Hybrid-Type Extensions for Actor-Oriented Modeling
(a.k.a. Semantic Data-types for Kepler)

Shawn Bowers & Bertram Ludäscher
University of California, Davis
Genome Center & CS Dept.
May, 2005

SEEK

Kepler

Outline

1. Hybrid Types
2. Hybrid Types and Scientific Workflow Design
3. Super Rapid Prototyping: The “Sparrow Family of Languages”
4. Next Steps: Adding a Hybrid-Type System to Kepler

Hybrid Types

Hybrid Types: Superimposing Semantics

Separation of Concerns:

Conventional Data Modeling (*Structural Data Types*)

- E.g., XML Schema / DTD, etc.

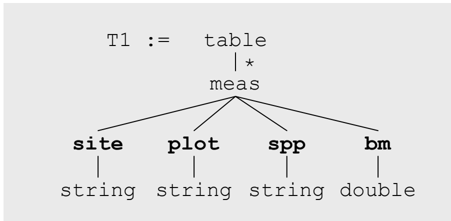
Conceptual Data Modeling (*Semantic Types*)

- Drawn from ontologies (expressed in Description Logic)
- Capturing domain knowledge (e.g., biodiversity, ecology)

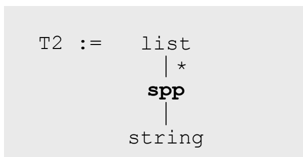
Explicit (external) linkages (*Semantic Annotations*)

- Simple links (one concept per item)
- Links expressed as constraints (logical mappings)

Hybrid Types: Superimposing Semantics



a relational table of measurements



a list of strings

Datatypes

Hybrid Types: Superimposing Semantics

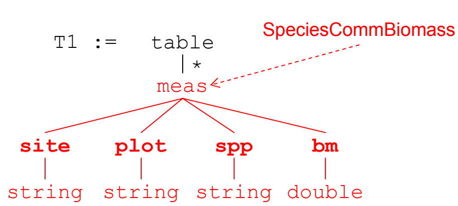
SpeciesBiomass ⊑ Measurement ⊓ ∃item.Species ⊓ ∃prop.Biomass ⊓ ∃loc.Location

"Species biomass is a measure of the amount of biomass of a particular species within a location."

SpeciesCommBiomass ⊑ SpeciesBiomass ⊓ ∃loc.Community

"Species community biomass is a species biomass within a community."

Semtypes



`table/X:meas => X:SpeciesCommBiomass`

each table/meas instance is a measurement

Hybrid Types: Superimposing Semantics

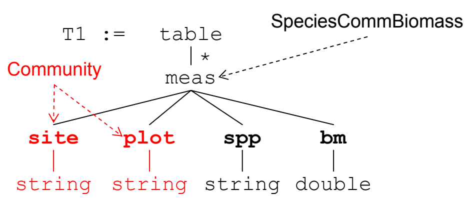
SpeciesBiomass ⊑ Measurement ⊓ ∃item.Species ⊓
∃prop.Biomass ⊓ ∃loc.Location

"Species biomass is a measure of the amount of biomass of a particular species within a location."

SpeciesCommBiomass ⊑ SpeciesBiomass ⊓ ∃loc.Community

"Species community biomass is a species biomass within a community."

Semtypes




```
table/X:meas, X/Y:site, X/Z:plot =>
   X:SpeciesCommBiomass, C=f(Y,Z),
   (X,C):loc, C:Community,
```

Hybrid Types: Superimposing Semantics

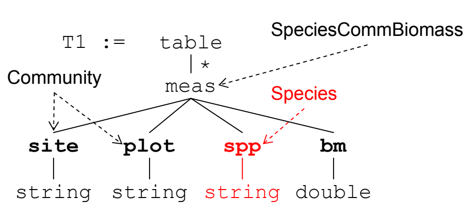
SpeciesBiomass ⊑ Measurement ⊓ ∃item.Species ⊓
∃prop.Biomass ⊓ ∃loc.Location

"Species biomass is a measure of the amount of biomass of a particular species within a location."

SpeciesCommBiomass ⊑ SpeciesBiomass ⊓ ∃loc.Community

"Species community biomass is a species biomass within a community."

