DFL designer — collection-oriented scientific workflows with Petri nets and nested relational calculus

Jacek Sroka, Piotr Włodarczyk and Łukasz Krupa
University of Warsaw, Poland
{sroka, p.wlodarczyk, lkrupa}@mimuw.edu.pl

Jan Hidders
Delft University of Technology, The Netherlands
a.j.h.hidders@tudelft.nl

ABSTRACT

In this paper we present DFL designer — a collection-oriented scientific workflow (COSW) tool based on the DFL notation which combines established formalisms from workflow modeling and databases, namely Petri nets and the nested relational calculus (NRC). COSW tools are used in applied sciences like bioinformatics where structured data is processed with the use of specialized services which are made available online by scientific institutions. They make such data processing experiments easier to conduct by the experimentators and easier to comprehend and repeat by the reviewers. The notations, models and techniques used for the construction of COSW tools are similar to the ones known from workflow modeling, but additional emphasis is put on the data manipulation aspects, e.g., the processing of nested collections of data. DFL designer not only allows design and enactment of complicated COSWs with the use of a huge library of supported bioinformatics services, but also provides a set of features for testing and analyzing workflow specifications that is unique for COSWs, including but not limited to interactive firing of transitions, hierarchical analysis of COSWs and translation of side-effect free COSWs to a query language like NRC.

Categories and Subject Descriptors

D.2 [**Software Engineering**]: Programming Environments, Design Tools and Techniques, Testing and Debugging

General Terms

Design, Theory, Verification

1. INTRODUCTION

In this paper we present DFL designer which is a tool for design, enactment and analysis of *collection-oriented scientific workflows* (COSWs) uniquely integrating ideas from workflow modeling and database research.

WANDS Indianapolis, IN, USA, June 6th, 2010

1.1 Collection-oriented scientific workflows

Information technology techniques and results developed for business applications are constantly challenged by new needs emerging from dynamically growing applied sciences. This is especially true for the domains of database systems and workflow processing, since even larger volumes of data have to be analyzed and the analysis processes become even more complex. Where those two domains coincide a new interesting field of research on collection-oriented scientific workflows has emerged.

In many sciences, like ecology, geology, chemistry, astronomy and especially bioinformatics, structured data is analyzed by a software system organized into a kind of network, through which the data flows and is processed, and where the nodes of the network carry out domain specific operations. This is similar to doing workflow processing in business, but here more emphasis is put on the processing of collections of data values and less on the control flow issues, hence the term collection-oriented scientific workflow (COSW).

The basic operations in such workflows are mainly specialized, domain-specific data analysis algorithms. Their efficient implementations are available as open source tools or are made freely accessible on dedicated Internet servers maintained by scientific institutions. The results produced by such workflows are used to form scientific hypotheses and to justify or invalidate them. The way a COSW is organized, i.e., which operations are executed and how they depend on each others results, is important and is usually published in some form together with the results of the data processing experiment. This is necessary for the reviewers and readers to understand what was done in the experiment, to effectively and objectively assess its merit, to repeat and verify it, and finally to adapt it for their own research projects.

Traditionally such data processing experiments, have been performed by copying and pasting data between local programs, e.g., the components of the EMBOSS package [21], and web accessible processing servers with WWW forms type user interfaces, like FASTA Sequence Comparison at the University of Virginia [27] and the Basic Local Alignment Search Tool at NCBI [17]. This method of experimenting is laborious and error prone. It has also been common to construct ad hoc scripts and programs to automatize this task, but for that at least some basic knowledge of programming in general and distributed programming issues is necessary. Furthermore, the produced software usually has not

been portable and poorly, if at all, documented.

Nowadays, specialized scientific workflow workbenches such as Taverna 2 [18, 10] and Kepler [14] are used. They are based on simple yet expressive graphical notations, integrate most important tools, services and databases from a given domain, and include various additional useful features like data provenance tracking or service discovery. The formal methods, techniques and tools used in COSW modeling have become an interesting topic of study themselves and are the focus of our research having many relationships with workflow modeling, business process modeling, databases, computational grids and many more established research areas.

1.2 Database query languages for collection-oriented scientific workflows

The COSW workbenches are being successfully applied to obtain publishable scientific results like the ones presented in [24]. The popularization of COSW techniques is progressing constantly. First, Internet repositories where scientists can share their workflows have been created [6]. Published workflows influence the creation of further workflows and sometimes are incorporated into them. Then, they are used to solve even more complicated problems. All this causes the need for the creation of still missing techniques and tools that aid COSW creators in understanding of how their workflows can precisely behave, debugging them, as well as detecting and helping to avoid common pitfalls.

