Computer-Assisted Scientific Workflow Design

Nadia Cerezo, Johan Montagnat, Mireille Blay-Fornarino

Journal of Grid Computing manuscript No.
(will be inserted by the editor)

Computer-assisted Scientific Workflow Design

Nadia Cerezo · Johan Montagnat · Mireille Blay-Fornarino

Received: date / Accepted: date

Abstract Workflows are increasingly adopted to describe large-scale data- and compute-intensive processes that can take advantage of today's Distributed Computing Infrastructures. Still, most Scientific Workflow formalisms are notoriously difficult to fully exploit, as they entangle the description of scientific processes and their implementation, blurring the lines between what is done and how it is done as well as between what is and what is not infrastructure-dependent.

This work addresses the problem of data-intensive Scientific Workflow design by describing scientific experiments at a higher level of abstraction, emphasizing scientific concepts over technicalities, easing the separation of functional and non-functional concerns and leveraging domain knowledge.

To achieve this goal, we propose a model-driven approach enhanced with Knowledge Engineering technologies. The main contributions of this work are a semantic Scientific Workflow model to capture user goals and a generative process assisting the transformation from high-level models to executable workflow artefacts.

Keywords Scientific Workflow Design · Workflow Modeling · Workflow Composition · Semantic Workflow

1 Introduction

Simulations, also known as "*in-silico*" experiments in the field of life sciences, are scientific experiments partially or entirely carried out via computers. In many fields, these experiments have become a major component of scientific research. Many factors contribute to the necessity of automating such experiments, most notably the volume of data to analyze and the exploratory nature of the analysis,

N. Cerezo · J. Montagnat · M. Blay-Fornarino
Université Nice Sophia Antipolis / CNRS, I3S laboratory, Sophia Antipolis, France
Tel.: +33-492965103
Fax: +33-492965155
E-mail: cerezo@i3s.unice.fr johan@i3s.unice.fr blay@polytech.unice.fr
http://modalis.i3s.unice.fr

which leads to frequent reuse and repurposing. For years, scientists have chained the various programs that composed their simulations through scripting. However Distributed Computing Infrastructures (DCIs) (*e.g.* service platforms, grids, clouds) make this hands-on ad-hoc approach very impractical: scientists find they need to become experts of distributed algorithms and web technologies in order to use the wealth of available distributed resources.

1.1 Workflows

In the corporate world, the need to automate the use of highly-distributed heterogeneous resources was answered by the concept of workflow, formally defined by the Workflow Management Coalition (WfMC) as "*the computerized facilitation or automation of a business process, in whole or part*". Originally geared towards the description of business processes, workflows have been increasingly used to describe scientific experiments, especially those performed on DCIs. Workflows meant to perform simulations are called Scientific Workflow.

The frontier between Business and Scientific Workflows is rather blurry and subjective. Nothing prevents a user from using a Business Workflow framework to model and perform a scientific experiment or a scientific workflow framework to capture and automate a business process. As detailed in [2] as well as in [3,4], the differences pertain essentially to priorities and context, notably:

- The need for **security and privacy**, which is extremely important in a business context, is still present but much less prevalent, in the scientific community, where peer validation and collaboration are common goals that imply **sharing, reuse and repurposing**.
- The need for **integrity and reliability** is a central aspect of business services and thus a top priority for Business Workflows, but the exploratory nature of research makes **flexibility** a much greater priority for Scientific Workflows.
- Many business contexts require the level of fine-grained control and flexibility provided by **control-driven** models (*i.e.* models relying on control constructs such as loops and conditionals). However, scientific data is most often the first-class citizen of a simulation, which makes **data-driven** models a better fit for most Scientific Workflows. Moreover, those models can leverage data parallelism implicitly [5], which is crucial to many data- and compute-intensive simulations.
- On the one hand, Business Workflow designers often face either a lack of suitable candidate services to perform a step in their process or a wealth of functionally close candidates which must be differentiated through considerations of Quality of Service and cost. On the other hand, Scientific Workflow designers often start modeling their simulations with the main services/programs already determined and generally find very few viable alternatives, since different candidates for a scientific process step often pertain to substantially different scientific approaches.

This work focuses on the design of Scientific Workflows. While the method proposed is generative and typical Business Workflow languages could be considered as targets, it is important to note that several hypotheses underlying this work derive from typical Scientific Workflow models and frameworks.

1.2 Scientific Workflow Models

Most Scientific Workflow models are, at their core, directed graphs whose nodes are activities (*i.e.* executable artefacts) and whose edges represent dependencies between these artefacts (*i.e.* control and data transfers). That core representation fits most Scientific Workflow models, see [6,7,8,9,10] for a few examples. There are other types of Scientific Workflow models, for instance those based on scripting [11] (graphical and scripting representations are highly similar and two-ways conversion is conceivable [12]) or petri nets [13].

We will, however, only consider graph-based Scientific Workflow representations where nodes represent activities and edges represent data or control dependencies, much like other works that consider Scientific Workflows at a computation-independent level, such as [14] and [15].

Typical graph-based workflow representations can be divided between Directed Acyclic Graphs (DAGs) (*e.g.* Taverna [6] and Pegasus [9]) and Directed Cyclic Graphs (DCGs) (*e.g.* Kepler [9] and MOTEUR [10]). Cycles are a common way to model loops, which is why our representation is not limited to DAGs.

1.3 Motivation

Workflow formalisms are appealing to scientists in that they provide means to formally describe complex experiments involving parallel or distributed processing. End-users of Scientific Workflows have come to expect a representation that is not too tightly coupled with an execution infrastructure, that is accessible and eases the design and implementation of scientific experimental protocols. Yet, despite laudable efforts to fulfill user wishes, it is widely recognized that most existing Scientific Workflow formalisms remain complex to use for scientists who are not experts in distributed algorithms [16,17].

Another important issue is the Separation of Concerns (SoC). Indeed, regardless of their scientific field and framework of choice, Scientific Workflow designers, in order to make executable workflows, will have to clutter their simulations with countless intermediary steps managing data and/or catering to non-functional concerns (*e.g.* optimization, reliability and security). This entanglement hinders sharing, reuse and interoperability by blurring the lines between goals and methods and binding the simulation to a given infrastructure.

One way to answer those problems would be to model simulations at an abstraction level that is closer to the end-user's domain(s). This solution is increasingly appealed for in the field of Scientific Workflows: quoting from [18], "*conceptual workflow modeling can provide a number of benefits to workflow designers by allowing workflows to be developed at a higher level of abstraction (e.g. without having to worry about implementation details), for workflow discovery... workflow reuse and interoperability [and] given the complexity of data, control-flow, and dataflow aspects of most Scientific Workflows, formal conceptual workflow modeling frameworks that integrate each of these aspects would have the potential to significantly help users design and implement complex scientific analyses.*"

Let us define what "*conceptual*" means by clarifying the distinction between Scientific Workflow abstraction levels.

