Scientific Workflows: Actor-Oriented Modeling & Design, Language Issues

• Different levels of SWF granularity:
  – “plumbing” level: connecting to remote resources, moving data around, launching local or remote applications, monitoring & restarting jobs, etc.
  – intermediate levels involving some database queries, data transformations, data analysis & visualization
  – design-level, conceptual diagrams

• What languages, programming metaphors, programming constructs, execution models are adequate for the different levels?

• How to combine different aspects, e.g. data-flow, control-flow, etc?

System Abstractions


Actor-Oriented Design & Dataflow

• Object orientation:

  ```
  class name
  data
  methods
  call
  return
  ```

  What flows through an object is sequential control (cf. CCA, MPI)

• Actor/Dataflow orientation:

  ```
  actor name
  data (state)
  ports
  input / output
  ```

  What flows through an object is a stream of data tokens (in SWFs/KEPLER also references!!)

Object-Oriented vs. Actor-Oriented Interfaces

Object Oriented

```java
public class TextToSpeech {
    public void initialize() {
        // code
    }
    public void notify() {
        // code
    }
    public boolean isReady() {
        // code
    }
    public double[] getSpeech() {
        // code
    }
}
```

Actor/Dataflow Oriented

```java
public void TextToSpeech(String text, String speech) {
    // code
}
```

OOP interface gives procedures that have to be invoked in an order not specified as part of the interface definition.

AO interface definition says “Give me text and I’ll give you speech”

Direction of flow is implied by I/O port type.

Source: Edward Lee et al., http://ptolemy.eecs.berkeley.edu
### Flow-based “Design Patterns”

- **Connection tokens “power line”**
  - JDBC connection tokens
  - SRB connection tokens
  - GSIDs, proxies, certificate tokens

  ![Flowchart of connection tokens](B. Ludaescher, ECS289F-W05, Topics in Scientific Data Management)

- **Exercise:** Design a WF that uses a “connection pool”

- **State changing pipeline:**

  ![State changing pipeline](B. Ludaescher, ECS289F-W05, Topics in Scientific Data Management)

### More Flow-based Design Patterns

- **Generality vs specialization of actors**
  - also loosely coupled vs tightly coupled

- **Data transformation pipelines**
  - alternate compute and data transformation steps

- **Stage-execute-fetch pattern (Grid-WFs)**

- **Loops, higher-order functions (map, foldr, …)**
  - cf. Taverna’s automatic loop insertion based on data types

### Why Ptolemy II (and thus KEPLER)?

- **Ptolemy II Objective:**
  - “The focus is on assembly of concurrent components. The key underlying principle in the project is the use of well-defined models of computation that govern the interaction between components. A major problem area being addressed is the use of heterogeneous mixtures of models of computation.”

- **Dataflow Process Networks w/ natural support for abstraction, pipelining (streaming) actor-orientation, actor and workflow reuse**
  - Component (“actor”) interaction semantics not hand-wired inside components, but “factored out” in a “director”
  - Component interaction semantics is not “emergent” (i.e., not an “accident” of the particular components working together)
  - Different directors for different modeling and execution needs (… can even be combined … to some extent)

- **User-Orientation**
  - Workflow design & exec console (Vergil GUI)
  - “Application/Glue-Ware”
  - excellent modeling and design support
  - run-time support, monitoring, …
  - not a middle-/underware (we use someone else’s, e.g. Globus, SRB, …)

- **Behavioral Polymorphism in Ptolemy**

  ![Behavioral Polymorphism](B. Ludaescher, ECS289F-W05, Topics in Scientific Data Management)

  These polymorphic methods implement the communication semantics of a domain in Ptolemy II. The receiver instance used in communication is **supplied by the director, not by the component** (cf. CCA, WS??, [G]BPL4??, … !)

  ![Source](Edward Lee et al. http://ptolemy.eecs.berkeley.edu/)

- **Pragmatics**
  - Ptolemy II is mature, continuously extended & improved, well-documented (500+pp), open source system, Ptolemy II folks actively participate in KEPLER

  ![Source](Edward Lee et al. http://ptolemy.eecs.berkeley.edu/)

---

*Note: This document appears to be a snapshot of a presentation or lecture, with various diagrams and text blocks. The content is technical and related to scientific data management and design patterns.*
Domains and Directors: Semantics for Component Interaction

- CI – Push/pull component interaction
- CSP – concurrent threads with rendezvous
- CT – continuous-time modeling
- DE – discrete-event systems
- DDE – distributed discrete events
- DT – discrete time (cycle driven)
- Giotto – synchronous periodic
- GR – 2-D and 3-D graphics
- PN – process networks
- SDF – synchronous dataflow
- SR – synchronous/reactive
- TM – timed multitasking

Source: Edward Lee et al. http://ptolemy.eecs.berkeley.edu/

Polymorphic Actor Components Working Across Data Types and Domains

- Actor Data Polymorphism:
  - Add numbers (int, float, double, Complex)
  - Add strings (concatenation)
  - Add complex types (arrays, records, matrices)
  - Add user-defined types

- Actor Behavioral Polymorphism:
  - In dataflow, add when all connected inputs have data
  - In a time-triggered model, add when the clock ticks
  - In discrete-event, add when any connected input has data, and add in zero time
  - In process networks, execute an infinite loop that performs rendezvous on input or output
  - In push/pull, ports are push or pull (declared or inferred) and behave accordingly
  - In real-time CORBA, priorities are associated with ports and a dispatcher determines when to add

Source: Edward Lee et al. http://ptolemy.eecs.berkeley.edu/

Component Composition & Interaction

Building Applications by Composition
- Connect uses Ports to Provides Ports

- Components linked via ports
- Dataflow (and msg/ctl-flow)
- Where is the component interaction semantics defined??
  - each component is its own director?
  - Might still be useful for special applications, e.g. parallel programs (MPI, ...)