In traditional workflow modeling such techniques and tools already exist and are constantly in the focus of attention. They are often based on the Petri net formalism [20, 16] which has a solid and proven theoretical foundation. Following [30] the three basic reasons for using Petri nets for workflow modeling are: (i) formal semantics despite the graphical nature, (ii) state-based instead of event-based, and (iii) an abundance of analysis techniques. The modeling and analyzing of workflows using a Petri net based approach is well described in literature, e.g., in [31], and many Petri nets tools support workflow modeling (see [25]). Even though notations used in some of the existing COWS tools are influenced by Petri nets, the majority of good ideas like verification and structural analysis algorithms have neither been adapted nor developed for COSWs at the moment of this writing.

Since COSWs are used to processes large amounts of data, it is also reasonable to expect that some of the ideas from database research on querying nested collections are applicable to them. As with workflow modeling, here again formalisms with solid and proven theoretical foundations exist, e.g., the nested relational calculus [1] (NRC). In fact, a NRC based *collection programming language* [33, 34] (CPL) has already been employed in the BioKleisli system [4] to combine biomedical data from diversity of sources used in the Human Genome Project [28]. This was possible since the precise specification of control flow was not necessary, i.e., the executed operations had no side effects, and only the computed result mattered. Thanks to relying on NRC, interesting optimization techniques, including but not limited to monad optimizations, could be implemented in BioKleisli and are explained in [33]. A side-effect free COSWs could also undergo such optimizations. For this either the database techniques have to be adapted to some COSW language or a language used to express workflows has to be translated to a query language over nested collections, like NRC or based on it CPL and XQuery.

2. DFL DESIGNER

It is our opinion that both the data manipulation and control flow aspects are important in the context of COSWs and that there are many interesting ideas from workflow modeling and database research that can benefit the COSWs community. In this paper we present DFL designer, a novel COSW tool that is aimed to be a framework for adapting and introducing into the COSW setting of such ideas. For that, as the COSW definition language, we have chosen the DataFlow Language (DFL) which we developed in our previous work [8, 7].

DFL combines Petri nets with NRC into a new COSW language with a formally defined semantics. From NRC it inherits the set of basic operators and the type system. To deal with the synchronization issues arising from processing of the data by distributed services it uses the Petri-net based formalism which has a clear and simple graphical notation and for which an abundance of correctness analysis results exist. From one point of view DFL can be considered as a graphical notation for NRC and a way of extending NRC expressions into descriptions of processes, which allows for the side-effects to be taken into account. On the other hand it can be viewed as an extension of Petri nets with explicit data manipulation aspects, such that the data flows through the net and is processed. Thus, in the rest of the paper COSWs specified in DFL will be called *dataflows*.

With DFL designer we aim to address three important for COSW workbenches issues: easier understanding of details of the expressed experiment, better control of flow aspects and transformation of side-effect free COSWs into queries. It is developed as a plugin for the Eclipse[1] platform and is an open source project. It extends Eclipse with two new perspectives, the DFL Edit and DFL Run (see Figure 1), which define the user interface configuration best suited for designing and enacting of dataflows respectively.

Apart from the core DFL operations for data manipulation (for details see [8]) DFL designer is provided with a huge library of bioinformatics services gathered in the very popular Taverna 2 workbench which is also an open source project. Thanks to this it is easy to adapt already existing real life workflows designed with Taverna 2, e.g., the ones available in myExperiment Internet repository [6], and test on them the usefulness of novel features. Since doing such tests with real life examples is one of our main goals, we included the functionality of importing Taverna 2 workflows into DFL designer.

