

From (Data-) Flow Based Programs to Web Service Composition using XSLT

Andrew L. Wendelborn
University of Adelaide

The PAGIS Grid Application Environment

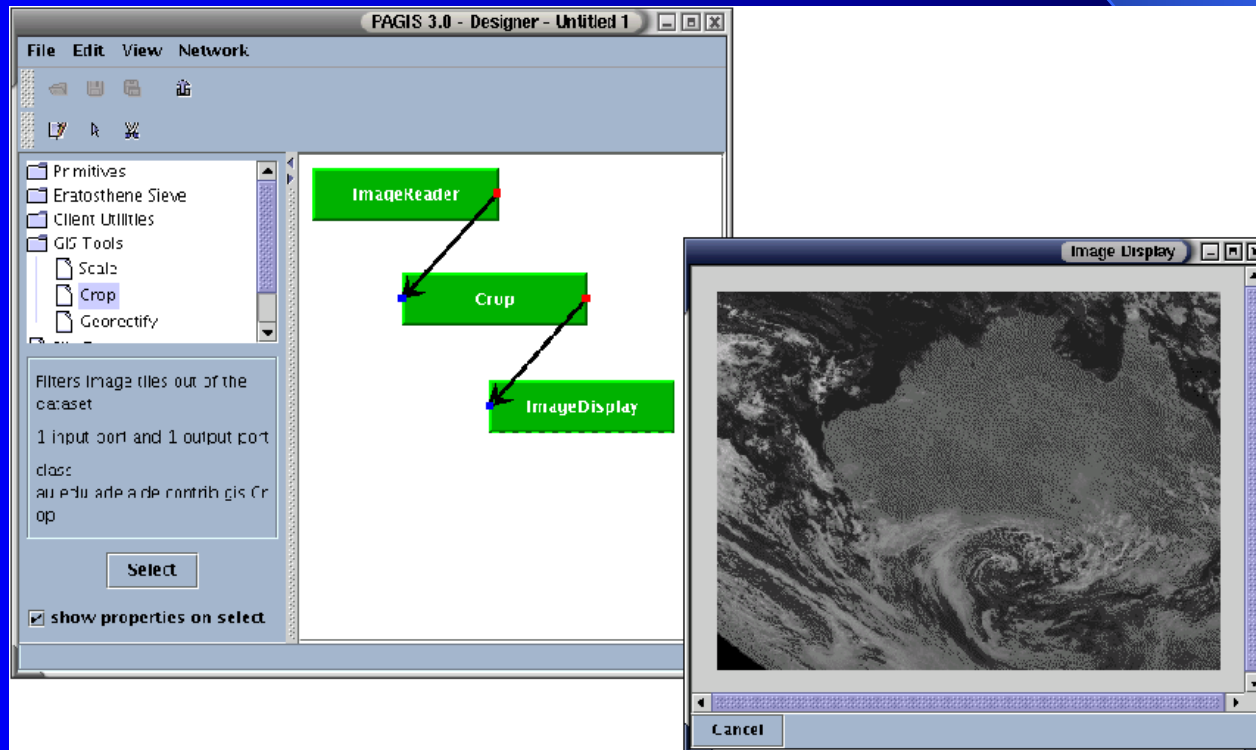
Darren Webb,
Andrew L. Wendelborn
University of Adelaide

Grid Programming Models

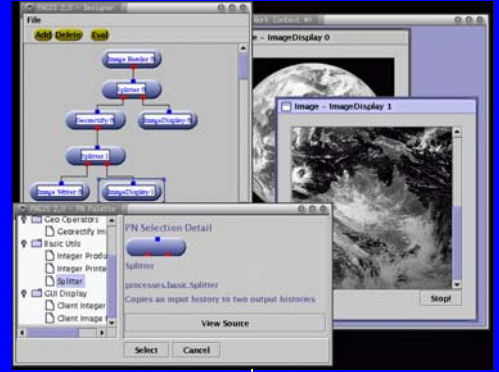
- Interested in *high level* models
- Focus on *adaptation* and *reconfiguration*
 - Requirements, resource quality and availability are dynamic: application must adapt
 - What can change?
 - The program itself;
 - Mapping to underlying environment.
 - How to guarantee reconfiguration *safety*?
 - A critical aspect