Semtypes




```
table/X:meas, X/Y:site, X/Z:plot,
X/U:spp =>
   X:SpeciesCommBiomass, C=f(Y,Z),
   (X,C):loc, C:Community,
   (X,U):item, U:Species,
```

Hybrid Types: Superimposing Semantics

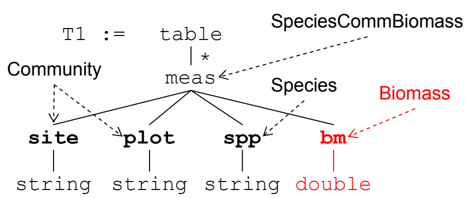
SpeciesBiomass $\sqsubseteq$ Measurement $\sqcap$ $\exists$item.Species $\sqcap$
$\exists$prop.Biomass $\sqcap$ $\exists$loc.Location

"Species biomass is a measure of the amount of biomass of a particular species within a location."

SpeciesCommBiomass $\sqsubseteq$ SpeciesBiomass $\sqcap$ $\exists$loc.Community

"Species community biomass is a species biomass within a community."

Semtypes




```
table/X:meas, X/Y:site, X/Z:plot,
X/U:spp, X/B:bm =>
   X:SpeciesCommBiomass, C=f(Y,Z),
   (X,C):loc, C:Community,
   (X,U):item, U:Species,
   (X,B):prop, B:Biomass.
```

Hybrid Types: Superimposing Semantics

Searching

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

Merging/Integrating

- 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

Semantic Annotation Propagation

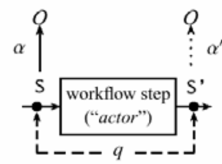
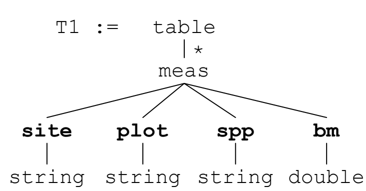


Figure 1. A workflow step whose semantic annotation α has been propagated via q to a new semantic annotation α'.

Capture I/O constraints

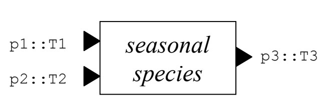
- Similar to unit type constraints
- Can enable automated metadata creation (annotation "propagation")
- Can help refine ontologies and existing annotations

Semantic Annotation Propagation



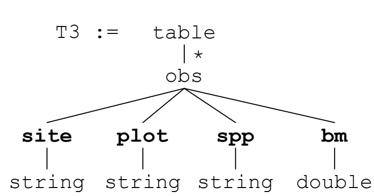
Port 1 Annotation

```
table/X:meas, X/Y:site, X/Z:plot,
X/U:spp, X/B:bm =>
   X:SpeciesCommBiomass, C=f(Y,Z),
   (X,C):loc, C:Community,
   (X,U):item, U:Species,
   (X,B):prop, B:Biomass.
```



Actor I/O constraint (approx.)

```
p3.obs(site, plot, spp, bm) :-
   p1.meas(site, plot, spp, bm),
   p2.spp(spp).
```



Port 2 "Chased" Annotation

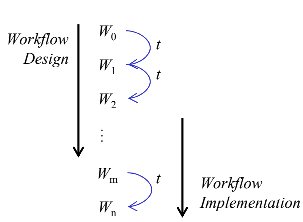
```
table/X:obs, X/Y:site, X/Z:plot,
X/U:spp, X/B:bm =>
   X:SpeciesCommBiomass, C=f(Y,Z),
   (X,C):loc, C:Community,
   (X,U):item, U:Species,
   (X,B):prop, B:Biomass.
```

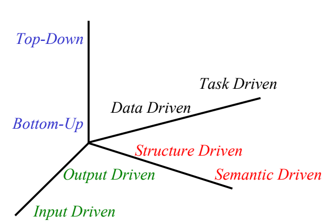
Hybrid Types and Scientific Workflow Design

Workflow Design Primitives

End-to-End Workflow Design and Implementation

- Viewed as a series of primitive "transformations"
- Each takes a WF and produces a new WF
- Can be combined to form design "strategies"





Workflow Design Primitives

Semantic types and Actor Oriented Modeling:

