Outline of the Tutorial Modules

I. Overview on Scientific Data Management (Gertz)
13:30—14:15

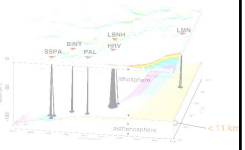
II. From Conventional to Scientific Data Integration (Ludaescher)
14:15—15:00

III. From Scientific Data Formats to Data Stream Processing (Gertz)
15:30—16:15

IV. Introduction to Scientific Workflows (Ludaescher)
16:15-17:00

Types of "Integration"

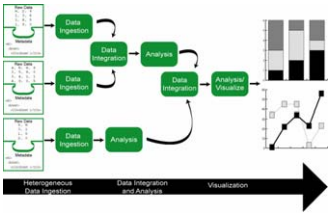
- Spatial (co-)registration/"overlay" of different data
  - from 2D, 3D, 4D (x,y,z,*t*), (4+n) D



- Conventional (DB-oriented) integration:
  - schema-based
  - view-based
  - at the data-level

- Extended DI approaches using "ontologies"
  - ontologies? controlled vocabularies? metadata/annotations?

- **Application/process integration**
  ➔ scientific workflows (Module IV)

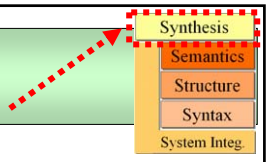


- Other mechanisms of "integration"
  - link-based (Aladin), clustering, …
  - statistics, data mining, visualization, …

IV. Introduction to Scientific Workflows (SWF)

- **Scientific Workflows in e-Science and CI**
- **SWF vs Business Workflows**
- **Features of a SWF System (Kepler)**
- **Flow-based Programming and Scientific Workflow Design**
- **Semantic Extensions**

Information Integration Challenges: *S[5] Heterogeneities*



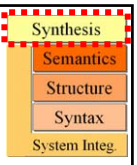
- ***S*ynthesis** of applications, analysis tools, data & query components, … into "**scientific workflows**"
  - How to put together components to solve a scientist's problem?

➔ **Scientific Problem Solving Environments (PSEs)**

➔ **Portals, Workbench ("scientist's view")**
+ ontology-enhanced data registration, discovery, manipulation
+ creation and registration of new data products from existing ones, …

➔ **Scientific Workflow System ("engineer's view")**
+ for designing, re-engineering, deploying analysis pipelines and scientific workflows; ***a tool to make new tools …***
+ e.g., creation of new datasets from existing ones, dataset registration, …

Motivation:
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**

Scientific Workflows

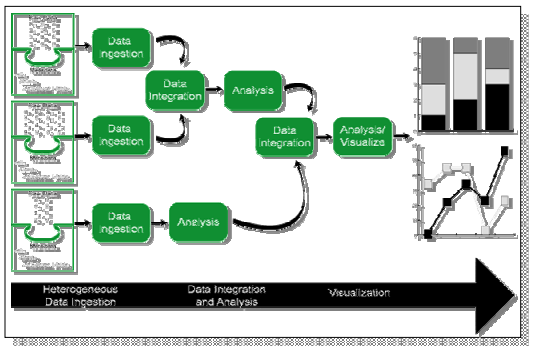
A model of the way a scientist works with their data and tools
- Mentally coordinate data export, import, analysis via software systems

Emphasize dataflow (≠ business workflows)

Metadata: automatic data ingestion, analysis, provenance tracking …

Goals:
- **SWF automation**
- **SWF component reuse**
- **SWF design & documentation**

… make scientific data analysis and management tasks easier for the scientist!



Types of Scientific Workflows (overlapping)

- **What do we use scientific workflow systems for?**
  - Short answer:

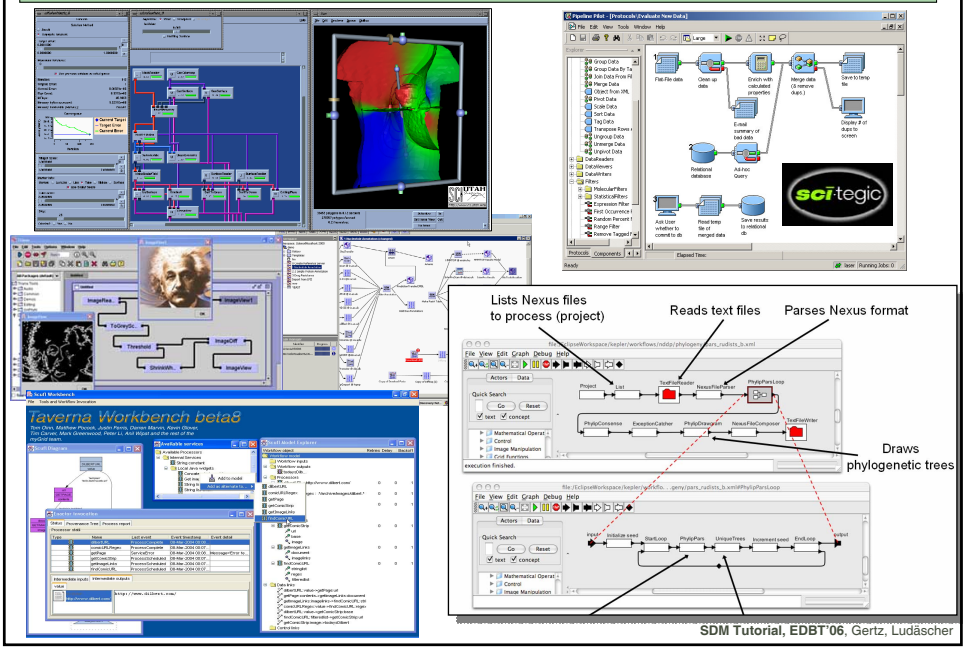
    … nearly everything …

- **Types of Scientific Workflows**
  - **"Modeling & Design"**: Capture or reverse-engineer processes and information flows at all levels
  - **"Knowledge discovery"**: Automate repetitive data access, retrieval, custom analysis (e.g. Blast), generic steps (PCA, cluster analysis, ..),

    Ex: PIW, Motif analysis, NDDP, …
  - **"Plumbing"**: Stage files, submit batch jobs, monitor progress, move files off XT3 to analysis and viz cluster, archive, steer computation, …

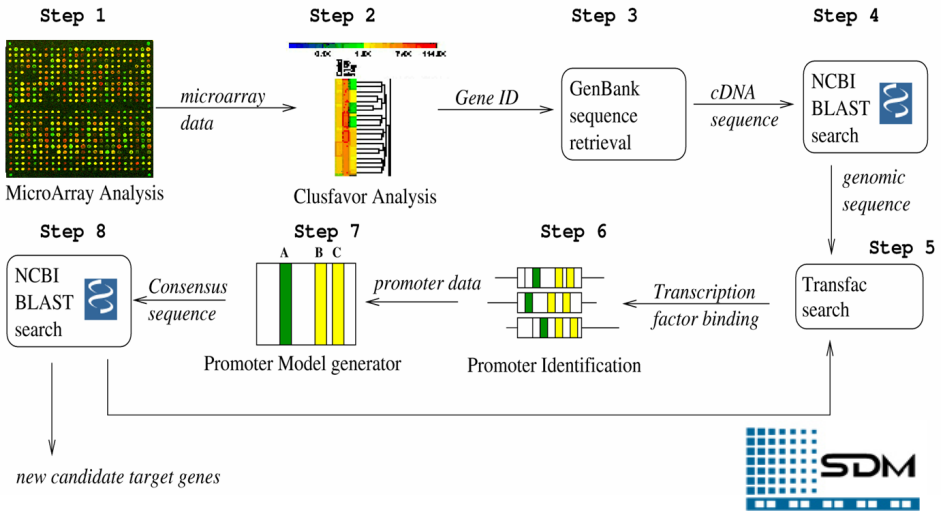
    Ex: Fusion simulation, Astrophysics (supernova simulation)
  - **"(Real-time) analysis pipelines"**: processing of environmental and earth science data from sensor networks

Commercial & Open Source Scientific Workflow Systems



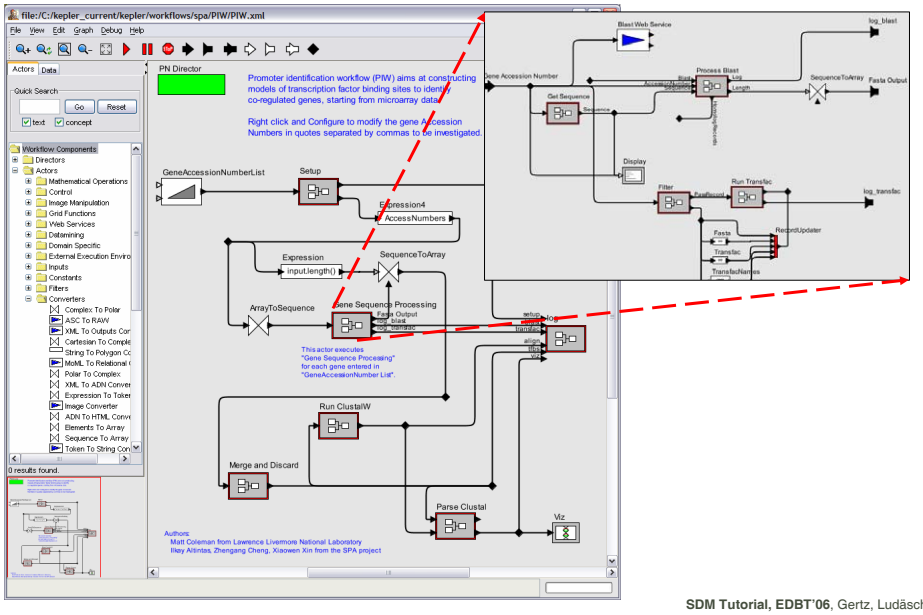
Promoter Identification Workflow (PIW)