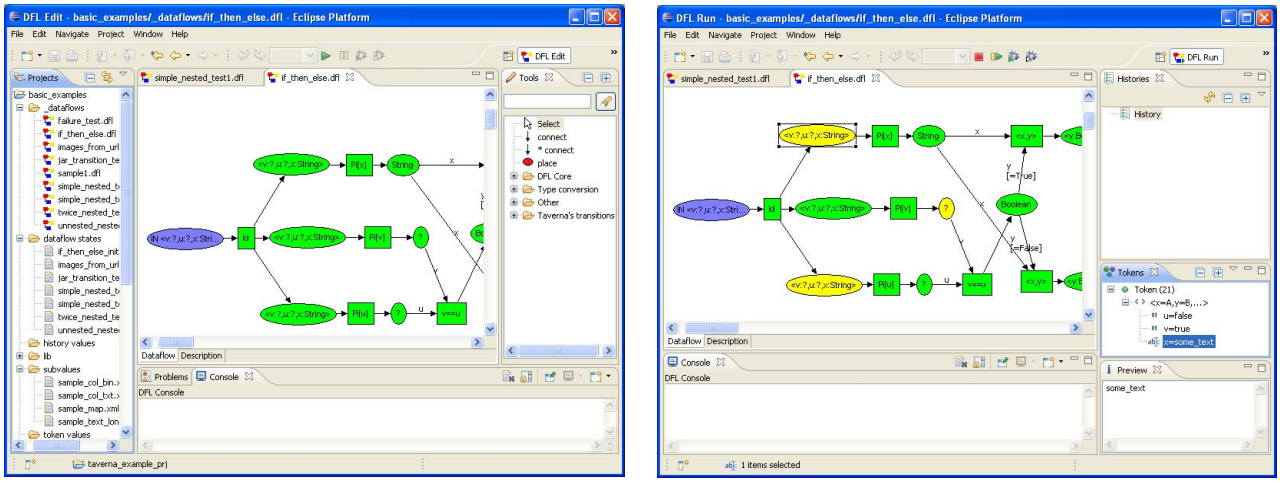
3. DFL DESIGNER FEATURES

In this section we describe the features of DFL designer that are in a COSW setting the most interesting and distinguishing. We organize the features into two groups: control flow oriented and data manipulation oriented. Because of the space constraints and for the accessibility of the presentation we provide only a limited number of examples. Further examples are available on the project web site at `http://code.google.com/p/dfldesigner/`. The site also contains installation instructions and multimedia materials tutorials.

3.1 Control-flow oriented features

COSWs are in fact distributed data processing systems in which data values can be transfered between web services in

[1]See `http://www.eclipse.org/`.



a) DFL Edit perspective

b) DFL Run perspective

Figure 1: DFL designer perspectives

adherence to some specific protocol and with possible synchronization constraints. Testing, debugging and often understanding of such distributed systems can be difficult and laborious.

Since DFL is based on a robust Petri net formalism, distributed protocols can be expressed in it very clearly. In fact the Petri net notation is often used in scientific publications to define the precise semantics of the control flow perspective of workflow systems, i.e., specify those systems explicitly and unambiguously, see for example [22, 12]. Transforming COSWs defined in other tools and notations into DFL designer can help users to understand them in detail. At the time of this writing DFL designer can import the Taverna 2 workflows with the exception of some aspects of the dispatch stack layers that in Taverna 2 can be associated with processors.

As for testing and debugging, the DFL Run perspective (see Figure 1) provides the *token game* functionality which is also common in Petri net tools. The abstraction of tokens transporting data and being transfered between places of the dataflow experiment allows for convenient step-by-step execution and enables the user to control the order of operations and what input values they consume. As in classical Petri net token game, the user can select which of the enabled transitions will fire next. Yet, since the tokens in DFL carry data values, an extra feature is present that allows to disable arbitrary tokens and thus determine not only which transition will fire, but also which data values it will consume. For user convenience it can be chosen whether the new tokens are produced as enabled or disabled. At every moment of the interactive execution the user has full control over the distribution of tokens and their values. It is possible to inspect and edit what tokens are in each place, what are their unnesting histories and what values they transport. The state of all places and the individual tokens can be saved to and loaded from an XML file.

For example, the user can influence the order in which elements of some collection are processed. This is possible thanks to the explicit iteration mechanism of DFL (see Figure 2), where a collection value can be first *unnested* by a specially marked edge, then the tokens representing the

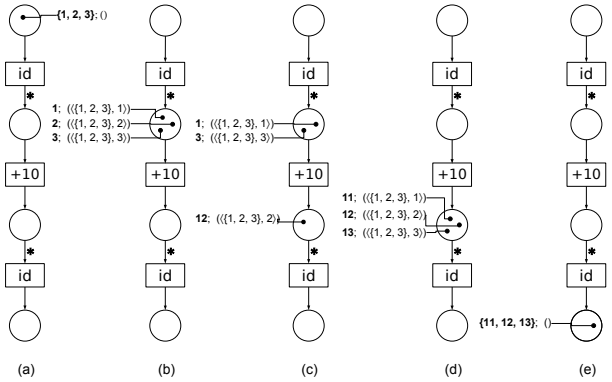


Figure 2: Abstract collection iteration example

collections elements are processed independently and finally are *nested* back into a collection. Determining if all the processed elements are ready to be nested is possible thanks to the *token unnesting history* mechanism of DFL where each token obtained from unnesting apart from its value carries a meta-data specifying to which element of the original collection does it correspond. This meta-data is a kind of token *provenance*. For further details on the token unnesting history mechanism see [8].