PAGIS

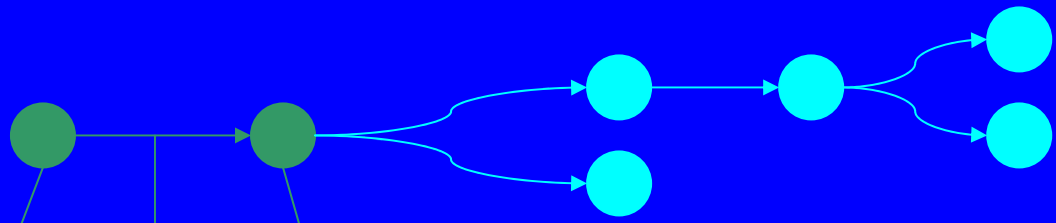
- Flow oriented programming model
 - Middleware to specify remote execution of computational tasks
 - Originally for GIS applications



Client

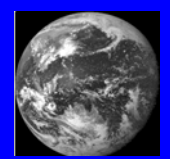


Scheduler

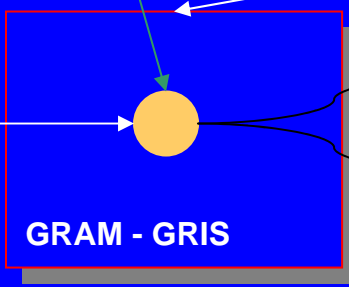


GIIS

DARC



GRAM - GRIS

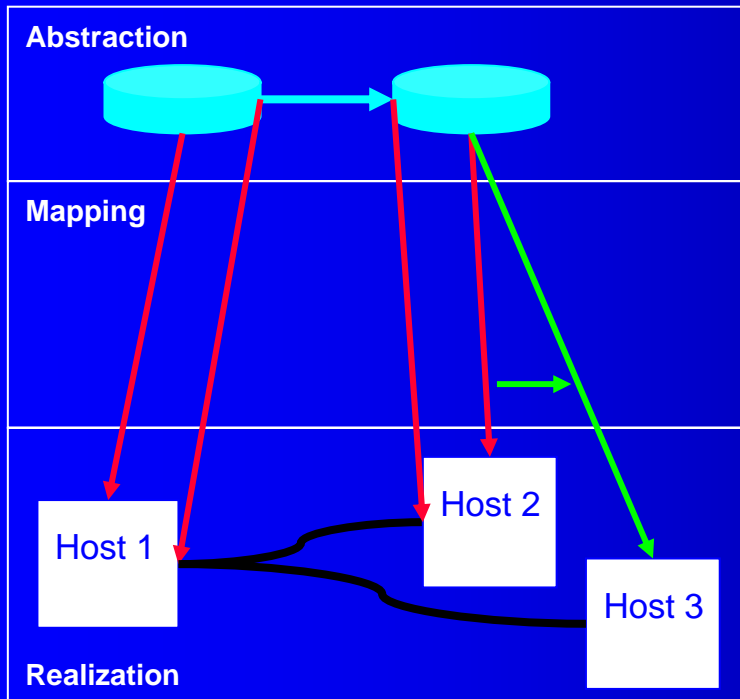


Evaluator

PAGIS

- Process networks as semantic foundation
 - Intuitive notation for Grid programming
 - Compositional
 - Implicitly parallel and distributed transparently
 - Determinate and reconfigurable
 - *Programmatic expression and semantics* of reconfiguration
- Intuitive user interface
- Simple Java API
- Reference implementation
 - Safe reconfiguration

The PAGIS Grid: Adaptation



We seek to model
safe application reconfiguration
triggered by changes in grid

The application
transparently evolves to match
resource availability and quality.

