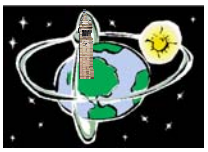
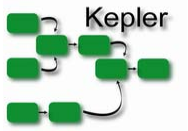
KEPLER: Overview and Project Status





Bertram Ludäscher
ludaesch@ucdavis.edu

Associate Professor
Dept. of Computer Science & Genome Center
University of California, Davis

UC DAVIS Department of Computer Science
genome center

Fellow
San Diego Supercomputer Center
University of California, San Diego

San Diego SDSC Supercomputer Center

6th Biennial Ptolemy Miniconference
Featuring the Kepler Project
May 12th, 2005, Berkeley, CA

Outline

- **Scientific Workflows (SWFs)**
  - Cyberinfrastructure, from bioinformatics to astrophysics
- **Some Kepler History**
  - … or why Ptolemy II rules
- **Current and Emerging Kepler Features**
  - from SWF plumbing/hacking to SWF design
- **Outlook**

Scientific Workflows: Pre-Cyberinfrastructure

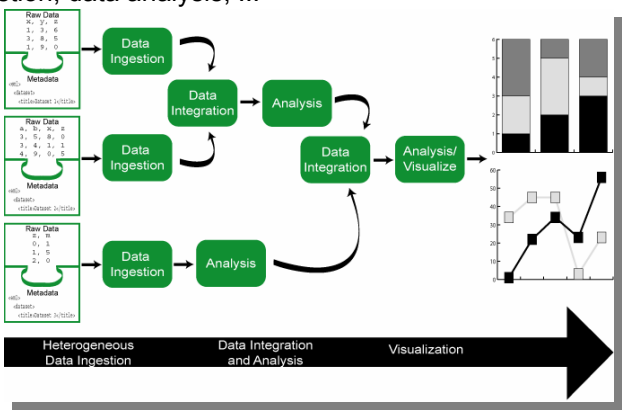
- **Data Federation & Grid "Plumbing":**
  - access, move, replicate, query … data (*Data-Grid*)
    - authenticate … SRB Sget/Sput … OPeNDAP, … Antelope/ORBs
  - schedule, launch, monitor jobs (*Compute-Grid*)
    - Globus, Condor, Nimrod, APST, …
- **Data Integration:**
  - Conceptual querying & integration, structure & semantics, e.g. mediation w/ SQL, XQuery + OWL (*Semantics-enabled Mediator*)
- **Data Analysis, Mining, Knowledge Discovery:**
  - manual/textbook (e.g. ternary diagrams), Excel, R, simulations, …
- **Visualization:**
  - 3-D (volume), 4-D (spatio-temporal), n-D (conceptual views) …

➔ **<u>one-of-a-kind custom apps., detached (island) solutions</u>**
➔ workflows are **hard to reproduce, maintain**
➔ **no/little** workflow design, automation, reuse, documentation

➔ need for an **integrated scientific workflow environment**

What is a Scientific Workflow (SWF)?

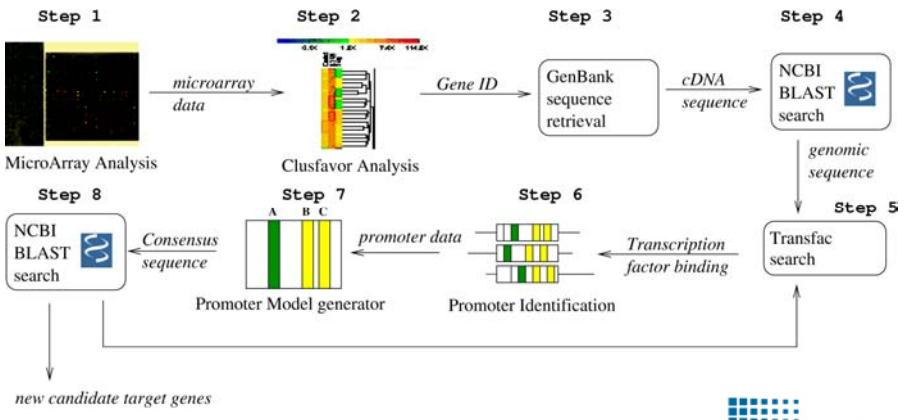
- **Model the way scientists work with their data and tools**
  - Mentally coordinate data export, import, analysis via software systems
- **Scientific workflows emphasize** data flow **(≠ business workflows)**
- **Metadata** (incl. provenance info, semantic types etc.) is crucial for automated data ingestion, data analysis, ...
- **Goals:**
  - SWF **automation**,
  - SWF & component **reuse**,
  - SWF **design** & **documentation**
  - **making scientists' data analysis and management easier**!