… or from a napkin drawing …

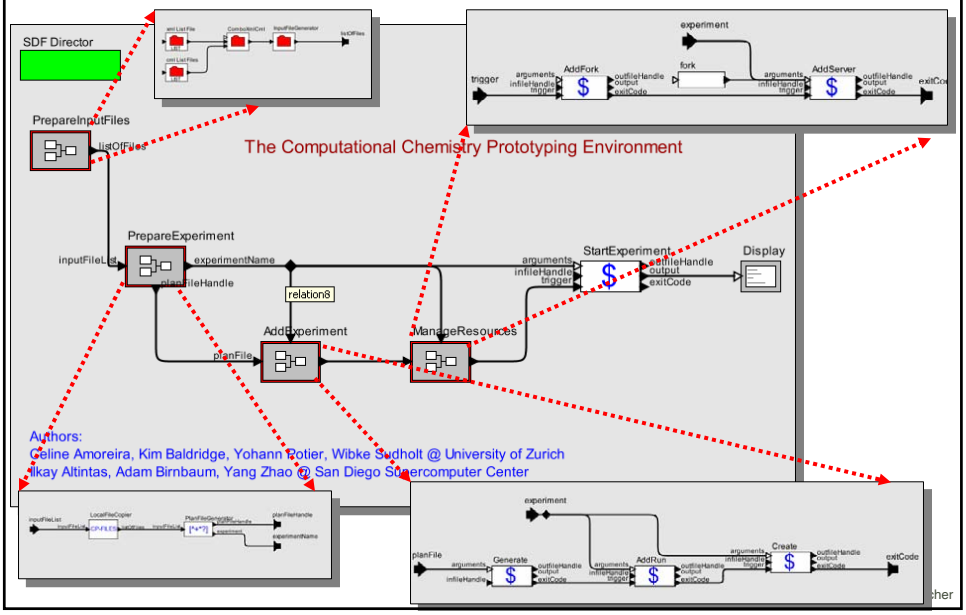


Source: Matt Coleman (LLNL)

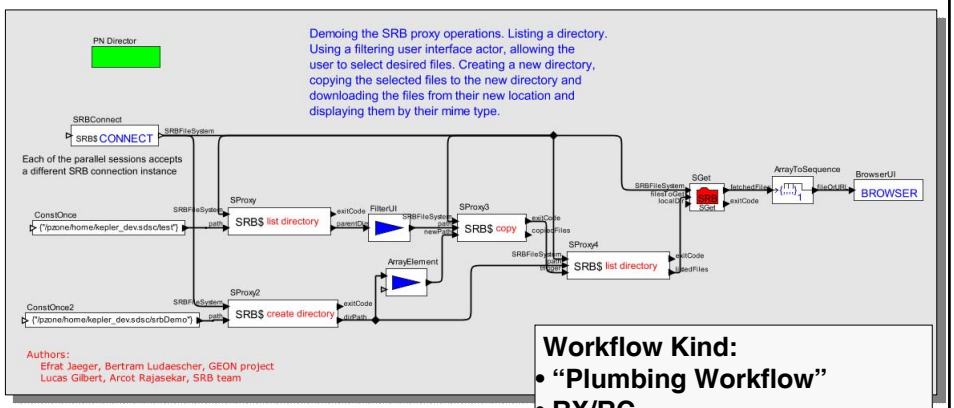
… to an executable workflow (here: in KEPLER)



… to a plumbing workflow (Job Mgmt w/ NIMROD)

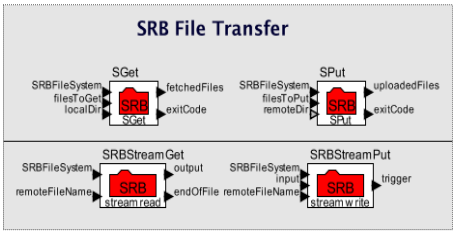


… more plumbing …



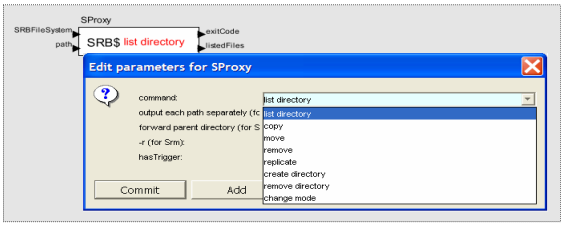
Workflow Kind:
- **"Plumbing Workflow"**
- **RX/RC**
  (remote exec, remote control)

… more plumbing …

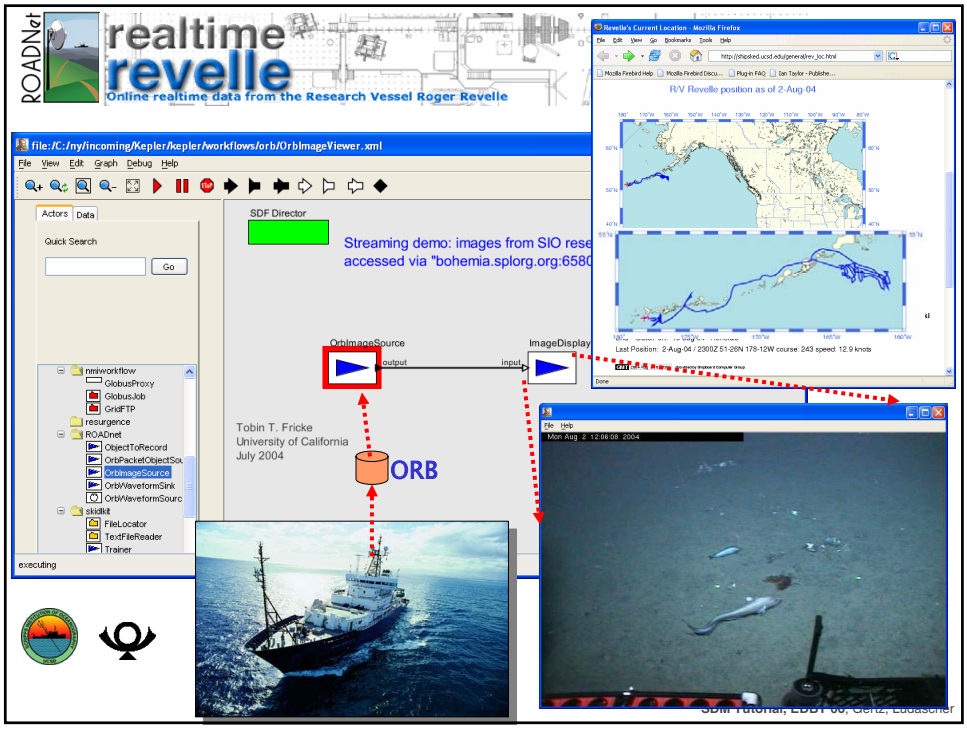


SPut/SGet use parallel put/get approaches as provided by JARGON API

Streaming Actors **Stream Put/Get** read & write files from and to SRB as sequence of byte arrays.



SRB **Proxy** Operations



Why not just a Python script?

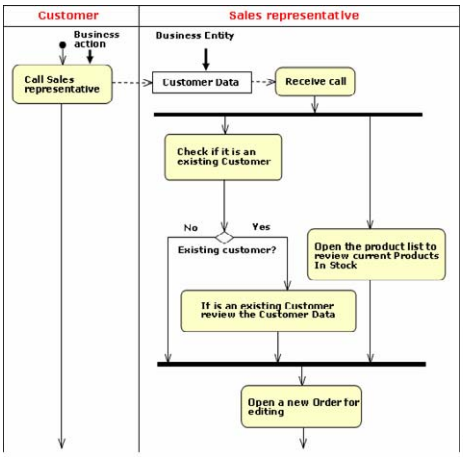
- **Users who can define, reuse, modify, specialize workflows may not be able to do the same (or as easily as) for Python scripts**
- **Other advantages to scientific workflows**
  - Modular reuse and application interoperability
  - Debugging and monitoring workflow execution
  - Automated data management (e.g., provenance)
  - Validation (e.g., data/structural/semantic typing)

  … From integrated modeling to execution, optimization, archival, etc

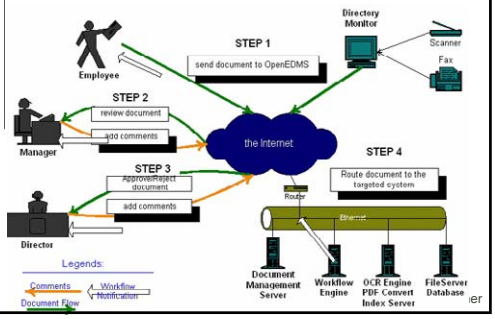
Business workflows *born-again*?

- **Yes, there are similarities**
  - And we can learn from BWF – e.g. transactions!
- **But also big differences:**
  - **Scientific Workflows:**
    - dataflow oriented
    - streaming/pipelined execution
    - cf. signal processing
    - popular Model of Computation (MoC):
      - Process Networks (PN), Synchronous DataFlow (SDF), Discrete Events (DE), Continuous Time (CT), …
  - **Business Workflows**:
    - task- and control-flow oriented
    - popular MoC (for theory, abstraction, modeling):
      - Petri-Nets, CSP!?, …

Sample Business Workflow

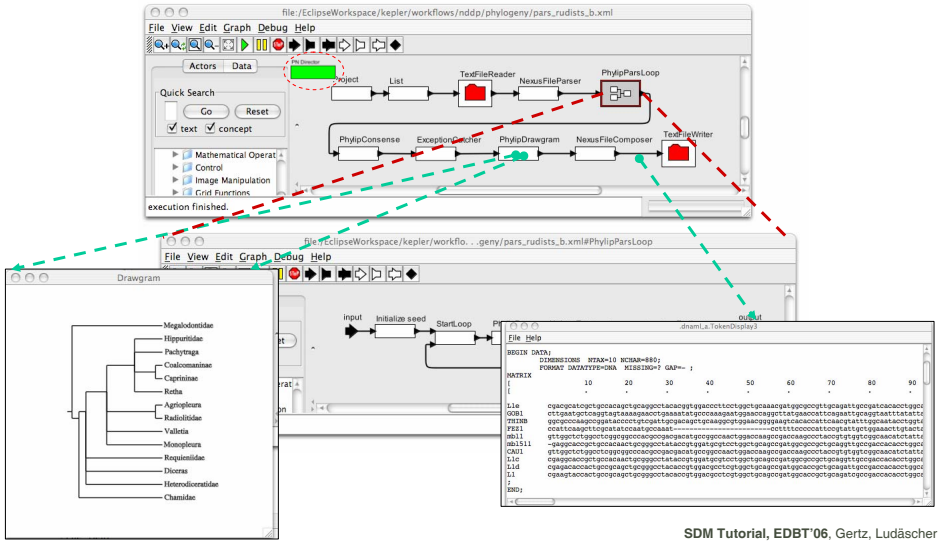


- **Focus is on …**
  - Tasks
  - Control-flow
  - Work items
- **Useful stuff:**
  - Transactions!
  - How to handle complex control-flow …