1.4 Abstraction Levels

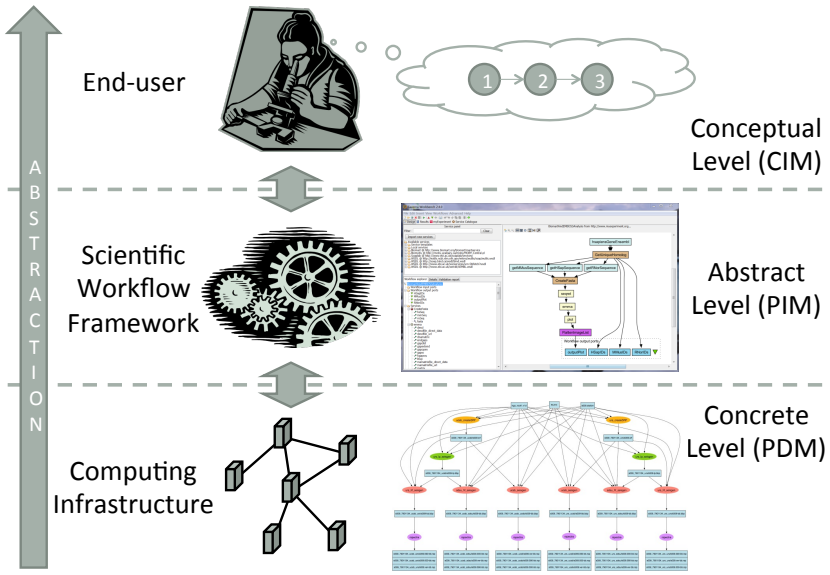


Fig. 1 Scientific Workflow Abstraction Levels (color online)

Simulations are modeled at three distinct levels of abstraction that align with those defined in the Model Driven Architecture (MDA), as shown in Figure 1:

- The Concrete Level is that of actual execution by an enactor over a DCI. At this level, models are tightly-coupled with the infrastructure and are referred to as Platform-Dependent Model (PDM) in the MDA.
- The Abstract Level is that of most Scientific Workflow models, ready to be automatically compiled or directly interpreted, but not entirely bound to specific resources and retaining some flexibility. Models at this level aim to be independent from computing infrastructures and are referred to as Platform-Independent Model (PIM) in the MDA. In practice though, frameworks end up tied to target infrastructures enough to warrant interoperability projects like SHIWA[1].
- The Conceptual Level is the one at which scientists conceive their scientific experiments in a vocabulary that is familiar to them. Conceptual models are referred to as Computation-Independent Model (CIM) in the MDA, for they remain independent from how the system is or will be implemented.

The distinction we make between **Abstract** and **Concrete** is nothing new [19]. But, as of yet, there is no consensus on the name of the highest level of abstraction: what we call **Conceptual** is even called "*Abstract*" in some works [14]. The distinction between CIM and PIM is much clearer in the MDA than it is in the

[1] SHIWA Platform: `http://www.shiwa-workflow.eu/`

field of Scientific Workflows, but few works have touched upon the transition from one to the other [20].

1.5 Contribution

The present work extends the preliminary work described in [21] and addresses the problem of Scientific Workflow design. Instead of building an entire scientific workflow framework, from design to enactment, our approach is to formalize simulations at a higher level of abstraction, upstream from already plentiful Scientific Workflow models, then transform those high-level models into low-level workflows, readily executable by existing systems. In that, we fit the gap described in [16] as the need for distinct "*dimensions of abstraction [that] are experiment-critical versus non-experiment-critical representations, where the former refers to scientific issues and the latter is more concerned with operational matters.*"

Our model-driven approach relies on Knowledge Engineering technologies to handle domain-specific information and automate the generative process as much as possible. The use of semantic metadata attached to scientific computing processes guides the semi-automated transformation process and enhances accessibility by emphasizing scientific goals over technical issues.

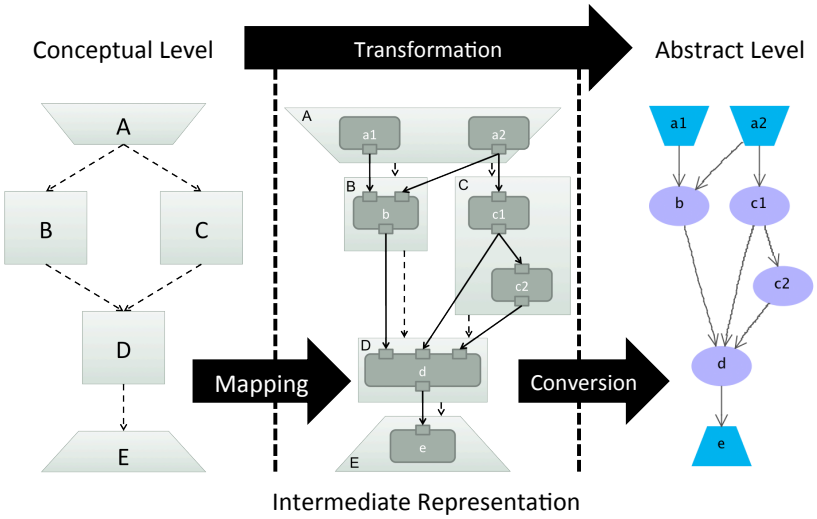


Fig. 2 Transformation Process (color online)

The transformation from a Conceptual to an Abstract workflow involves two steps as illustrated in Figure 2. The first step, **Mapping**, is the computer-assisted transformation from a simulation modeled at the Conceptual Level into an Intermediate Representation that contains both conceptual and abstract elements, allowing us to maintain traceability between the two representations. The **Conversion** step is then needed to transform the workflow from the Intermediate Representation to a target Abstract Workflow language, so that execution can be

delegated to an existing scientific workflow framework. We built a working prototype for the **Conversion** step that transforms Intermediate Representations into GWENDIA [5] workflows. The focus of this paper, however, is the **Mapping** step.

Most examples presented in this paper come from the Virtual Imaging Platform (VIP) project [22], whose goal is the integration of multiple modalities and organ models into a cohesive medical image simulation platform.

1.6 Outline

The **Conceptual Workflow Meta-model** is shown in Unified Modeling Language[2] (UML) on Figure 3. It is divided in three parts: `Semantic`, `Conceptual` and `Abstract`. The following Section 2 describes the `Semantic` part, and details which related technologies we use in our approach as well as why and how we use them. Section 3 describes the `Conceptual` part, meant for high-level simulation modeling. Section 4 describes the `Abstract` part, which defines the low-level elements needed for implementation. Those elements are either provided by the user or retrieved from the Knowledge Base during the Mapping transformation step described in Section 5. Section 6 illustrates the Transformation process on a real use case. Section 7 addresses the specific issues of "*cold-start problem*" and "*meta-data-based discovery*". Related works are discussed in Section 8 and we conclude this paper with perspectives in Section 9.

2 Semantics

One issue with Abstract Workflow representations, regarding simulation modeling, is that they assume a rather close relationship between the description of a scientific process and its implementation. There is a one-to-one mapping from operations in the process modeled in the Abstract Workflow by processing nodes (called Activities in ASKALON [23], Actors in Kepler [9], Processors in MOTEUR [10], Tasks in Pegasus [7], Services in Taverna [24], Units/Tools in Triana [25] and so on) to executable artefacts (e.g. services, executable binaries, grid jobs, sub-workflows). Because those processing nodes are black boxes, there is still some flexibility in that a single node or a pre-defined construct, such as a Taverna "*Layer*" [26], may hide a complex sub-workflow, but there is purposely little structural flexibility.

Abstract Workflows are not only bound to a specific set of executable artefacts and a specific target execution infrastructure, they also indiscriminately mix domain concerns (*i.e.* high-level concepts pertaining to user scientific domain(s)), technical concerns (*i.e.* functional in that they are needed to actually enact the workflow, but not directly relevant to the modeled scientific experiment) and non-functional concerns (*e.g.* Quality of Service considerations). There is simply no difference between a major scientific algorithm, a small format conversion step and a logging operation: all steps are treated equally, which makes it harder to distinguish goals from methods and thus lowers legibility.