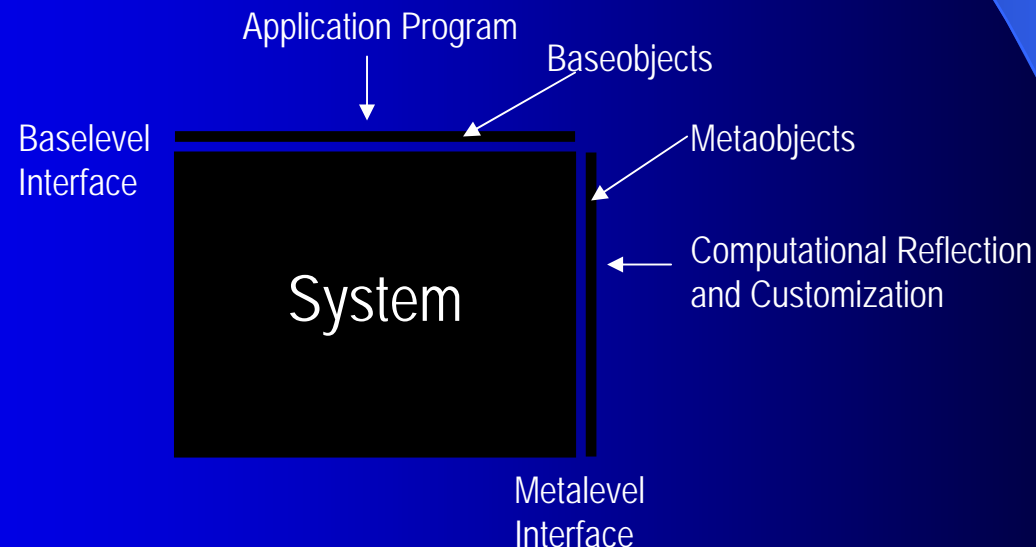
Reflective architecture

A Reflective Approach

- Separation of concerns
 - *Functional* concerns: baselevel program
 - *Non-functional* concerns:
 - metalevel program (meta-behaviours)
 - Reflects separation in process network model
- Reify *appropriate* aspects of baselevel
 - reflect upon *selected* aspects to customize critical aspects
- Software engineering for the grid

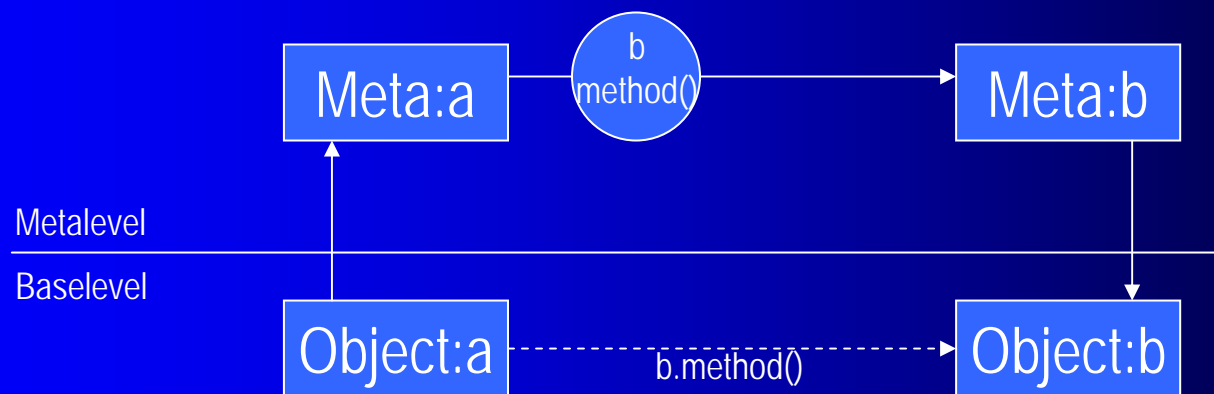
Reflective Interface

- Separate interfaces for app and customization concerns
- Metalevel exposes system elements to customization
- Design choices:
 - Which aspects to expose?



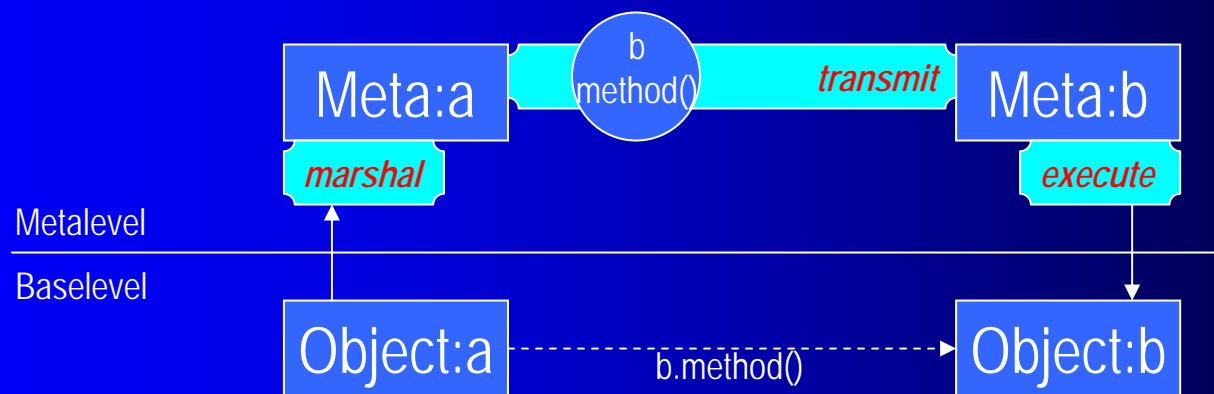
Enigma

- A *general* metalevel (reflective) interface for an object-oriented interpreter
 - Structural and behavioural reification
 - Reifies and exposes method invocation to customization
- Aids metalevel design by providing hooks for customization



Enigma Customization Phases

- Decomposes method invocation into phases
 - Marshal, Transmit, Execute
- Each phase is customized independently
 - Metabehaviour attached to relevant phase
- e.g. tracing metabehaviour



Enigma and Process Networks

- We use Enigma to develop grid applications
 - define *metabehaviours* for a given process network
- How to do it?
 - Poor decisions can lead to cumbersome metalevel
- Need to closely mirror the language model
 - How to expose reconfiguration operations?
 - How to represent a PN sub-computation?
 - For example ...