Some Scientific Workflow Features

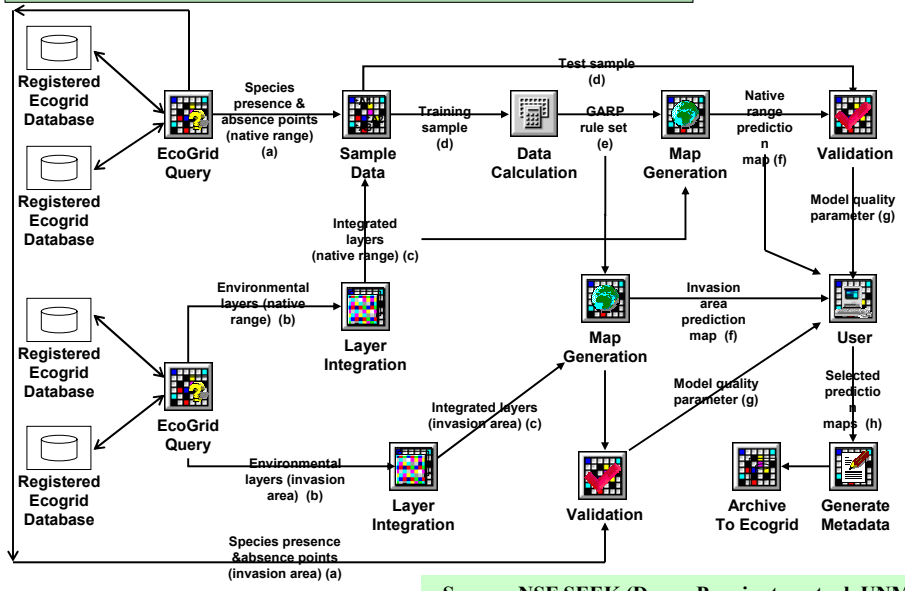
- Typical requirements/characteristics:
  - data-intensive and/or compute-intensive
  - plumbing-intensive
  - dataflow-oriented
  - distribution (data, processing)
  - user-interaction “in the middle”, …
  - … vs. (C-z; bg; fg)-ing (“detach” and reconnect)
  - advanced programming constructs (map(f), zip, takewhile, …)
  - logging, provenance, “registering back” (intermediate) products
  - …
- … easy to recognize a SWF when you see one!

Promoter Identification Workflow (Napkin Drawing)



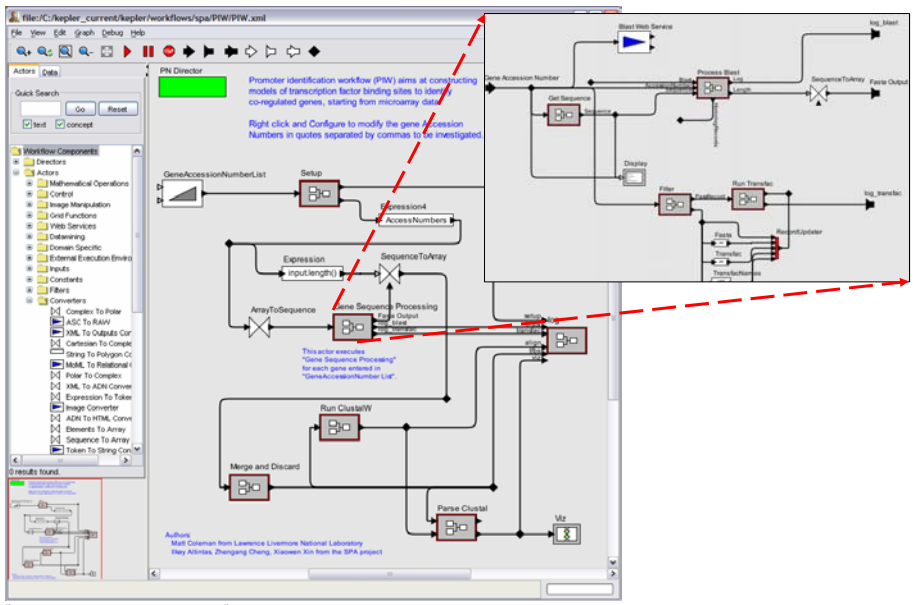
Source: Matt Coleman (LLNL)

Ecology: Analysis Pipeline for Invasive Species Prediction (Napkin Drawing)