The **Conceptual Workflow Model** addresses the shortcomings previously mentioned by separating elements into two abstraction layers: a high-level one

[2] UML: `http://www.omg.org/spec/UML/Current`

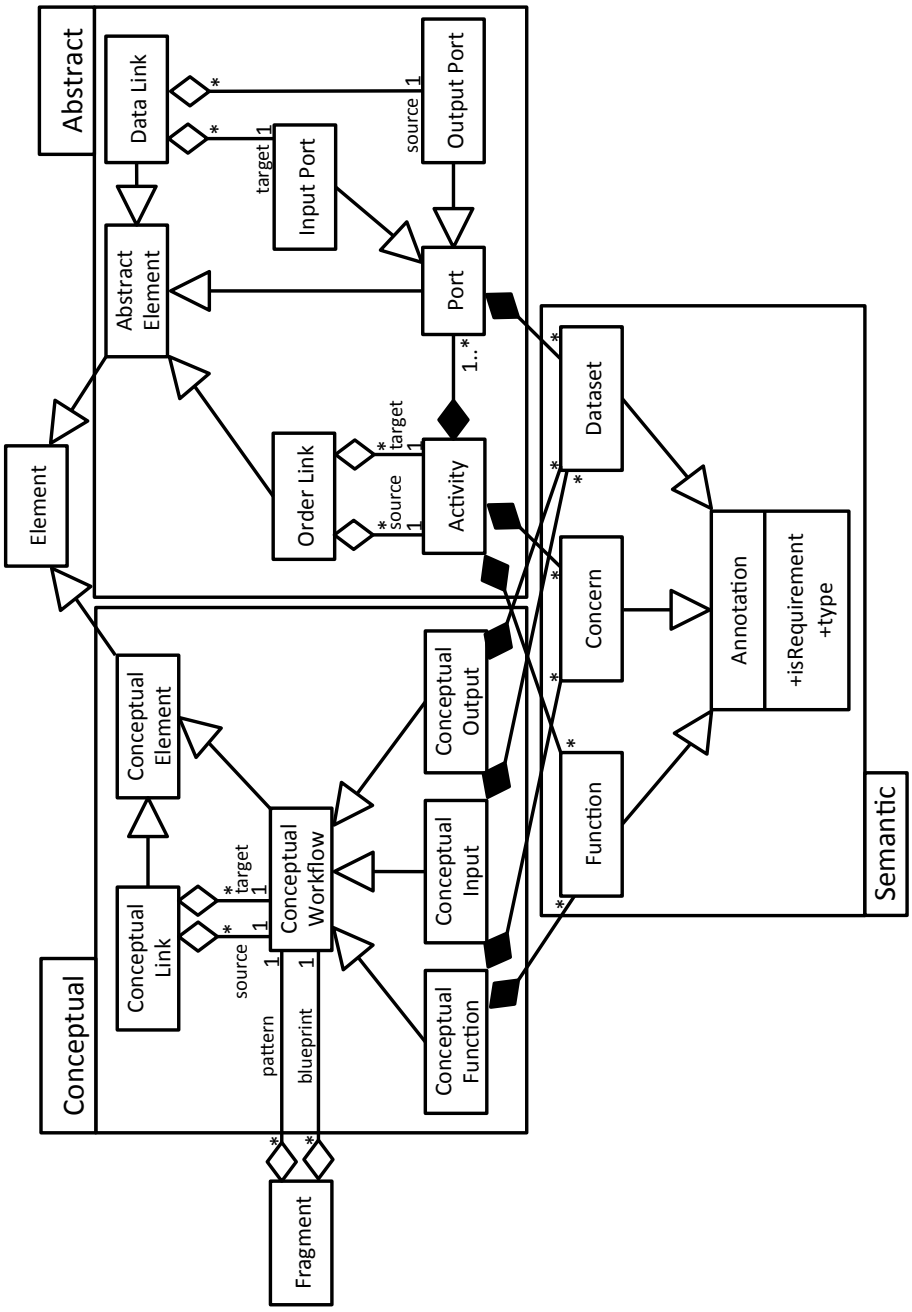


Fig. 3 Conceptual Workflow Meta-model

where there are only functional considerations that pertain to the user domain(s) and a low-level one where nodes are bound to artefacts, much like in Abstract Workflow formalisms. The two levels are integrated into one model so as to provide a coherent and complete view of a simulation, without losing sight of the underlying scientific process.

To capture the scientific process itself and assist the user in implementing it implies handling knowledge and know-how from multiple domains, in a computer-legible and yet accessible way. Semantic Web technologies and tools were created precisely to address such problems of knowledge representation production [27].

2.1 Standards

At the basis of the Semantic Web is the notion of resource, *i.e.* a concept identified by a Uniform Resource Identifier (URI). Resources are described and connected through `subject predicate object` triples, *i.e.* assertions that the resource `subject` has the property `predicate` with the literal or resource `object` as value. Triples can be stored in dedicated repositories called "*triple stores*" or used to annotate the resources themselves or their models. The recommendation of the World Wide Web Consortium (W3C)[3] to define triples is the Resource Description Framework (RDF). A set of RDF triples *de facto* constitutes a graph, possibly made of several disjoint components, and is commonly referred to as a "*knowledge graph*".

SPARQL Protocol and RDF Query Language (SPARQL) is the standard semantic query language used to manipulate knowledge graphs in four main ways, all based on pattern matching:

- `SELECT` queries return all matches for a given graph pattern.
- `CONSTRUCT` queries return one new knowledge graph, specified by a given template, for each match found with a given graph pattern.
- `ASK` queries check whether there is at least one match for a given graph pattern.
- `DESCRIBE` queries return a description (made of various available information, depending on the query processor) of each match found for a given graph pattern.

Vocabularies and hierarchies of resource classes can be defined with RDF Schema (RDFS) or more expressive languages such as the Web Ontology Language (OWL). Semantic representations are meant to formalize knowledge in a computer-legible way, so that it can be collected into repositories where it can be queried and inferred upon to deduce new knowledge from asserted facts. More expressive ontology languages make inference engines more powerful, with the drawbacks of increased computational time and lower decidability.

Semantic Web technologies fit in our approach in two ways. First, they enable Conceptual Workflow modeling by helping capture domain knowledge. Second, the SPARQL language eases Conceptual Workflow transformations through graph pattern matching and graph construction. In our work:

[3] W3C: `http://www.w3.org/`

– the Conceptual Workflow Model itself is described in an RDFS-based COnceptual WORKflow (COWORK) ontology[4], so that Conceptual Workflows can be treated like knowledge graphs and queried as such;
– external ontologies (in any RDF-compatible language) are used to annotate the elements of Conceptual Workflows with the domain-specific functions they implement, the data types and contents they manipulate and non-functional criteria they fulfill, as detailed in Section 2.2; and
– SPARQL is used to combine components with workflows as detailed in Section 5.2 and to retrieve those components, as detailed in Section 5.3.

All created or imported elements are stored and retrieved from a repository that is called the **Knowledge Base**.

2.2 Annotations

Most elements of the Conceptual Workflow Model (detailed in Sections 3 and 4) can be annotated with types defined in external domain ontologies, as shown on the `Semantic` part of Figure 3.

An **Annotation** is defined by three things in the Conceptual Workflow Model:

– its **Type**, *i.e.* its class in an external ontology;
– its **Role**:
  either **Requirement** if it is a goal to achieve or a criterion to satisfy
  or **Specification** if it is an achieved goal or a satisfied criterion; and
– its **Meaning**:
  either **Function** if it is a domain method or objective (*e.g.* `fourier transformation`, `standard deviation calculation` and `regional cerebral blood volume estimation`),
  or **Concern** if it is a non-functional criterion (*e.g.* `SplitAndMerge`, `Log` and `SecureConnection`)
  or **Dataset** if it characterizes data (*e.g.* `PET simulated image`, `regional cerebral blood flow dataset` and `displacement field dataset`).

At the Conceptual Level, which is implementation-independent, Conceptual Workflows are annotated only with **Requirements**, since they do not yet achieve any goals or fulfill any criteria. During the Mapping transformation step (detailed in Section 5), **Requirements** are progressively transformed into **Specifications**, as they are fulfilled.

Fig. 4 Graphical Convention - Semantic elements (color online)

[4] COWORK ontology: `http://www.i3s.unice.fr/~cerezo/cowork/latest/cowork.rdfs`

Figure 4 shows the graphical convention for **Annotations**. Both **Type** and **Role** are displayed explicitly, but the **Meaning** is inferred through the **Type**.

3 Conceptual Elements

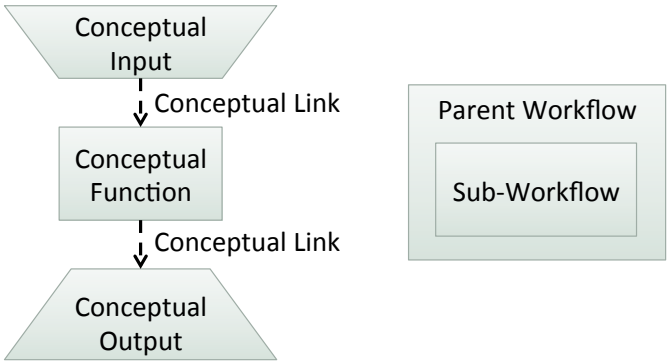


Fig. 5 Graphical Convention - Conceptual elements (color online)

The `Conceptual` part of the Conceptual Workflow Meta-model, shown on Figure 3, pertains to simulation modeling at the highest level of abstraction, with no information at all about implementation. Figure 5 illustrates the graphical convention for all Conceptual elements of the Conceptual Workflow Model.