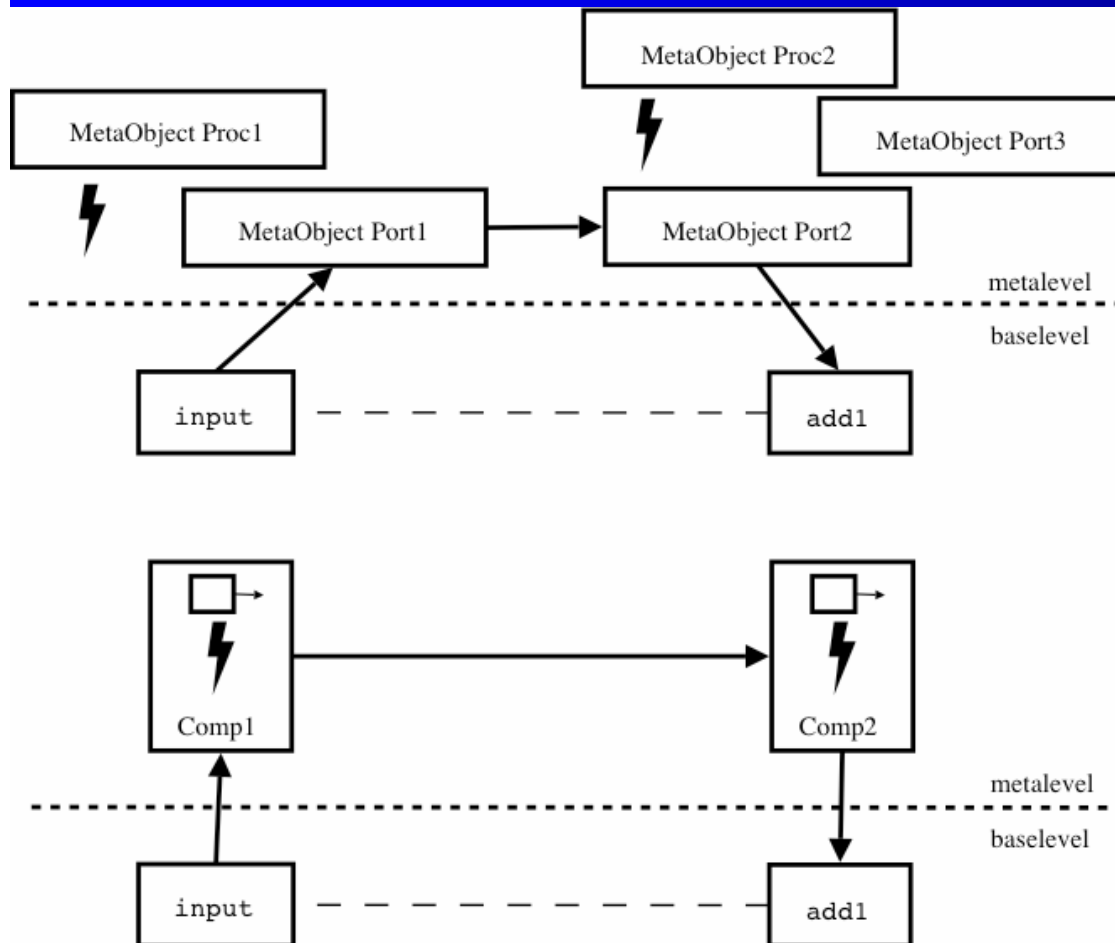
Metalevel design choices

- Metalevel objects ...

- Correspond to baselevel objects? ...