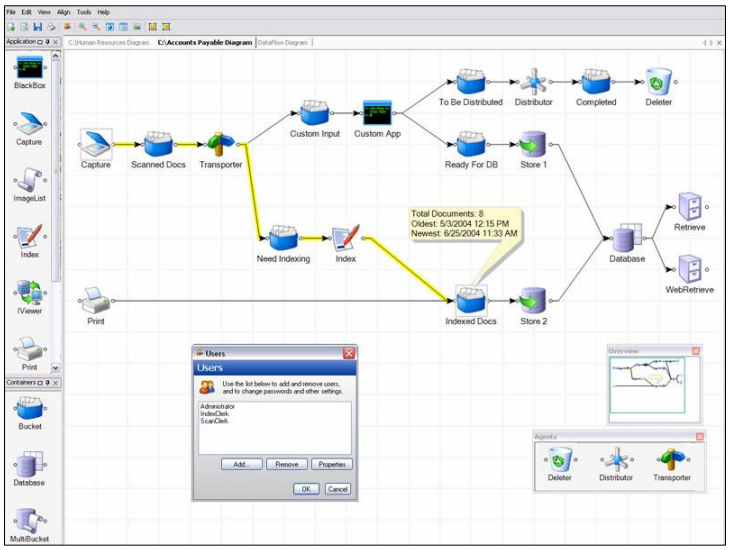


A Simple Scientific Workflow

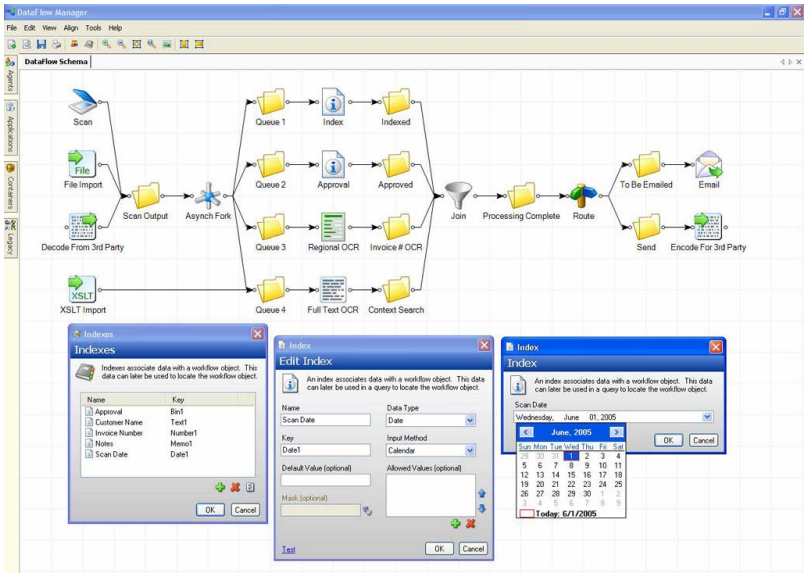
Example scientific workflow run, executed as a Dataflow Process Network



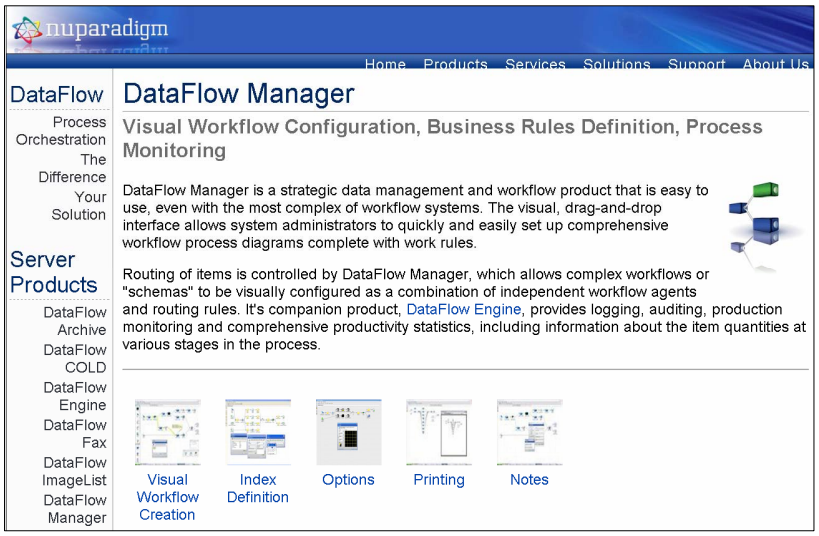
Pop Quiz! BWF? SWF?



And the answer is …

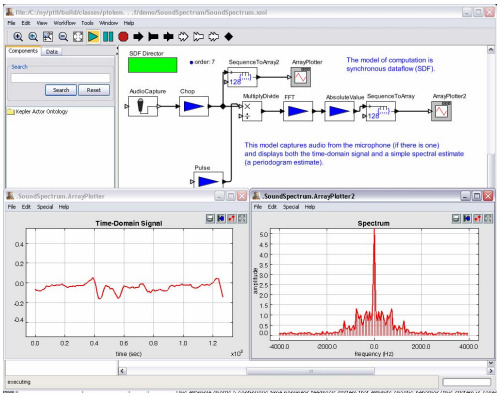


Wrong Question! (not just SWF vs. BWF, but Dataflow and Data-orientation vs. Control-Flow/Orientation)



Driving the point home…

- **Dataflow-oriented scientific workflows have features of**
  - … stream-processing
  - … data-, task-, and **pipeline-parallelism**
  - … signal processing systems
  - … visual PSEs: AVS/Express, IBM DataExplorer, OpenDX, LabView, …



Some Features of Scientific Workflows

- **Actor-oriented, Hierarchical Modeling & Design**
  - study interactions of MoCs, compose, nest WFs, …
  - simulate those systems
- **Automated Provenance System**
  - Keep track of data & workflow provenance
  - ➔ "smart" re-run, crash recovery, live & post-mortem analysis & debugging …
- **Flow-Based Programming Paradigm**
  - data-stream oriented processing
- **Basis for End-to-End Experiment Life-Cycle Management**
  - from design, semantic types, to monitoring & control ("dashboard"), and optimization

Some Basic SWF Terminology (evolving)

- ***Workflow definition W* (≈ WF graph we see)**
  - **partial** specification of a workflow (cf. *program*)
  - **parameters *P*** need to be **instantiated**
  - **data-bindings *D*** can be viewed as special parameters
- ***Model of Computation* (MoC – PT terminology)**
  - Looking at *W, P, D* we still do **not** know how to execute *W(P,D)* to compute result *R*
  - A MoC is an algorithm telling us how to apply *W* on *P* and *D* to obtain *R.*
  - Examples:
    - MoC PN (Process Network):
      - Network of independent processes, communicating through (infinite) unidirectional buffers (queues), prefix-monotonic behavior; given a PN and an input stream and prefix-monotonic, deterministic actors, the output stream is determined! (lots of flexibility for execution!)
    - MoC SDF (Synchronous Dataflow):
      - Similar to PN, but actors must statically declare there token production/consumption rates; solving for pos. int. solutions of balance equations ("LGS") yields static schedule guaranteeing fixed buffer size

Some Basic SWF Terminology (evolving)

- **WF *Run*: completed computation**
- **WF *Execution*: ongoing computation**
- ***Computation graph*: graph data structure keeping track of which token has been computed from which other one(s)**
  - Simple examples: evaluating an arithmetic expression; running a "job DAG"
  - But keeping track of "real dependencies" can be tricky
    - Ex: output tuples of an SQL query have "witness tuples" in multiple relations; clear for positive existential queries; what are witnesses for universal and negated queries? R = A \ B ; witnesses anybody?
    - Similar to the notion of "proof tree" in logic (and LP); negation-as-failure looms it's ugly (beautiful?) head!

Research Area: Provenance

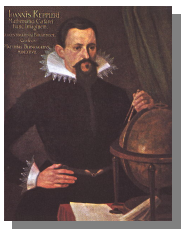
- **(Abstract) Use Cases:**
  - "Total Recall": capture everything the MoC can observe
    - … and more: MoC-inherent plus addtl. ***observables***
      - Example: time-stamp *token-in*, *token-out* events ➔ benchmark actor exec time, data movement time, …
      - The 7 W's: Who, What, Where, Why, When, Which, (W)how (C. Goble)
  - Smart Re-run:
    - after Pause or Stop, followed by parameter changes: rerun only relevant parts
  - Fault tolerance, crash recovery (cf. checkpointing)
  - Result interpretation and post-mortem analysis

Research Area: Provenance (2)

- **Research Questions:**
  - Given a use case (as a query U) and a provenance schema PS, can U be answered using PS?
    (related to *query answering using views* – a *reasoning* problem!)
  - Ultimately: design PS with U in mind! Also: optimize/specialize PS if U is known/limited
  - Note: the MoC can make a difference! For example, some MoCs have explicit notion of "firing" or might exploit actor declarations ("I'm a function! I have no state!") This info is relevant e.g. for checkpointing (Need to save state or not? When to save state..)