3.1 Nested Directed Cyclic Graphs

Conceptual Workflows are modeled through nested directed cyclic graphs whose vertices represent operations to be performed and edges represent dependencies between operations. Because Conceptual Workflows are meant to be iteratively transformed into Abstract Workflows that can be enacted on a DCI, the Conceptual Workflow Model also features Abstract Elements that are described later in Section 4.

A vertex in a Conceptual Workflow may itself contain nodes and edges. The nested workflow is then relatively called a **Sub-workflow** and the workflow that contains it is called the **Parent workflow**. Because Sub-workflows can in turn contain Sub-workflows, the relationship is transitive. Where it is necessary to distinguish between direct ancestor/descendants (where the relationship is defined directly) and indirect ancestors/descendants (where the relationship is derived by transitivity), the direct ancestor is called the **immediate Parent Workflow**. That nesting is needed to model simulations at multiple abstraction levels in an integrated way.

3.2 Conceptual Workflow Types

There are three distinct types of Conceptual Workflows:

- **Conceptual Functions** represent simulations or steps in the simulation modeled by their Parent Workflow;
- **Conceptual Inputs** represent data fed as input to the scientific experiment modeled by their Parent Workflow and
- **Conceptual Outputs** represent data produced by the scientific experiment modeled by their Parent Workflow.

Only Conceptual Functions can contain Sub-workflows; the meta-model (Figure 3) thus exhibits a "*Composite Pattern*" [28] with Conceptual Workflows as "*Components*", Conceptual Functions as quotComposites and Conceptual Inputs/Outputs as quotLeaves.

All types of Conceptual Workflows can embed Abstract Elements and be annotated, but the Meaning (defined in Section 2.2) is restricted by type: Conceptual Functions can only bear **Functions** and **Concerns**, whereas Conceptual Inputs/Outputs can only bear **Datasets**.

3.3 Conceptual Links

An edge that connects two Conceptual Workflows is a **Conceptual Link**. A Conceptual Link cannot be annotated and it models a dependency between its source and target. It represents either:

- a data transfer, *i.e.* the target uses data produced by the source;
- a precedence order, *i.e.* the source must be done before the target;
- a causal link, *i.e.* the target must run only if the source has terminated or
- a combination of all of the above.

If the source and target have the same immediate Parent Workflow, then the source cannot be a Conceptual Output and the target cannot be a Conceptual Input, by definition. That restriction becomes somewhat complex if multiple nesting levels are involved: a Conceptual Input (resp. Output) can be seen and used as an entry (resp. exit) point of a Sub-workflow, as illustrated on Figure 6.

The restriction can be expressed thusly: if the source (resp. target) of a Conceptual Link is a Conceptual Output (resp. Input), then the immediate Parent Workflow of the source is neither the immediate Parent Workflow of the target, nor one of its Sub-workflows.

4 Abstract Elements

The end goal of Conceptual Workflow design is to produce an executable workflow and delegate it to an existing Abstract Workflow framework, but there are many targets to choose from (*e.g.* Taverna's Scufl [24], Kepler's MoML [9], MOTEUR's GWENDIA [5], Askalon's AGWL [23], Pegasus' DAX [7]) and each has its own advantages, its own target infrastructures and its own user base.

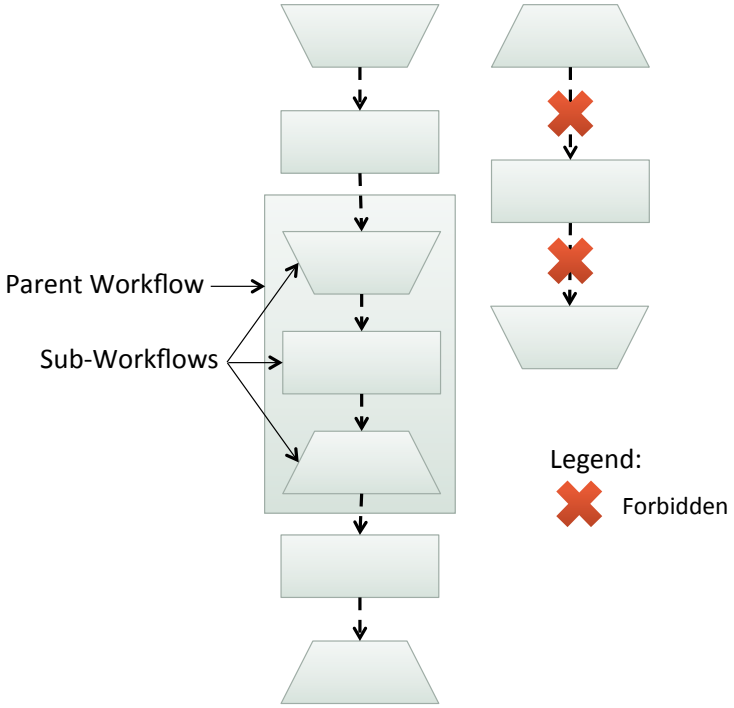


Fig. 6 Conceptual Link Restriction (color online)

As shown on Figure 2, the Transformation process is divided in two steps to avoid building an entirely separate process for each target language. The Conceptual Workflow is first populated with Abstract Elements, during the Mapping phase, and actual language conversion is postponed as much as possible.

The `Abstract` part of the Conceptual Workflow Meta-model shown on Figure 3 pertains to those Abstract Elements. Figure 7 illustrates the graphical convention for all Abstract elements of the Conceptual Workflow Model.

4.1 Activities

An Activity represents an "atomic" task (a black box from the viewpoint of the Conceptual Workflow that embeds it), *e.g.* a web service, a grid job or an executable artefact. Like Conceptual Functions, Activities can only bear Functions and Concerns. An Activity has an **Input Port** for each of its arguments and an **Output Port** for each of its products; it must have at least one **Port**. Ports can only bear Datasets.

Some arguments may not be explicit in the artefact's description. For instance, a web service might take a folder path as input and import files that are in that folder: those files are also arguments of the Activity, but they are implicit. The notion of **Implicit Ports** is meant to clarify those implicit relations and expose the related knowledge.

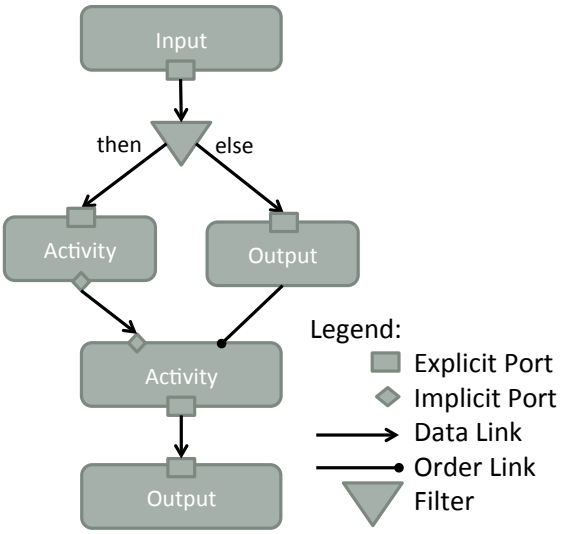


Fig. 7 Graphical Convention - Abstract elements (color online)

Implicit Ports are not automatically detected. If the Activity has such an implicit dependency, but is created without the corresponding Implicit Port and the end-user (who embeds the Activity in his or her Conceptual Workflow) does not know of it, then in all likelihood the resulting workflow will fail. Implicit Ports are a tool that can be used to make required knowledge explicit.