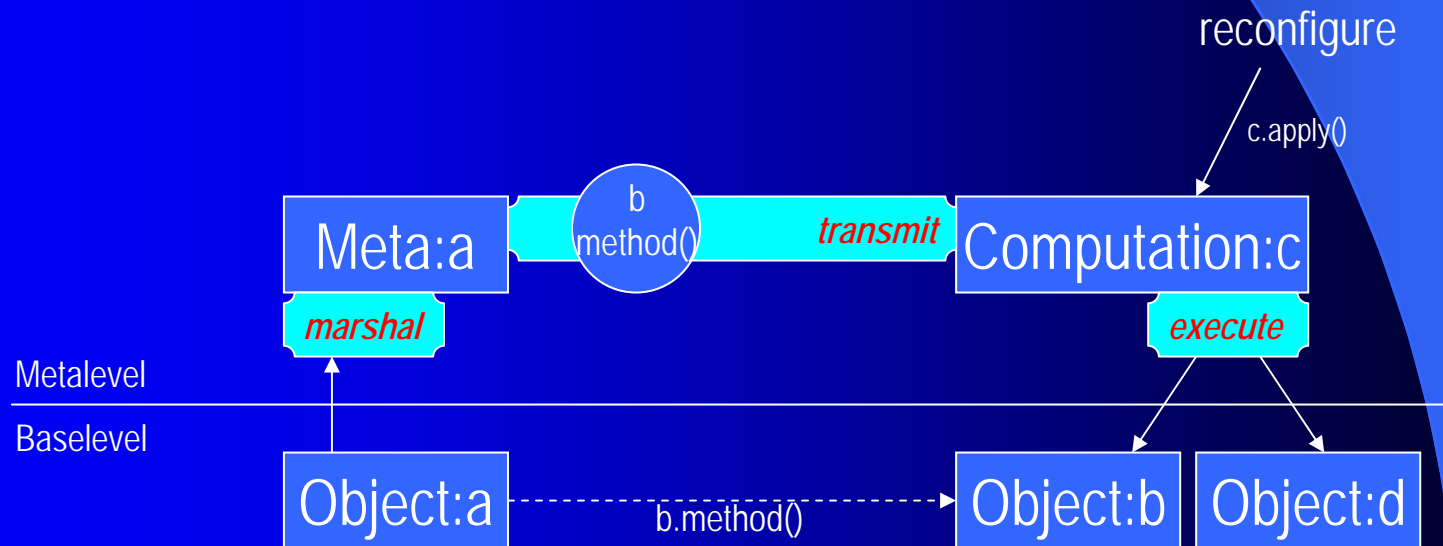
- Or to language (PN) level structures?

- Choose the latter



Process Network Metalevel

- *Computation* metaobject to represent baselevel process and port elements
 - They aggregate sub-computation elements
- Customize computation like any other metaobject
- Expose operations for adaptive programming
 - e.g *reconfiguration* specified by a “recipe” written to a Reconfiguration API and initiated by *apply()*



Example Metabeaviour

- Compressing method parameters
 - Compress large parameters prior to transmission
 - Implemented as marshal phase metabeaviour
 - Reified method invocation encapsulated in a new, compressed representation
 - Parameter values decompressed as needed
 - Attached to Computation metaobject to affect all sub-computation elements
- Similar approach to compress method results

Example Metabeaviour

- The process applies to many other Grid-related metabeaviours
 - Evaluation mode: eager / lazy / hybrid
 - Communication Integrity/Confidentiality
 - Data/Process Migration
 - Software Adaptation

Summary

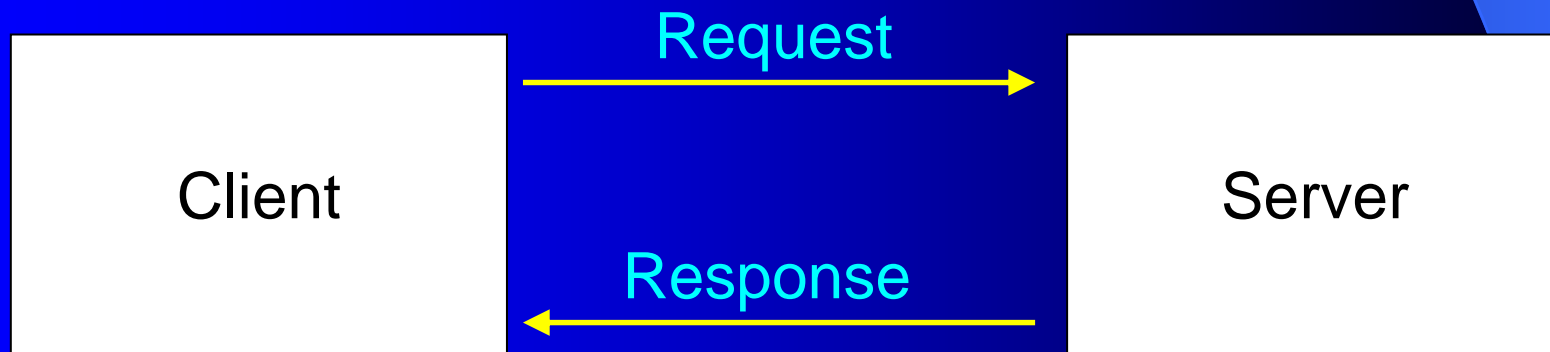
- PAGIS is a high-level programming environment, useful for deploying existing systems on the Grid, for building new Grid applications, and for supporting adaptive software
- Implemented with Globus plugin; demonstrated Berlin 2002
- Lessons
 - Insight into reconfiguration and metalevel design
- Improvements
 - Stronger component view
 - But need supporting semantics
 - ASP calculus; Fractal; formal analysis
 - ProActive plugin ??

