

Using the Functional Data Model to Store and Query Recursive Biological Data

Graham J.L. Kemp, Selpi and Merja Karjalainen

Department of Computing Science
Chalmers University of Technology
SE-412 96, Göteborg, Sweden

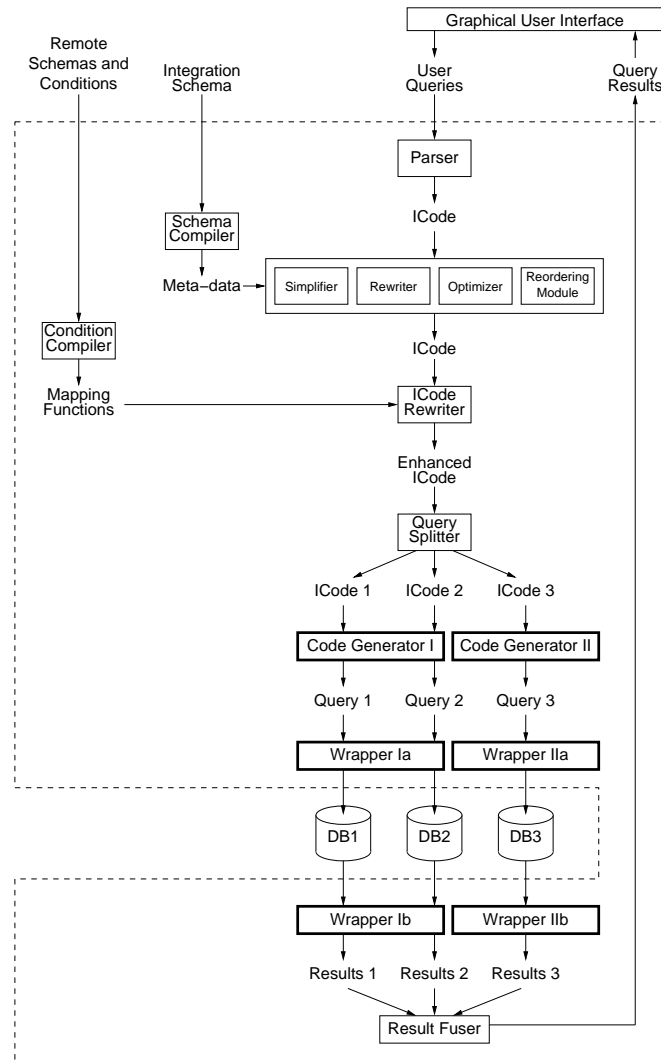
<http://www.cs.chalmers.se/~kemp/>

Workshop on Database Issues in Biological Databases (DBiBD)
January 8-9, 2005
Edinburgh, Scotland