4.2 Activity Types

In addition to regular Activities, there are specific ones defined in the Conceptual Workflow Model:

- **Inputs** are Activities with at least one Output Port and no Input Port;
- **Outputs** are Activities with at least one Input Port and no Output Port and
- **Filters** are special Activities implementing conditional constructs: they have a **Guard** (*i.e.* a logical condition), one Input Port and two Output Ports `then` and `else`. Whenever a piece of data `d` is transferred to a Filter, the associated Guard is evaluated and `d` is passed along the `then` branch if the Guard is True, along the `else` branch otherwise.

In practice, Inputs and Outputs are most often data constants or references to files, but they may also be typical activities (such as web services) that either only produce or only consume data.

4.3 Links

At the Abstract Level, there are two types of Links:

– a **Data Link** represents a data transfer from an Output Port to an Input Port and
– an **Order Link** represents a precedence constraint between two Activities, *i.e.* the target Activity cannot start before the source Activity has completed.

Neither Data Links nor Order Links can bear Annotations.

5 Mapping

The **Mapping** transformation step introduced in this work is computer-assisted and iterative. Its objective is to transform all Requirements of a Conceptual Workflow into Specifications, by embedding Abstract Elements or inserting Sub-workflows that fulfill those Requirements, and compose these elements to form a functional Intermediate Representation.

The Activities and Sub-workflows used to fulfill the Requirements of Conceptual Workflows are either provided by the user or retrieved from the Knowledge Base where they are stored as **Fragments**.

Other approaches like the "*semantic framework*" [29] extending WINGS [30], work under the assumption that encapsulation and substitution are the only mechanisms needed to combine Scientific Workflows, using fixed templates and one-to-one mapping from the yet-undetermined steps in those templates to activities in the catalog. We believe that this assumption works in many cases, especially scientific areas where most simulations are data processing pipelines, but that it will not hold in the more general case and cannot tackle SoC.

A common way to tackle SoC is to use "*aspects*" [31]. It has even been suggested specifically for simulations [32]. However, one of the assumptions of Aspect-Oriented Programming (AOP) is that aspects are completely decoupled from functional processes: aspects are designed and maintained separately from one another and from core processes and weaving happens as late and as automatically as possible. While that makes a lot of sense in a business context, where different aspects are often entrusted to different experts who develop concurrently, Scientific Workflows are often designed and maintained by one scientist.

Another option is the use of "*process fragments*" [33,34], that can be extracted from a functional process or created from scratch and are woven into a base process through model transformation at any time during design. Instead of different people working concurrently on modules, the assumption of people reusing parts of other users' workflows and integrating them into their own seems a closer match to the context of Scientific Workflows. Therefore we opted for the somewhat less constrained notion of "*fragments*" over "*aspects*".

Fragments are meant to be (i) woven into Conceptual Workflows (ii) retrieved from the Knowledge Base in order to meet Requirements. The formal definition of Fragments is introduced in Section 5.1, their weaving process in Section 5.2 and their retrieval is detailed in Section 5.3. After weaving Fragments, the resulting workflows must be fully composed and Section 5.4 details that process.

5.1 Fragments

A Fragment is composed of two distinct Conceptual Workflows:

– the **Pattern** describes where the Fragment must be woven;
– the **Blueprint** describes what must be woven and
– their combination specifies what must be left untouched, deleted, generated or preserved in the base workflow into which the Fragment is woven, as detailed in Table 1. Note: A preserved element is not necessarily untouched: in some cases it will be modified (*e.g.* a link could be switched to a different target) but it will never be deleted.

Table 1 Pattern/Blueprint combinations

Pattern	Blueprint	Resulting Graph
		Untouched
✓		Deleted
	✓	Generated
✓	✓	Preserved

Fragments and Conceptual Workflows are modeled as knowledge graphs, based on the COWORK ontology. This representation allows the use of sophisticated SPARQL queries to match Fragments with Requirements and to weave them into Conceptual Workflows. Indeed, SPARQL's powerful pattern matching and graph constructing features can be leveraged to:

– retrieve Fragments from the Knowledge Base that match specific Requirements;
– look for patterns in a base Conceptual Workflow to identify where Fragments should be woven; and
– construct the graphs resulting from weaving Fragments into base Conceptual Workflows.

Although any graph manipulation mechanism would likely work, we use SPARQL because the transformation from a Fragment to a `SPARQL CONSTRUCT` query is very straightforward, as detailed in Section 5.2.

When an Activity, provided by the user or retrieved from the Knowledge Base, fulfills all the Requirements of a Conceptual Workflow, embedding the Activity inside the Conceptual Workflow is enough to map it. The Weaving process is trivial in that case: the Activity is inserted into the Conceptual Workflow and the Requirements become Specifications. However, there are much more complex cases where SPARQL advanced graph manipulation capabilities are warranted, notably to weave cross-cutting concerns.

5.2 Weaving

When the Weaving mechanism for a given Fragment is a simple substitution, the Pattern is an empty Conceptual Workflow with Requirements and/or Specifications. Activities stored into the Knowledge Base are thus stored into Fragments with trivial Patterns: empty Conceptual Workflow bearing the same Specifications as the Activity.

Figure 8 is an example of such a Fragment, where the Activity `sorteo_single` is taken from Sorteo, the PET simulator seminally included in the VIP project.

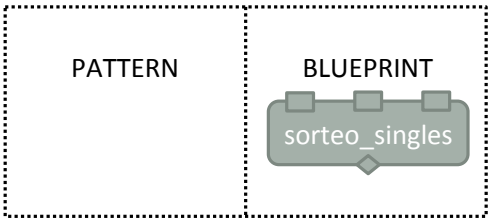


Fig. 8 Typical Activity Fragment (color online)

Substitution is sufficient in many cases, but more complex mechanisms are sometimes required, especially with non-functional concerns which are often cross-cutting. For instance, `SplitAndMerge` is a Concern meaning the workflow should exploit data parallelism and leverage multiple available computing resources by splitting data, distributing it for processing and then merging the results. It is non-functional since it pertains to computation time rather than the simulation itself.

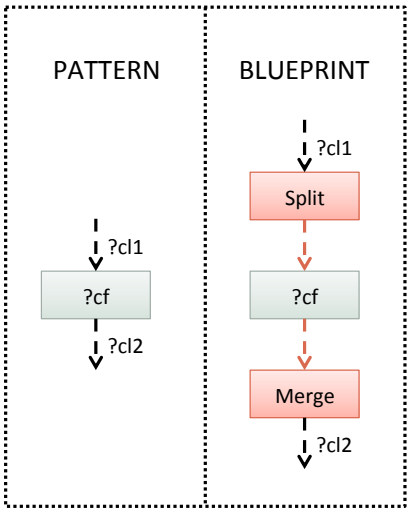


Fig. 9 Split and Merge Fragment (color online)

Because data parallelism is application-dependent, if we want to describe how to fulfill this Concern in general, we can only do so at the highest level of abstraction: any Abstract Element we would use would restrict the Fragment to a few applications at best. At a purely Conceptual Level, there is no way to fulfill the `SplitAndMerge` Concern more straightforwardly than by weaving a `Split` step before the target Conceptual Function and a `Merge` step after it. Figure 9 shows an example Fragment that describes the `SplitAndMerge` concern.

This is obviously a cross-cutting concern: the structure of the base workflow itself must be altered to accommodate the non-functional property. There is no way to fulfill it by inserting just one node anywhere. Still, in this case, one might think we could still use encapsulation through layers. However, the "layer" itself might be a Sub-workflow. In fact, out of 4 simulators included at launch in the VIP project, 3 require full-fledged Sub-workflows to fulfill `SplitAndMerge` and one uses an Activity that already fulfills the Concern. Layering does not seem appropriate in any of those 4 cases. Weaving thus requires more sophisticated mechanisms than encapsulation and substitution.

