Bioinformatics Advance Access published April 1, 2004

TITLE PAGE

‘Designing and executing scientific workflows with a programmable integrator’

M. Chagoyen* (1), M.E. Kurul (2), P.A. De-Alarcón(3), J.M. Carazo (1), A. Gupta (2)

(1) Biocomputing Unit, Centro Nacional de Biotecnología, Madrid 28049, Spain
(2) San Diego Supercomputer Center – U.C. San Diego, La Jolla CA 92093-0505, USA
(3) Integromics S.L. – Parque Científico Madrid, Madrid 28049, Spain

Keywords: information integration, programmable integrator, scientific workflows, workflow design
Running title: scientific workflows integrator
* To whom correspondence should be addressed

ABSTRACT

Motivation:

As in many other fields of science, computational methods in molecular biology need to intersperse information access and algorithm execution in a computational workflow. Users often find difficulties when transferring data between data sources and applications. In most cases there is no standard solution for workflow design and execution and tailored scripting mechanisms are implemented in a case by case basis.

Results:

In this paper we present a general purpose “programmable integrator” that can access information from a variety of sources in a coordinated manner. Its usefulness in complex bioinformatics applications is claimed and supported by some application examples.

Availability:

Tools are freely available to non-profit educations and research institutions. Usage by commercial organizations requires a license agreement. Contact**:** gupta@sdsc.edu
Software requirements: Java v1.3 (http://java.sun.com), Xerces XML Parser (http://xml.apache.org/xerces-j), Kweelt implementation of XQuery (http://kweelt.sourceforge.net/).

INTRODUCTION

Bioinformatics involves the use and transformation of great amounts of data. To accomplish a given task, different computational tools are used over different data elements in a particular cascade of computational steps. Difficulties in designing and running a study arise when data are of heterogeneous nature and distributed and when the tools available to the scientist provide their own peculiar input/output mechanisms, often incompatible with each other. As the survey on bioinformatics tasks performed by (Stevens *et al*. 2001) highlighted, moving data between repositories and analysis tools is of great importance when building complex queries.

In order to assist the researchers in building and managing computational processes, several researchers investigated the use of workflows (Ailamaki *et al*., 1998; Ludäscher *et al*., 2003). Standard Workflow Management Systems (WFMSs) are designed for the definition, execution and monitoring of complex processes in fairly stable environments (such as industry and business applications). These systems are process-oriented with a transactional focal point. In contrast, most applications in computational biology focus their attention on the information being processed, that is, they are data-oriented (Ailamaki *et al*., 1998). As they are discovery driven, they demand both powerful and expressive workflow definition languages, as well as rich means of data access and integration.

In this work we introduce the use of a procedural programmable integrator suitable for the creation of bioinformatics workflows. Our integrator allows the definition and execution of a cascade of data access, querying/filtering and algorithm invocation events. This integrator is currently based on a few mature techniques allowing distributed access to most resources available in the Molecular Biology community. The use of emerging accessing mechanisms, such as web services and grid-based services, recently introduced in bioinformatics, e.g. BioMOBY and MyGrid projects (Stein *et al*., 2003), is under evaluation.

Our procedural programmable integrator includes the essential functionality for the definition and implementation of *executable workflows*, and has been designed as part of larger computer infrastructure aimed at the support of scientific workflows (Ludäscher *et al*., 2003). In this global infrastructure, an executable workflow, which is now directly written by a workflow designer, could also be automatically created by a workflow compiler from an *abstract workflow*. This means that, when fully integrated into the workflow infrastructure, a scientist could use the appropriate high-level design tool to create an *abstract workflow* using domain language and concepts. Since it might seem that a high-level design would completely hide away the details of an *executable workflow*, we show the need for making parts of this workflow explicit through two examples in computational biology.

SYSTEM AND METHODS

PLAN – a Language for a Programmable Integrator

PLAN is a simple XML-based language for the definition of executable workflows that simplifies data search and analysis by providing a uniform XML view on both data sources and analytical applications. The use of internal XML data structures is very reasonable, since an increasing number of data providers in the molecular biology domain offer the possibility of downloading information in XML format. Furthermore, many programs in the field also provide input/output XML-based mechanisms. In the case in which the data are not provided in XML format, a wrapper mechanism should be implemented to provide the necessary translation.

System architecture

The main strength of PLAN is to significantly reduce the complexity of information integration from multiple sources. To do so, PLAN combines a declarative query language with the additional power of a procedural instruction set using a uniform and easy to manipulate XML format. The overall system architecture is shown in Figure 1. We briefly describe the main components of our approach:

- Data source registration:

Data sources (web pages, local files, applications or relational databases) are registered in PLAN using the *resource catalogue*. This catalogue contains information relevant to each data source such as type, base URL, path information, parameters needed and wrapper assigned to it. Data sources are considered to be distinct if they have different invocations and/or provide structurally different content (e.g. access to a Swiss-Prot entry and a hit list page resulting from a search performed on the Swiss-Prot in the Expasy (www.expasy.org) server are considered two distinct data sources). For a relational database, it contains information about the database URL, tables, fields, types and so forth to ensure that a query can be issued against it.