- Actors and Workflows: can have semantic types conceptually describing their “function”
- Ports: can have semantic types conceptually describing what they consume and produce
- I/O Constraints: a general form of constraint between input and output (e.g., like unit constraints) … approximating the function of an actor

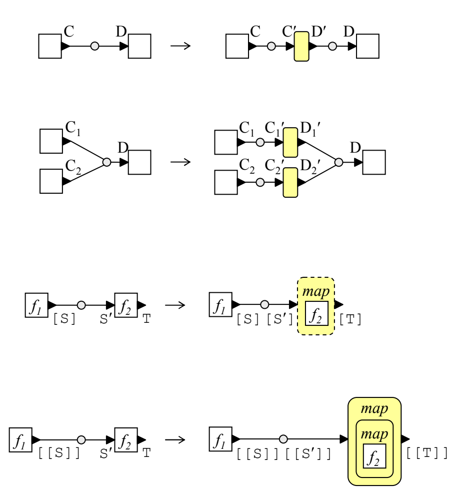
Basic Design Primitives Inherited from Ptolemy

Basic Transformations	*Starting Workflow*	*Resulting Workflow*	*Resulting Workflow*
t_1: Entity Introduction (actor or data connection)			
t_2: Port Introduction			
t_3: Datatype Refinement ($s' \preceq s$, $t' \preceq t$)	s t	s′ t	s t′
t_4: Hierarchical Abstraction			
t_5: Hierarchical Refinement			
t_6: Data Connection			
t_7: Director Introduction			

Additional (Planned) 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 s D t α_2	α'_1 C s D t α_2	α_1 C s D t α'_2
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	

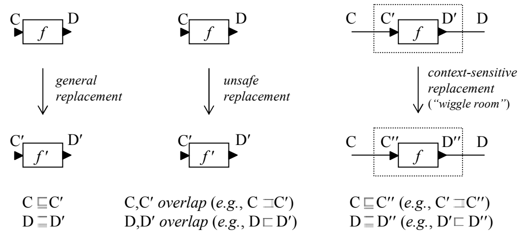
Adapters for Semantic and Structural Incompatibility



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)

Applying the Replacement Primitive



- General replacement doesn’t consider surrounding connections
- Context-sensitive replacement gives more “wiggle room” by “tuning” the actors semtypes based on connections

Workflow Elaboration

Adapter insertion, replacement, and search provide a powerful mechanism for workflow “elaboration”:

1. Given an initial, user specified set of connected “abstract” actors
2. Repeatedly search for replacement “concrete” actors (atomic and composite)
3. At each step, insert adapters when necessary
4. Allow user to select returned workflows to be combined

Super Rapid Prototyping:
The Sparrow “Family of Languages”

The “Sparrow Family of Language”

The Sparrow Family of Languages (v0.0)

Shawn Bowers Bertram Ludäscher

May 1, 2005

1 Introduction

The Sparrow family of languages provide a simple and uniform syntax for defining ontology-aware scientific workflows. The languages are meant to be both convenient and easy to use, and in particular, human and machine readable. This document serves as a manual for Sparrow and also discusses the Sparrow-Toolkit, a rapid prototyping system for designing scientific workflows using the languages of Sparrow.

The Sparrow languages adopt Ptolemy II's actor-oriented modeling paradigm and extend it by providing rich conceptual modeling primitives for enabling scientific workflow design and development. The following list briefly summarizes the Sparrow family of languages.

- Sparrow-DL was the first language of the Sparrow family. It provides a lightweight variant of OWL-DL, the interchange format for semantic web ontologies.
- Sparrow-DTD is a simple variant of XML schema for describing the structural (input/output) datatypes of actors. Note that we view data sets as special cases of "source" actors in Sparrow.
- Sparrow-Annotate is a language for annotating actors with semantic information. The language provides the means for various parts of actors (ports, datatypes, etc.) to be semantically typed, i.e., linked with ontologies.
- Sparrow-SWF is a simple variant of Ptolemy II's MoML language for describing actors and workflows (note that SWF stands for "Scientific Workflow").
- Sparrow-Rule is an extension to Sparrow-Annotate for describing additional logic-based rules and constraints over actors.

Sections [illegible] through [illegible] define each of the languages in more detail. Section [illegible] describes the functions available in Sparrow-Toolkit.