The first Weaving step is to translate the Fragment we want to weave into a `SPARQL CONSTRUCT` query. Elements present in the Pattern become "*variables*", easily recognizable by the question mark preceding their name. Elements that appear in the Blueprint become "*blank nodes*", declared with an underscore and colon before a placeholder identifier that will only hold for one match. Because they are created without a fixed identifier, different matches will produce different elements. The Pattern itself becomes the `WHERE` part of the query and the Blueprint becomes the `CONSTRUCT` part. Listing 1 presents a simplified version (omitting all type declarations) of the `SPARQL CONSTRUCT` query obtained by transforming the aforementioned Split and Merge Fragment.

Listing 1 Spint and Merge query

```
CONSTRUCT {
  ?cl1  cowork:hasConceptualTarget      _:Split .
  _:l1  cowork:hasConceptualSource      _:Split .
  _:l1  cowork:hasConceptualTarget      ?cf     .
  _:l2  cowork:hasConceptualSource      ?cf     .
  _:l2  cowork:hasConceptualTarget      _:Merge .
  ?cl2  cowork:hasConceptualSource      _:Merge .
}
WHERE {
  ?cl1  cowork:hasConceptualTarget      ?cf     .
  ?cl2  cowork:hasConceptualSource      ?cf     .
}
```

The second step is to run the `SPARQL CONSTRUCT` query, merge the resulting knowledge graphs with that of the base workflow and then automatically fix conflicts such as links with multiple targets or sources.

Now let us consider the simple workflow shown on Figure 10, with two Conceptual Inputs `Object Model` and `Simulation Parameters`, one Conceptual Output `Sinogram` (*i.e.* the output of a PET scan) and a simple chain of two Conceptual Functions `Generate Singles` then `Generate Emissions`. This workflow is one way to describe Sorteo as a Conceptual Workflow.

The Pattern part of the Split and Merge Fragment matches this workflow in three places: twice on `Generate Singles` because it is bound to two distinct Conceptual Inputs and once on `Generate Emissions`. The result of the SPARQL query, shown on Figure 11(a), is not directly satisfying, as it creates too many Split and Merge steps around `Generate Singles`. It is expected behavior, since the pattern is indeed matched twice there, but it is not the desired result (*i.e.* only one Split step before and only one Merge step after `Generate Singles`).

To achieve the desired result requires merging the extra Conceptual Functions. In order to do that, an automated tool is available that takes two Conceptual

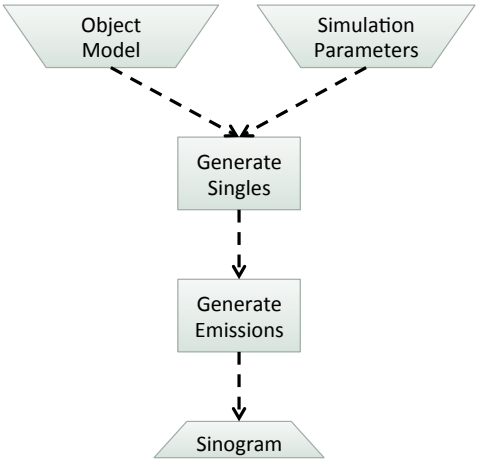
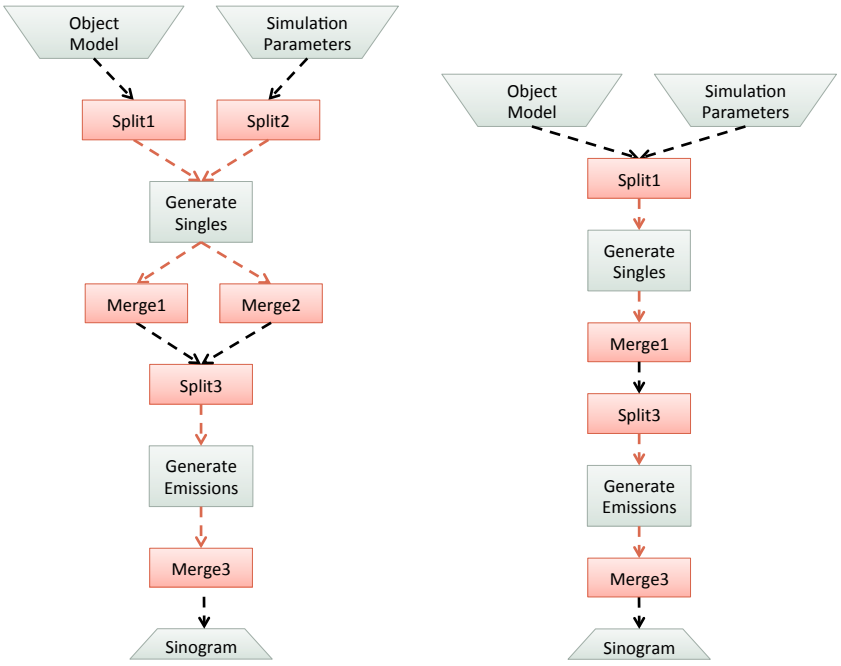


Fig. 10 Weaving Split and Merge - Base Workflow (color online)



(a) Query Result (b) Final Result

Fig. 11 Weaving Split and Merge (color online)

Workflows of the same type (with no embedded Abstract Elements) and merges them into one. On our running example it produces the desired result shown on Figure 11(b).

5.3 Discovery and Selection

When a user designs a Conceptual Workflow, chances are he or she has a few specific Activities (*e.g.* web services, legacy applications) or even Sub-workflows that he or she wants to use, but it is unlikely that those will be enough to perform all the Functions and fulfill all the Concerns of the Conceptual Workflow.

The other Activities and Sub-workflows the user needs might already be in the Knowledge Base. An assistance is needed to help the user find the Fragments needed to build an executable Scientific Workflow.

Our approach to assist the user in discovering and selecting Fragments to weave into his or her Conceptual Workflow consists of two steps:

1. **Matching**: for each Requirement, find and score all Fragments whose Specifications match, as detailed in Section 5.3.1.
2. **Sorting**: based on the nature of the Conceptual Workflow, the number and nature of annotations it bears and the combined scores of each candidate, sort candidates so as to prioritize the most likely relevant, as detailed in Section 5.3.2.

Once the candidates are found and sorted, they are presented to the user who may select the candidate(s) he or she wants to weave into the Conceptual Workflow. It is during Weaving that Requirements are transformed into Specifications.

5.3.1 Matching

Matching a Conceptual Workflow means matching all of its Requirements. There are three distinct ways to do that: (1) query the Knowledge Base for Fragments matching all Requirements simultaneously, (2) query the Knowledge Base for Fragments matching every possible combination of Requirements and then getting rid of the redundancies (a Fragment matching two Requirements will also match them separately) or (3) query the Knowledge Base for each Requirement separately and merge redundant results while sorting them. We did not opt for (1), because partial matches are desirable and we chose (3) over (2) because it puts less load on the Knowledge Base and thus seems a more scalable solution.

We consider three types of matches, ordered by decreasing quality:

- **Exact match** applies when both annotations are of the same type,
- **Narrower match** applies when the Specification's type is a subtype of the Requirement's and
- **Broader match** applies when the Specifications's type is a supertype of the Requirement's.

To illustrate those matches, let us consider the small excerpt of the Virtual Imaging Platform[5] (VIP) ontology shown on Figure 12. This excerpt defines a taxonomy of the registration image alignment procedure. The VIP ontology itself is written in OWL and is based on the foundational ontology DOLCE [35] (interested readers please refer to [36] for more details).

