Seamless Coarse Grained Parallelism Integration in Intensive Bioinformatics Workflows

Francois Moreews, Dominique Lavenier

Seamless coarse grained parallelism integration in intensive bioinformatics workflows

Francois Moreews
GENSCALE
IRISA/INRA
Rennes, France
fmoreews@irisa.fr

Dominique Lavenier
GENSCALE
IRISA/INRIA
Rennes, France
lavenier@irisa.fr

ABSTRACT

To be easily constructed, shared and maintained, complex in silico bioinformatics analysis are structured as workflows. Furthermore, the growth of computational power and storage demand from this domain, requires workflows to be efficiently executed. However, workflow performances usually rely on the ability of the designer to extract potential parallelism. But atomic bioinformatics tasks do not often exhibit direct parallelism which may appears later in the workflow design process.

In this paper, we propose a Model-Driven Architecture approach for capturing the complete design process of bioinformatics workflows. More precisely, two workflow models are specified: the first one, called design model, graphically captures a low throughput prototype. The second one, called execution model, specifies multiple levels of coarse grained parallelism. The execution model is automatically generated from the design model using annotation derived from the EDAM ontology. These annotations describe the data types connecting differents elementary tasks. The execution model can then be interpreted by a workflow engine and executed on hardware having intensive computation facility.

1. INTRODUCTION

Bioinformatics applications challenge today's computation resources by raising the amount of data to process and computation requirement to a new level. As an illustration, many algorithms used for Next Generation Sequencing (NGS) data processing, like genome assembly or polymorphism discovery, are computationally intensive and have to deal with a huge amount of data.

The common answer to such challenge is to use large storage facilities associated with classical computer clusters that combine the processing power of multiples machines. A job scheduler is then in charge of dispatching the processing demand onto the available processing resources. Such architecture takes advantage of coarse grain parallelism to speed-up computation. This parallelism can either be used by running multiple applications in parallel or by designing the application in such way that it can be divided into smaller grain tasks. Even if more and more tools like mpiBlast [5] use distributed computational resources like cluster-nodes or CPU-cores, the data parallelism pattern often need to be manually coded. For this purpose, APIs that eases the implementation of coarse grain data parallelism patterns have been developped [4].

Bioinformatics analysis usually consists in writing a script calling heterogeneous specialized softwares provided by the research community. Such "pipelines" are usually described as a sequence of operations represented as a dataflow. In that case, applications are modeled as a network of operations connected through their data dependencies. Such representations exploit the available parallelism by analyzing data dependencies between operations.

Workflow management systems (WMS) used in bioinformatics were often limited to educational or low throughput usage. With the development of middlewares designed to integrate the power of intensive computation infrastructure in client softwares, the orchestration of bioinformatics services deployed on a computation intensive production environment became realistic [15]. Among these middlewares, the DRMAA API [2] standardizes the access to job shedulers and the Opal toolkit [11] wraps command lines and manages job posting through web services. Thus, nowadays, WMS that enable cluster or cloud job submissions have emerged as an interesting solution to face the high throughput sequencing challenge.

Taverna [18] is a service oriented graphical workflow authoring and execution tool, able to orchestrate remote web services or local components. Associated with appropriate OPAL or PBS middleware clients embedded in actors, Taverna, but also Kepler [17], can be run on clusters. But many actors must be laboriously defined by the designer for a single intensive computation job execution (upload , submission, wait, get result, download). The resulting graph representation contains many technical actor nodes that do not clearly display the main analysis steps.

In contrast, Galaxy [8] has been successfully adopted by the scientific community as a bioinformatics production environment. Galaxy can integrate intensive computation tools and support a large collection of predefined bioinformatics components. It is used more as a tools repository where each task is independently manually launched than for its orchestration capabilities. Galaxy has proved to be a solution for workflow prototyping using, for example, a conversion of the user activities as a "pipeline". Galaxy workflow module, like

many dataflow based scientific WMS, proposes to ease the workflow specifications through a graph editing GUI and can schedule and run workflows, addressing clusters through job scheduler.