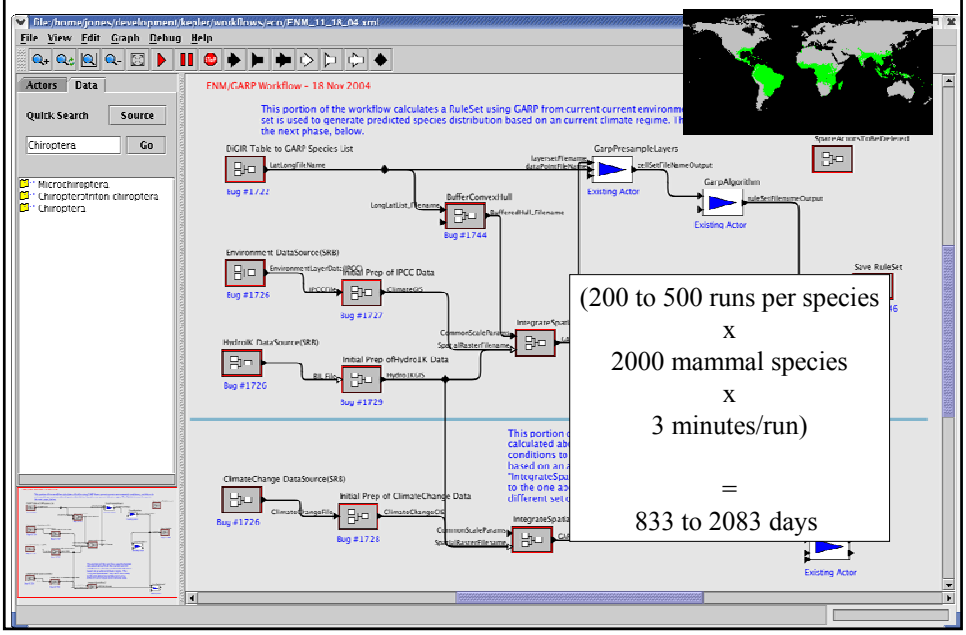


Source: NSF SEEK (Deana Pennington et. al, UNM)

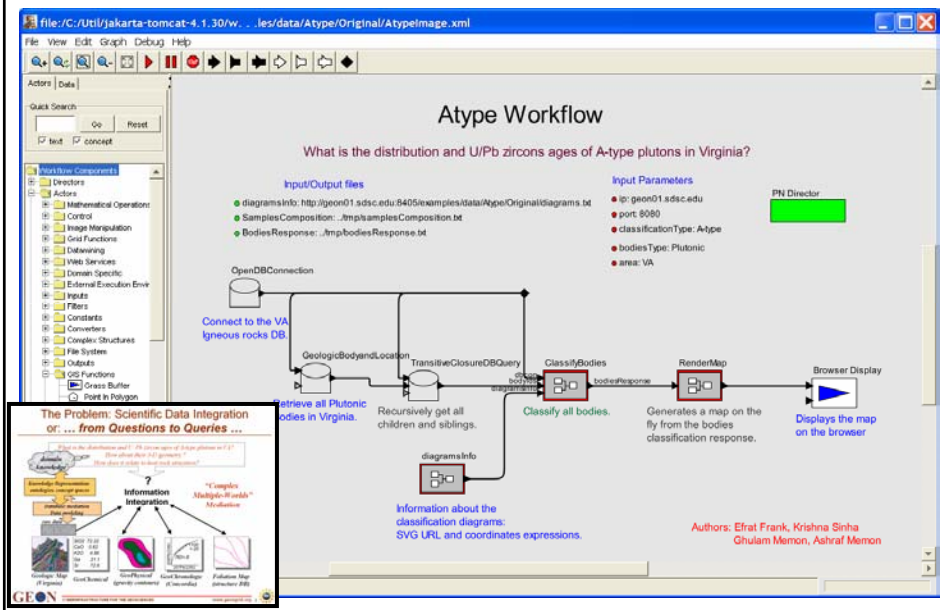
Promoter Identification Workflow in Kepler



