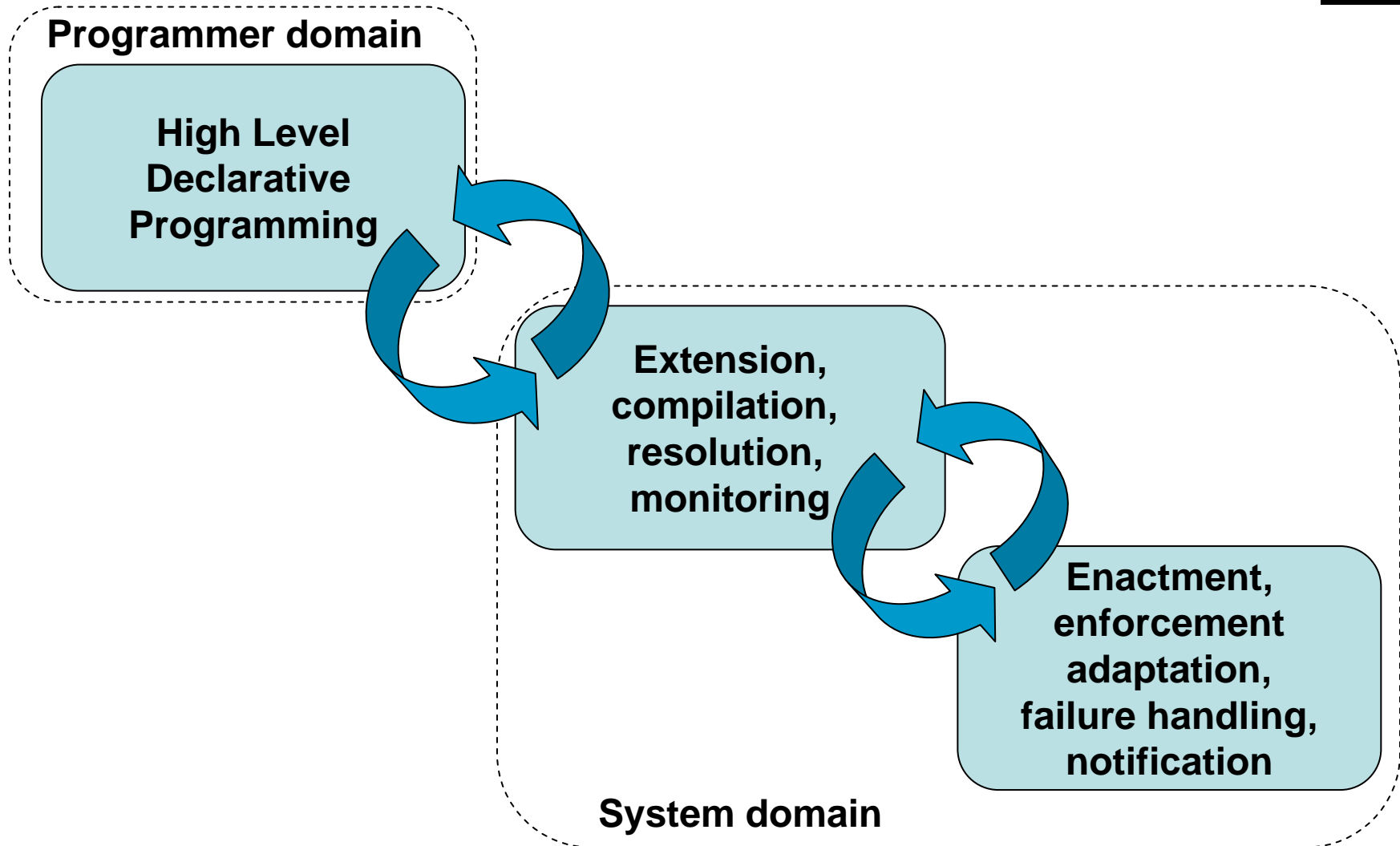
## Fully transparent integration at the semantic level

- Semantic matchmaking
- Automatic composition through pre- and post- conditions
- Theorem proven driven QoS
- Automatic, world wide service search

## This will not work any time soon:

- It is not a real problem
- Ignores that the service matching is not interface based
- “AI mistakes” revisited = it is not the logic but the underlying data
- Ignores the power of vertical standards in this area

# A Dream: Automatic Software Generation





# The ugly city = System Integration

**SOA is not an abstract idea, it is a response to**

- Very well defined needs in enterprise computing
- Changes in technology (cluster, networking, distribution)
- The Internet
- Business-to-Business integration
- Enterprise Computing coming of age
- Large scale distributed information systems

**To understand SOA, the key is to understand the underlying problems, not just the technology involved in solving them**

# Automatic *Plumbing* Generation

**The promise of SOA is to facilitate integration by**

- Letting the system automatically decide on integration decisions
  - protocols to use
  - intermediate processing needed
  - routing and distribution
  - data transformations
- Enforcing standards for defining and operating with services
- Enforcing the use of service interfaces

**Related to this, there are many additional opportunities:**

- Languages for orchestration and composition
- Reasoning about compositions
- Integration by non-experts (IBM's situational integration)