Overview

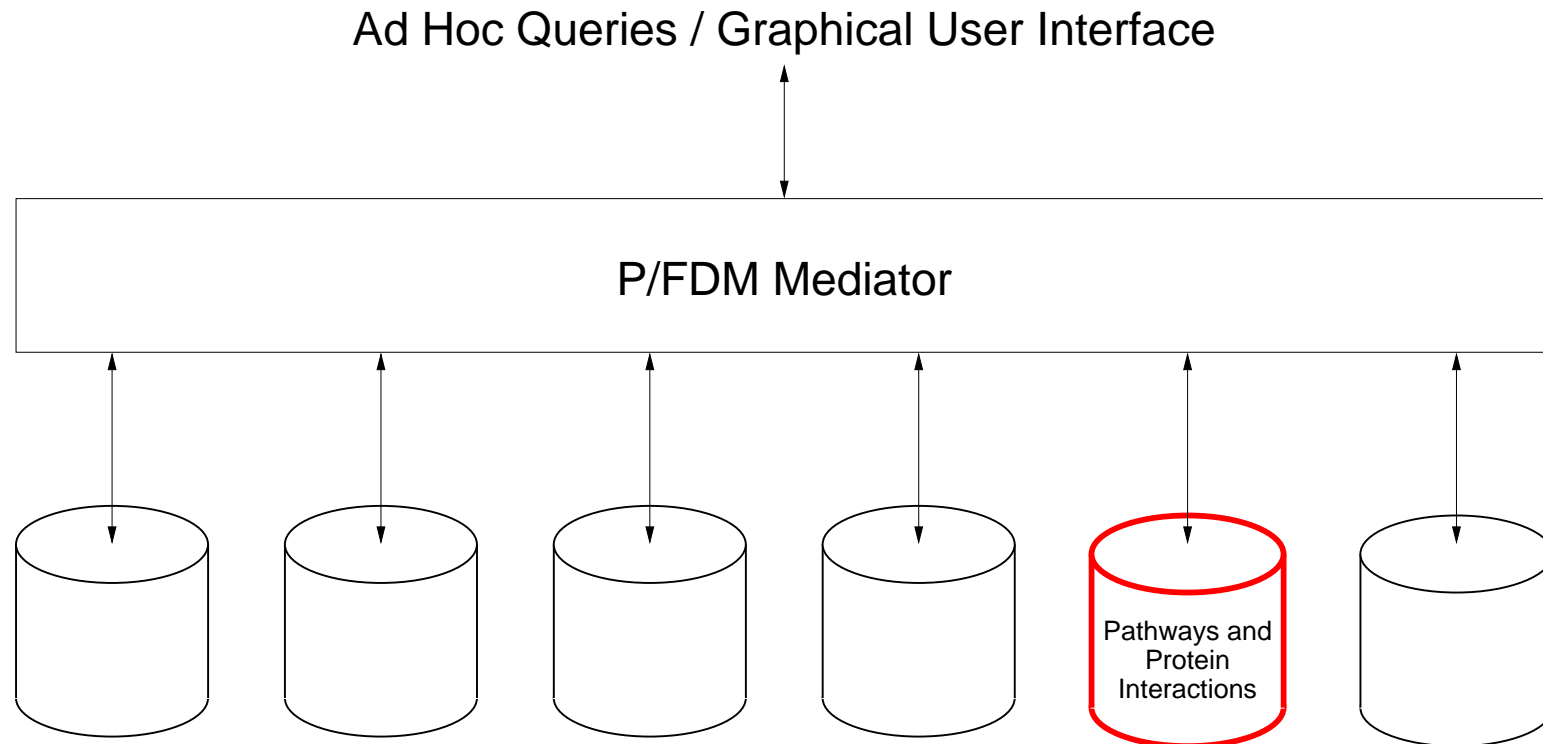
- Pathways and interaction networks
 - project background and aims
 - working with BIND XML files
 - recursive queries
- Taxonomy example
- Gene Ontology example

P/FDM Mediator architecture



Another talk ...

P/FDM Mediator



- determine what DBs are relevant to answering the query
- translate query into language(s) of the underlying DBs
- combine results and present these to the user

Pathways and interaction networks — project aims

- (i) to investigate the structures of several pathway and protein interaction data resources;
- (ii) to design a database schema for pathway and protein interaction data;
- (iii) to build a small prototype pathway and protein interaction database using the P/FDM object database;
- (iv) to investigate the usefulness of proposed data exchange formats for automatically loading data sets into a database;
- (v) to use the prototype database to perform some simple pathway and protein interaction studies.

P/FDM

Object-oriented database implemented mainly in Prolog

Based on the Functional Data Model (FDM)

- entities (classes; hierarchies)
- functions
 - attributes and relationships
 - stored and derived values

Daplex

- data definition language (DDL)
- data manipulation language (DML)

Is XML the answer?

“Many companies report a strong interest in XML. XML however, is so flexible that this is similar to expressing a strong interest in ASCII characters.”

From BizTalk Framework Overview

<http://www.oasis-open.org/cover/BiztalkFrameworkOverviewFinal.html>

Biological pathway and interaction data sources

BIND

- a data specification for information about biomolecular interactions, complexes and pathways;
- initially defined in ASN.1;
- specification transformed systematically into XML DTD;
- data files transformed systematically into XML.

