A Platform for Exchanging Bioinformatics Data

Todd J. Green, Grigoris Karvounarakis, Nicholas E. Taylor, Zachary G. Ives, Val Tannen, Olivier Biton, Sarah Cohen-Boulakia

University of Pennsylvania

http://www.cis.upenn.edu/~zives/orchestra/

Funded by NSF IIS-0477972, 0513778, 0629846

pPOD Investigators Meeting
September 12, 2007
A Problem Familiar to All of You

Biological knowledge is fragmented across many databases

By organism:
  Spiders, beetles, fungi, etc.

By focus:
  Protein, gene, disease, phylogenetics matrix, etc.

By group:
  GenBank vs. EMBL, Morphbank vs. MorphoBank

By curation level:
  SWISS-PROT vs. PIR

Putting it together is the key to scientific breakthroughs

- Cross-organism comparisons (e.g. to determine common ancestry); systems-level biology; etc.
- But the databases are fundamentally different, in design, terminology, etc.

How do we get the data together?
Approach 1: Standards and Data Warehouses

Build a grand unified data model and clearinghouse for all data

- Worth aspiring to for certain data – standards help immensely!
- Probably the predominant approach in bioinformatics

Standards help but don’t fully solve problems of data exchange and integration

- Needs change, science changes – need new versions of the standard (hence no one standard)!
  - And standardizing across standards?
- Standards evolve slowly – not well-suited for cutting-edge science
- Some have different needs for their data – want a different schema

* Schema is sometimes called “data model”

Even if we have a standard schema, data may be of different levels of quality – how do we agree on the “standard” version of the data?
Approach II: Exchange Among Cooperating Sites

Everyone keeps their database, and uses point-to-point translators between them
e.g., based on Web services, FTP, export files, etc.

Much more flexible – each site can control its own schema and do its own data curation

But also poses new challenges:

- Requires significant expertise at each site, to write the translators (often in both directions)
- Translators seldom work on incremental changes: “refreshes” are a major task
- Conflicting data; violated constraints; unexpected dependencies
- No tracking of changes or where data originated (provenance)
The Orchestra System: A Platform for Data Sharing Among Collaborators

Assumes a set of cooperating sites (peers), each of whom wants to incorporate data from the others.

Allows each site to, according to its own policy:

- **Import** data from trusted data sources
  - original database instance OR a set of recent updates
- **Curate** data
- Use data to run (local) algorithms and **analysis**
- **Track** data origin / **provenance** (e.g., original database)

How?

- **Schema mappings** describe data transformations concisely and using constraints (instead of complex code)
- **Trust policies** describe site policies about data quality
  - Administrator can replace data imported from elsewhere
**Orchestra: Collaborative Data Sharing**

**Collaborating peers** in a logical network

No one DB is the final authority – each peer may reject or override data from elsewhere.

- PlasmoDB
- NCBI Taxonomy
- NCBI RefSeq
- EBI InterPro
- GO terms
Orchestra: Collaborative Data Sharing

Collaborating peers in a logical network
No one DB is the final authority – each peer may reject or override data from elsewhere

Schema mappings specify how to share data
Data and updates are translated across mappings

M0: RefSeq imports scientific names from the NCBI taxonomy
M1: PlasmoDB imports the entire NCBI taxonomy (all attributes)
M2: PlasmoDB maintains a copy of the RefSeq catalog
M3: PlasmoDB needs some sequences which used to be in RefSeq but have been removed
M4: PlasmoDB gets from InterPro references to domains in PFAM & PRODOM
M5: PlasmoDB gets terms from GO
Orchestra: Collaborative Data Sharing

Collaborating peers in a logical network
No one DB is the final authority — each peer may reject or override data from elsewhere

Schema mappings specify how to share data
Data and updates are translated across mappings

Trust policies filter data a site doesn’t want & resolve conflicts

- PLASMODB only accepts data from EBI if it originated from PFAM or PRODOM, but nowhere else
What Is a Schema Mapping?

A constraint specifying that data must exist in one set of tables, given data in another set of tables, e.g.:

RefSeq:Species (taxId, taxName)
← Taxo:Names (taxId, taxName, 'scientific name')

One can think of it as resembling a query

- Just as SQL queries offer benefits over Java code to maintain stored data, mappings offer benefits to data translation
What Can Schema Mappings Do?