The interactive firing of transitions together with the full control over the state, allows the user to precisely understand the execution semantics of the defined dataflow and gives means to effectively debug and test them. In particular the user can: (1) check what values are returned by transitions and make sure that services represented by them behave as expected, (2) repeat the experiments from saved partial states, (3) enforce and test all possible variants of execution which is especially useful when complex synchronization protocol is being defined, and (4) experiment in a "what would happen if" way, by redefining the state during the execution.

The final feature of DFL designer presented in this section deals with automatic checking if constructed dataflows follow certain good practices and are thus not exposed to some

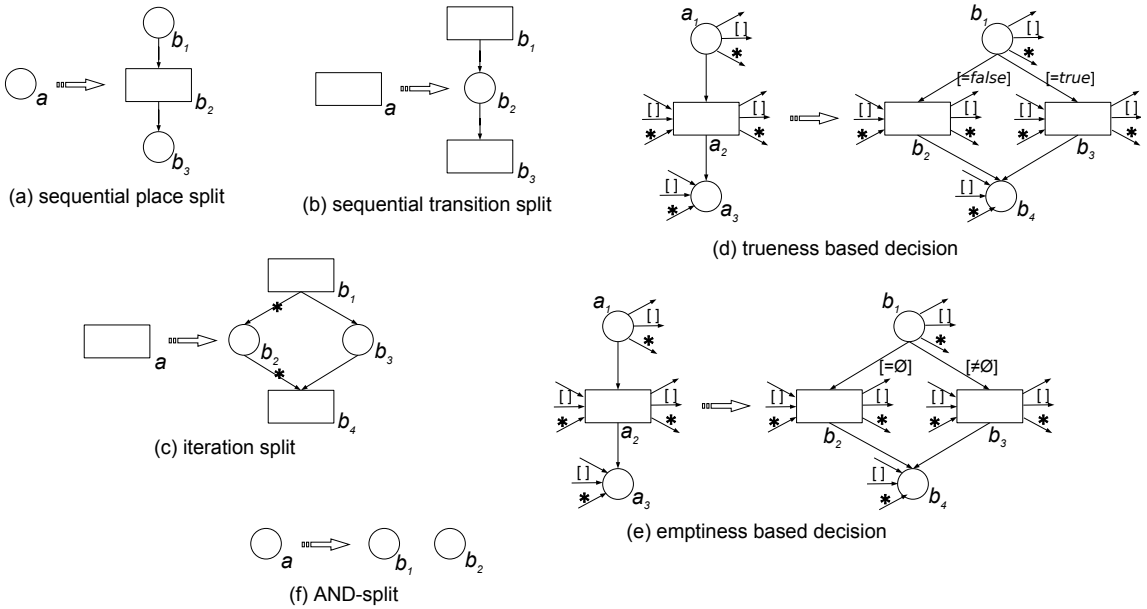


Figure 3: Refinement rules

typical problems. In [8] we adapted for DFL the *soundness* and *hierarchicality* notions known for Petri nets (see [3]). Dataflows constructed in a hierarchical manner, according to a set of refinement rules we proposed (see Figure 3), are *semi-sound*, i.e., initiated with a single token (which may represent a complex scientific data collection) in the input node, terminate with a single token in the output node (which represents the output data collection). In particular they never leave any "debris data" behind and an output is always eventually computed regardless of how the computation proceeds. DFL designer can automatically test by reversing the rules if dataflows are hierarchical, and if not, suggest the source of the problem.

Unfortunately not all semi-sound dataflows are hierarchical, but the majority of the ones used in practice can be represented hierarchically and are easier to comprehend this way. In [8] we have given an example of a real life dataflow which has some subtle flaws that are difficult to notice by a human reader. Those can be detected by checking the dataflow for hierarchicality and easily fixed by adhering to the refinement rules.