Ecological Niche Modeling in Kepler

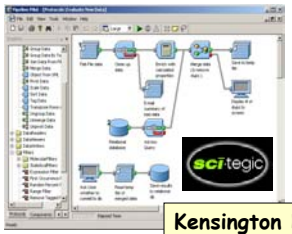


GEON Analysis Workflow in KEPLER

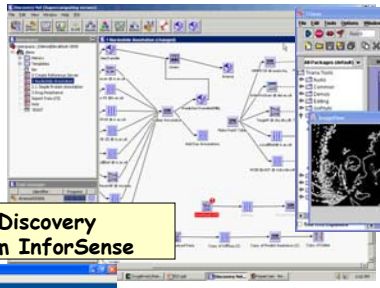


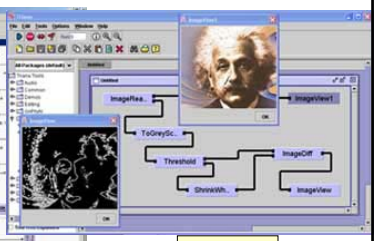
Commercial & Open Source Scientific Workflow and (**Dataflow**) Systems & Problem Solving Environments

Kepler



Kensington Discovery Edition from InforSense

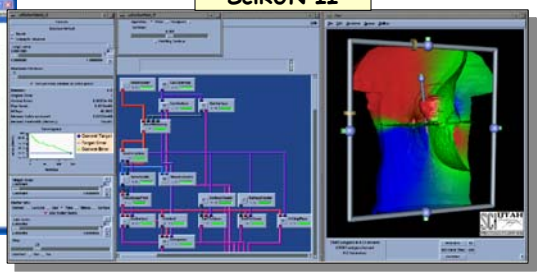




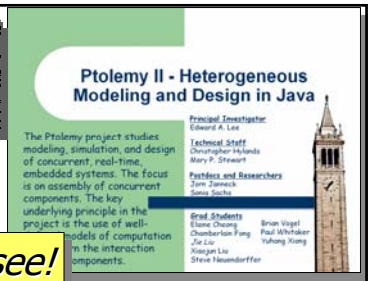
Triana

Taverna

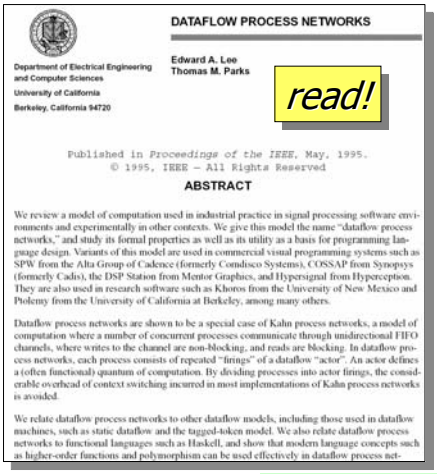
SciRUN II



Our Starting Point: Ptolemy II



see!



DATAFLOW PROCESS NETWORKS

Edward A. Lee
Thomas M. Parks

Department of Electrical Engineering and Computer Sciences
University of California
Berkeley, California 94720

read!

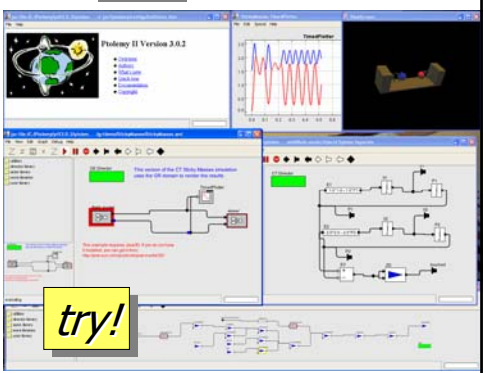
Published in *Proceedings of the IEEE*, May, 1995.
© 1995, IEEE – All Rights Reserved

ABSTRACT

We review a model of computation used in industrial practice in signal processing software environments and experimentally in other contexts. We give this model the name "dataflow process networks," and study its formal properties as well as its utility as a basis for programming language design. Variants of this model are used in commercial visual programming systems such as SPW from the Alta Group of Cadence (formerly Comdisco Systems), COSSAP from Synopsys (formerly Cadis), the DSP Station from Mentor Graphics, and Hypersignal from Hyperception. They are also used in research software such as Khoros from the University of New Mexico and Ptolemy from the University of California at Berkeley, among many others.

Dataflow process networks are shown to be a special case of Kahn process networks, a model of computation where a number of concurrent processes communicate through unidirectional FIFO channels, where writes to the channel are non-blocking, and reads are blocking. In dataflow process networks, each process consists of repeated "firings" of a dataflow "actor". An actor defines a (often functional) quantum of computation. By dividing processes into actor firings, the considerable overhead of context switching incurred in most implementations of Kahn process networks is avoided.