- Query declaration:

PLAN uses XQuery (XQuery, 2003) to support the execution of declarative queries for XML data . Internally PLAN provides an XML view on any data source, therefore queries and filters are written by the programmer using this query language. In this way, the system is not limited to filtering out operations on data but it allows the declaration and execution of complex queries including aggregate operations on data elements. PLAN query language can be "extended" through the implementation of User Defined Functions and registration into the PLAN system (equivalent to PL/SQL routines in Relational Database).
In the particular case of relational data sources XQuery searches are automatically translated to SQL statements (by *to SQL* processing unit) and executed on the corresponding source RDBMS.

- Wrapping non-XML sources:

A generic wrapper utility (Gupta *et al*. 2003) automatically translates resulting data from relational database searches into XML format (*to XML* processing unit).
In case the data source is neither XML nor relational, an external wrapper is used to convert the data (HTML, raw text, etc…) to XML format. Transformation of web pages and local files is based on the content and structure of the file. If two sites/files provide structurally different contents they require the implementation of two wrapping mechanisms. For the work described in this paper, we used the Minerva wrapper toolkit as a freeware technology for accessing and transforming web pages (Crescenzy and Mecca, 1998). Nevertheless, the choice of a particular technology to wrap external information sources does not affect either the PLAN language or the PLAN execution model.
No wrapper is needed for those data sources providing information in XML format.

• Query execution:
The execution of a PLAN program is done by a stack machine where data to be accessed are pushed to the *execution stack* by its unique access link (or access instruction). Once the stack is constructed, it is traversed in order to access the actual data one by one. The *wrapper selector* chooses the corresponding wrapper from the *resource catalogue* and makes the appropriate wrapper call (if any). Data are retrieved and if needed, converted to XML by the wrapper. This is automatically done during run time.
Retrieved data are placed in the *global buffer*. Now, a further query can be executed on the buffer resulting in a new data structure that is name tagged and put into the *global data table*. Unlike the *resource catalogue*, which is a permanent data structure, the *execution stack*, *global buffer* and *global data table* are virtual XML data structures. “Virtual” means that they exist while the PLAN program is running.
Final results (*XML output*) can be saved locally on an XML document by the *XML writer*.

• Error handling
PLAN also provides some error handling mechanisms. To handle exceptions due to data source unavailability and data source policies, the user can use “delay” and “trial” attributes in a particular access. If data is still not available, PLAN just bypasses that particular data access and reports the unresolved unique access link (this allows the sequential retrieval of a set of entries from a web site, even if some of them are corrupted). Finally, some of the error handling on data format might be pushed to the wrappers (e.g. Exception Handling mechanisms provided by the Minerva Wrapper Generator).

PLAN at work
Creation and execution of a workflow using PLAN requires:
(1) Declaration of new data sources that are not already registered in the *resource catalogue*. If they are non-XML/non-relational appropriate wrappers should be implemented.
(2) Definition of the process workflow using PLAN syntax and commands.
(3) Execution of the workflow

The set of commands provided by PLAN can be grouped into three categories: (i) memory management, (ii) filtering/querying and (iii) input/output procedures. In the simplest case, a workflow in the PLAN language involves:
1) Construct the unique access link to access data in the data source.
2) Launch an HTTP request, a program invocation or relational query to retrieve data and convert it into XML. The last task is transparent to the user since the wrapper module for that particular source will perform the translation (if needed).
3) Allocate a name for the buffer where information will be internally stored.
4) Query the resulting XML document in order to filter out interesting information.
5) Present results back to the user by writing to an XML file.

We explain the PLAN language through a simple workflow: “Retrieve those sequences in Swiss-Prot corresponding to proteins which contain the InterPro motif IPR001478”. In this particular example data was not available in XML format, therefore we used two wrappers: one for the Swiss-Prot entry format and other for the InterPro site which provides Swiss-Prot matches of a given InterPro entry. The basic plan to execute this workflow is:

```
Get all SwissProt entries from InterPro matching IPR001478
For each SwissProt entry
        get sequence information from SwissProt
```

The first step in executing this request is to construct an URL and send an HTTP request to the InterPro site, get the matches from the site and store the results in the global data table to be used in the following part of the query. In PLAN, this is specified as:

```
..<QUERY>
     <RESULT>
        LET $x := set("","ipr","IPR001478"),
            $x := set($x,"display","n"),
            $x := set($x,"dmax","20000"),
            $y :=  constructURL("GET","http://www.ebi.ac.uk/interpro/ISpy",$x)
        RETURN $y
     </RESULT>
  </QUERY>
  <TRAVERSE>POP</TRAVERSE>
  <QUERY>
     <RESULT>
        <DATA NAME="InterProMatches" TYPE="Add">
           RETURN stream()
        </DATA>
     </RESULT>
  </QUERY>
```

The first <QUERY> uses the internal variables $x and $y to build the URL with the proper parameters in order to get the desired data. The built hyperlinks (or Xlinks) are identified, parsed and checked against the *resources catalogue* to verify the resource entry and the wrapper class to be executed for this link. Finally, they are pushed to the *execution stack* with all related information attached to it.

<TRAVERSE> obtains each link from the *execution stack*, traverses it, gets the data from the source and converts it into XML if needed. Two options are available for this statement: either links are removed from the stack once traversed (POP) like in the example, or keeping the link in the stack (PEEK). The main reason to keep a link in the stack is that more than a <QUERY> can be sequentially applied to the data source in the stack.