1

Basic Idea: Have both Machine *and* Human readable syntax

Sparrow-DL

Description logic

Sparrow-DTD

Datatypes, variant of XML DTDs

Sparrow-Annotate

Configuring concepts; linking datatypes and ontologies

Sparrow-SWF

KSW-Based MoML Metadata

Sparrow-Rule

Fancy stuff, like type constraints (a la unit types), function approximation, and misc. other constraints

The "Sparrow Family of Languages"

Sparrow-DTD

```
datatype d1 := 'list' elem * ['string' elem string].   % list of stri
datatype d2 := 'string' elem string.                   % string
datatype d3 := 'int' elem int.                         % int
datatype d4 := 'AMSQuote' elem ['price' elem double, 'cond' elem stri
datatype d5 := 'seller' elem ['id' elem string, 'rating' elem int, 'a
```

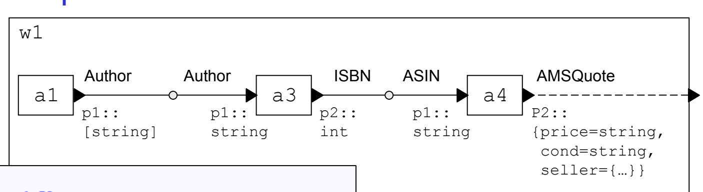
Sparrow-SWF

```
% a simple datasource: -> Author
actor a1 := [
  name 'authors',
  concrete true,
  port [ name 'p1', direction 'out', datatype d1, semtype s1 ],
  semtype s7
].

% a concrete actor: ISBN -> ASIN
actor a2 := [
  name 'isbn to asin',
  concrete true,
  port [ name 'p1', direction 'in', datatype d3, semtype s5 ],
  port [ name 'p2', direction 'out', datatype d2, semtype s6 ],
  semtype s9
].
```

Sparrow-DL

```
concept 'Book' isa 'has-author' only 'Author' and
                   'has-author' some 'Author' and
                   'has-book-id' only 'BookID' and
                   'has-book-id' exactly 1 and
                   'has-publisher' only 'Publisher' and
                   'has-publisher' exactly 1.


concept 'USDollarAmount'.

concept 'Author' isa 'Person'.

concept 'BookVendor' isa 'Vendor' and 'sells' some 'Book'.

concept 'AmazonMarketplaceSeller' isa 'BookVendor'.

concept 'ASIN' isa 'BookID'.

concept 'ISBN' isa 'BookID'.

concept 'Quote' eqv 'has-vendor' some 'BookVendor' and
                    'has-price' some 'USDollarAmount'.
```

Sparrow-Toolkit Operations




```
% a badly formed workflow:
workflow w1 := [
  actors [auth ref a1, book ref a3, vend ref a4],
  connection [auth:'p1', book:'p1'],
  connection [book:'p2', vend:'p1'],
  signature [vend:'p2'],
  semtype s7
].
```

Sparrow-Toolkit (example) Operations

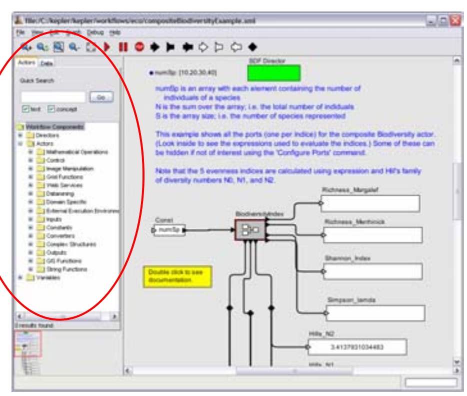
- Is w1 semantically and/or structurally well typed?
- What can be semantically connected to a3?
- Insert "abstract" adapter between a3 and a4
- What can replace (e.g., implement) the adapter?
- …

Future Steps: Adding a Hybrid-Type System to Kepler

Current and Future Implementation

Concept-based Actor Search

- Implemented as proof-of-concept
  - About to undergo major revision
  - Additional operations slated for next Kepler Release (data search, actor-based port search, etc.)



Biggest Challenges

- Building/searching a repository …
- Making changes to MoML (see KSW)
- GUI changes
- Ontology management