We relate dataflow process networks to other dataflow models, including those used in dataflow machines, such as static dataflow and the tagged-token model. We also relate dataflow process networks to functional languages such as Haskell, and show that modern language concepts such as higher-order functions and polymorphism can be used effectively in dataflow process net-



try!

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

Why Ptolemy II ?

- **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** reuse
- **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, …)
    - but middle-/underware is conveniently accessible through actors!
- **PRAGMATICS**
  - Ptolemy II is mature, continuously extended & improved, well-documented (500+pp)
  - open source system
  - many research results
  - Ptolemy II participation in Kepler

KEPLER/CSP: Contributors, Sponsors, Projects

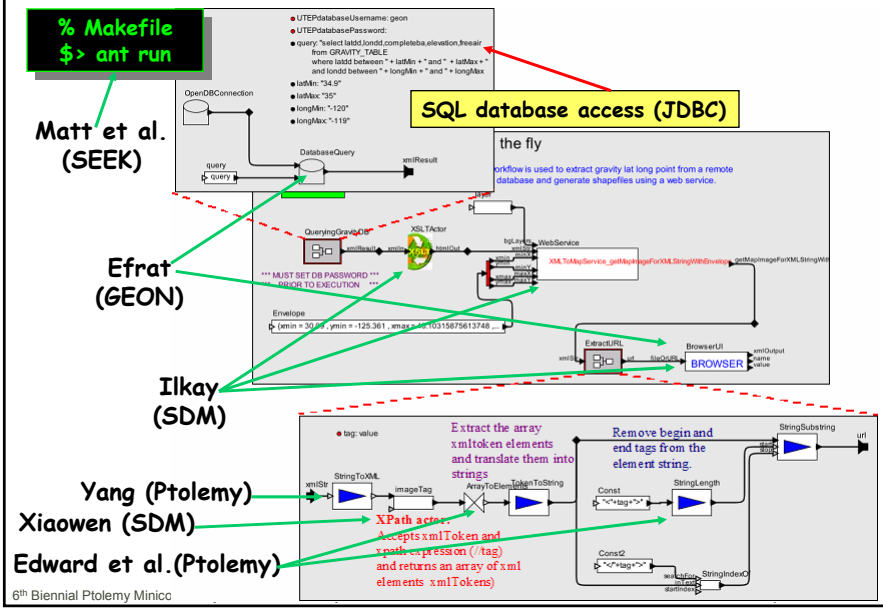
Ilkay Altintas *SDM, NLADR, Resurgence, EOL, …*
Kim Baldridge *Resurgence, NMI*
Chad Berkley *SEEK*
Shawn Bowers *SEEK*
Terence Critchlow *SDM*
Tobin Fricke *ROADNet*
Jeffrey Grethe *BIRN*
Christopher H. Brooks *Ptolemy II*
Zhengang Cheng *SDM*
Dan Higgins *SEEK*
Efrat Jaeger *GEON*
Matt Jones *SEEK*
Werner Krebs, *EOL*
Edward A. Lee *Ptolemy II*
Kai Lin *GEON*
Bertram Ludaescher *SDM, SEEK, GEON, BIRN, ROADNet*
Mark Miller *EOL*
Steve Mock *NMI*
Steve Neuendorffer *Ptolemy II*
Jing Tao *SEEK*
Mladen Vouk *SDM*
Xiaowen Xin *SDM*
Yang Zhao *Ptolemy II*
Bing Zhu *SEEK*
…

www.kepler-project.org

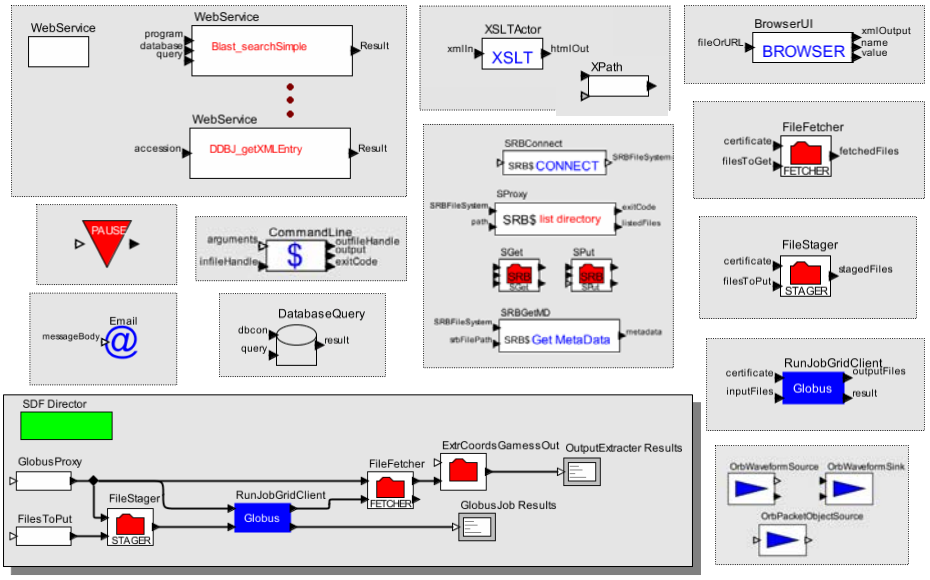
LLNL, NCSU, SDSC, UCB, UCD, UCSB, UCSD, U Man… Utah,…, UTEP, …, Zurich