A second <QUERY> is used to further process data: the <DATA> operation builds a new entry in the *global data table* by adding the result data into the structure called "InterProMatches". In this case the entire data stream is saved by the RETURN construct.

To continue with the example workflow, we need now to get sequence data from Swiss-Prot. The corresponding pseudo-code is:

```
For each Swiss-Prot entry in "InterProMatches"
        construct link to Swiss-Prot
```

```
For each link in stack
    traverse by popping
    retrieve Swiss-Prot sequence
    add to “spwSeq” XML data structure
```

This is done in PLAN by first bringing the results of the virtual structure “InterProMatches” (stored previously in the *global data table*) to the *global buffer* as the working register using:

```
  <XMLBUFFER NAME="InterproMatches" />
```

Next, we need to fetch data from Swiss-Prot. In order to do so, we should first create the corresponding Xlinks to Swiss-Prot data with a similar query statement to the first <QUERY> above (code not shown). Links to Swiss-Prot constructed are located in the *execution stack.*

Now we can traverse each link in the execution stack to get the corresponding Swiss-Prot entry and select the information of interest in the RETURN statement (in this particular case we only want the sequence data, corresponding to the <seq> tag provided by the Swiss-Prot wrapper).

```
..<WHILE>
     <CONDITION>
        <STACK>
           <CONDITION>NONEMPTY</CONDITION>
        </STACK>
     </CONDITION>
     <DO>
        <TRAVERSE>POP</TRAVERSE>
        <QUERY>
           <RESULT>
              <DATA NAME="spwSeq" TYPE="Add">
                 FOR $doc in stream ()
                 RETURN (<swp_entry> $doc/seq </swp_entry>)
              </DATA>
           </RESULT>
        </QUERY>
     </DO>
  </WHILE>
```

The WHILE-DO construct has exactly the same semantics as in any standard procedural language. The termination condition for the while construct is to check if the stack of links is not empty. Since PLAN is a stack-based language, this check is performed explicitly.

Finally, the results are returned to the user by writing out a result file.

```
  <CONSTRUCT>
     <DATA NAME="spwSeq" />
  </CONSTRUCT>
  <DELETE FILE="./wf_example.xml" />
  <PRINTOUT FILE="./wf_example.xml" />
```

The <CONSTRUCT> element gathers all results in the virtual XML sources and puts together a composite XML document named ‘spwSeq’ in the *global data table* that is finally written in a file.

Besides web sources as illustrated in this example, PLAN can also access relational databases and local programs. For a relational source, a <RELATIONAL_QUERY> construct is used. Local programs can be transparently integrated into the engine so the system will "see" them as any other web or relational source. To do so, we have developed a software library that provides an interface to request the execution of a particular program. Externally, the user would make an information request in the same way as for "static" sources. Internally, the engine will query the wrapper catalogue and recognize this request as a local execution request so it will perform a background execution of the requested program instead of an HTTP request. After the execution is completed, results will be retrieved by the engine in XML format.

Concatenation of workflows is straightforward in PLAN, as results are always stored in the *global data table* and can be further invocated as working registers (streams) in the *global buffer* using the <XMLBUFFER /> element.

RESULTS

Motivating examples

We will argue in favour of the usefulness of a programmable integrator like PLAN by showing how two typical tasks performed in bioinformatics and computational biology applications can be implemented. The first task illustrates a classic information extraction that requires the integration of several data sources and data types (this is the case of sequence and structural data), while the second, involving more complex computations and relationships among data and searching criteria, demonstrates complex searches and filter operations on biological data.

Example A: Transfer of annotations between two biological domains: from sequence to structure.
The first workflow is designed to study the structure of scaffold proteins and their interacting domains, and will be illustrated by a search of information about PDZ domain containing proteins. PDZ domains are modular protein interaction domains that bind in a sequence-specific fashion to short C-terminal peptides or internal peptides that fold in a beta-finger (Shen and Sala, 2001). They are frequently found in multiple copies or associated with other protein-binding motifs in multidomain scaffold proteins. PDZ containing proteins are typically involved in the assembly of macromolecular complexes that perform localized signalling functions at particular cellular locations. They are widespread and show considerable sequence variation.
A wide picture of the structural characterization of PDZ domains can be obtained by searching and integrating information from different data sources, such as InterPro (Apwailer *et al*. 2000) containing information on protein sequences having such domains, Swiss-Prot and TrEMBL (Bairoch and Apwailer, 2000) that contain the actual protein sequences and, finally, PDB (Berman *et al*., 2000) and derived databases e.g. CATH (Orengo *et al*., 1997) that contain structural information.
This workflow is an example of how features of different data types can be transformed and mapped using PLAN, once the appropriate relationship among data has been established. In this case, we wanted to transfer annotations on a protein sequence data (in Swiss-Prot or TrEMBL through the InterPro) to atomic coordinates (in PDB) by performing a sequence alignment using BLAST (Altschul *et al*., 1997). The actual biological question that we are addressing in the search is to find all solved structures of PDZ domains.