An Example System & Project

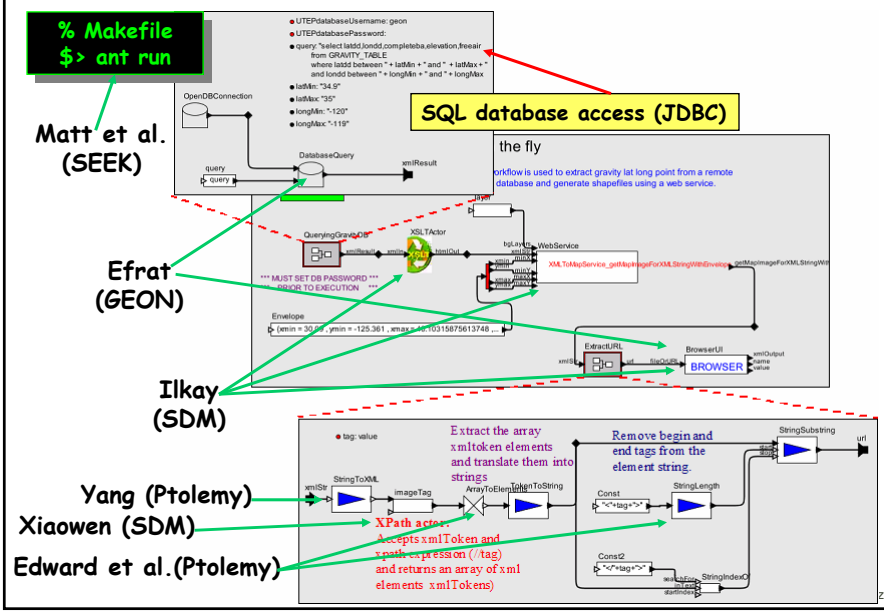
- **Open-source cross-project collaboration:**
  - NSF/ITR SEEK, GEON, DOE SciDAC/SDM, …
  - based on Ptolemy II Modeling & Simulation system
  - R&D at Berkeley, SDSC, UCSB, NCSU, UCD, LLNL, Utah, Rutgers, Penn State, Zurich, …



www.kepler-project.org

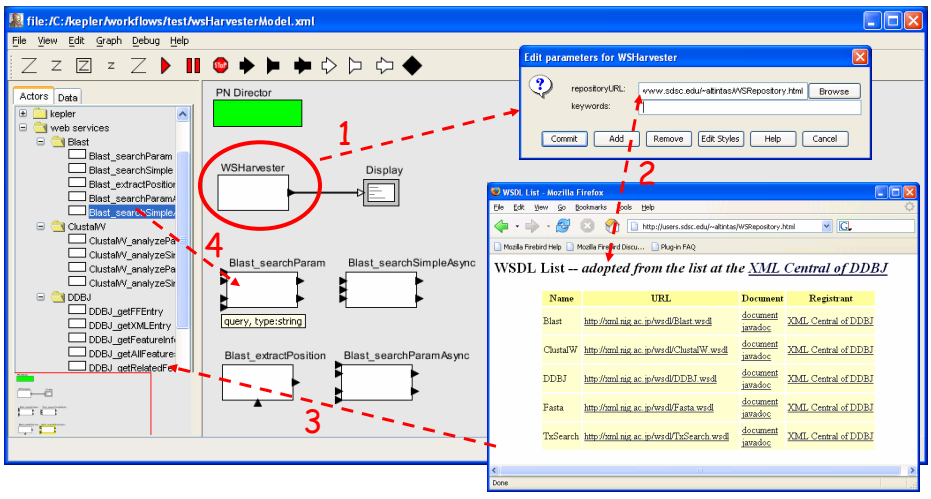
Collab. tools: IRC, cvs, Wiki, FAQs, ..

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



z, Ludäscher

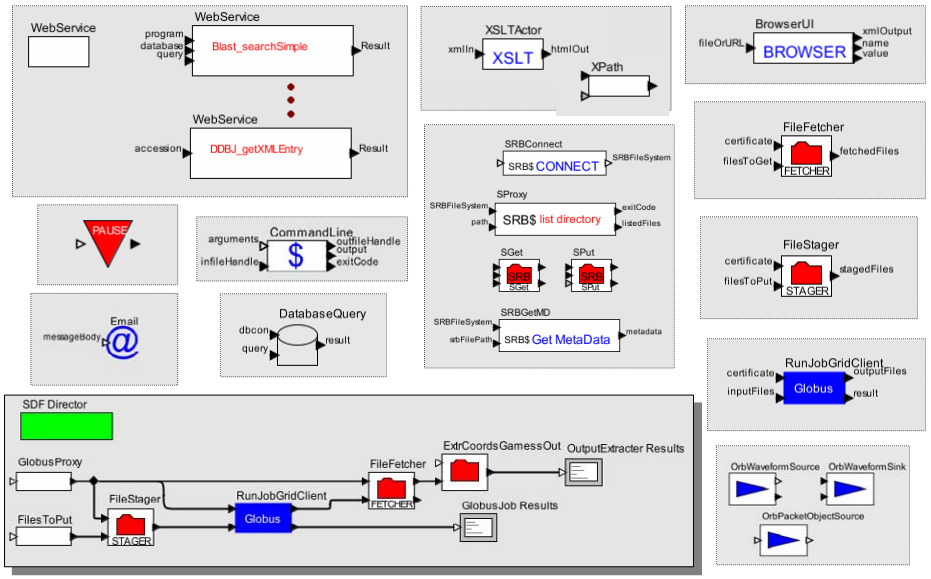
Web Services ➔ Actors (WS Harvester)



➔ "Minute-made" (MM) WS-based application integration

- Similarly: MM workflow design & sharing w/o implemented components

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

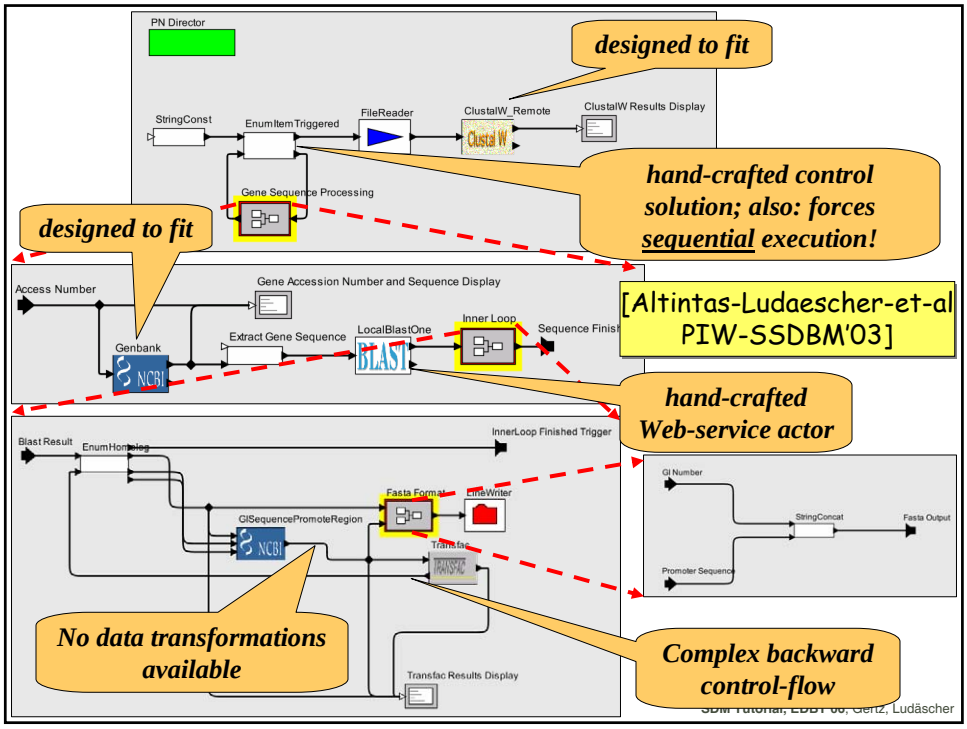


Flow-Based Programming & Design for SWF

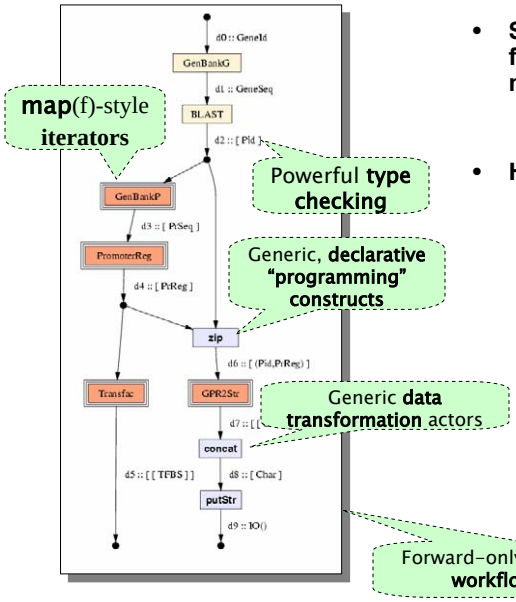
- Just doing visual-programming by itself does **not** lead to modular, re-usable, maintainable workflows!
- To fully exploit the dataflow paradigm …
  … **think dataflow!**

➔Flow-based Programming
➔… combined w/ Functional, Collection-oriented Programming

- **… similar to assembly-line metaphor**

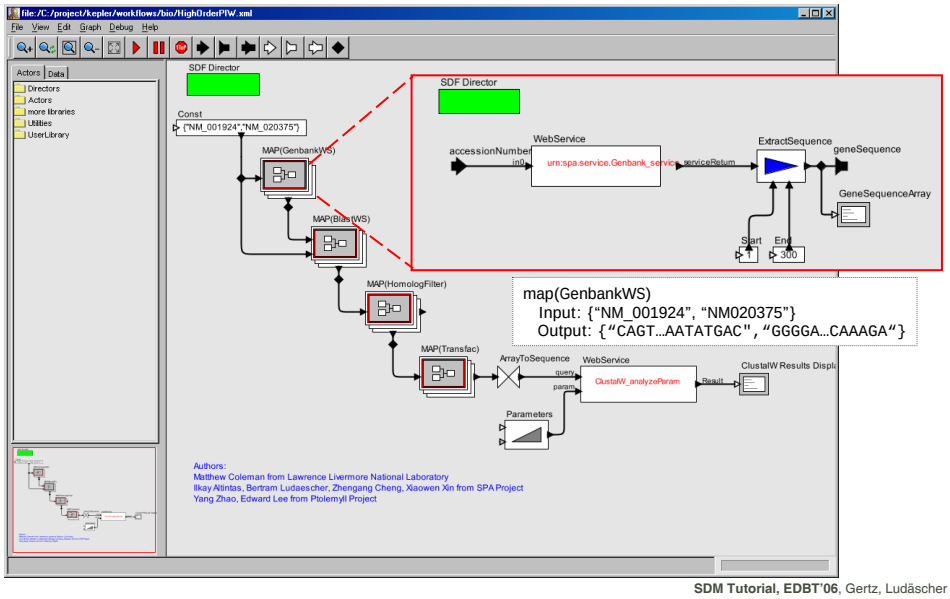


A Scientific Workflow Problem: More Solved
(Computer Scientist's view)