All these tools authorize the creation of atomic tasks, that internally manage parallelism. Thus, when available, coarse grain data parallelism is based on the implementation of each components. But these environments do not propose a way to integrate coarse grain data parallelism without hard coding or complex manipulations. They do not offer a general mechanism for data parallelism extraction.

In contrast, some generic grid computing oriented environments like P-GRADE [7] offer a unified access to multiple complementary middlewares, each one dedicted to a level of parallelism extraction. The configuration of these tools is user-defined. Building highly intensive bioinformatics workflow still relies on the technical ability of the designer to take advantage of the available data parallelism.

The purpose of this work is to greatly simplify the design process of bioinformatics workflows. Integrating a general mechanism of data parallelism extraction from a captured workflow prototype remains challenging. The goal is to hide to the designers (bioanalyst people) time consumming and error prone tasks dedicated to technical aspect of parallel implementation. This seamless parallelism integration could not only speed up the treatments but also facilitate the design process and disseminate the usage of parallel workflows.

To achieve this objective, we propose an approach based on a Model-Driven Architecture allowing the designer to easily capture bioinformatics workflows using a high level model from which a parallel execution model is semi automatically derived.

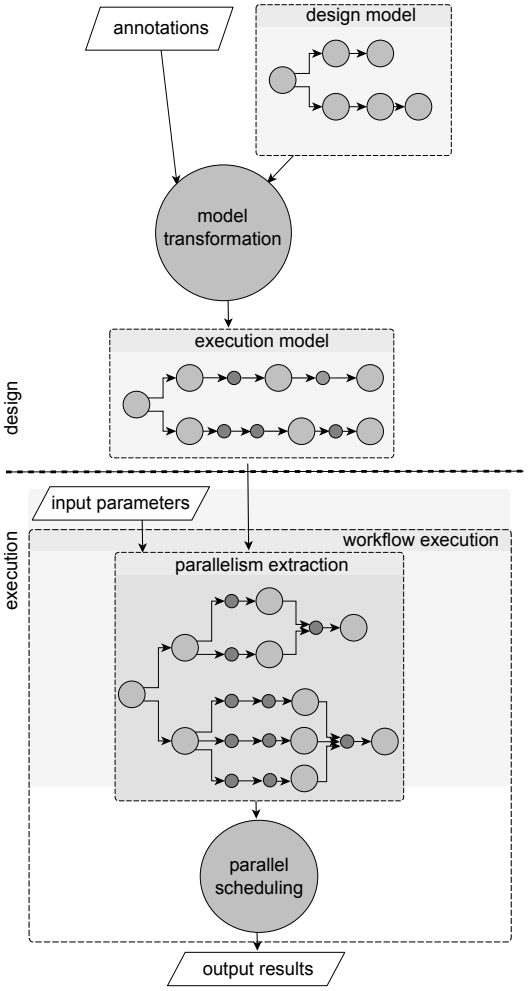

Figure 1: Overview of the workflow design and parallel execution steps

Figure 1 illustrates the approach: the user first specifies a workflow using a graphical interface. He connects bioinformatics tools as a dataflow graph. The user is asked to integrate annotations for specifying data types flowing between nodes (bioinformatics tasks) of the graph. From this graph, and using a model transformation, an execution graph is generated from which parallelism can be automatically extracted.

The rest of the paper is organized as follows: section 2 and 3 respectively present the workflow design model and the way the model is transformed. Section 4 describes the execution model. Section 5 discusses the integration of this Model-Driven approach in a new WMS dedicated to bioinformatics intensive data treatment and gives directions for future works.