When building links between two different data sources in a given workflow, it is essential that the user have full control over all relevant parameters of the association being built between the two different data types, as it is allowed in PLAN by means of a declarative query language like XQuery. In this example this statement can be exemplified by the fact that just providing all structures in PDB corresponding to PDZ containing proteins in Swiss-Prot/TrEMBL doesn't bring the correct answer to our question (see Figure 2). Only those PDB structures that contain the full polypeptide segment of a PDZ domain should be supplied. This requirement can be fully evaluated using PLAN by making a comparison of the amino acid ranges in the InterPro matches (that correspond to a range in the BLAST query sequence), and the corresponding positions in the aligned sequences in the PDB (the BLAST hit sequences) in the filtering step. From a total of 1376 matches in Swiss-Prot/TrEMBL corresponding to the PDZ/DHR/GLGF domain in InterPro (IPR001478), 44 structures are found in the PDB (using a BLASTp search mechanism and 90% identity threshold). The transfer of the amino acid segments of the PDZ domain from the query sequence to the hit sequence during the search alignment, filtering out those structures not containing the PDZ domain, results in the true 27 matching structures (see Table 1).

The information that we obtain by this process would provide grounds to state that all PDZ domains have similar architectures, that is to say, their overall fold approximates a six-strand beta-sandwich flanked by two alpha-helices (Shen and Sala, 2001). This information is highlighted with our system when performing an additional search on the CATH database: of the 27 structures corresponding to the PDZ domain, 13 are classified in CATH. All of them under code 2.30.42.10 , that corresponds to 'Mainly Beta' class, 'Roll' architecture, 'Pdz3 domain' topology and 'Signal transduction' homologous superfamily. Of course, this is an example with only one specific domain illustrating the very rich information that can be obtained by integrating with PLAN, although any other sequence domain can be analysed as well.

Example B: conservation of residues at protein-protein interfaces. The case of homodimeric structures
The study was conducted by Valdar and Thornton, 2001 to assess the conservation of residues at protein-protein interfaces compared with other residues on the protein surface. To carry out this work the authors chose a set of solved protein structures, from which a group of functionally equivalent homologues were identified along with a multiple alignment. The search criterion established by Valdar and Thornton for the creation of this first data set was the following:

- the protomer to be studied must form a stable, symmetric complex with one other protomer to which it is identical (or nearly identical) such as the oligomer is homodimeric and the conservation of only one chain need to be considered;
- the full wild-type complex must be available in PDB or PQS (Henrick and Thornton, 1998);
- Among all the structures available for the complex, the structure chosen must have the best combination of the following properties:
    - high resolution, inclusion of any bound cofactors that occur naturally,
    - if applicable, the inclusion of a ligand similar in size and shape to that of the natural substrate.
- to enable the robust identification of a diverse set of homologues, the promoter should be represented in CATH
- the promoter sequence must have non-fragment homologues in the Swiss-Prot that are numerous (>10) and diverse (<70% mean pairwise sequence identity) and, as judged by their annotation, share its function and multimeric state

Now, we are going to show how this search can be automatically performed using PLAN. The requirements can be translated into a sequence of steps of information access and post-processing as given in Figure 3. We point out the repeated occurrence of *search-collect-filter* patterns in this workflow and that, in Step 3, grouping structures by protein has the standard group-by semantics in database systems – thus, for every value of protein, we need to collect the structures that belong to the protein.

Valdar and Thornton identified a total of six homodimer families. We have implemented, using PLAN, a complete workflow search including those requirements of the authors (see figure 3). The design of the workflow aims at minimizing the amount of data to be accessed and processed, maximizing the overall performance.

- Step 1*: The oligomer is homodimeric*

To find structures that meet the first criterion a search is done on the Protein Quaternary Structure service of the Macromolecular Structure Database at the EBI (www.ebi.ac.uk/msd) with the following parameters: quaternary type = 'dimeric', homo or hetero = 'homo' and mean delta ASA per chain = 'greater than 400 Å^2' (according to the PQS methods, values below this cutoff may indicate that PQS file describes a significant crystal packing arrangement rather than an oligomeric assembly) A total of 5659 structures where found.

- Step 2: *The protomer should be available in CATH*

The authors of the study used the CATH's Homologous Superfamilies to identify a robust set of homologues. For all the homodimeric wild-type structures in the list so far, a filtering is performed to get only those available in the CATH classification. From the first set of homodimeric structures, 3406 are already classified within the CATH database.

- Step 3: *Group structures corresponding to the same protein*

Further processing and filtering of the data should be done on the basis of structures corresponding to the same protein. In order to carefully identify the biological content of the structures, a BLASTp search should be therefore done against a non-redundant sequence database (Swiss-Prot in our case) using the sequence of the corresponding PDB/PQS hits as query.
EC numbers corresponding to enzyme entries in Swiss-Prot are retrieved for further use. BLASTp searches against Swiss-Prot are also useful for the evaluation of the following requirements:

- Step 3a: *Proteins should have numerous distant homologues*

The evaluation of the number of distant homologues for each protein sequence obtained in step 4 is being done by means of a BLASTp search against Swiss-Prot and counting of the alignments with an identity between 30% and 70%, with an alignment length greater than 100 amino acids.