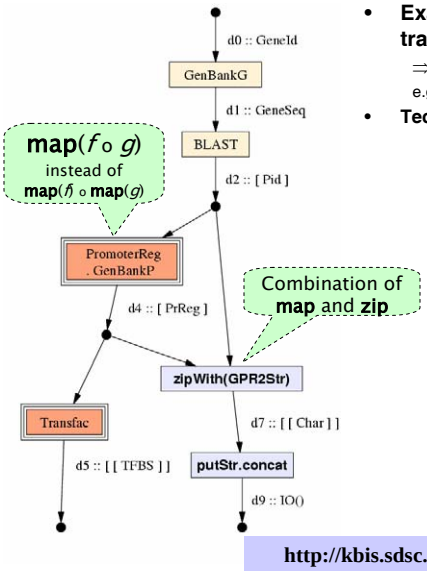


- **Solution based on declarative, functional dataflow process network**
  (= also a ***data streaming model***!)
- **Higher-order constructs:** map(***f***)
  - ⇒ **no control-flow spaghetti**
  - ⇒ **data-intensive apps**
  - ⇒ **free concurrent execution**
  - ⇒ **free type checking**
  - ⇒ automatic support to go from piw(GeneId) to PIW :=**map**(piw) over [GeneId]

A Scientific Workflow Problem: Even More Solved (domain&CS coming together!)



Research Problem: Optimization by Rewriting



- **Example: PIW as a declarative, referentially transparent functional process**
  - ⇒ optimization via functional rewriting possible
  - e.g. map($f \circ g$) = map(f) ∘ map(g)
- **Technical report &PIW specification in Haskell**

http://kbis.sdsc.edu/SciDAC-SDM/scidac-tn-map-constructs.pdf

Optimizing II: Streams & Pipelines

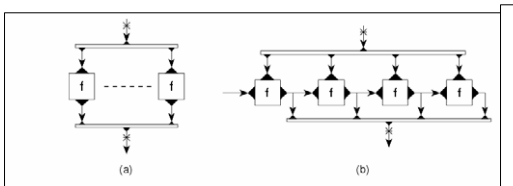
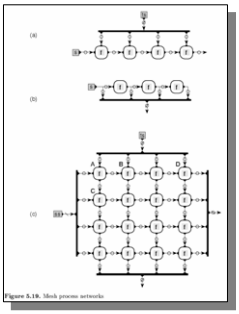


Figure 4.24. Unfolded higher-order functions: a) *map*; b) *scanl*

$$
\begin{aligned}
(:\text{-}) &:: \alpha \to \text{Stream}^n\ \alpha \to \text{Stream}^n\ \alpha \\
\text{groupS} &:: \text{Int} \to \text{Stream}^{nk}\ \alpha \to \text{Stream}^n\ (\text{Vector}^k\ \alpha) \\
\text{concatS} &:: \text{Stream}^n\ (\text{Vector}^k\ \alpha) \to \text{Stream}^{nk}\ \alpha \\
\text{zipS} &:: \text{Stream}^n\ \alpha \to \text{Stream}^n\ \beta \to \text{Stream}^n\ (\alpha, \beta) \\
\text{unzipS} &:: \text{Stream}^n\ (\alpha, \beta) \to (\text{Stream}^n\ \alpha, \text{Stream}^n\ \beta) \\
\text{mapS} &:: (\alpha \to \beta) \to \text{Stream}^n\ \alpha \to \text{Stream}^n\ \beta
\end{aligned}
$$

Figure 5.10. Types of stream functions

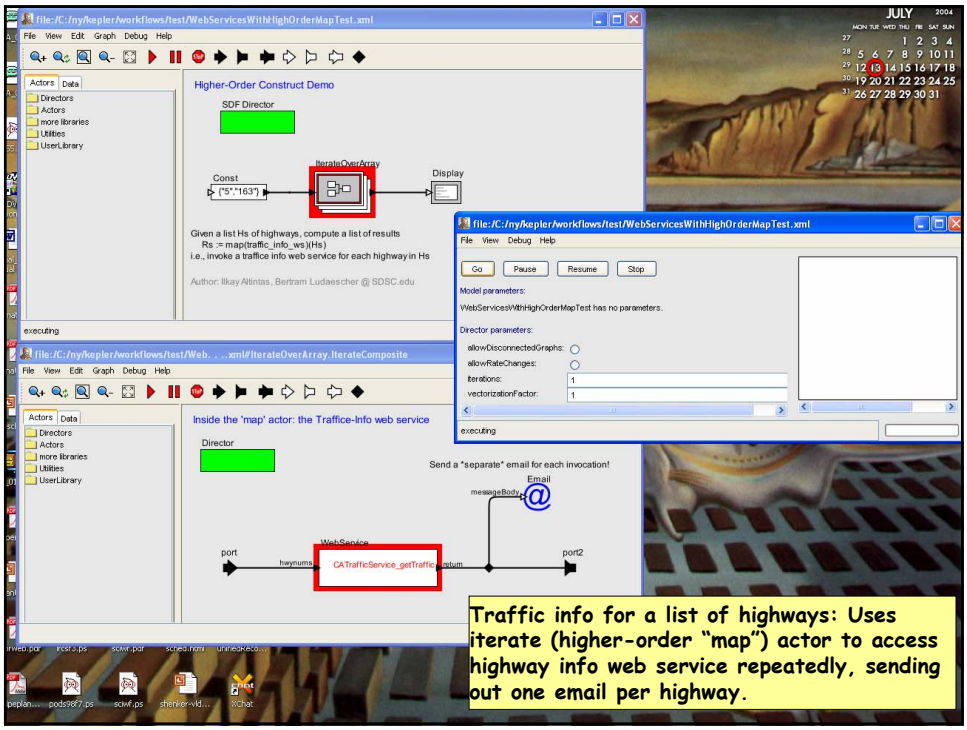


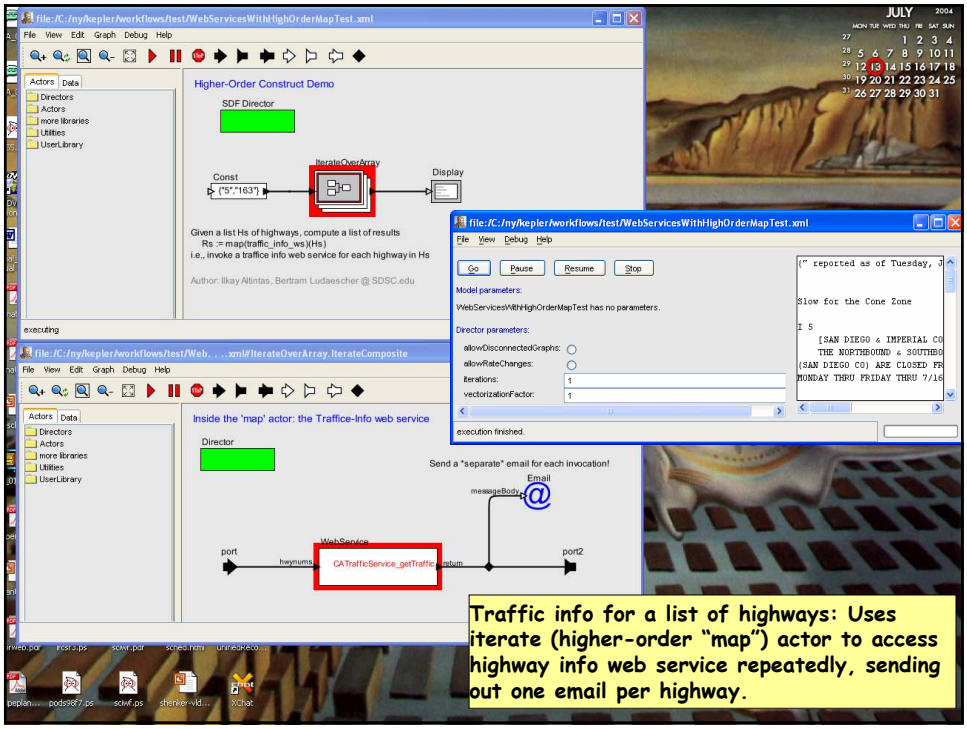
Source: Real-Time Signal Processing: Dataflow, Visual, and Functional Programming, Hideki John Reekie, University of Technology, Sydney