3.2 Data manipulation oriented features

Source code editors in modern integrated development environments for strongly typed languages, like the one used in Eclipse to edit Java code, prevent the programmers from making certain types of errors by checking if operations are applied to parameters of correct types and if the result is assigned to a variable of a matching type. This type of aid is build in DFL designer and prevents the construction of illegal dataflows in which the input and output types of connected services do not match. While connecting the nodes on the diagram, illegal constructions are prevented.

The enforcement of legality in the designer is more complex than for DFL itself. This is because in DFL the core operations are parametrized, i.e., there are multiple instances of every operation with different input and output types. Yet, in the designer we have made those operations polymorphic to free the user from the inconvenience of dealing with many variants of the same operation. By doing this the dataflows became polymorphic themselves and the types used usually cannot be determined exactly, but rather have to be expressed as patterns. Thus, while connecting the nodes on the diagram it is checked if corresponding patterns can be unified. For example, in Figure 4 we see a beginning of an example dataflow. Copies of the input value are projected on fields u, v and x, so the input value has to be a record with at least such fields. Similarly, because the operations $f()$ and $g()$ consume and produce a string value, then string is also the type of field x. Checking if the combined restrictions can be met together with the computation of patters for each place is a variant of the *type inference* problem [15, 19]. There are many results for type inference in industrial-strength functional programming languages like ML and Haskell. Some results are also available for a certain extension of NRC [29], but there the problem is proven to be NP-complete. Yet, for the polymorphic DFL the construction of a polynomial algorithm was possible thanks to: (1) a slightly different definition of the Cartesian product, which does not require that input values are sets of records with disjunctive sets of field labels, and (2) a simple equality test for only arguments of the same basic type. A comprehensive discussion of which particular operations make the

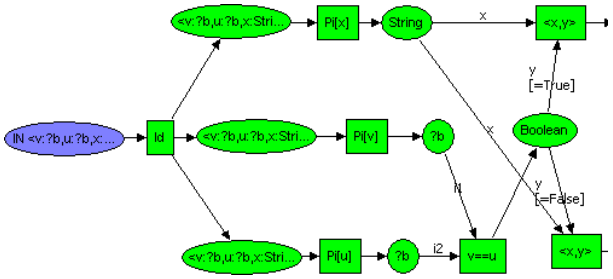


Figure 4: Beginning of an example DFL dataflow

type inference problem NP-hard can be found in [32].

Many tricks can be usually applied before executing a program, e.g., while compiling or interpreting it, to speed up its execution. Optimization techniques are also developed, applied and studied in the context of scientific workflows enactment, but most often this research concentrates on aspects shared with business workflows and grid techniques, e.g., which of many semantically equivalent services to choose [9] or how to execute the workflow on a distributed resources such as the Grid [5]. Since we emphasize the data processing aspects of COSWs, we are more interested in application of the query optimization results like the ones for NRC. It should be noted here, that the most effective optimization results for query languages in general and for NRC in particular depend upon algebraic identities that only hold if the involved operations are side-effect free. It is our observation that most operations used in COSWs are indeed side-effect free. Also the basic operators for which these algebraic identities are know, like products, are present in COSW languages, sometimes explicitly as in DFL and sometimes implicitly as in Taverna 2, so these identities can also be used here for optimization.

The application of the query optimization results to COSWs can be done by either developing similar results for COSW specification languages or by translating COSWs to some query language for which optimized execution engines exist. For hierarchical dataflows and for COSWs imported from Taverna 2 a mapping to NRC is possible. Also NRC has many stable execution engines [33], though by the time of this writing we have implemented the mapping algorithm but have not integrated DFL designer with an NRC execution engine yet. The mapping is achieved by an extension of the algorithm that checks if a dataflow is hierarchical. Such a check is done by reversing the refinements until either only one transition with a source and a sink place is left, i.e., the dataflow is hierarchical, or no further reversal is possible, i.e., the dataflows is not hierarchical. During the merging of dataflow nodes according to the reversed refinement rules, operations represented by transitions can be composed, so that in each step the dataflow as a whole computes the same function.

For example, in the case of the iteration split of rule (c), the function computed by b_4 can be provided on both of its arguments with the result of function computed by b_1. Thus, the transition on the right hand side which represents their combination computes the function $b_4 \circ pair_{k,l} \circ b_1$, where $pair_{k,l}$ returns a record with field labels k, l and the same value on both of them, i.e., the input value. Here the "$*$" annotated edges are unimportant since no processing occurs between the unnesting and nesting. If a structural recursion indeed takes place in the dataflow it is incorporated into the function of b_4 by the reversal of the sequential place split of rule (a).