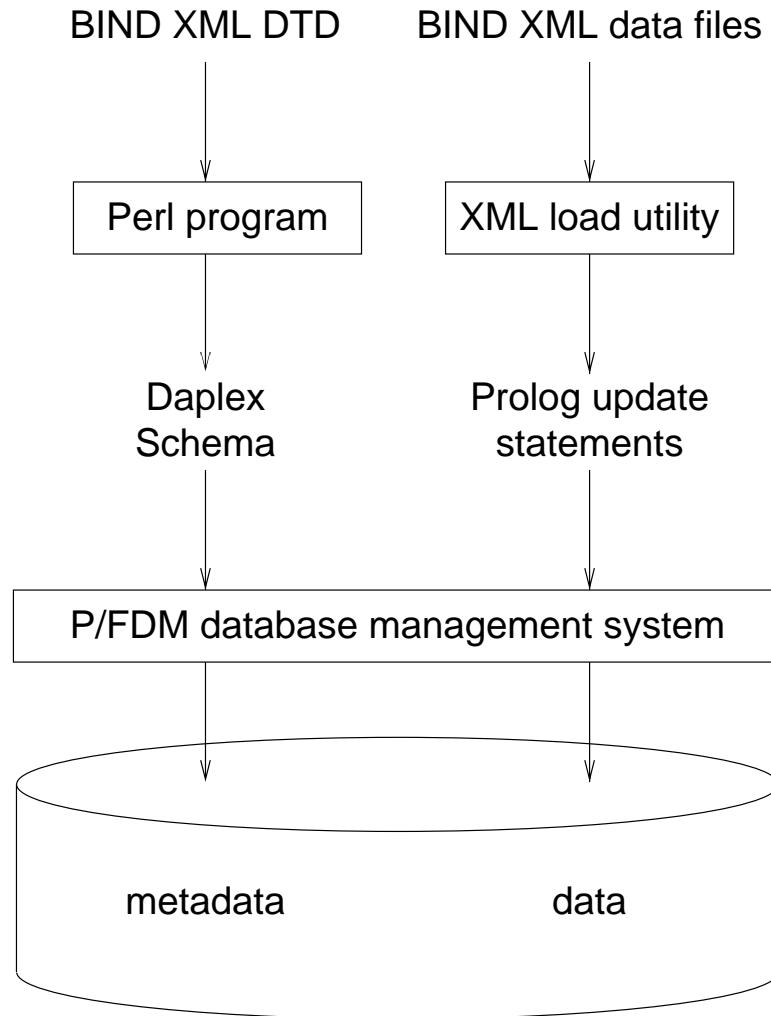
KEGG

- data available in KGML (KEGG Markup Language) format and XML;
- focus is on the graphical presentation of pathway diagrams and the XML tags include layout and presentation elements.

MIPS Comprehensive Yeast Genome Database ... no XML

BioPAX ... no data

Implementation



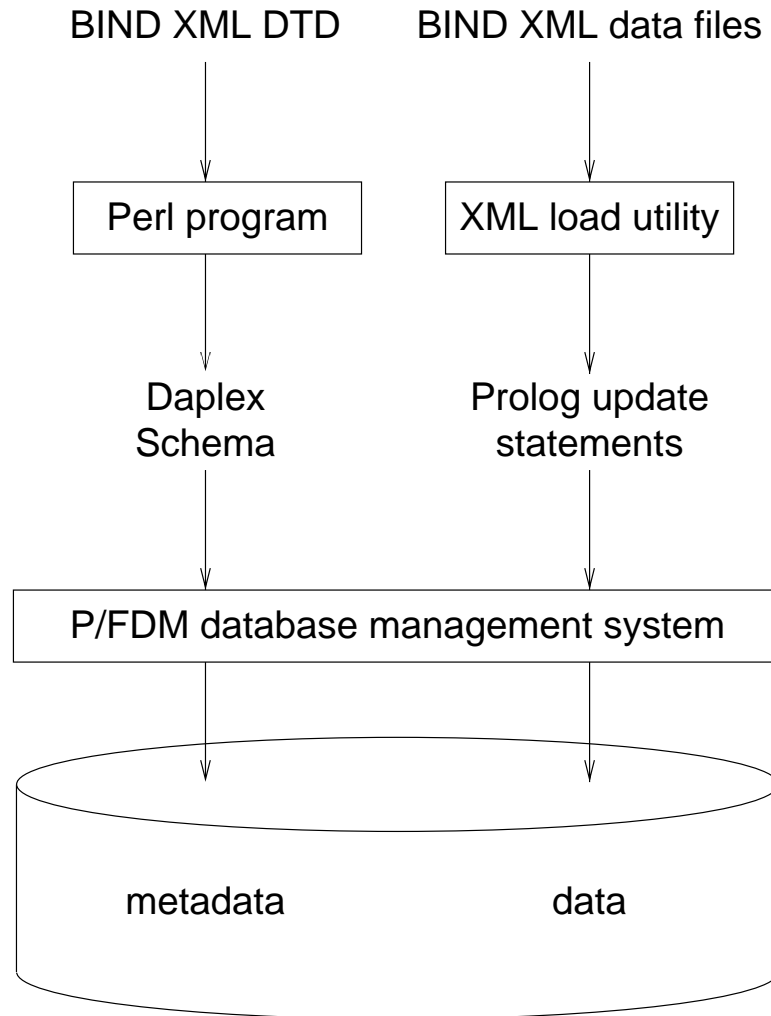
Generating a Daplex schema from BIND's DTD