Collab. tools: IRC, cvs, skype, Wiki: hotTopics, FAQs, ..

GEON Dataset Generation & Registration
(and co-development in KEPLER)



Some KEPLER Actors (out of 160+ … and counting…)



KEPLER Today

- Support for SWF life cycle
  - Design, share, prototype, run, monitor, deploy, …
- Coarse-grained **scientific workflows, e.g.,**
  - web service actors, grid actors, command-line actors, …
- Fine grained **workflows and simulations, e.g.,**
  - Database access, XSLT transformations, …
- Kepler Extensions
  - support for data- and compute-intensive workflows (SDM/SPA, SEEK)
  - real-time data streaming (ROADNet)
  - other special and generic extensions (e.g. GEON, SEEK)
- Status
  - first release (alpha) was in May 2004
  - nightly builds w/ version tests
  - "Link-Up Sister Project" w/ other SWF systems (myGrid/Taverna, Triana, …), SciRUN II (DOE SciDAC/SDM)
  - Participation in various workshops and conferences (GGF10, SSDBMs, eScience WF workshop, …)

Kepler Today: Some Numbers

- **#Actors:**
  - Kepler: ~160 new + ~120 inherited (PTII)
  - soon there can be thousands (harvested from web services, R packages, etc.)
- **#Developers:**
  - ~ 24+, ~10 very active; more coming… (we think :-)
- **#CVS Repositories: ~2**
  - hopefully not increasing… :-{
- **# "Production-level" WFs:**
  - currently ~8, expected to increase quite a bit …

KEPLER Tomorrow

- Application-driven extensions (here: SDM):
  - access to/integration with other IDMAF components
    - PnetCDF?, PVFS(2)?, MPI-IO?, parallel-R?, ASPECT?, FastBit, …
  - support for execution of new SWF domains
    - Astrophysics, Fusion, ….
- Further generic extensions:
  - addtl. support for data-intensive and compute-intensive workflows (all SRB Scommands, CCA support, …)
  - semantics-intensive workflows
  - (C-z; bg; fg)-ing (“detach” and reconnect)
  - workflow deployment models
  - distributed execution
- Additional “domain awareness” (esp. via new directors)
  - time series, parameter sweeps, job scheduling (CONDOR, Globus, …)
  - hybrid type system with semantic types (“Sparrow” extensions)
- Consolidation
  - More installers, regular releases, improved usability, documentation, …

A User’s Wish List

- Usability
- Closing the “lid” (cf. vnc)
- Dynamic plug-in of actors (cf. actor & data registries/repositories)
- Distributed WF execution
- Collection-based programming
- Grid awareness
- Semantics awareness
- WF Deployment (as a web site, as a web service, …)
- “Power apps” (➔ SciRUN II)
- …

Separation of Concerns

- **A shining example:**
  - Ptolemy Directors – "factoring out" the concern of workflow "orchestration" (MoC)
  - common aspects of overall execution **not** left to the actors
- **Similarly:**
  - The "Black Box" ("flight recorder")
    - a kind of "recording central" to avoid wiring 100's of components to recording-actor(s)
  - The "Red Box" (error handling, fault tolerance)
    - ……
  - The "Yellow Box" (type checking)
    - ……
  - The "Blue Box" (shipping-and-handling)
    - central handling of data transport (by value, by reference, by scp, SRB, GridFTP, …)