- Step 3b: *Full wild-type complex must be available*

The intended analysis of the dataset requires that atomic coordinates should correspond to wild-type proteins for all the structures of step1. Mutants are interpreted as those PDB chains with less than a 100% identity with the first hit in the BLASTp search against the Swiss-Prot. A further search is done in PDB to filter out structures annotated as Fragments in the PDB.
Step 3 reduced the number of structures to 973, corresponding to 335 unique sequences in the Swiss-Prot. Information of the multimeric state (CC -!- SUBUNIT) of Swiss-Prot matches is evaluated for cross-validation. The number of sequences annotated as "HOMODIMER" in Swiss-Prot reduces the set to 185 sequences.

- Step 4: *Homologues should share multimeric state and function*

Although the authors required homologues to share both multimeric state and function, we have relaxed this requirement due to the difficulty of evaluating an agreed “similar” relationship in the functional domain. From the set of 185 sequences so far, only 78 have more than 10 distant homologues in the Swiss-Prot (30-70% identity) which are also annotated as HOMODIMER.

- Step 5: *Search natural cofactor and substrate*

A search is performed in ENZYME (Bairoch, 2000) to find cofactors and substrates for each of the distinct EC numbers.

- Step 6: *Final selection*

For each enzyme protein, the corresponding PDB entries are searched to retrieve ligands. The structure chosen will be the one with the highest resolution; ligand corresponding to natural cofactor; ligand similar to natural substrate.

The search performed provided all the families identified by Valdar and Thornton (Table 2a) plus a number of new families (Table 2b). The additional number of families identified might correspond to the greater number of data in the PDB, CATH and Swiss-Prot databases from the time the original study was performed in 2001. Although the requirements of the authors have been relaxed in the current workflow implementation, a manual check has been performed on homologue proteins by visual inspection of the Swiss-Prot ID and corresponding EC numbers.

This example illustrates how a complex search can be automated and some of the virtues arising from this automation. In particular, the capability to perform the search with new releases of the different databases makes it possible to keep gathering new information as more sequences and structures are deposited.

DISCUSSION AND CONCLUSIONS

Scientific workflow infrastructures demand particular means of data access, analysis and integration, as well as powerful and flexible workflow definition languages. General purpose programmable integrators, such us PLAN, are very valuable tools to control information fetching, filtering and result construction in a workflow engine designed for performing typical tasks in bioinformatics and computational biology.

PLAN is highly flexible due to its modular design and programmable interface. It is designed to easily handle heterogeneous data sources (facilitated by the use of the resource catalogue and wrapping mechanisms for non-XML data sources), while providing powerful mechanisms for data integration and filtering (through the use of an internal XML data structure, a declarative query language and a procedural instruction set). Information can be kept in its original location and accessed only during run time. Only a resource catalogue defining access mechanisms and properties of the data is required. PLAN can be easily extended allowing the incorporation of additional data sources by registration to the catalogue. Available sources in the catalogue can seamlessly be used together in a computational workflow. The use of a declarative query language allows filtering operations on data, as well as any other complex queries provided by XQuery. Custom user defined functions can be easily added to be used in the query language.

Current limitations of the system are due to the distributed nature of the data sources. First, if a data source is not available it cannot be accessed. Implementing mechanisms for handling of alternative locations for the same data could solve this limitation. Second, non-XML and non-relational data sources need the design and implementation of wrappers. Although this requires an additional programming effort today, the number of such sources will be decreased in the near future as more data providers and applications in molecular biology supply data in XML format. Finally, the current system doesn't provide any mechanism to automatically handle changes in the data schema/format of the registered sources, therefore corresponding updates in the wrappers and already designed workflows should be manually performed.

Executable workflow engines like PLAN will be used in their full extent when combined with higher level languages defining abstract workflows, as described in (Ludäscher, 2003). Nevertheless, we have shown that there is a number of bioinformatics applications that demand the explicit definition of low-level procedures (exact procedure of retrieving and filtering a data object) even if embedded in high-level languages.

ACKNOWLEDGEMENTS

This work has been partially funded by CICYT grant BIO2001-1237 and NIH grant 1R01HL67465-01. Special thanks to Simone Santini for fruitful discussions and comments, as well as Alberto Pascual-Montano for careful reading of the manuscript.

REFERENCES

Ailamaki A, Ioannidis YE, Livny M. (1998) Scientific Workflow Management by Database Management. *Statistical and Scientific Database Management*, 190-199.

Altschul SF, Madden TL, Schäffer AA, Zhang J, Zhang Z, Miller W, Lipman DJ. (1997) Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res* **25**, 3389-3402

Apweiler R, *et al*. (2000) InterPro--an integrated documentation resource for protein families, domains and functional sites. *Bioinformatics* **16**, 1145-1150

Bairoch A, Apweiler R (2000) The SWISS-PROT protein sequence database and its supplement TrEMBL in 2000. *Nucleic Acids Res* **28**, 45-48

Bairoch A. (2000) The ENZYME database in 2000. *Nucleic Acids Res* **28**, 304-305.

Berman HM, Westbrook J, Feng Z, Gilliland G, Bhat TN, Weissig H, Shindyalov IN, Bourne PE (2000) The Protein Data Bank, *Nucleic Acids Res* **28**, 235-242

Gupta A, Ludäscher B, Martone ME *et al* (2003) BIRN-M: A Semantic Mediator for Solving Real-World Neuroscience Problems. *Proc. ACM SIGMOD 2003*

Henrick K, Thornton JM (1998) PQS: a protein quaternary structure file server, *Trends Biochem Sci*. **23**, 358-61

Ludäscher B, Altintas I, Gupta A. (2003) Compiling Abstract Scientific Workflows into Web Service Workflows. *15th Intl. Conf Sci Stat Database Management (SSDBM) Boston, MA, 2003*.