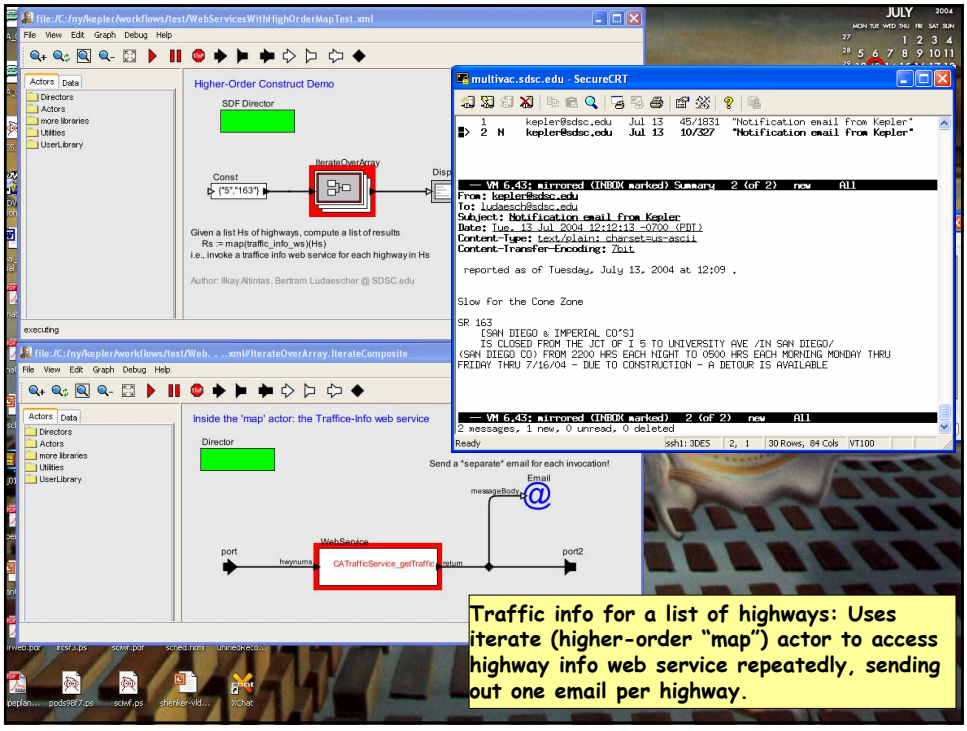
```
zipWithS :: (α → β → γ) → Stream α → Stream β → Stream γ
zipWithS f xs ys  = mapS (\(x,y) -> f x y) (zipS xs ys)

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

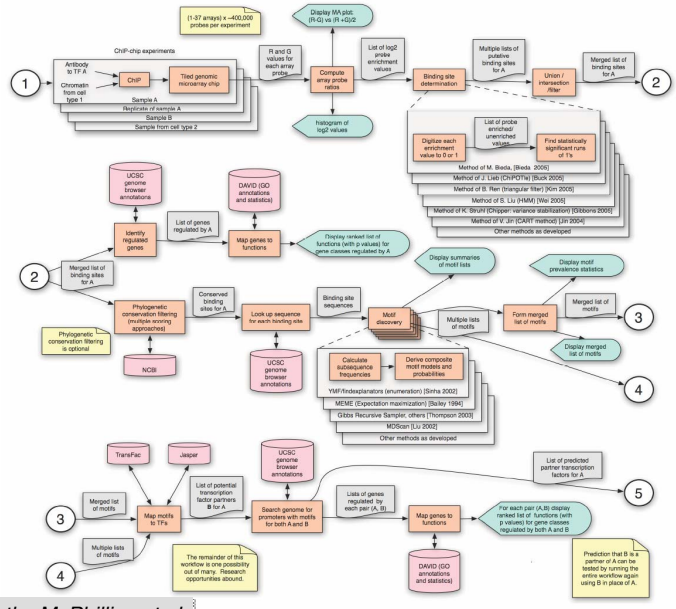
- **Clean functional semantics facilitates *algebraic workflow (program) transformations* (Bird-Meertens); e.g. mapS *f* • mapS *g* ➔ mapS (*f* • *g*)**




Higher-Order Construct Demo
SDF Director
Const
IterateOverArray
Display
Given a list Hs of highways, compute a list of results
Rs := map(traffic_info_ws)(Hs)
i.e., invoke a traffice info web service for each highway in Hs
Author: Ilkay Altintas, Bertram Ludaescher @ SDSC.edu
Inside the 'map' actor: the Traffice-Info web service
Director
Send a "separate" email for each invocation!
Email
WebService
port
port2
Traffic info for a list of highways: Uses iterate (higher-order "map") actor to access highway info web service repeatedly, sending out one email per highway.


multivac.sdsc.edu - SecureCRT
Higher-Order Construct Demo
SDF Director
Const
IterateOverArray
Given a list Hs of highways, compute a list of results
Rs := map(traffic_info_ws)(Hs)
i.e., invoke a traffice info web service for each highway in Hs
Author: Ilkay Altintas, Bertram Ludaescher @ SDSC.edu
Inside the 'map' actor: the Traffice-Info web service
Director
Send a "separate" email for each invocation!
Email
WebService
port
port2
Traffic info for a list of highways: Uses iterate (higher-order "map") actor to access highway info web service repeatedly, sending out one email per highway.

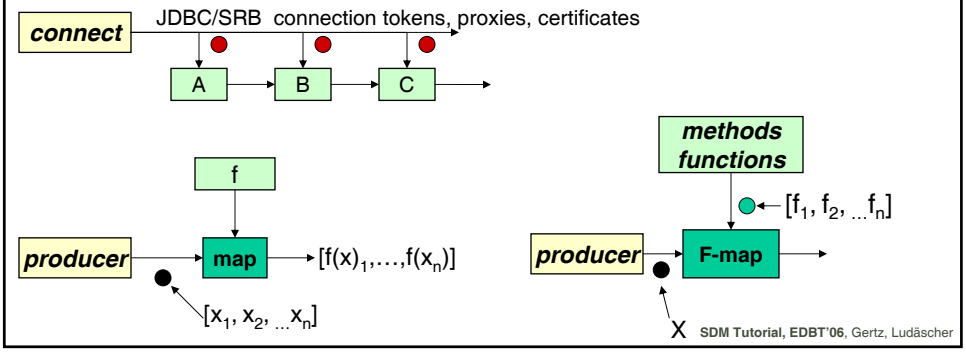
A Bioinformatics Workflow



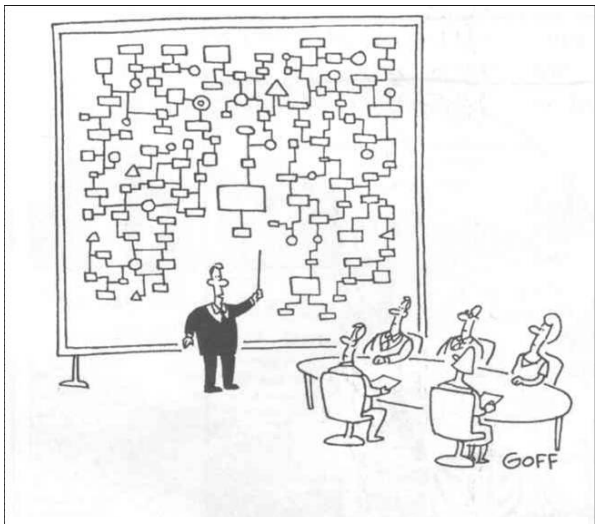
Src: *Timothy McPhillips et al.*

Towards 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/HPC/HTC-WFs)**
- **Loops, higher-order functions (map, foldr, …)**
  - cf. Taverna's automatic loop insertion based on data types

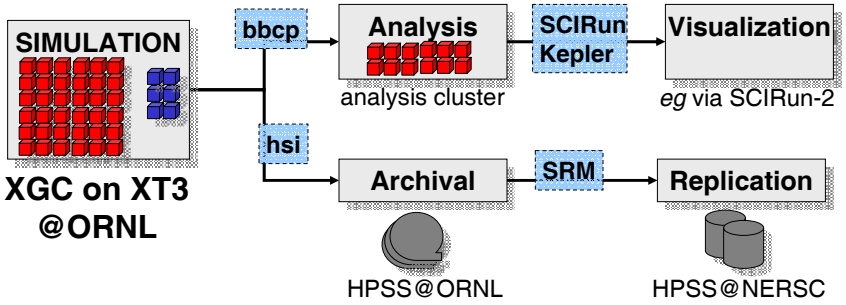


Scientific Workflow Design

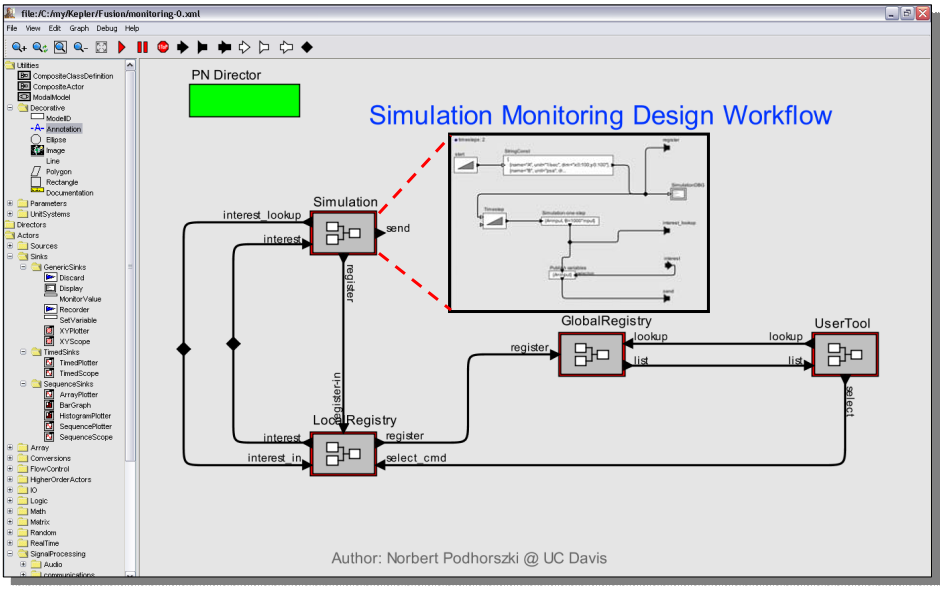


"And that's why our scientific workflows are much easier to understand and maintain!"

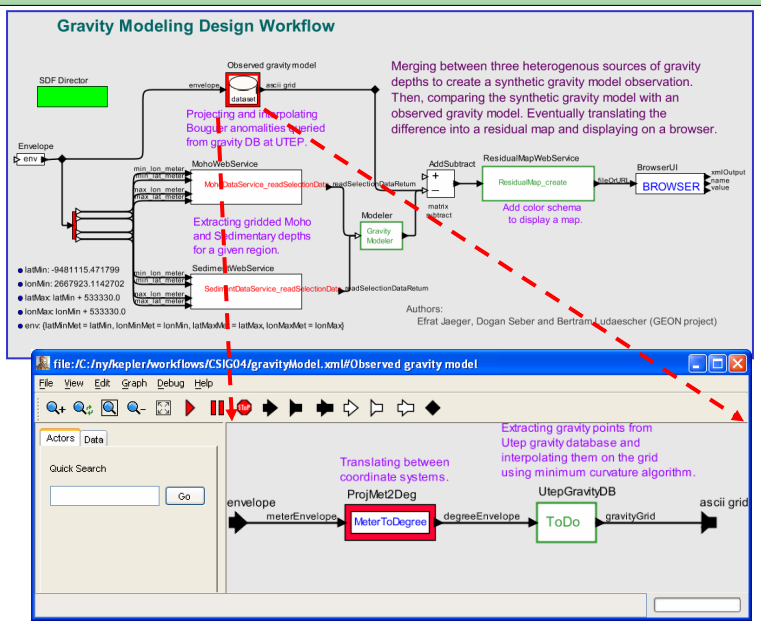
Simulation Monitoring WF: Design V0



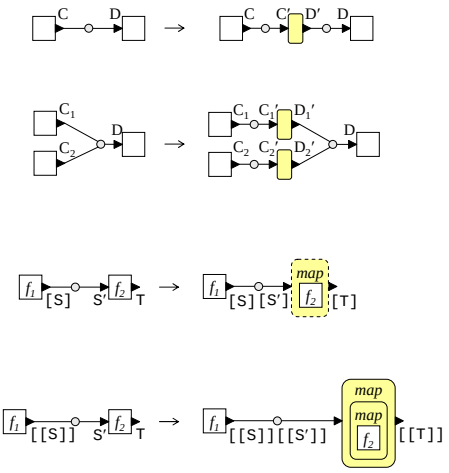
WF Design, Simulation, Prototyping …



Blurring Design (ToDo) and Execution



WF-Design: Adapters for Semantic & Structural "Gaps"



Adapters may:

- be abstract (no impl.)
- be concrete
- bridge a semantic gap
- fix a structural mismatch
- be generated automatically
  (e.g., Taverna's "list mismatch")
- be reused components
  (based on signatures)

Source: [Bowers-Ludaescher, ER'05]

Additional Design Primitives for Semantic Types

Extended Transformations	*Starting Workflow*	*Resulting Workflow*	*Resulting Workflow*
t_9: Actor Semantic Type Refinement ($T' \sqsubseteq T$)	T	T'	
t_{10}: Port Semantic Type Refinement ($C' \sqsubseteq C$, $D' \sqsubseteq D$)	C D	C′ D	C D′
t_{11}: Annotation Constraint Refinement ($\alpha' \rightarrow \alpha$)	α_1 C D α_2 s t	α'_1 C D α_2 s t	α_1 C D α'_2 s t