Parallel composition of web services using XSLT

Peter Kelly, Andrew Wendelborn,
Paul Coddington
University of Adelaide

Web services

- Service Oriented Architecture (SOA) – a rapidly emerging model of grid computing
- Each machine on the network provides a set of services that can be accessed by clients
- Standards based – XML, SOAP, WSDL etc. used to provide interoperability between different platforms



Web service composition

- A distributed programming model
- A composition program consists of functionality implemented by several different services
- Composition program acts as a client to other services, and can also (optionally) be exposed as a service itself
- Can be done in any programming language; some are more suited than others

Existing approaches

- Object-oriented, compiled languages, e.g. Java and C++ - heavyweight and complex to develop services with, and lack network transparency
 - Data semantics
- Scripting languages, e.g. Perl and PHP – simple and easy to use, but lack static typing and have poor support for concurrency
- Web service composition/workflow languages, e.g. BPEL – programming model often restrictive, low level

Desirable features for a Grid programming language

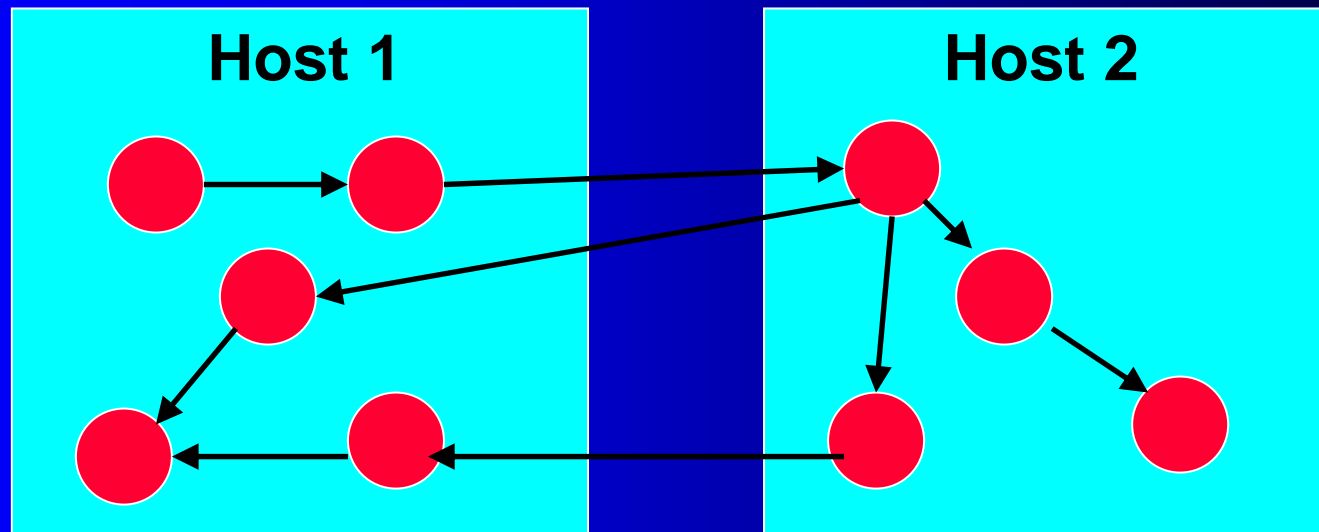
- Automatic parallelization
- Network transparency - hide details of distribution and communication from programmer
- Service exposure/consumption should be built-in to the language interpreter, not an add on
- Execution environment should handle details of distribution and scheduling: more declarative
- We believe XSLT is suited to this model
 - Above aspects not included in language, but
 - We believe support is straightforward

XSLT for web services development

- Extensible Stylesheet Language Transformations
- Designed for transforming XML data
- Uses XML Schema type system – matches that used by web service interfaces
- Functional language – ideal for automatic parallelization
- Well established language (W3C standard)
- Current implementations do not support web services or parallelism; this is a new area for the language

Execution model

- XSLT program is compiled into dataflow graph
- Graph is partitioned into sections
 - each section is run on a different machine
- Nodes in the graph can correspond either to local operations or web service calls