Orengo CA, Michie AD, Jones S, Jones DT, Swindells MB, Thornton JM. (1997) CATH- A Hierarchic Classification of Protein Domain Structures, *Structure*, **5**, 1093-1108

Shen M and Sala C (2001) PDZ domains and the organization of supramolecular complexes. *Annu Rev Neurosci*, **24**, 1-29.

Stein LD (2003) Integrating biological databases. *Nat Rev Genet*, **4**,337-345.

Stevens R, Goble C, Baker P, Brass A (2001) A classification of tasks in bioinformatics. *Bioinformatics*, **17**, 180-188

Valdar WSJ, Thornton JM (2001) Protein–Protein Interfaces: Analysis of Amino Acid Conservation in Homodimers, *Proteins* **42**, 108–124

XQuery 1.0: An XML Query Language W3C Working Draft. (2003) Available at http://www.w3.org/TR/xquery/

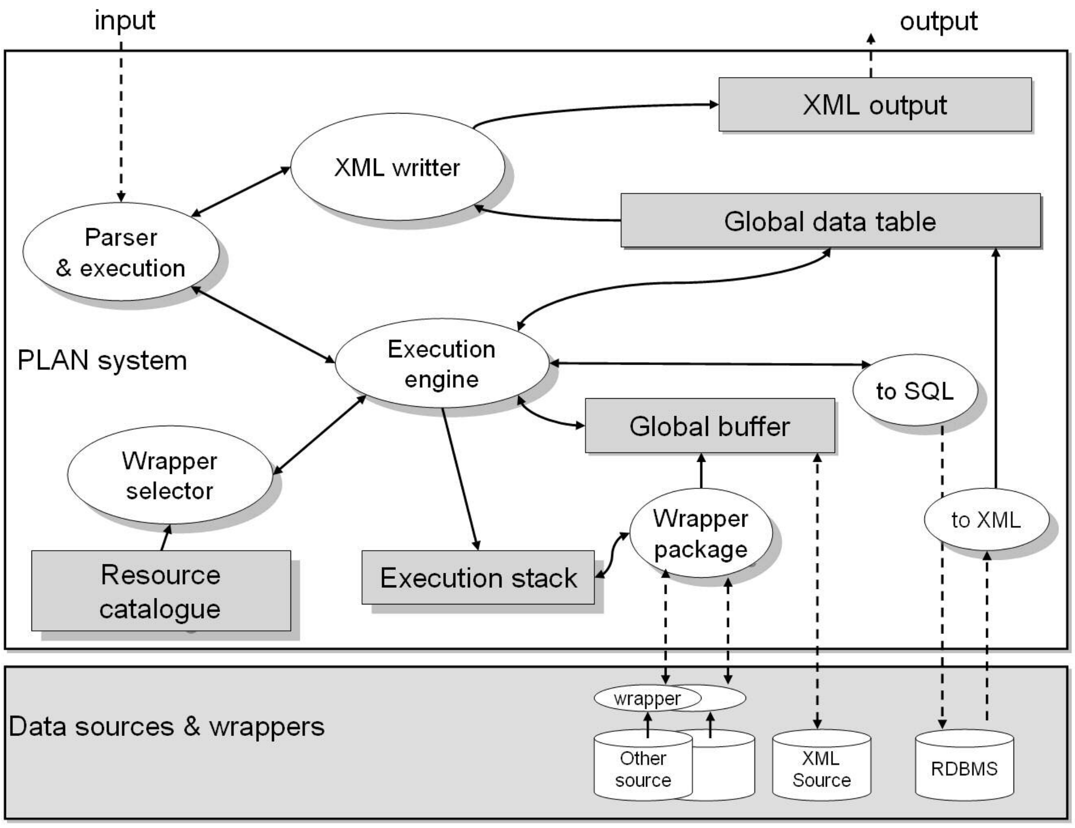

Figure 1: PLAN system architecture (oval shapes represent processing units and rectangles represent data structures). Data sources are registered in the *resource catalogue*. Data access/computation instructions are handled in the *execution stack.* Retrieved data is temporary stored in the *global buffer* for further filtering/querying. Resulting data from each access/computation step are managed in the *global data table*. Data transfer between consecutive steps in a workflow is performed by invocation of the desired named *global data table* structures into the *global buffer*.