SDF/PN/DE/…

Recorder

On Error

Static Analysis

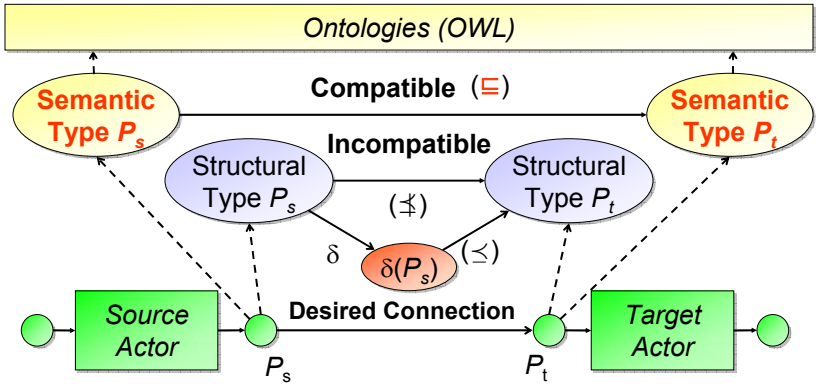
SHA @

Separation of Concerns: Port Types

- **Token consumption (& production) "type"**
  - a director's concern
- **Token "transport type"**
  - by value, reference (which one), protocol (SOAP, scp, GridFTP, scp, SRB, …)
  - a SHA concern
- **Structural and semantic types**
  - SAT (static analysis & typing) concern
  - built after static unit type system…
    - static unit type system as a special case!?

Hybrid Types (Structure + Semantics)

- Services can be ***semantically compatible***, but ***structurally incompatible***



Source: [Bowers-Ludaescher, DILS'04]

Scientific Workflow Design

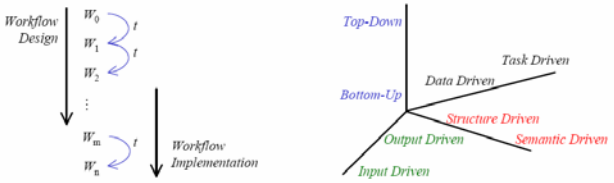
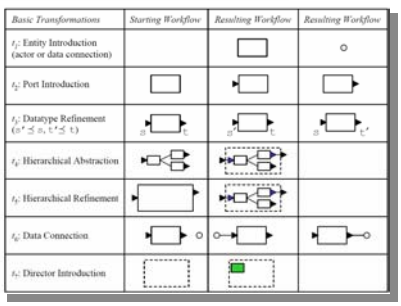


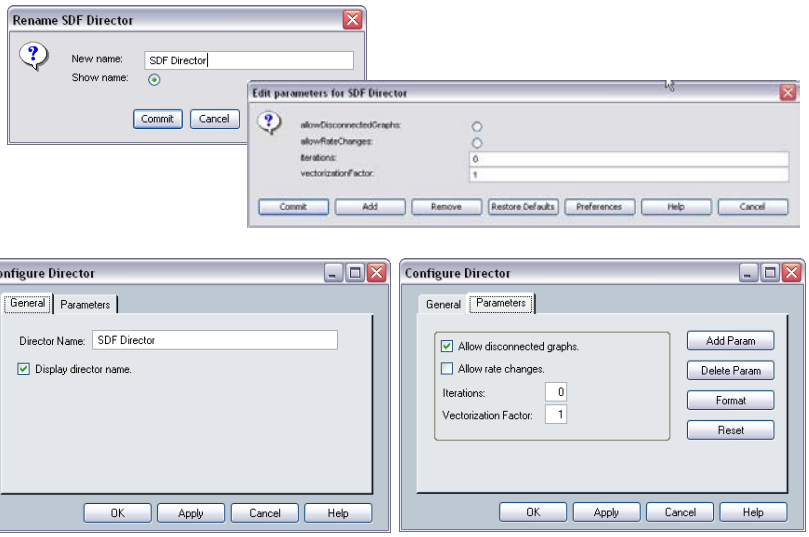
Fig. 2. Workflow engineers evolve workflows by applying design primtives (left), shown as transformations *t*; and certain primitives can be grouped to form design strategies (right), where each design strategy is shown as a distinct dimension of a design space.

- **Support SWF design & reuse, via:**
  - Structural data types
  - Semantic types
  - Associations (=constraints) between them
  - Type checking, inference, propagation

  ➔Separation of concerns:
  - structure, semantics, WF orchestration, etc.