<pre><!ELEMENT BIND-Interaction (BIND-Interaction_iid , BIND-Interaction_a , BIND-Interaction_b , BIND-Interaction_priv?)> <!ELEMENT BIND-Interaction_iid (Interaction-id)> <!ELEMENT BIND-Interaction_a (BIND-object)> <!ELEMENT BIND-Interaction_b (BIND-object)> <!ELEMENT BIND-Interaction_priv %BOOLEAN; > <!ATTLIST BIND-Interaction_priv value (true false) "false"> <!ELEMENT Interaction-id (%INTEGER;)> <!ELEMENT BIND-object (BIND-object_short-label , BIND-object_descr? , BIND-object_user-id?)> <!ELEMENT BIND-object_short-label (#PCDATA)> <!ELEMENT BIND-object_descr (#PCDATA)> <!ELEMENT BIND-object_user-id (%INTEGER;)></pre>	<pre>declare BIND_Interaction ->> entity declare BIND_object ->> entity declare iid(BIND_Interaction) -> integer declare a(BIND_Interaction) -> BIND_object declare b(BIND_Interaction) -> BIND_object declare priv(BIND_Interaction) -> boolean declare short_label(BIND_object) -> string declare descr(BIND_object) -> string declare user_id(BIND_object) -> integer key_of BIND_Interaction is iid key_of BIND_object is short_label, descr</pre>
---	--

“Guessing” keys

- union of all non-optional attributes and relationships defined on that class in the DTD;
- in some case, this produces a super-key
 - have to remove attributes/relationships to form a candidate key;
- in some case, insufficient to uniquely identify instances
 - have to add attributes/relationships to form a candidate key;
- fewer than 10 “guessed” keys had to be altered by hand.

Implementation



XML load utility

- XML document is parsed and compiled into a nested Prolog term structure;
- recursively traverse tree to find data items to load, guided by metadata;
- values for key attribute(s) are found first, and a new entity instance is created;
- values for any other attributes and relationships are added;
- relationship functions:
 - if the related instance already exists
then the function value can be added immediately
else the related entity instance must be created first.

BIND XML data and Prolog update goals

```
<BIND_Interaction>
  <BIND_Interaction-iid>
    <Interaction_id>118</Interaction_id>
  </BIND_Interaction-iid>
  <BIND_Interaction-a>
    <BIND_object>
      <BIND_object-short_label>EGF_EGFR complex</BIND_object-short_label>
      <BIND_object-descr>Epidermal Growth Factor (EGF) bound to Epidermal
      Growth Factor Receptor (EGFR)</BIND_object-descr>
      <BIND_object-user_id>0</BIND_object-user_id>
    </BIND_object>
  </BIND_Interaction-a>
  <BIND_Interaction-priv value=""false""/>
</BIND_Interaction>
```

```
newentity('BIND_Interaction', [118], 'BIND_Interaction'(10)),
newentity('BIND_object', ['EGF_EGFR complex', 'Epidermal Growth Factor (EGF)
bound to Epidermal Growth Factor Receptor (EGFR)'], 'BIND_object'(18)),
addfnval(user_id, ['BIND_object'(18)], 0),
addfnval(a, ['BIND_Interaction'(10)], 'BIND_object'(18)),
addfnval(priv, ['BIND_Interaction'(10)], false)
```

Daplex function definitions

```
define interaction_objects(i in BIND_Interaction) ->> BIND_object  
  {a(i), b(i)};
```

```
define complex_interactions(c in BIND_Molecular_Complex) ->> BIND_Interaction  
  i in BIND_Interaction such that iid(i) in interaction_list(c);
```

```
define complex_objects(c in BIND_Molecular_Complex) ->> BIND_object  
  interaction_objects(complex_interactions(c));
```

```
define complex_subcomplexes(c in BIND_Molecular_Complex) ->> BIND_Molecular_Complex  
  s in BIND_Molecular_Complex such that  
    descr(s) in short_label(complex_objects(c));
```

```
define all_complex_subcomplexes(c in BIND_Molecular_Complex) ->> BIND_Molecular_Complex  
  ( complex_subcomplexes(c)  
  union  
    all_complex_subcomplexes(complex_subcomplexes(c)) );
```