t_{12}: I/O Constraint Strengthening ($\psi \rightarrow \varphi$)	φ	ψ	
t_{13}: Data Connection Refinement			
t_{14}: Adapter Insertion			
t_{15}: Actor Replacement	f	f'	
t_{16}: Workflow Combination (Map)	f_1 f_2	f_1 f_2	

Source: [Bowers-Ludaescher, ER'05]

Scientific Workflow Design

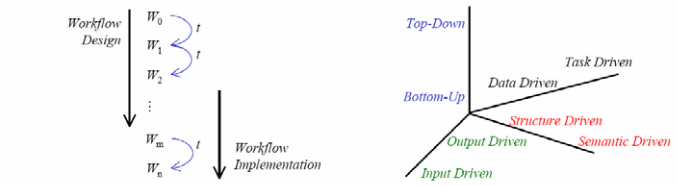
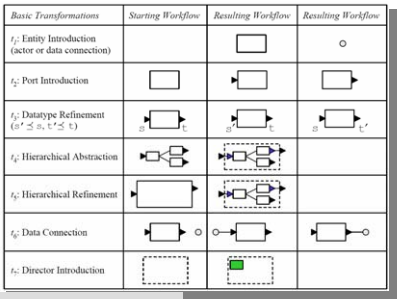


Fig. 2. Workflow engineers evolve workflows by applying design primtives (left), shown as transformations t; and certain primitives can be grouped to form design strategies (right), where each design strategy is shown as a distinct dimension of a design space.

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

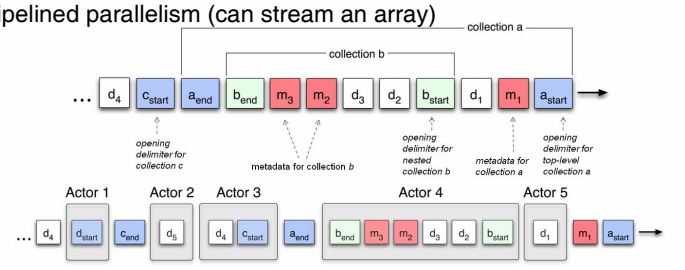
  ➔Separation of concerns:
  - structure, semantics, WF orchestration, etc.



Source: [Bowers-Ludaescher, ER'05]

Collection-Oriented SWF Modeling & Design

- **Assembly line metaphor + Signal Processing + XML + …**
  - Streams are nested collections (≈ XML)
  - Stream data schema is "registered" to a WF data model (really need this)
  - use ideas from "punctuated streams" to delineate collections
- **Advantages:**
  - Less "messy" WFs (more linear, less branching)
  - "Add-only" mode (inject new derived information); *augmentation instead of transformation*
  - Tagging data for downstream processing (instead of "bombing", pass on "dirty" / faulty / strange data with a relevant tag
  - Pipelined parallelism (can stream an array)



Fault Tolerance & Maintenance Challenges

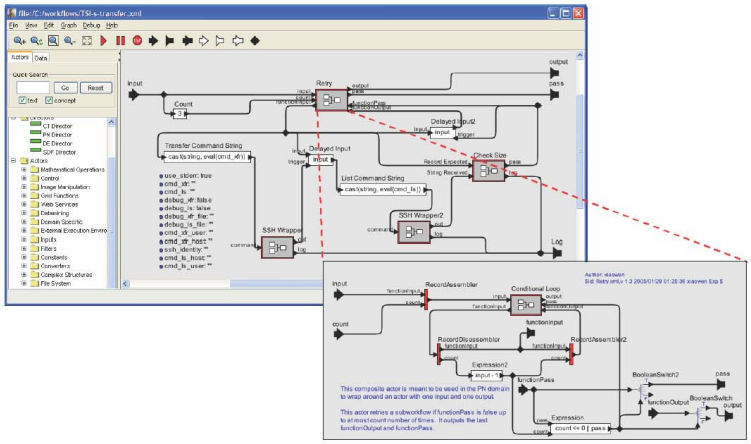


Figure 1: Control-flow intensive astrophysics workflow in KEPLER [21]. "Retry", a composite actor for fault-tolerant data transfer (top), contains a subworkflow (bottom), which itself contains a "ConditionalLoop" subworkflow (inside not shown). Complex feedback loops and the use of boolean switches illustrate the complexity of modeling control-flow directly in a dataflow process network.

Workflow Templates and Patterns

New Ingredients

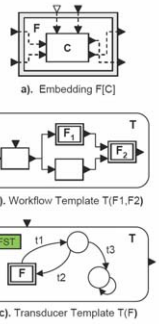


Figure 2: a) Embedding of component C in frame F; b) workflow template $T(F_1,F_2)$; c) finite state transducer template T(F).

work w/ Anne Ngu, Shawn Bowers, Terence Critchlow

Proposed Layered Architecture

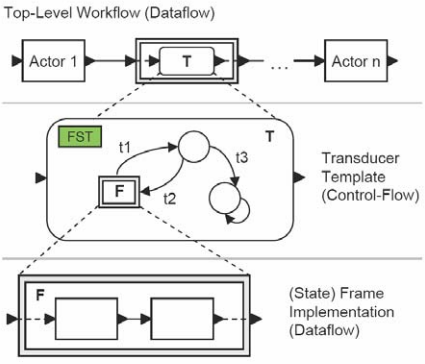


Figure 3: A pattern for modeling generic control-flow comoponents that consists of an outer frame (top), a nested transducer template (middle), and state-frame embeddings (bottom)

Use of Semantics in SWF…

"Smart" Search

- Concept-based, e.g., "find all datasets containing biomass measurements"

Improved Linking, Merging, Integration

- Establishing links between data ***through*** semantic annotations & ontologies
- Combining heterogeneous sources based on annotations
- Concatenate, Union (merge), Join, etc.

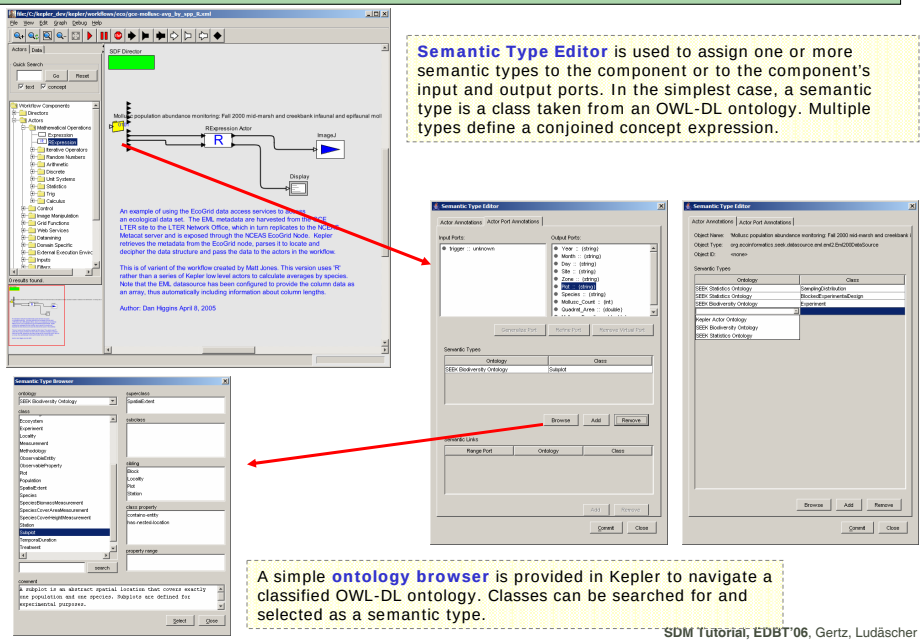
Transforming

- Construct mappings from schema S1 to S2 based on annotations

Semantic Propagation

- "Pushing" semantic annotations through transformations/queries

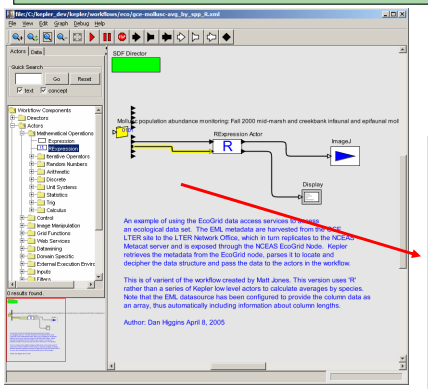