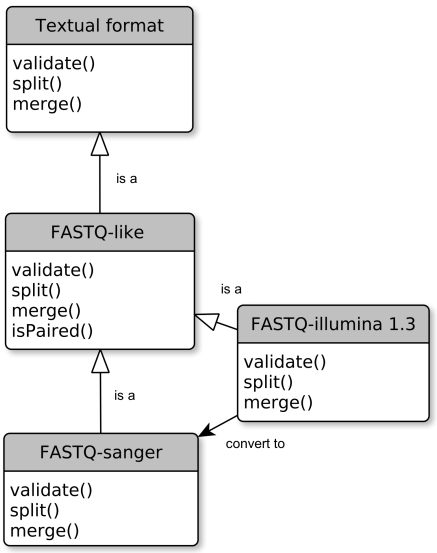

Figure 2: We use a data format hierarchy derived from EDAM ontology. Split and merge methods are implemented for each relevant data format.

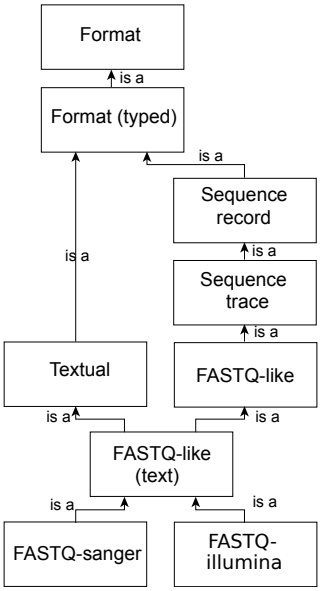

Figure 3: Example of the FASTQ-sanger data format hierarchy in EDAM (EMBRACE Data And Methods), an ontology of bioinformatics operations, types of data and formats.

2. THE DESIGN MODEL

To initiate the design process, we defined a "design model" that eases the capture of a worfklow, omitting technical tasks such as paralellization.

The workflow design model is a simple dataflow graph where processes are the nodes and the data dependencies the edges of a direct acyclic graph (DAG). Each node in the DAG is named actor. We use 3 majors classes of actors: input, execution, output. An execution actor wraps a script or a command line tool, that, without any additional semantic, will be seen as a black box, called here a user-defined function (UDF). Each edge represents a data dependency. An edge links a source actor output port to a target actor input port and represents a channel of data tokens. Actors can have multiple input and output ports. The scheduling aspects are implicit.

2.1 Prototype capture

During the prototype capture, only major processing steps are represented as UDF actors. Utilitary tools which perform operations related to data validation, transformation and conversion methods and which do not aggregate or generate additional knowledge are called here Data Format Function (DFF). DFF actors must be omitted during the prototype capture. Pre-existing tools and scripts that fit theses conditions are embedded within UDF actors following a template syntax.

2.2 Annotations

The input and output ports of the UDF actors need to be partially or fully annotated by the designer using a data format hierarchy provided by the framework (Figure 2). This additional semantic defines a strong data typing that enables the integration of DFF actors.

Predefined DFF actors are proposed to the designer as pre-processing and post-processing methods applied to the ports of an actor node. It is similar to a sequence of methods applied on the edges of the dataflow DAG. The resulting separation of workflow tasks between UDF and DFF permits to display a clear workflow "design" view (Figure 4). This "analysis" or "design" view represents only the input, output and UDF actors, related to the domain tasks (here bioinformatics methods...). DFF actors (validation, conversion and other utilities) remains masked. This results in a better overview and semantic analysis of the workflow. It is also a way to limit the proliferation of visible technical actors or data adaptors, sometimes called the "shim problem" [16]. We obtain a complete semi concrete workflow model that will subsequently be transformed in a fully executable model. We have generated the domain-specific data format hierarchy, derived from the EDAM ontology [9]. EDAM (EMBRACE Data And Methods) is an ontology of bioinformatics operations, types of data and formats (Figure 3). The vocabulary of terms and relations provided have already been used to classify tools and also for workflow composition purpose [13].