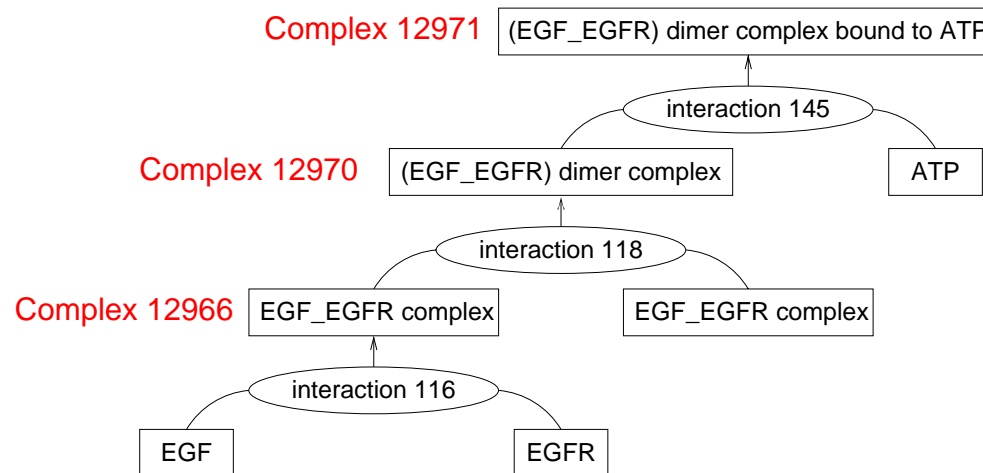
Finding all complex-subcomplex relationships

Daplex query:

```
for each c in BIND_Molecular_Complex
  for each s in all_complex_subcomplexes(c)
    print(mcid(c), descr(c), mcid(s), descr(s));
```

Query results:

12971	(EGF_EGFR) dimer complex bound to ATP	12970	(EGF_EGFR) dimer complex
12971	(EGF_EGFR) dimer complex bound to ATP	12966	EGF_EGFR complex
12970	(EGF_EGFR) dimer complex	12966	EGF_EGFR complex



Taxonomy example (1)

Extract from a P/FDM implementation of the MIAME schema:

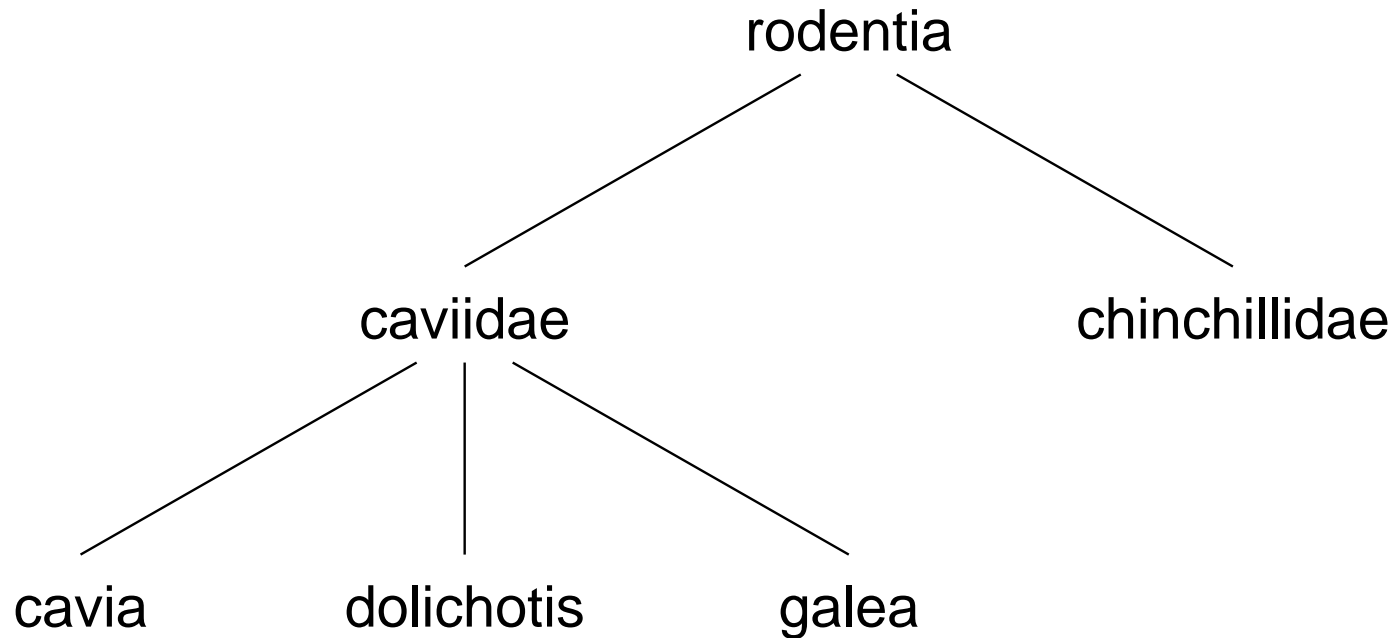
```
declare sample                ->> entity
  declare id(sample)          -> string
  declare organism_ncbi(sample) -> string
  declare treatment_protocol(sample) -> string
  declare used_in_experiment(sample) -> experiment
  declare sample_parameters(sample) ->> parameter
key_of sample is id
```