The reversal process takes into account an additional rule that was not included in the rules for hierarchical dataflows, which allows it to deal with non-hierarchical dataflows where transactions are organized in an arbitrary directed acyclic graph, as is possible in Taverna.

4. CONCLUSIONS

In this paper we have presented DFL designer — a novel tool for specifying and enacting COSWs which uses a notation based on a well studied formalisms, namely Petri nets and NRC. Combining those two made it possible to implement in DFL designer many features like correctness enforcement, enactment optimization, and debugging and testing support that are unique in the COSW setting.

There are many interesting ways of extending DFL designer and continuing our research. In another paper of one of the authors [23] it was already shown that an XQuery [26] (a standard query language for XML data) engine can be integrated with a COSW workbench in such a way that parts of the workflow can be expressed as a query and this query can access the operations provided by the workbench. Similar integration with an NRC or XQuery engine is a natural next step in the development of DFL designer. It would also allow to enact the dataflow more effectively since hierarchical fragments of dataflows which use only side-effect free operations could be automatically translated into a query language and optimized. Another interesting topic is adding provenance support, so that information about how each data item has been computed, i.e., by which services and from what input values, is collected. This is important for scientists using COSWs for knowing how much they can rely on the data, i.e., if it is very reliable data obtained in laboratory or less reliable data resulting from several approximation algorithms applied subsequently. There already exist some work [11] connecting NRC with the Open Provenance Model [13]. We believe similar ideas can be adapted for DFL. Finally, further analysis algorithms can be designed for DFL by following the results available for Petri nets. For example, there are interesting results by Piotr Chrząstowski-Wachtel [2] showing that token distributions can also be checked for soundness, i.e., that starting from such a marking the computation can always be correctly completed regardless of how it proceeds.

Apart from testing new ideas and setting new directions DFL designer can be further developed in two ways: as a general use COSW tool and as a supplement for other existing systems. The first requires additional work on user friendliness, support of further domain specific operations and enriching the DFL notation with syntactic sugar and implicit features like an implicit iteration mechanism. The second requires the definition of a mapping from notations used in other systems to DFL such that COSWs defined in other tools could be automatically translated to DFL to take advantage of the analysis algorithms that are available for it.

Acknowledgments: The authors thank Łukasz Krupa and Paolo Missier for their contribution to the implementation.

5. REFERENCES

[1] P. Buneman, S. Naqvi, V. Tannen, and L. Wong. Principles of programming with complex objects and collection types. *Theoretical Computer Science*, 149(1):3–48, 1995.

[2] P. Chrząstowski-Wachtel. Determining sound markings in structured nets. *Fundamenta Informaticae*, 72(1-3):65–79, 2006.

[3] P. Chrząstowski-Wachtel, B. Benatallah, R. Hamadi, M. O'Dell, and A. Susanto. A top-down Petri net-based approach for dynamic workflow modeling. In W. M. P. van der Aalst, A. H. M. ter Hofstede, and M. Weske, editors, *Business Process Management*, volume 2678 of *Lecture Notes in Computer Science*,

pages 336–353. Springer, 2003.

[4] S. B. Davidson, G. C. Overton, V. Tannen, and L. Wong. BioKleisli: A Digital Library for Biomedical Researchers. *Int. J. on Digital Libraries*, 1(1):36–53, 1997.

[5] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. C. Laity, J. C. Jacob, and D. S. Katz. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005.

[6] C. A. Goble and D. C. De Roure. myExperiment: social networking for workflow-using e-scientists. In *WORKS '07: Proceedings of the 2nd workshop on Workflows in support of large-scale science*, pages 1–2, New York, NY, USA, 2007. ACM Press.

[7] J. Hidders, N. Kwasnikowska, J. Sroka, J. Tyszkiewicz, and J. Van den Bussche. Petri net + nested relational calculus = dataflow. In *OTM Conferences (1)*, pages 220–237, 2005.

[8] J. Hidders, N. Kwasnikowska, J. Sroka, J. Tyszkiewicz, and J. Van den Bussche. DFL: A dataflow language based on Petri nets and nested relational calculus. *Information Systems*, 33(3):261–284, 2008.