3. MODEL TRANSFORMATION

Model transformation is used in Model-Driven Architecture for code generation including automatic parallelisation [12]. It is commonly used in graphical capture of processes and has already been applied to workflow formats conversion [6]. We are not aware of any previous use of this method for a concrete coarse grain parallelisation of workflows.

Basically our approach depends on the prior definition of a library of efficient split and merge methods, related to most common data format types used in the application domain. To efficiently organise this set of utilities, the format hierarchy previously introduced is used (Figure 2). This set of utilities is manually created and associated to related formats within the format hierarchy.

For each format and its variants, a custom set of functions is defined including the implementations of the split and merge methods, with appropriated parameters. Split and merge methods are DFF. For an actor port tagged with a data format, all methods related to the data format and their predecessors in the hierarchy can then be used to semantically annotate the related edges of the design model (Figure 4-A). This annotation process is user-defined.

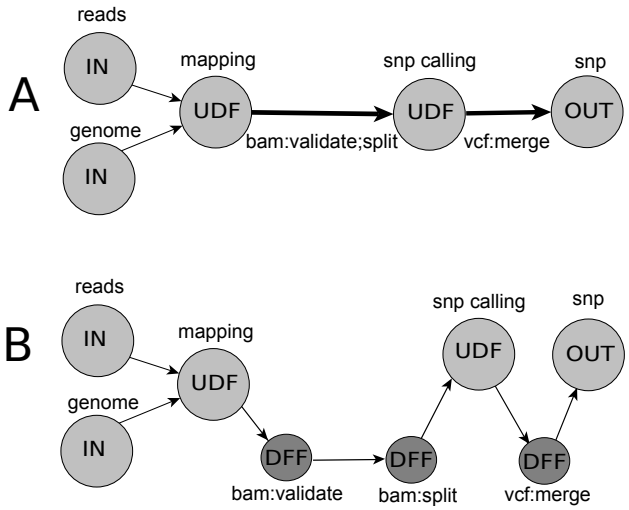

Figure 4: The definition of UDF and DFF actors enables the creation of two different dataflow views dedicated to design (A) or execution and monitoring (B).

A graph transformation mecanism, based on graph patterns, converts the annotated design model to an execution model, after checking the constraints. To each model corresponds a view, the "design view" (Figure 4-A) and the "execution view" (Figure 4-B). The "execution view" is a technical view where all actors are represented. UDF and DFF actors are all represented as nodes. This view is closer to the implementation and useful for monitoring execution. Because all DFF are generated as new actors in the execution model, the split and merge actors methods become represented as actors. This means that when input and output port data formats have been specified by user-defined annotations, a map reduce pattern is consequently applied using split and merge methods related to each data format (Figure 2).

Finally, the execution of the resulting model by a dedicated engine corresponds to the execution of a parallelised implementation of the captured workflow.

4. THE EXECUTION MODEL

The execution model enables the parallelism extraction and a high level of task scheduling. The execution model is also a dataflow with input, execution, and output actors (Figure 5-A), but each actor has 6 different states (undefined, initialized, submitted, done, error, finalized) (Figure 5-B). The states are defined to support asynchronous remote calls, a required pattern for long running job submissions. At the transition between states, a specific method is launched. The transition between the state "submitted" and the state "done" corresponds to the UDF, here a bioinformatics tool. Each edge represents a data dependency. An edge links an output port of a source actor to an input port of a target actor and represents a channel of data tokens. Actors can have multiple input and output ports. Each workflow input is represented as a list of tokens. Conditional statements like "if else" structures can be represented using the propagation of a null token.

A dataflow model consumes and emits data tokens [14]. Two data token classes have been defined, a simple string for short values and a dataset Uniform Resource Identifier (URI). A dataset URI is an identifier that abstracts a file or a group of related files, like all files generated by an execution. This unified data container can seamlessly be used by the orchestration layer that manipulates abstract data tokens or by the services called internally by actors for data movement and the generation of data subsets.