Daplex query:

```
for each s in sample such that organism_ncbi(s)="rodentia"
  print(id(s),treatment_protocol(s));
```

The system returns information about all samples where the recorded taxonomy name is “rodentia” **or below**.

Taxonomy example (2)



If a user asks for information about samples where the source is identified by the taxonomy term “rodentia”, then we want the system also to return information on samples where the recorded taxonomy term is below “rodentia” in the hierarchical vocabulary.

Taxonomy example (3)

Extract from schema of database module storing the taxonomy:

```
declare taxonomy_entry ->> entity
declare name(taxonomy_entry) -> string
declare is_of_taxa(taxonomy_entry) -> taxonomy_entry
```

and derived functions:

```
define acceptable_alternative_names(s in string) ->> string
  name(self_and_more_specialised(t in taxonomy_entry such that
                                     name(t)=s));
```

```
define self_and_more_specialised(t in taxonomy_entry) ->>
  taxonomy_entry
  ({t} union self_and_more_specialised(is_of_taxa_inv(t)));
```

Taxonomy example (4)

1. Analyse query

Is a value for a string-valued attribute specified in the query?

2. Check database metadata

Are the values stored for that attribute chosen from a hierarchical vocabulary?

3. Find alternative acceptable values

If “yes” to both of the above questions, find the set of more specialised values for this attribute that would also satisfy the query by examining the sub-tree of terms below the given term.