[9] L. Huang, A. Akram, R. Allan, D. W. Walker, O. F. Rana, and Y. Huang. A workflow portal supporting multi-language interoperation and optimization: Research articles. *Concurr. Comput. : Pract. Exper.*, 19(12):1583–1595, 2007.

[10] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. R. Pocock, P. Li, and T. Oinn. Taverna: a tool for building and running workflows of services. *Nucl. Acids Res.*, 34:W729–732, 2006.

[11] N. Kwasnikowska and J. V. den Bussche. Mapping the nrc dataflow model to the open provenance model. In J. Freire, D. Koop, and L. Moreau, editors, *IPAW*, volume 5272 of *Lecture Notes in Computer Science*, pages 3–16. Springer, 2008.

[12] N. Lohmann. A feature-complete petri net semantics for ws-bpel 2.0 and its compiler bpel2owfn. Technical report, In WS-FM 2007, LNCS, 2007.

[13] J. F. R. E. M. J. M. P. P. Luc Moreau, Juliana Freire. The open provenance model. Technical report, University of Southampton, 2007.

[14] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the Kepler system: Research Articles. *Concurr. Comput. : Pract. Exper.*, 18(10):1039–1065, 2006.

[15] J. C. Mitchell. *Foundations of programming languages*. MIT Press, Cambridge, MA, USA, 1996.

[16] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[17] National Center for Biotechnology Information. NCBI Blast. `http://www.ncbi.nlm.nih.gov/blast/Blast.cgi`.

[18] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, November 2004.

[19] B. C. Pierce. *Types and programming languages*. MIT Press, Cambridge, MA, USA, 2002.

[20] W. Reisig. *Petri nets: an introduction*. Springer-Verlag New York, Inc., New York, NY, USA, 1985.

[21] P. Rice, I. Longden, and A. Bleasby. EMBOSS: the European Molecular Biology Open Software Suite. *Trends in Genetics*, 16(6):276–277, June 2000.

[22] N. Russell, Arthur, W. M. P. van der Aalst, and N. Mulyar. Workflow control-flow patterns: A revised view. Technical report, BPMcenter.org, 2006.

[23] J. Sroka, G. Kaczor, J. Tyszkiewicz, and A. M. Kierzek. XQTav: an XQuery processor for Taverna environment. *Bioinformatics*, 22(10):1280–1281, May 2006.

[24] R. Stevens, H. Tipney, C. Wroe, T. Oinn, M. Senger, C. Goble, P. Lord, A. Brass, and M. Tassabehji. Exploring Williams-Beuren syndrome using ^{my}Grid. In *Proceedings of 12th International Conference on Intelligent Systems in Molecular Biology*, 2004.

[25] TGI group at the University of Hamburg, Germany. Petri nets tool database. `http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/db.html`.

[26] The World Wide Web Consortium. XML query working group public page. `http://www.w3.org/XML/Query`.

[27] University of Virginia. FASTA Sequence Comparison. `http://wrpmg5c.bioch.virginia.edu/fasta_www2/fasta_list2.shtml`.

[28] U.S. Department of Energy Human Genome Program. Genomics and its impact on science and society: The Human Genome Project and beyond. `http://www.ornl.gov/TechResources/Human_Genome/publicat/primer2001/index.html`.

[29] J. Van den Bussche and S. Vansummeren. Polymorphic type inference for the named nested relational calculus. *ACM Trans. Comput. Logic*, 9(1):3, 2007.

[30] W. M. P. van der Aalst. Three good reasons for using a Petri-net-based workflow management system. In S. Navathe and T. Wakayama, editors, *Proceedings of the International Working Conference on Information and Process Integration in Enterprises (IPIC'96)*, pages 179–201, Camebridge, Massachusetts, 1996.

[31] W. M. P. van der Aalst, K. M. van Hee, and G.-J. Houben. Modelling and analysing workflow using a Petri-net based approach. In G. De Michelis, C. Ellis, and G. Memmi, editors, *Proceedings of the second Workshop on Computer-Supported Cooperative Work, Petri nets and related formalisms*, pages 31–50, 1994.

[32] S. Vansummeren. On the complexity of deciding typability in the relational algebra. *Acta Informatica*, 41(6):367–381, 2005.

[33] L. Wong. Querying nested collections. PhD thesis, U.Pennsylvania, 1994.

[34] L. Wong. The collection programming language reference manual. `http://sdmc.iss.nus.sg/kleisli/psZ/cpl-defn.ps`, 1995.