Mappings allow us to:

- **Restructure** data – combine or split tables, flatten or add hierarchy
- Perform lookups, **cross-references**, or link traversals
- **Translate** synonyms or IDs
- Create special markers for **unknown** data (analogous to SQL’s NULL, but more powerful)

Mappings can be:

- **Composed** (we don’t need mappings between every pair of sites for them to share data!)
- Inverted (if we want bidirectional data sharing)
- Tested for certain kinds of correctness
- Automatically expanded to add **provenance**
- etc.
An Example: pPOD Schema (Subset)

Consists of Trees with:
Sets of “OTUs”
Provenance with:
Software Matrices
... with their own OTU lists
TreeBase II Schema (Subset)

Consists of Studies with Analyses: Consisting of steps From Software Including Data: Derived from Matrices with TaxonLabels Producing PhyloTrees with TaxonLabels
Mapping from TreeBase to pPOD
Mapping The Schemas

TREE(treeId, treeTitle),
PROVENANCE(provId, treeId, matrixId, soft, params, -,-),
MATRIX(matrixId, matrixTitle),
matrixotu(matrixId, otuId, -),
OTU(otuId, otuLabel),
treeotu(treeId, inOtulId),
OTU(inOtulId, inLabel)
Mapping The Schemas

study(studyId,studyName,studyAcc),
analysis(anId,studyId,anName,_),
ANALYSISSTEP(stepId,anId,softId, dataId,stepNam,_,parms),
SOFTWARE(softId,soft,-,-,-,-),
analyzeddata(dataId,false,-,len),
MATRIX(matrixId,dataId,matrixTitle, -,-,-),
matrixrow(rowId,matrixId,mLabelId),
TAXONLABEL(mLabelId,inLabel),
PHYLOTREE(treeId,dataId,treeTitle, -,-,-),
**A TreeBase-pPOD Schema Mapping**

TREE(treeId, treeTitle),
PROVENANCE(provId, treeId, matrixId, soft, parms, -,-),
MATRIX(matrixId, matrixTitle), matrixotu(matrixId, otuId, -),
OTU(otuId, otuLabel),
treeotu(treeId, inOtuId), OTU(inOtuId, inLabel)
←
study(studyId, studyName, studyAcc),
analysis(anId, studyId, anName, _),
ANALYSISSTEP(stepId, anId, softId, dataId, stepNam, _, parms),
SOFTWARE(softId, soft, -, -, -, -), analyzeddata(dataId, false, -, len),

MATRIX(matrixId, dataId, matrixTitle, -, -, -),
matrixrow(rowId, matrixId, mLabelId), TAXONLABEL(mLabelId, inLabel),
PHYLOTREE(treeId, dataId, treeTitle, -, -, -, -),
treenode(nodeId, treeId, tLabelId, nodeLabel, -, -),
TAXONLABEL(nodeId, otuLabel)
Updates are Logged and Translated across Mappings

Orchestra is a “middleware” layer that, upon request:

Publishes updates to a P2P storage system for archival

Imports others’ published updates $\Delta_*$ to a target
Mapping Provenance

We record:

- *Which* mapping was used to derive a value
- *What* input tuples were processed
- This can be enhanced with versioning on the mappings, time on the original base data, etc.
Trust

Each site can specify a **policy** about what data it is willing to import

- Since data “flows” along any mappings, we might get data from sites we don’t know
- We can have a default policy of not trusting data unless we explicitly say to

Policies filter based on data’s value and provenance:

- PlasmoDB trusts protein data from SWISS-PROT
  ... or even, from SWISS-PROT with a preference over data from GenBank
- PlasmoDB trusts data that has been imported into INTERPRO regardless of where it originated
A common problem with import/exchange tools:

- PlasmoDB imports data from GenBank
- It makes a correction to that data
- The next time it imports, that change is lost!
- ... Or things go awry if GenBank later makes its own correction to the data!

Orchestra tracks how data is changed at each site

If a data item is updated by someone other than the originator, an independent “branch” is created for this data

- Such changes are visible to anyone who imports data from PlasmoDB
- Such changes “stay” even after later updates are made
The Orchestra Engine

Designed to “layer over existing relational DBMSs”

- Your old applications will continue to work
- Tracks changes made through one of two schemes:
  - Do a diff between old and new
  - (Preferred) Use database logging facilities (DB2, Oracle, ...)

Published data is permanently archived elsewhere

When a site imports data:
- Requests data from the archive
- Uses a DBMS or other query processor to translate all of the data using the mappings
- Updates the local DBMS using JDBC
A Brief Demo

Let’s see the Orchestra prototype in two contexts:

Core Data Model and TreeBase schemas
(No data yet, but we can browse)

Genetics/proteomics space
Greg will demonstrate the administrative console, how updates are propagated, and provenance
Biological data sources share their data as peers.

Each peer is able to:

- **Map** its schema to other peers' schema
- **Publish its updates** to the other peers
- **Reconcile** automatically its data with other peers' data through mappings
- **Curate** other peers data when imported
- Define **trust policies** over the related databases

It is easy to change the mappings to adapt to schema changes, new data needs, ...

A much simpler development task.
Ongoing Work on Orchestra

Short-term

Augment the mapping language to deal with:

- User-defined functions (appearing in Perl/Python scripts)
  - Arithmetic, substring extraction, calling BLAST, etc.
- XML data

Longer-term

- Support bidirectional mappings (site A can delete data out of site B’s database)
- Support mappings with learned rankings (which are more trustworthy?)
http://www.cis.upenn.edu/~zives/orchestra/