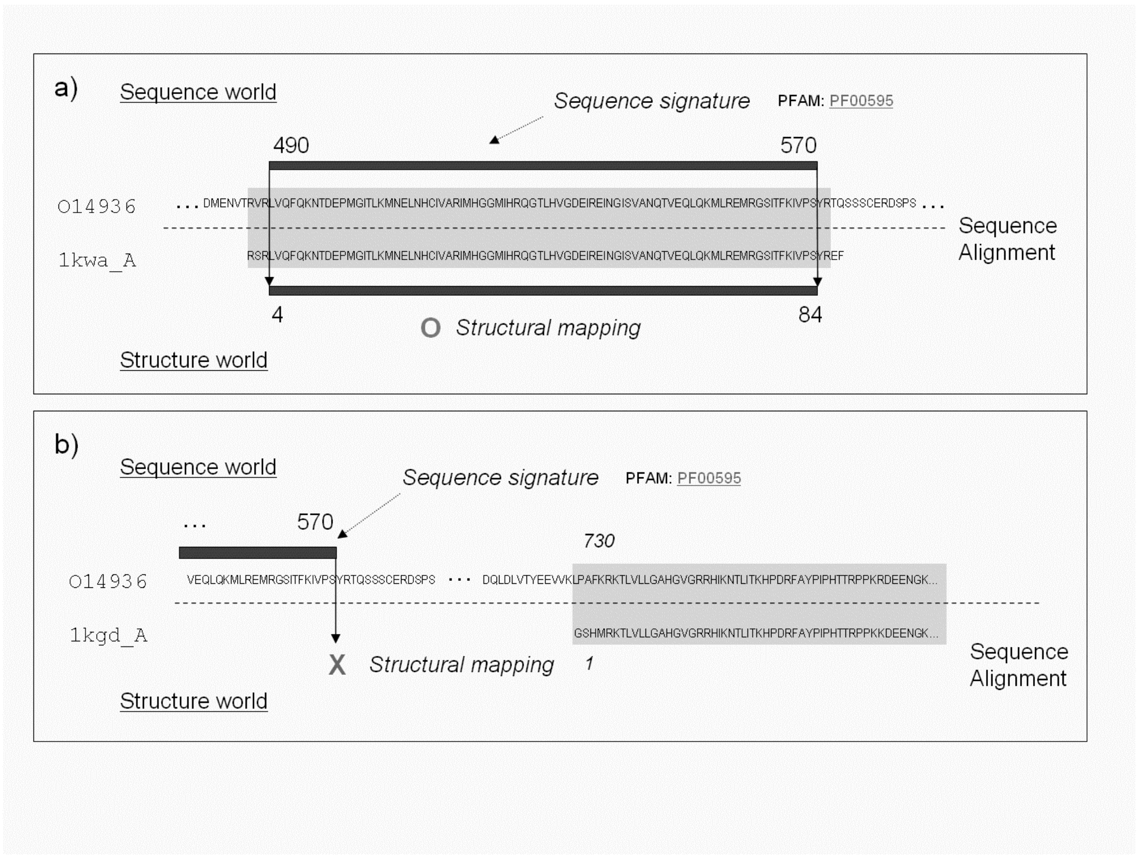
a)
Sequence world
Sequence signature
PFAM: PF00595
490
570
O14936
...DMENVTRVRLVQFQKNTDEPMGITLKMNELNHCIVARIMHGGMIHRQGTLHVGDEIREINGISVANQTVEQLQKMLREMRGSITFKIVPSYRTQSSSCERDSPS...
1kwa_A
RSRLVQFQKNTDEPMGITLKMNELNHCIVARIMHGGMIHRQGTLHVGDEIREINGISVANQTVEQLQKMLREMRGSITFKIVPSYREF
Sequence Alignment
4
84
O Structural mapping
Structure world
b)
Sequence world
Sequence signature
PFAM: PF00595
...
570
730
O14936
VEQLQKMLREMRGSITFKIVPSYRTQSSSCERDSPS ... DQLDLVTYEEVVKLPAFKRKTLVLLGAHGVGRRHIKNTLITKHPDRFAYPIPHTTRPPKRDEENGK...
1kgd_A
GSHMRKTLVLLGAHGVGRRHIKNTLITKHPDRFAYPIPHTTRPPKKDEENGK...
X Structural mapping
1
Sequence Alignment
Structure world

Figure 2: Transfer of sequence signature (PDZ domain) found in Swiss-Prot entry O14936 to structures in the PDB through a sequence alignment. It shows how this process can be controlled by correct filtering of BLAST output parameters, such as enabled by PLAN. a) A true positive hit in PDB is found when the sequence signature is within the limits of amino acid range being aligned. b) False positive hit: although BLAST has found a structure in PDB, it doesn't contain the PDZ domain.