An actor can emit one job output URI per invocation. In other words, each execution of an actor returns one URI for one command execution result set. At the scheduler level, this means that the number of tokens is predictable, allowing static scheduling and compliance with the Synchronous Data Flow model (SDF) [14].

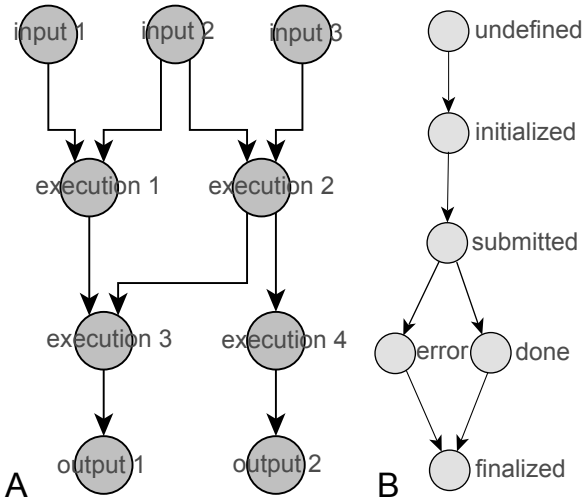

Figure 5: The execution model corresponds to a dataflow DAG (A) where execution actors wrap user-defined functions (UDF). Each actor has different states (B) to enable the control of asynchronous remote calls, directly during the orchestration.

4.1 Parallelism extraction

We focused on coarse grained parallelism. Defining implementation specific parallelism is out of the scope of this work but existing ones can be wrapped as UDF actors. For example, MPI or GPU implementation can be called using a particular job scheduler queue defined by an actor property.

We now explains how the execution model specifically targets different complementary levels of parallelism using:
(i) iterations (ii) data dependencies (iii) map-reduce

4.1.1 Iterations

All elements of a list of input tokens can be submitted in parallel. When an actor depends on multiple parameters (ports), each one linked to a list of input tokens, we can compute all parameter sets corresponding to all independent job submissions. As an example, it is similar to multiple parallel workflow instances execution with different parameter sets.

The sequence of all parameter sets U of an actor is obtained by computing a cartesian product of all lists of input tokens connected to its input ports (Figure 6). In the case of an UDF actor *F1* with two input ports, respectively populated with the lists of input tokens R and S, it corresponds to the generation of all possible pairs formed from the two lists, $U = R \times S$. The resulting jobs are $F1(U)$. Each element of U is a set of parameters which can be applied to the function *F1* in parallel.

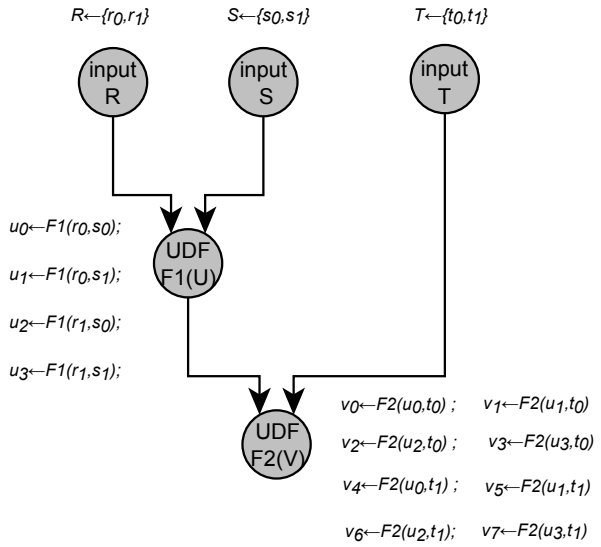

Figure 6: Static computation of UDF actors job parameter sets.