Typing Workflow Components



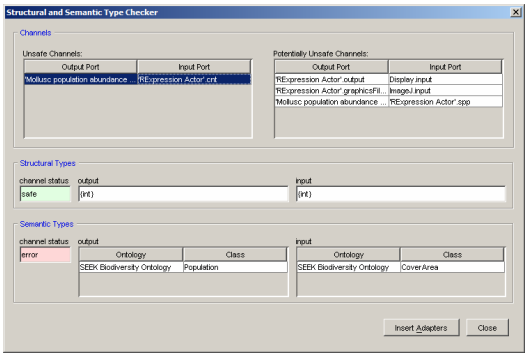
Semantic Type Editor is used to assign one or more semantic types to the component or to the component's input and output ports. In the simplest case, a semantic type is a class taken from an OWL-DL ontology. Multiple types define a conjoined concept expression.

A simple **ontology browser** is provided in Kepler to navigate a classified OWL-DL ontology. Classes can be searched for and selected as a semantic type.

Checking Type Constraints



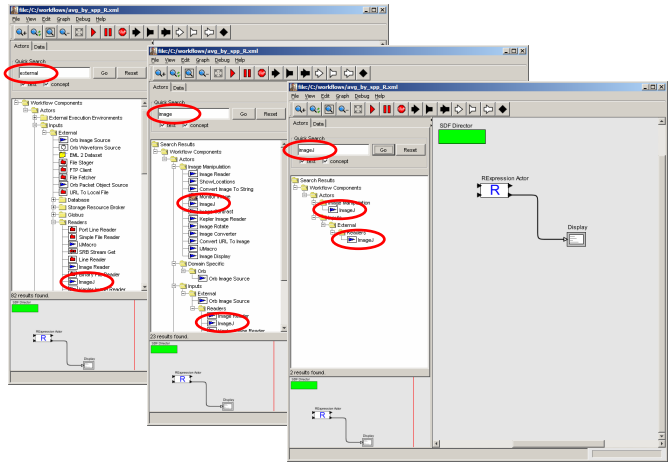
Kepler can **statically perform semantic and structural type checking** of connections. A type checker allows the user to see potentially mismatched port connections as well as known type conflicts before workflow execution.



The user can **navigate** the **unsafe and potentially unsafe** channels using the Kepler Type Checker dialog. When a channel is selected: (a) it is highlighted on the canvas, (b) the structural type and status is shown (here, the channel is structurally well typed), and (c) the semantic type and status is shown (here, the connection produce a semantic type error).

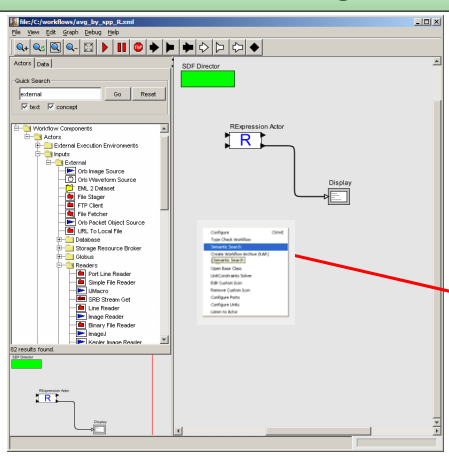
Kepler Actor-Library

- **Ontology-based actor organization / browsing**
- **Customizable libraries based on ontologies**
- **Text search with concept-based expansion**

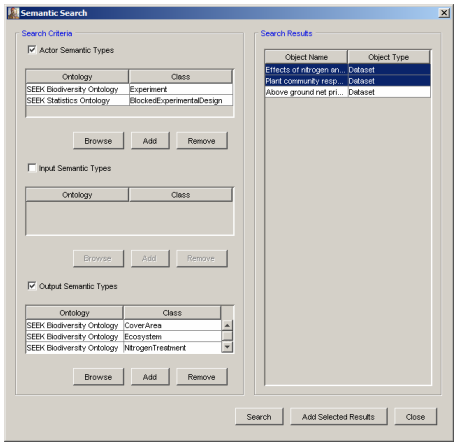


Users can discover ImageJ using various search terms. Here, ImageJ shows up in multiple tree locations based on its given annotations. The library search permits text-based matching against the component's metadata (its given name and certain properties), expanded with concept matches.

Semantic Searching

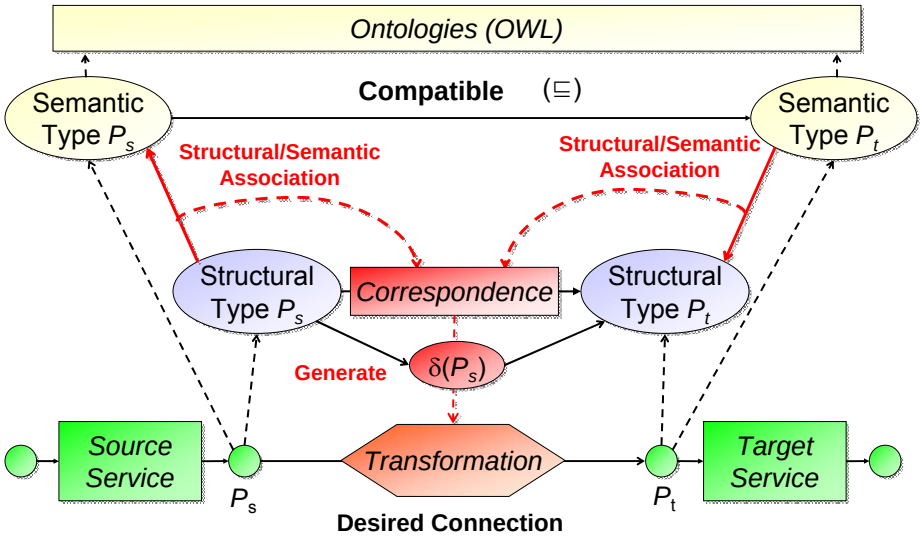


Kepler provides a more advanced ontology-based search mechanism. Users can start the **Semantic Search** dialog, where components can be search for based on their semantic types.



The Semantic Search dialog allows a user to search components by any combination of actor, input, and output semantic types.

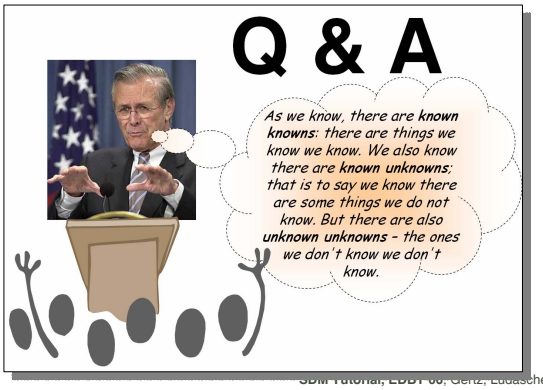
Ontology-Guided Data Transformation



Source: [Bowers-Ludaescher, DILS'04]

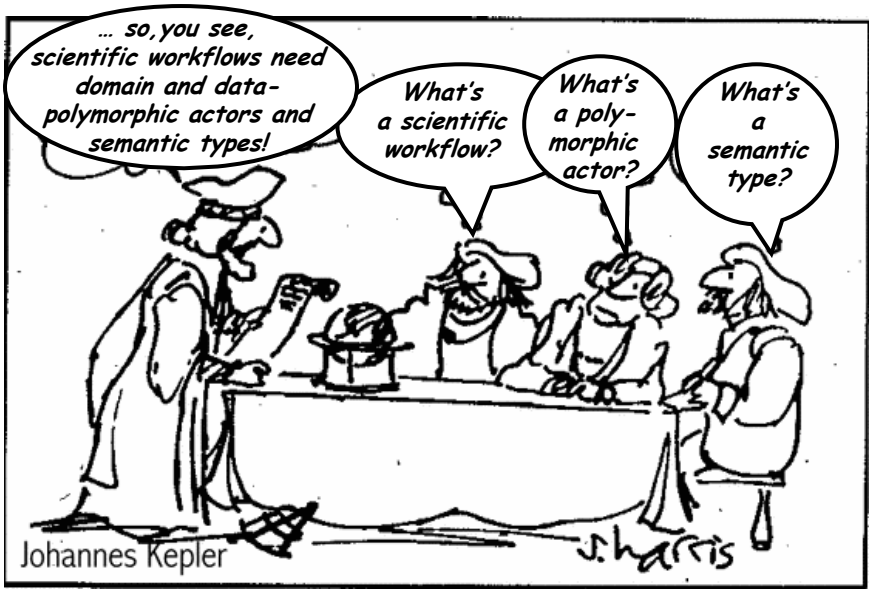
Summary: Scientific Workflows

- **Scientific Workflows in e-Science and CI**
- **SWF vs Business Workflows**
- **Features of a SWF System (Kepler)**
- **Flow-based Programming and Scientific Workflow Design**
- **Semantic Extensions**



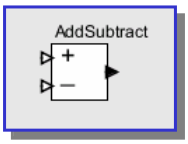
Back then … KEPLER… was ahead of his time …

… but such is life … ;-)



Polymorphic Actors: 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 in a thread that blocks when reading empty inputs
  - In ***CSP***, 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

*hey, Ptolemy has been out for long!

By not choosing among these when defining the component, we get a huge increment in component re-usability. But how do we ensure that the component will work in all these circumstances?

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