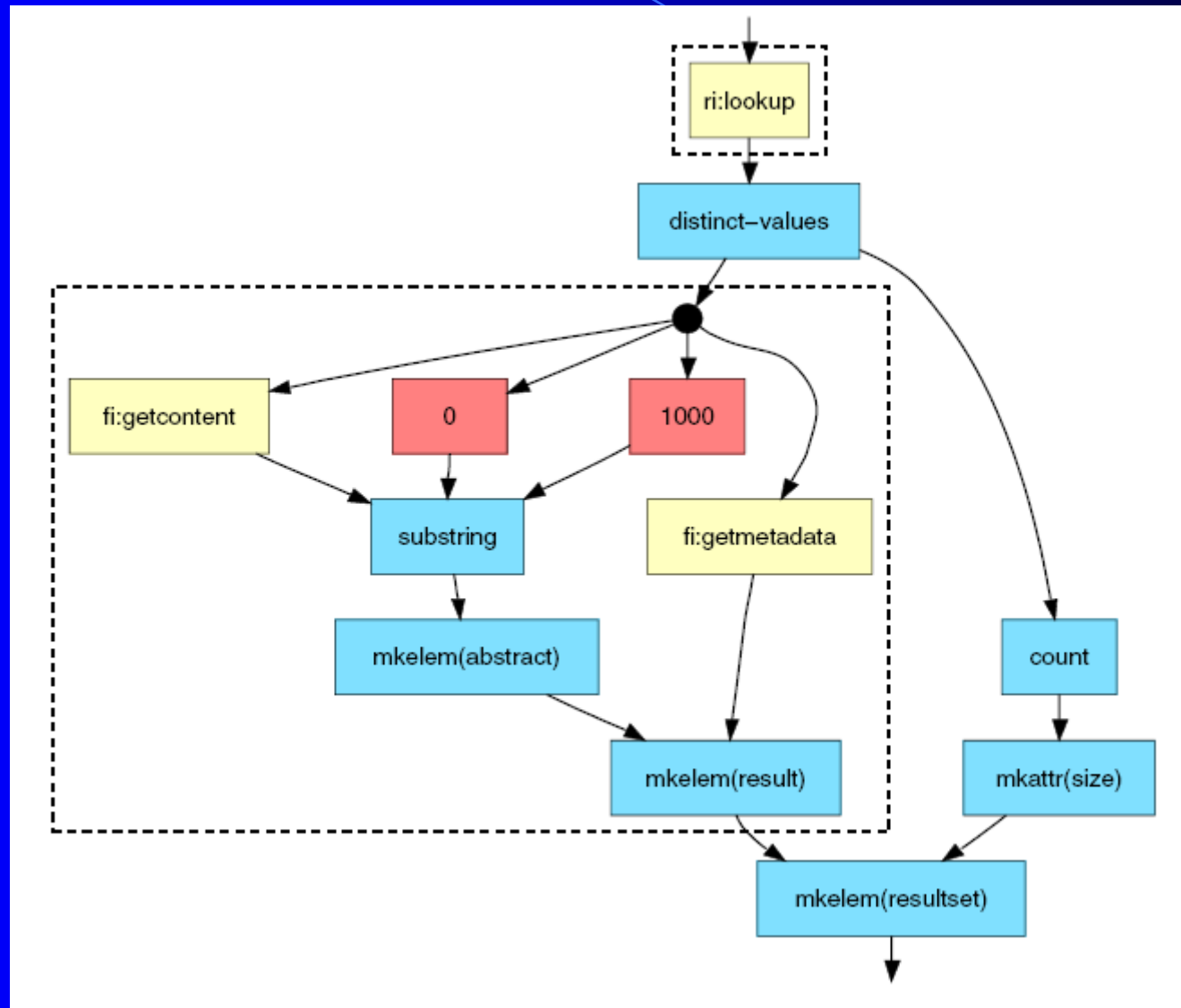


Distributed execution

Two types of distribution:

1. Multiple instances of the execution engine on different machines
 - Each handles some portion of the graph (potentially multiple nodes)
 - Cooperative scheduling between execution engines determines load balancing
2. Web services as dataflow nodes
 - Remote hosts do not require execution engine to be installed; they just provide a web service with a defined interface
 - Uses standard protocols (SOAP/HTTP messaging)

Search engine – dataflow graph



Exposing a program as a web service

- Designed to be as easy to use as Perl/PHP
- Many web service development environments are complex
- In our implementation, you just place the file on a web server and it is immediately available
- WSDL definitions for a service are automatically generated on demand when requested by a client
- Web service compositions written in XSLT
 - act as clients to other services
 - can be accessed as a service themselves
- *Hierarchical* composition is possible
 - other services called may themselves be compositions