4.1.2 Data dependencies

Parallel executions among workflow actors is obtained by exploiting the graph topology. Following the dependency graph, any actor can be fired when all its predecessors have been fired. Each actor n associated with a set of parameters represents a job submission s. The current states of all job submissions are stored in a synchronized data structure. Reading the data structure holding the state of each defined job submission and the states of the related predecessors allows the scheduler to dynamically identify all job submissions that can be illegible for execution (algorithm 1). Then jobs can be submitted in parallel to a cluster job scheduler, following other implemented constraints, defined by the sheduling strategy.

Algorithm 1 parallelism extraction obtains from dynamic scheduling using the data dependencies defined by the DAG topology.

```
D ← setWorklowGraph() //workflow DAG
A ← getAllJobSubmissions(D)//all job submissions
for each n in D do
  if (n is UDFActor ) then
    P ← getPredecessors(n) //get all predecessors of n
    J ← getJobSubmissions(n) //get reserved job
    // submission array
    for each s in J do
      if (s is not submitted ) then
        for each m in P do
          t ← getLinkedJobSubmission(m, s)
          //each submission of n is related to another
          //job submission of the predecessor actor m
          if (t is finished ) then
            A(s) ← TRUE //Job is defined as available
            // for future launch
for each s in A do
  if (A(s) is TRUE ) then
    if (externalSchedulingCondition(s) is TRUE ) then
      p ← new thread()
      e ← fire(p, s) //execute the job submission in a
      //new thread. Its results in multiple parallel
      //job executions on the target cluster
```

4.1.3 Map-Reduce patterns

In many application cases, a simple Map-Reduce pattern leads to the extraction of a massive coarse grain data parallelism from the workflow prototype. As previously defined, split and merge methods are Data Format Function (DFF). Within the execution model, DFF actors do not differ from UDF actors. It means that after model transformation, according to the user-defined annotations, a map reduce pattern is automatically inserted in the dataflow. For example, the actor node $UDF(F1)$ is replaced by a sub graph :

$$split \rightarrow UDF(F1) \rightarrow merge$$

A split method and a merge method can be applied on the same UDF actor on different UDF actors. In this last case, all data chucks are simply grouped using a shared dataset URI, and, thanks to this abstract data token, transparently routed in the dataflow graph.

5. CONCLUSION

We have implemented a workflow engine based on our models. This engine can be integrated in a graphical WMS dedicated to intensive computation [1]. It generates an execution model from the design model, previously created with a web workflow design GUI (Figure 7), and subsequently executes the workflow [3].

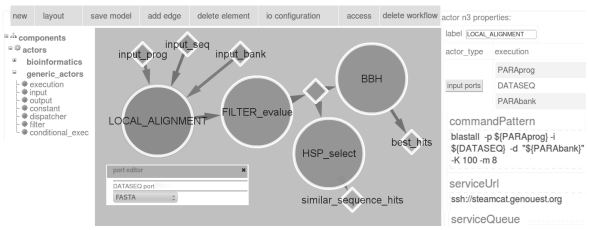

Figure 7: Web workflow design GUI. The DAG corresponds to an intensive bioinformatics workflow captured using a graphical editor. For execution, the underlying workflow engine integrates a dedicated middleware, SLICEE (Service Layer for Intensive Computation Execution Environment), embedded in actors

In this paper, we have proposed a modeling process of intensive bioinformatics workflows, closely related to the actual observed development process. The transformation of an annotated design model to an execution model enables the extraction of different levels of coarse grain parallelism with minimum human intervention. By this way, the conversion of a low throughput prototype to a workflow adapted to intensive computation can be more easily achieved. It could also result in a larger dissemation of coarse grain parallelism usage in bioinformatics treatments, especially within popular graphical WMS dedicated to non-technical users. In addition, we have defined a minimal classification of the workflow actors in two classes (UDF, DFF) that clearly highlights what can be automated or must stay user-defined.

However, the effectiveness of our approach depends on the structuration of a large corpus of utilitary methods related to bioinformatics types and formats. Major efforts still need to be made in those directions, especially, to study if the annotation process of the design model, actually user-defined, could be automated. It would also be worth to investigate how model transformations could be used to seamlessly target a larger scope of execution environments [10].