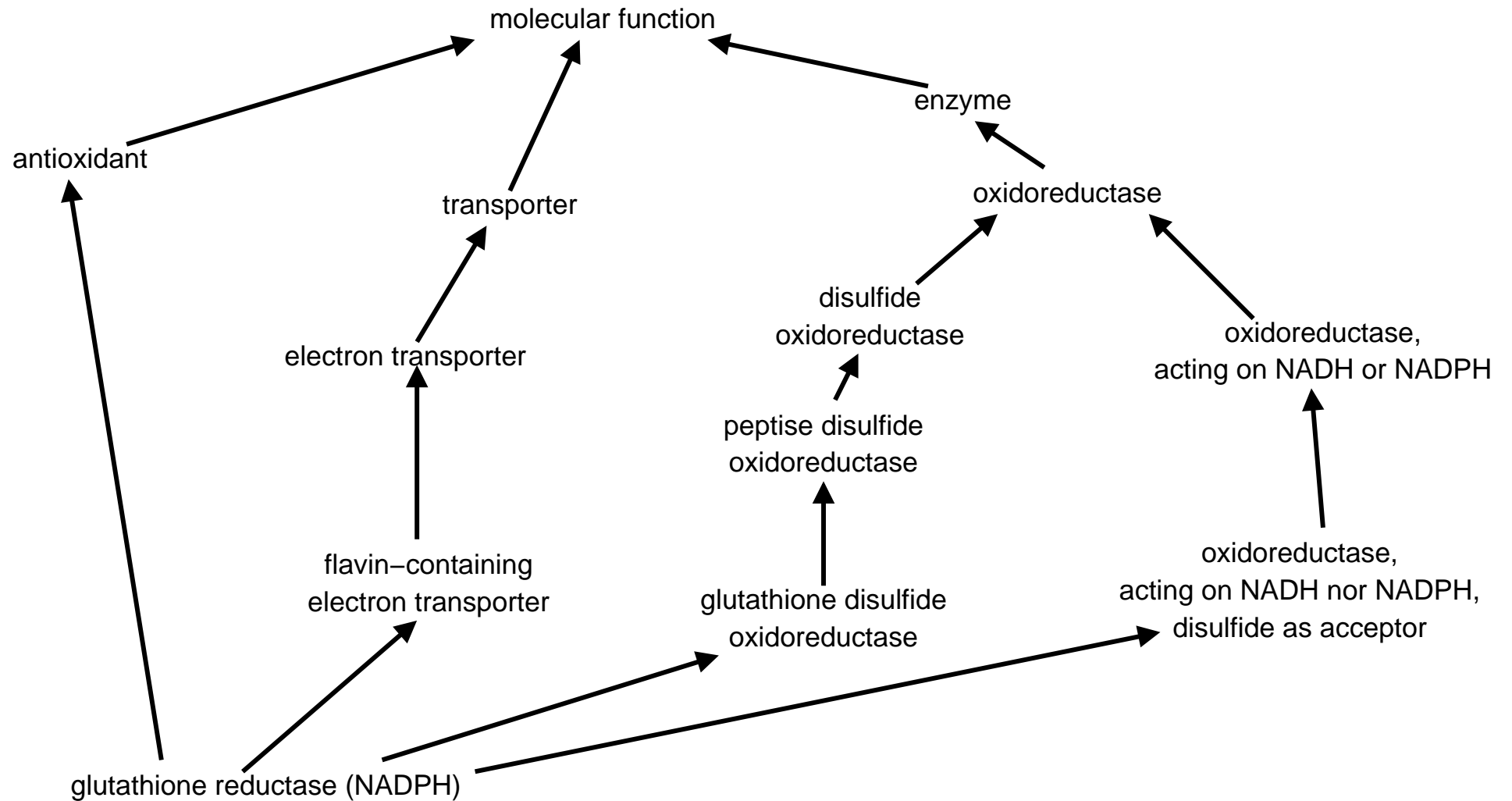
Daplex schema for GO terms and relationships

```
declare go_term          ->> entity
  declare name(go_term)  -> string
  declare term_type(go_term) -> string
  declare isa(go_term)   ->> go_term
  declare partof(go_term) ->> go_term
  key_of go_term is name, term_type;
```

The functions *isa* and *partof* are multi-valued (indicated by the double-headed arrows), as are the inverse relationships (*isa_inv* and *partof_inv*) that are maintained automatically by the P/FDM system.

Formerly, a GO term was not uniquely identified by its name. For example, “B-cell receptor” was both the name of a function and the name of a component, and “tubulin folding” was both the name of a function and the name of a process.