If the Requirement we are trying to match is `rigid-registration`, then:

- Fragments specified as `rigid-registration` would be exact matches,

[5] VIP: `http://www.creatis.insa-lyon.fr/vip/`

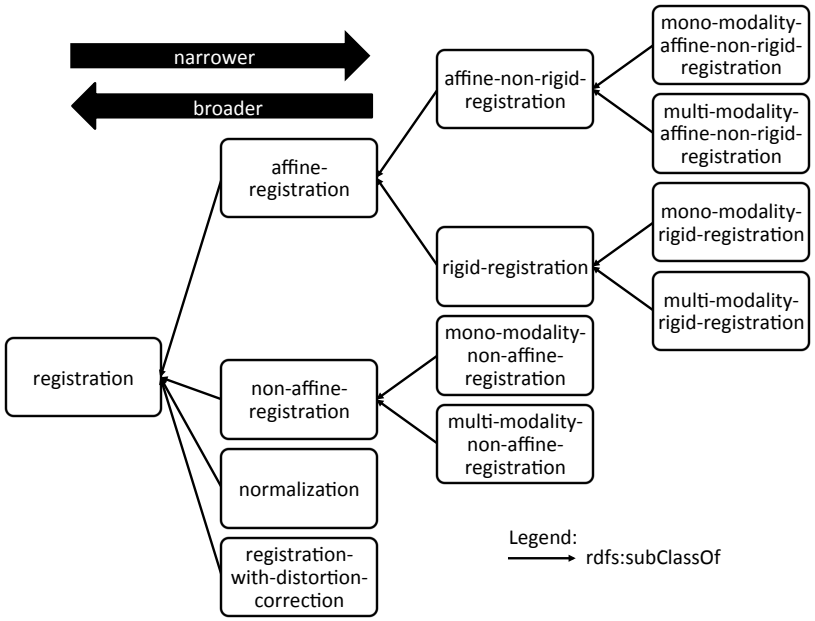


Fig. 12 VIP Ontology - Registration Processes Taxonomy

– Fragments specified as `mono-modality-rigid-registration` or as `multi-modality-rigid-registration` would be narrower matches and
– Fragments specified as `affine-registration` would be broader matches.

The case for narrower matches seems pretty straightforward, at first glance: a rigid registration process matches regardless of whether it is mono or multi modality. Indeed, the definition of `rdfs:subClassOf` - the property that links subtypes to supertypes in RDFS and all ontology languages built upon it such as OWL - makes finding narrower matches easy: "`A rdfs:subClassOf B`" means that all instances of B are also instances of A, hereby ensuring that narrower matches will be found as long as inferences were made. For instance, if we assert that `:Socrates rdf:type :Man` and `:Man rdfs:subClassOf :Mortal`, run the inference engine, then look for instances of the Class `:Mortal`, we find `:Socrates`.

Broader matches may be less relevant. They are deemed moderately worthy (their quality is lower than that of the other matches) for two reasons: (1) the user might over-specify, *e.g.* look for a rigid registration algorithm where an affine but non-rigid one would do just fine, and (2) there might be valuable insight in workflows that are too broad for the work at hand.

To avoid burdening the user with all levels of supertypes (ultimately, the type `owl:Thing` from which all Classes are derived can always be reached, thus retrieving the entire Knowledge Base), only level 1 broader matches are considered: candidates whose type is the direct supertype of the Requirement's.

Listing 2 Match(R) Query

```
SELECT ?f
WHERE {
```

```
  ?f rdf:type cowork:Fragment .
  ?f cowork:hasBlueprint ?b .
  { ?b cowork:hasSpecification ?s . }
  UNION
  { ?b cowork:contains ?cw .
    ?cw cowork:hasSpecification ?s . }
  { ?s rdf:type R . }
  UNION
  { ?s rdf:type ?t .
    R rdfs:subClassOf ?t . }
}
```

Listing 2 is the SPARQL query template we can use to find all candidate Fragments for a given Requirement of type `R`.

5.3.2 Sorting

The way we sort candidates depends partly on the type of Conceptual Workflow considered. The Sorting process has the following goals:

- **prioritize match quality**, *i.e.* prioritize exact matches over narrower matches and both of them over broader matches;
- **penalize partial matches**, *i.e.* candidates matching fewer Requirements must come after candidates matching more of them;
- **penalize extra inputs/outputs/functions**, depending on the type of Conceptual Workflow considered, *i.e.* make sure that Fragments that would introduce new inputs/outputs or unnecessary Functions when they are woven come after Fragments that do not; and
- **prioritize Functions over Concerns**, *i.e.* favor Fragments that fulfill Functions over those that fulfill Concerns, since it makes more sense to first ensure functionality and then cater to non-functional properties than the other way around.

Algorithm 1 Discovering and sorting candidates for a Conceptual Input

function MATCH($ci : ConceptualInput$)
  $weightedMatches : Dictionary(Fragment, Float)$
  for all $r \in req(ci)$ **do** ▷ $\forall r$ requirement of ci
    for all $m \in match(r)$ **do** ▷ Fetch all candidates for r
      $w \leftarrow max_{s \in spec(m)}(score(r, s))$ ▷ Compute best matching score
      if $m \in keys(weightedMatches)$ **then**
        $weightedMatches[m] \leftarrow weightedMatches[m] + w$ ▷ Update weight
      else
        $weightedMatches[m] \leftarrow w$ ▷ Add new key and initial weight
      end if
    end for
  end for
  for all $(m, w) \in weightedMatches$ **do**
    $weightedMatches[m] \leftarrow weightedMatches[m]/|req(ci)|$ ▷ Penalize partial matches
    $weightedMatches[m] \leftarrow \frac{weightedMatches[m]}{|inputs(m)|+1}$ ▷ Penalize inputs
  end for
  return $sort(weightedMatches)$ ▷ Sort by decreasing weight
end function

Algorithm 1 details how the system sorts candidates for Conceptual Inputs and uses the following sub-functions:

- $req(e : Element)$ returns the set of Requirements e is annotated with;
- $spec(e : Element)$ returns the set of Specifications e is annotated with;
- $match(r : Requirement)$ uses the SPARQL query detailed in Listing 2 to fetch all candidates matching the Requirement r;
- $score(r : Requirement, s : Specification)$ affects a score to the pair of annotations based on the relationship between them, so that $score(exact) > score(narrower) > score(broader)$ and $score(other) = 0$; and
- $inputs(f : Fragment)$ returns the set of Conceptual Inputs and unattached Input Ports contained in the Fragment f.

The algorithm to sort candidates for Conceptual Outputs is almost identical to Algorithm 1, but it penalizes outputs instead of inputs.

Algorithm 2 Finding candidates for a Conceptual Function

function MATCH($cf : ConceptualFunction$)
  if $\exists c \rightarrow cf$ **then** ▷ At least one sub-workflow
    $weightedMatches : Dictionary(ConceptualWorkflow, Dictionary(Fragment, Float))$
    for all $c \rightarrow cf$ **do** ▷ Recursive propagation
      $weightedMatches[c] \leftarrow$ MATCH(c)
    end for
  else
    $weightedMatches : Dictionary(Element, Float)$
    for all $r \in req(cf)$ **do** ▷ $\forall r$ requirement of cf
      for all $m \in match(r)$ **do** ▷ Fetch all candidates for r
        if $rinstanceofFunction$ **then** ▷ r is a Function
          $w' \leftarrow K_F * w$ ▷ Prioritize Functions over Concerns
        else ▷ r is a Concern
          $w' \leftarrow w$
        end if
        if $m \in keys(weightedMatches)$ **then**
          $weightedMatches[m] \leftarrow weightedMatches[m] + w'$ ▷ Update weight
        else
          $weightedMatches[m] \leftarrow w'$ ▷ Adding new key and initial weight
        end if
      end for
    end for
    for all $(m, w) \in weightedMatches$ **do**
      $weightedMatches[m] \leftarrow \frac{weightedMatches[m]}{K_F * |req(cf) \cap \Omega_F| + |req(cf) \cap \Omega_C|}$ ▷ Penalize partials
      $weightedMatches[m] \leftarrow \frac{weightedMatches[m]}{1 + |spec(m) \setminus req(cf)|}$ ▷ Penalize extra Functions
    end for
  end if
  return $sort(weightedMatches)$ ▷ Sort by decreasing weight
end function

Algorithm 2 details how the system sorts candidates for Conceptual Functions and uses the sub-functions req, $spec$ and $match(r : Requirement)$ described previously, but also the constant K_F which is a factor characterizing how much Functions should be prioritized over Concerns.

Once all found candidates are sorted, they can be presented as an ordered list the user can freely browse to pick his or her choice of Fragment to weave into the base workflow.