6. REFERENCES

[1] Bioinformatics Intensive Workflow Portal. *http://workflow.genouest.org*.

[2] Drmaa working group. *http://www.drmaa.org/*.

[3] Slicee - Service Layer for Intensive Computation. *http://vapor.gforge.inria.fr/*.

[4] P. Bui, L. Yu, and D. Thain. Weaver: integrating distributed computing abstractions into scientific workflows using python. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 636–643, New York, NY, USA, 2010. ACM.

[5] A. Darling, L. Carey, and W.-c. Feng. The design, implementation, and evaluation of mpiBLAST. In *Proceedings of ClusterWorld Conference*, pages 13–15, San Jose, California, June 2003.

[6] J. Eder, W. Gruber, and H. Pichler. Transforming workflow graphs. In *Proceedings of the first international conference on interoperability of enterprise software and applications INTEROP-ESA 2005 ,Geneva, Switzerland*, pages 203–215, 2005.

[7] Z. Farkas and P. Kacsuk. P-grade portal: A generic workflow system to support user communities. *Future Generation Computer Systems*, 27(5):454 – 465, 2011.

[8] J. Goecks, A. Nekrutenko, J. Taylor, and T. Team. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biol*, 11(8):86, 2010.

[9] J. Ison, M. Kalas, I. Jonassen, D. Bolser, M. Uludag, H. McWilliam, J. Malone, R. Lopez, S. Pettifer, and P. Rice. EDAM: an ontology of bioinformatics operations, types of data and identifiers, topics and formats. *Bioinformatics*, 29(10):1325–32, May 2013.

[10] L. Jourdren, M. Bernard, M.-A. Dillies, and S. Le Crom. Eoulsan: a cloud computing-based framework facilitating high throughput sequencing analyses. *Bioinformatics*, 28(11):1542–3, June 2012.

[11] S. Krishnan, B. Stearn, K. Bhatia, K. Baldridge, W. Li, and P. Arzberger. Opal: Simpleweb services wrappers for scientific applications. In *Proceedings of ICWS '06. International Conference on Web Services, 2006.*, pages 823–832, 2006.

[12] O. Labbani, J. luc Dekeyser, P. Boulet, and E. Rutten. Introducing control in the gaspard2 data-parallel metamodel: Synchronous approach. In *Proceedings of the 1st International Workshop on Modeling and Analysis of Real-Time and Embedded Systems*, 2005.

[13] A.-L. Lamprecht, S. Naujokat, B. Steffen, and T. Margaria. Constraint-guided workflow composition based on the edam ontology. In *Proceedings of the Workshop on Semantic Web Applications and Tools for Life Sciences*, 2010.

[14] E. Lee and D. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Transactions on Computers*, 36(1), Jan. 1987.

[15] W. Li, S. Krishnan, K. Mueller, K. Ichikawa, S. Date, S. Dallakyan, M. Sanner, C. Misleh, Z. Ding, X. Wei, O. Tatebe, and P. Arzberger. Building cyberinfrastructure for bioinformatics using service oriented architecture. In *Proceedings of the sixth IEEE International Symposium on Cluster Computing and the Grid, 2006. CCGRID 06.*, volume 2, pages 8 pp.–39, 2006.

[16] C. Lin, S. Lu, X. Fei, D. Pai, and J. Hua. A task abstraction and mapping approach to the shimming problem in scientific workflows. In *Proceedings of SCC '09. IEEE International Conference on Services Computing, 2009.*, pages 284–291, 2009.

[17] 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. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065, Aug. 2006.

[18] T. Oinn, M. Greenwood, M. Addis, M. N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M. R. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe. Taverna: lessons in creating a workflow environment for the life sciences. *Concurrency and Computation: Practice and Experience*, 18(10):1067–1100, Aug. 2006.