Accessing other web services

- Accessing other web services is made as convenient as a local function call
- To call a web service operation
 - programmer associates a *namespace prefix* with a WSDL file
 - then uses that prefix in function calls
- No need to generate proxy/serialization classes – all WSDL parsing and parameter marshalling is handled internally within the interpreter

```
namespace store
    "http://store.com/api?WSDL" ;

int $price = store:getPrice("shoes");
```

Conclusion

- Simpler environment for development of grid applications than many existing approaches
- Designed to integrate closely with existing systems through XML and web services standards
- Aims to provide an *easy to use* way of developing large-scale distributed, parallel programs for grid environments, especially in eScience context

For further information

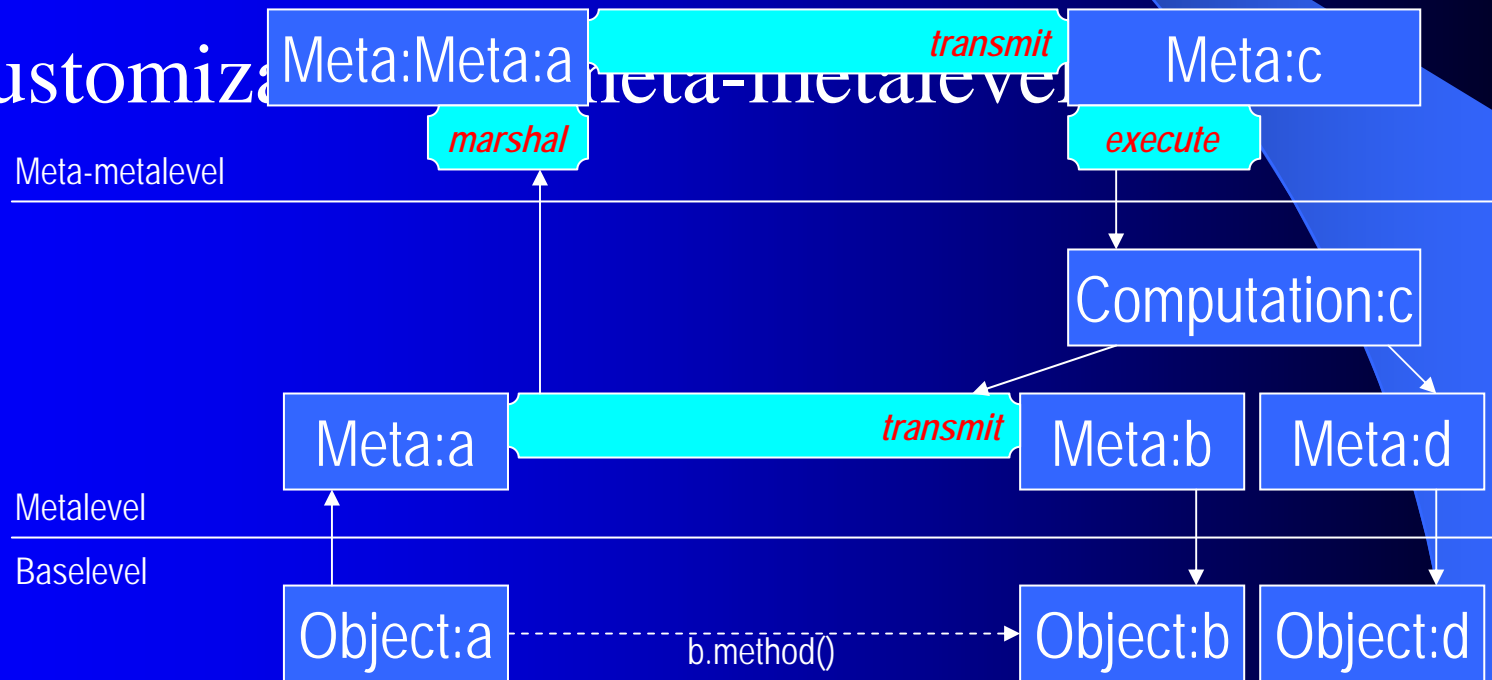
<http://gridxslt.sourceforge.net>

pmk@cs.adelaide.edu.au



Alternative Solution

- Introduce Computation metaobject by customizing a phase
- Customization



Dataflow computation

- Program represented as a directed graph
- Each node corresponds to an operation that takes a set of input values and produces a set of output values
- Edges connecting nodes represent the flow of data from one operation to another