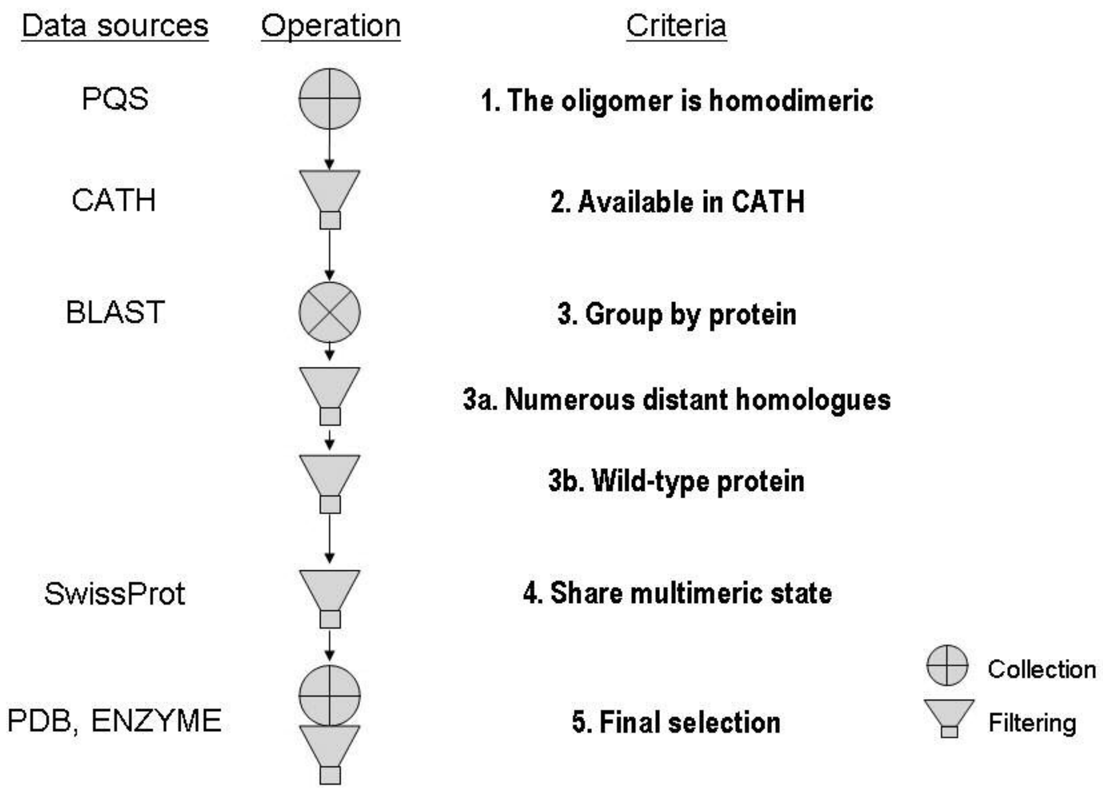

Figure 3: Conceptual workflow corresponding to the process of finding the representative data set for the study performed by Valdar and Thornton, 2001: data sources (databases and computations), operations (creation of collections and filtering) and description of criteria are included.

Table 1: PDZ domain containing structures in PDB retrieved by the corresponding workflow written with PLAN

PDB chain	Organism	Method	SwissProt/TrEMBL
1kwa A,B,C,D	Homo sapiens	X ray (1.9)	O14936
1b8q A 1qav B 1qau A	Rattus norvegicus	NMR X-ray (1.9) X-ray (1.2)	P29476
1lcy A,B,C	Homo sapiens	X-ray (2.0)	O43464
1ky9 A,B,C,D,E,F	Escherichia coli	X-ray (2.8)	P09376
1bfe A,B 1be9 A,C,D,E,F,G 1qlc A	Rattus norvegicus	X-ray (2.3) X-ray (1.8) NMR	P31016
1kef A	Homo sapiens	NMR	P78352
1pdr A,B,C,D,E,F	Homo sapiens	X-ray (2.8)	Q12959
3pdz A 1d5g A	Homo sapiens	NMR NMR	Q12923
1gm1 A	Mus musculus	NMR	Q64512
2pdz A	Mus musculus	NMR	Q61234
1i16 A	Homo sapiens	NMR	Q14005
1ihj A,B	Drosophila melanogaster	X-ray (1.8)	Q24008
1fcf A 1fc9 A 1fc7 A 1fc6 A	Scenedesmus obliquus	X-ray (2.19 X-ray (1.9) X-ray (2.0) X-ray (1.8)	O04073
1g9o A 1i92 A 1gq4 A 1gq5 A	Homo sapiens	X-ray (1.5) X-ray (1.7) X-ray (1.9) X-ray (2.2)	O14745
1htj F	Homo sapiens	X-ray (2.2)	O15085
1i16	Homo sapiens	NMR	Q14005

Table 2a: Families identified by Valdar & Thornton:

Description	Enzyme Commission Number
Superoxide dismutase.	1.15.1.1
Glutathione transferase.	2.5.1.18
Alkaline phosphatase.	3.1.3.1
Phosphopyruvate hydratase.	4.2.1.11
Triosephosphate isomerase.	5.3.1.1
Subtilism inhibitor (SSI type)	n.a.

Table 2b: Additional families identified with the corresponding workflow written with PLAN

Description	Enzyme Commission Number
Malate dehydrogenase	1.1.1.37
Malate dehydrogenase (NADP+).	1.1.1.82
3-isopropylmalate dehydrogenase	1.1.1.85
Trypanothione-disulfide reductase.	1.8.1.12
Dihydrolipoamide dehydrogenase.	1.8.1.4
Glutathione-disulfide reductase.	1.8.1.7
Thioredoxin-disulfide reductase.	1.8.1.9
Transketolase.	2.2.1.1
Orotate phosphoribosyltransferase.	2.4.2.10
Uracil phosphoribosyltransferase	2.4.2.9
Dihydropteroate synthase	2.5.1.15
Ribosylhomocysteinase	3.2.1.148
Orotidine-5'-phosphate decarboxylase	4.1.1.23
Citrate (si)-synthase.	4.1.3.7
UDP-glucose 4-epimerase	5.1.3.2
Tyrosine--tRNA ligase.	6.1.1.1
Histidine--tRNA ligase.	6.1.1.21
Threonine--tRNA ligase.	6.1.1.3
Interferon gamma (IFN-gamma)	n.a.
Transforming growth factor beta 2 (TGF-beta 2)	n.a.
Transforming growth factor beta 3 (TGF-beta 3)	n.a.