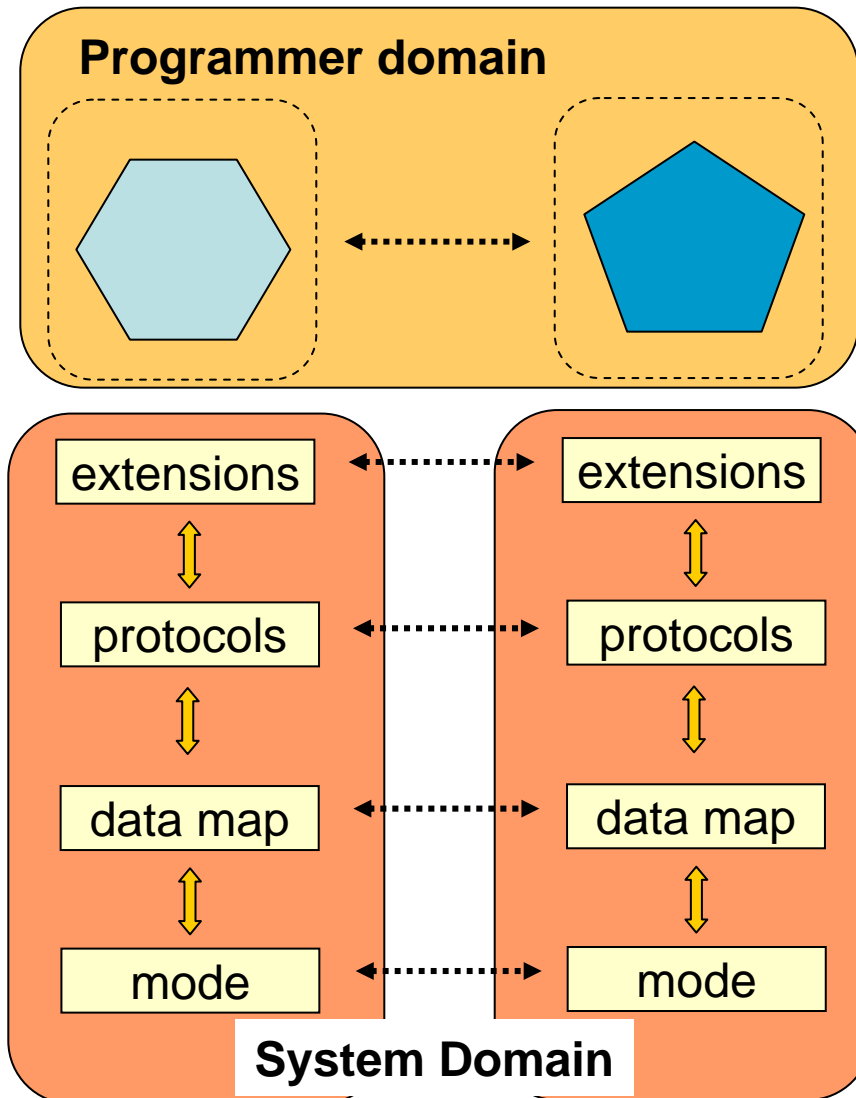
# Example of easier integration



## WS Invocation Framework

- Use WSDL to describe a service
- Use WSIF to let the system decide what to do when the service is invoked:
  - If the call is to a local EJB then do nothing
  - If the call is to a remote EJB then use RMI
  - If the call is to a queue then use JMS
  - If the call is to a remote Web service then use SOAP and XML
- There is a single interface description, the system decides on the binding
- This type of functionality is at the core of the notion of Service Oriented Architecture

# The new challenge



Can we provide a platform for integration that:

- covers the whole stack
- amenable to formal treatment
- considers the services
- considers the data flow



# Services = run time Software Engineering

A Service contract involves the interface, the Service Level agreement and QoS

Contracts are key to be able to develop, debug, optimize and maintain systems developed as a combination of services

**Service contracts are not the static, compile time pre- and post-conditions of conventional programming languages**

**They are an additional software layer in charge of the dynamic aspects of using services**

**The challenge is to make them part of the language**

# Problems are deeply interrelated

The language needs to describe a complete system at all levels

The language needs to be complete (no WS to add two numbers)

The language needs to include data management

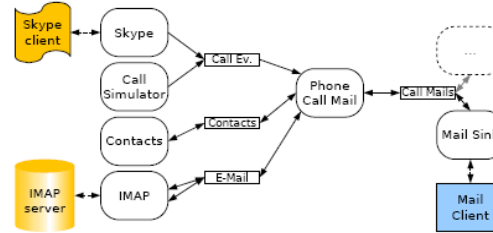
The language needs to reason about composite, multi-layer systems

The language has to be amenable to optimization

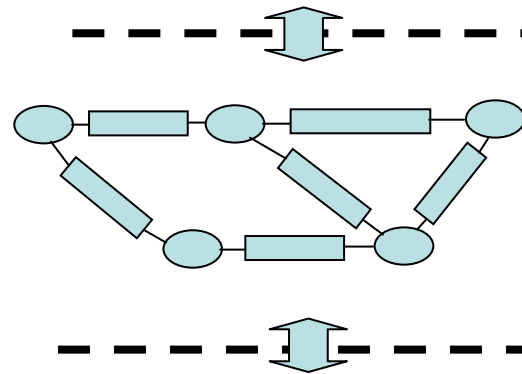
The language should be as declarative as possible

## This has to be our goal

# An example



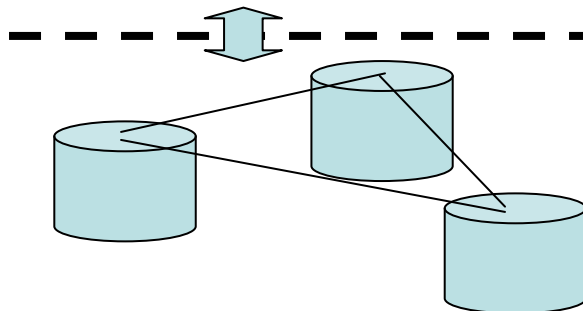
Workflow  
(programming language)



Mesh data stream network  
multi-query optimization  
operator placement  
compiler optimization  
dynamic adaptation



Slet and channel  
data stream processing  
window operators



Distributed storage  
Relational Algebra  
SQL, XPath, XQuery

# Real use cases