Everything Flows! But what exactly?

- Dataflow
  - Data flows through operations (zoom into your CPU...)
  - Activity diagrams: data flows through actions
  - Process networks: data flows between processes

- Control-flow
  - Nodes are control-flow operations that start other operations on a state

- Mixed approaches
  - Statecharts: events trigger state transitions
  - Petri nets: tokens mark control and dataflow
  - Workflow languages: mix control and dataflow
  - ... many others ...

Source: GRIST/SC4DEVO workshop, July 2004, Caltech
A Closer Look at Dataflow ...
(or: Do you know what's going on under your carpet?)

- Dataflow: what you see is what you get (almost...)
- Need a general systematic way to handle references!

control tokens flow, e.g., from "$"-actor to FileReader and ImageReader actors
actual dataflow is "under the carpet" and through handles (file system, GridFTP, scp, SRB, ...)

• Dataflow: what you see is what you get (almost...)
• Need a general systematic way to handle references!

Dataflow Spectrum

- "clean" data/(ctl)-flow special tokens flow message passing, control flow
- Data (tokens) flow
  - (almost) no other side effects
  - WYSIWYG (usually)
- References flow
  - token reference type may be "http-get", "ftp-get", "hsi put"...
  - generic handling still possible
- Application specific tokens flow
  - e.g. current Nimrod job management in Resurgence
  - "invisible contract" between components
  - Director is unaware of what's going on ... (sounds familiar? ;-)
- Specific message passing protocols (e.g., CSP, MPI)
  - for systems of tightly coupled components

A Scientific Workflow Problem: More Solved (Computer Scientist's view)

- Solution based on declarative, functional dataflow process network
  (= also a data streaming model!)
- Higher-order constructs: \( \text{map}(f) \)
  \( \Rightarrow \) Rewritings & optimizations, e.g. \( \text{map}(f \circ g) = \text{map}(f) \circ \text{map}(g) \)
  \( \Rightarrow \) no control-flow spaghetti
  \( \Rightarrow \) data-intensive apps
  \( \Rightarrow \) free concurrent execution
  \( \Rightarrow \) free type checking
  \( \Rightarrow \) automatic support to go from \( \text{piw}(\text{GeneId}) \) to \( \text{PIW} := \text{map}(\piw) \overline{\text{over}} \{\text{GeneId}\} \)

- Solution based on declarative, functional dataflow process network
  (= also a data streaming model!)
- Higher-order constructs: \( \text{map}(f) \)
  \( \Rightarrow \) Rewritings & optimizations, e.g. \( \text{map}(f \circ g) = \text{map}(f) \circ \text{map}(g) \)
  \( \Rightarrow \) no control-flow spaghetti
  \( \Rightarrow \) data-intensive apps
  \( \Rightarrow \) free concurrent execution
  \( \Rightarrow \) free type checking
  \( \Rightarrow \) automatic support to go from \( \text{piw}(\text{GeneId}) \) to \( \text{PIW} := \text{map}(\piw) \overline{\text{over}} \{\text{GeneId}\} \)
A Scientific Workflow Problem:
Even More Solved (domain&CS coming together!)

Scientific “Workflows”: Some Findings

- Very different granularities: from high-level design to lowest level plumbing
- More dataflow than (business control-) workflow
  - DiscoveryNet, Kepler, SciRun, SciTegic, Triana, Taverna, …
- Need for “programming extensions”
  - Iterations over lists (foreach); filtering; functional composition; generic & higher-order operations (zip, map(f), …)
- Need for abstraction and nested workflows
- Need for data transformations (WS1 → DT → WS2)
- Need for rich user interaction & workflow steering:
  - Pause / revise / resume
  - Select & branch; e.g., web browser capability at specific steps as part of a coordinated SWF
- Need for high-throughput data transfers and CPU cycles: “(Data-)Grid-enabling”, “streaming”
- Need for persistence of intermediate products and provenance

Scientific “Workflows” vs Business Workflows

- Business Workflows (BPEL4WS* …)
  - Task-orientation: travel reservations; credit approval; BPM; …
  - Tasks, documents, etc. undergo modifications (e.g., flight reservation from reserved to ticketed), but modified WF objects still identifiable throughout
  - Complex control flow, complex process composition (danger of control flow/dataflow “spaghetti”)
  - Dataflow and control-flow are often divorced!
- Scientific “Workflows”
  - Dataflow and data transformations
  - Data problems: volume, complexity, heterogeneity
  - Grid-aspects
    - Distributed computation
    - Distributed data
    - User-interactions/WF steering
    - Data, tool, and analysis integration
  - Dataflow and control-flow are often married! (can be a happy marriage… at times…)

*Business Process Execution Language for Web Services (in case you wondered)