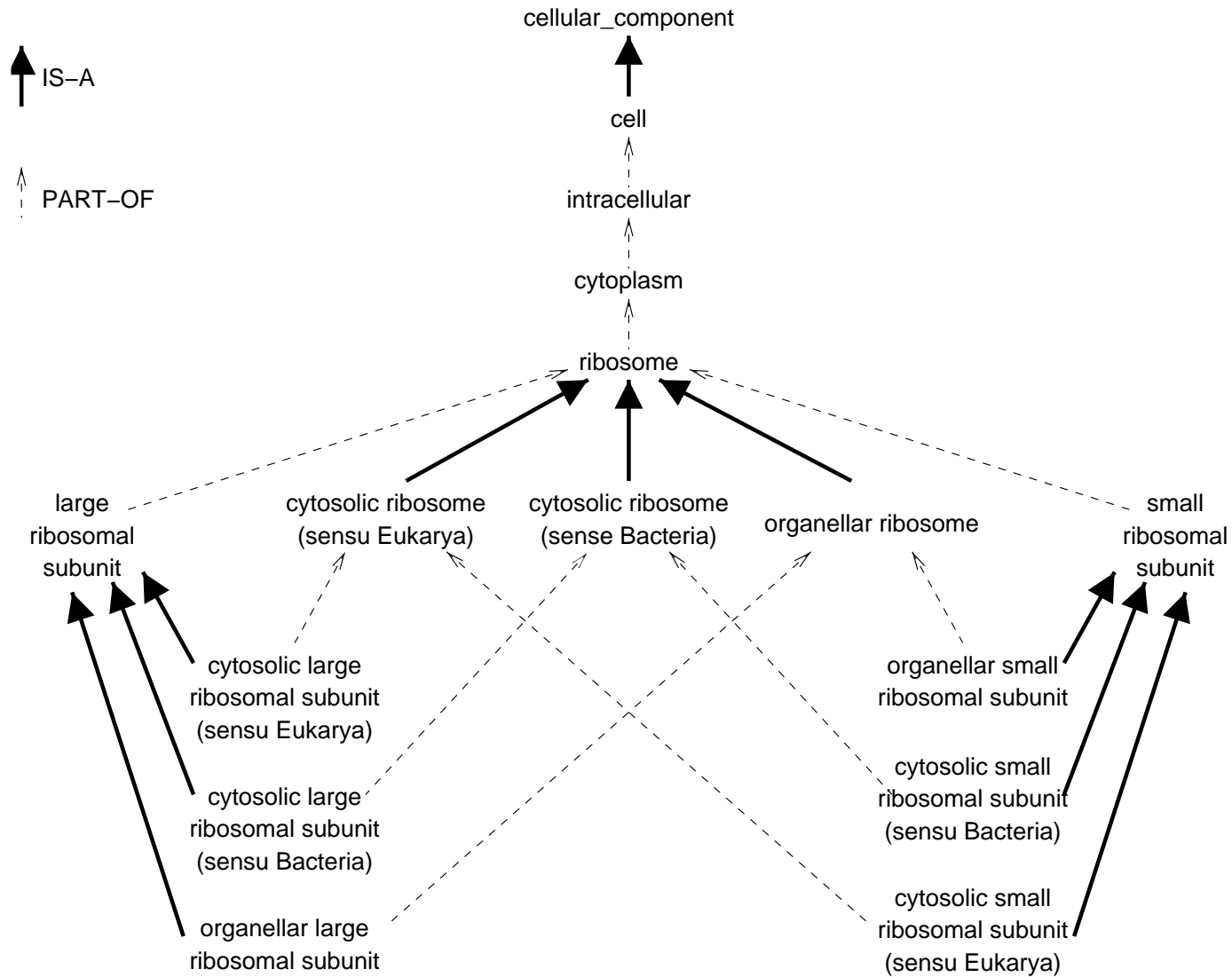
GO: function example



Finding all more specialised terms

```
define all_more_specialised_terms(t in go_term) ->> go_term  
  (isa_inv(t) union all_more_specialised_terms(isa_inv(t)));
```

GO: component example



Finding all sub-parts

```
define all_subparts(t in go_term) ->> go_term
  in tempmod
  ((partof_inv(t)
   union all_subparts(partof_inv(t)))
   union all_subparts(isa(t)));
```

Example query:

```
|: for the t in go_term such that
|:   name(t) = "organellar ribosome" and
|:   term_type(t) = "component"
|:   print(name(all_subparts(t)));
```

```
organellar small ribosomal subunit
organellar large ribosomal subunit
small ribosomal subunit
large ribosomal subunit
```

Species-specific terms

```
declare go_term                ->> entity
  declare name(go_term)        -> string
  declare term_type(go_term)   -> string
  declare taxonomy_entry(go_term) -> taxonomy_entry
  declare isa(go_term)         ->> go_term
  declare partof(go_term)      ->> go_term
  key_of go_term is name, term_type, key_of(taxonomy_entry);
```

Conclusions

- The functional data model is convenient for describing and querying data with a recursive structure.
- Derived functions can make explicit those relationships that are present but which are not fully described in a DTD file.
- Implementing taxonomies and ontologies in a database facilitates querying their contents and performing integrity checks.
- In P/FDM, queries and derived functions can be written in Daplex or Prolog.
- Providing a simple graphical user interface that enables non-expert users to express recursive queries against biological networks and tree structures in an easy way remains a challenge.

Acknowledgements

The Swedish Foundation for International Cooperation in Research and Higher Education (STINT)

The Swedish Foundation for Strategic Research (SSF)