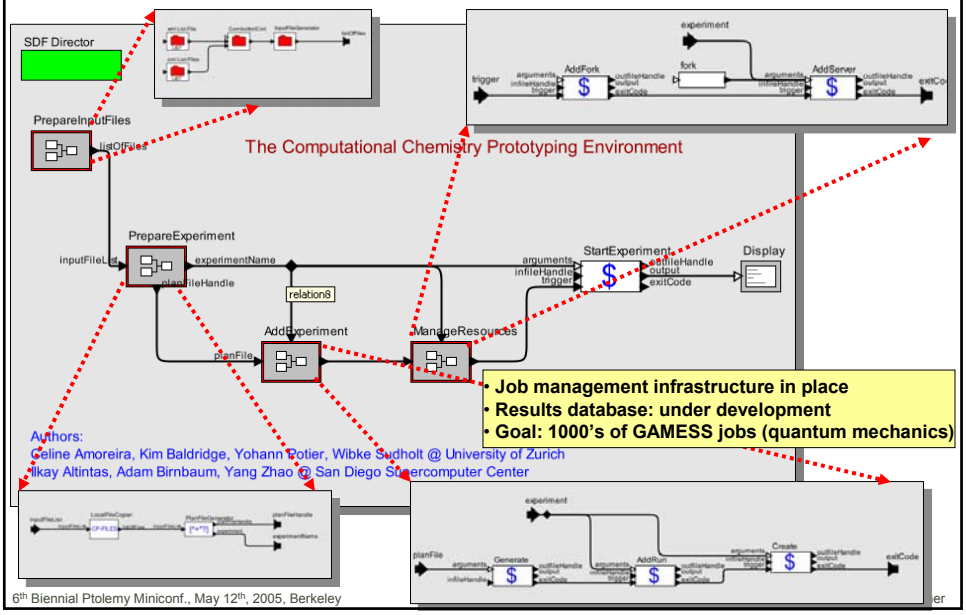


Usability Engineering



Source: Laura Downey, SEEK/LTER

Job Management (here: NIMROD)



Breaking into the Parallel (e.g. MPI) and Stream Processing Worlds!?

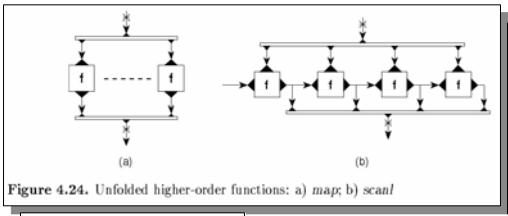
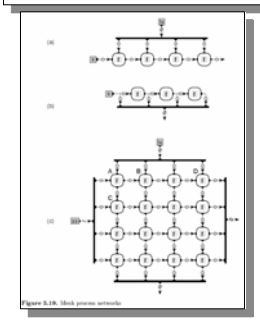


Figure 4.24. Unfolded higher-order functions: a) *map*; b) *scanl*

```
(:-)     :: α → Stream^n α → Stream^n α
groupS   :: Int → Stream^nk α → Stream^n (Vector^k α)
concatS  :: Stream^n (Vector^k α) → Stream^nk α
zipS     :: Stream^n α → Stream^n β → Stream^n (α, β)
unzipS   :: Stream^n (α, β) → (Stream^n α, Stream^n β)
mapS     :: (α → β) → Stream^n α → Stream^n β
```



Source: Real-Time Signal Processing: Dataflow, Visual, and Functional Programming, Hideki John Reekie, University of Technology, Sydney

```
zipWithS :: (α → β → γ) → Stream α → Stream β → Stream γ
zipWithS f xs ys  = mapS (\(x,y) -> f x y) (zipS xs ys)

zipOutS :: (α → (β, γ)) → Stream α → (Stream β, Stream γ)
zipOutS f xs = unzipS (mapS f xs)

zipOutWithS :: (α → β → (γ, δ)) → Stream α → Stream β
                                → (Stream γ, Stream δ)
zipOutWithS f xs ys = unzipS (mapS (\(x,y) -> f x y) (zipS xs ys))

iterateS :: (α → α) → α → Stream α
iterateS f a  = let ys = a :- (mapS f ys) in xs

generateS :: (α → (α, β)) → α → Stream β
generateS f a = let (zs,ys) = zipOutS f (a :- zs) in ys

scanS :: (α → β → α) → α → Stream β → Stream α
scanS f a xs = let ys = zipWithS f (a :- ys) xs in ys

stateS :: (α → β → (α, γ)) → α → Stream β → Stream γ
stateS f a xs = let (zs,ys) = zipOutWithS f (a :- zs) xs in ys
```

Figure 5.12. Process constructor definitions

- **Clean functional semantics facilitates *algebraic workflow (program) transformations* (Bird-Meertens); e.g. mapS *f* • mapS *g* ➔ mapS (*f* • *g*)**