5.4 Composition

When all Requirements are fulfilled the resulting workflow is still likely to be incomplete: the Activities and Sub-workflows woven into the base Conceptual Workflow must be composed. This is the level where most technical issues have to be dealt with, most notably conversion problems.

Indeed, only domain and non-functional issues have been catered to, purposely leaving out technicalities so as to emphasize the scientific experiment over its implementation. However, to transform the Conceptual Workflow into a full executable Abstract Workflow requires tackling all technical issues.

This process cannot be fully automated for two reasons: (1) this would assume that all annotations are perfectly accurate and (2) even if they were, they would be likely to provide non-exhaustive knowledge and know-how. There is most often a gap between the physical layer where the workflow enactor handles file formats, error codes and transfer protocols and the semantic layer where annotations describe high-level goals and methods. Theoretically, one could hope to fill this gap to reach complete automation by extending both layers, *e.g.* adding meta-data to the physical files and extending ontologies with technical notions as they arise. In practice, however, there is a trade-off between the scope of the system, in terms of domains and types of workflows supported, and the level of automation that can be provided.

Projects like [37] assist the user very thoroughly through the creation of workflows, but are restricted to curated application domains (data mining pipelines in the case of [37]). Ultimately, user input is needed to sort out non-modeled requirements and fill in uncaptured knowledge and know-how. Nonetheless, user workflow design can be assisted in all cases where file formats (and other technical information available) and ontologies provide enough information to detect and/or solve technical issues, by suggesting links and converters.

Assistance to composition is divided in three distinct and complementary parts: Link Suggestion, Mismatch Detection and Converter Suggestion.

5.4.1 Link Suggestion

When a Port is not bound to any Data Link, it is **unattached**. The end goal of Link Suggestion is to attach all Input Ports.

Unattached Output Ports are not a problem in and of themselves: there might well be by-products of an Activity that are irrelevant to the simulation considered. Unattached Input Ports, however, induce unreachable parts in the workflow. Indeed, if no data is ever directed to an Activity, then it will never be fired. The algorithm thus examines all unattached Input Ports in the Conceptual Workflow and suggests Data Links between them and Output Ports already present in the workflow.

Only links that preserve the order defined by Conceptual Links are suggested. If a link from an Activity inside a Conceptual Workflow s to one inside a Conceptual

Workflow t is suggested, then either $s = t$ or there is a path (made of Conceptual Links) from s to t.

Using the same *score* function introduced in Section 5.3.2, the system can compare the Specifications of an unattached Input Port with those of all preceding Output Ports. The main difference with sorting candidate Fragments is that partial matches are not just penalized, they are skipped entirely. Indeed, multiple Requirements on a same Conceptual Workflow might be fulfilled by different Fragments woven together, but multiple Specifications on an Input Port must be fulfilled all at the same time.

5.4.2 Mismatch Detection

A **mismatch** characterizes a Data Link whose source and target are incompatible at either the physical level (*e.g.* the source is a `list` and the target is a `string`), the semantic level (*e.g.* the source is a `BMP-format` and the target is a `JPEG-format`) or both at the same time. Note that narrower matches are compatible. For instance, if an Output Port specified by `PET-image-storage-SOP-class` is connected to an Input Port specified by `DICOM-SOP-class`, since the former is a subclass of the latter, there is no mismatch.

5.4.3 Converter Suggestion

When a mismatch is detected at the physical level, the user is warned and provided with a modest collection of converters meant to tackle the most common mismatches, such as `List2String` and `String2Integer`.

When a mismatch is detected at the semantic level, the system looks for an appropriate converter in the Knowledge Base. The search for a converter is similar to the Discovery and Selection process described in Section 5.3, but all three types of annotations need to be considered simultaneously: the aim is to match input and output Datasets while penalizing Functions and Concerns that are unrelated to conversion so as to avoid converters that would alter data.

There is also a difference in scoring: the *exact* > *narrower* > *broader* hierarchy still stands for the target type of the converter, but broaderMatches are better than narrowerMatches when it comes to the source type.

As with matches in Section 5.3, potential converters are found in the Knowledge Base via SPARQL queries, but in this case more than one query is needed, for the optimal number of conversion steps is unknown. Listing 3 is a template for a simple conversion with no intermediary step, whereas Listing 4 is a template for a conversion with one intermediary step. Such query templates can be generated automatically up to an arbitrary number of intermediary steps, but each intermediary type gone through increases the likelihood of data loss or alteration. Therefore the system looks for converter chains of length up to an arbitrary threshold and penalizes longer chains.

Listing 3 Convert $X \rightarrow Y$ Query

```
SELECT ?f
WHERE {
 ?f rdf:type cowork:Fragment .
 ?f cowork:hasBlueprint ?b .
```

```
  ?b cowork:contains ?ci .
  ?ci rdf:type cowork:ConceptualInput .
  ?ci cowork:hasSpecification ?si .
  { ?si rdf:type X . }
  UNION
  { ?si rdf:type ?ti .
    X rdfs:subClassOf ?ti . }
  ?b cowork:contains ?co .
  ?co rdf:type cowork:ConceptualOutput .
  ?co cowork:hasSpecification ?so .
  { ?so rdf:type Y . }
  UNION
  { ?so rdf:type ?to .
    Y rdfs:subClassOf ?to . }
}
```

Listing 4 Convert $X \rightarrow ? \rightarrow Y$ Query

```
SELECT ?f1, ?f2
WHERE {
  ?f1 rdf:type cowork:Fragment .
  ?f1 cowork:hasBlueprint ?b1 .
  ?b1 cowork:contains ?ci1 .
  ?ci1 rdf:type cowork:ConceptualInput .
  ?ci1 cowork:hasSpecification ?si1 .
  { ?si1 rdf:type X . }
  UNION
  { ?si1 rdf:type ?ti1 . }
    X rdfs:subClassOf ?ti1 .}
  ?b1 cowork:contains ?co1 .
  ?co1 rdf:type cowork:ConceptualOutput .
  ?co1 cowork:hasSpecification ?so1 .
  ?so1 rdf:type ?to1 .
  ?f2 rdf:type cowork:Fragment .
  ?f2 cowork:hasBlueprint ?b2 .
  ?b2 cowork:contains ?ci2 .
  ?ci2 rdf:type cowork:ConceptualInput .
  ?ci2 cowork:hasSpecification ?si2 .
  ?si2 rdf:type ti2 .
  { ?to1 rdfs:subClassOf ?ti2 . }
  UNION
  { ?ti2 rdfs:subClassOf ?to1 . }
  ?b2 cowork:contains ?co2 .
  ?co2 rdf:type cowork:ConceptualOutput .
  ?co2 cowork:hasSpecification ?so2 .
  { ?so2 rdf:type Y . }
  UNION
  { ?so2 rdf:type ?to2 .
    Y rdfs:subClassOf ?to2 . }
}
```

Algorithm 3 details how to find and sort potential converters (*i.e.* list of Fragments) from a list of source types to a list of target types and uses the following sub-functions:

- $convert(source : Specification, target : Specification)$ generates queries similar to those shown on Listing 3 and Listing 4 and returns a list of potential converters.

Algorithm 3 Finding converters

```
1: function CONVERT(S : List(Specification), T : List(Specification))
2:     weightedConverters : Dictionary(List(Fragment), Float)
3:     for all s ∈ S do                                          ▷ For all source types
4:         for all t ∈ T do                                      ▷ For all target types
5:             for all f ∈ convert(s, t) do                      ▷ Find s → t converters
6:                 w : Float ← score(s, f, t)                    ▷ Compute base score
7:                 w ← w / |f|                                   ▷ Penalize chain length
8:                 w ← w / |nonConvert(f)|   ▷ Penalize non-conversion Functions and Concerns
9:                 if f ∈ keys(weightedConverters) then
10:                    weightedConverters[f] ← weightedConverters[f] + w ▷ Update weight
11:                else
12:                    weightedConverters[f] ← w                 ▷ Add new key and initial weight
13:                end if
14:            end for
15:        end for
16:    end for
17:    return sort(weightedConverters)                           ▷ Sort by decreasing weight
18: end function
```

- $score(source : Specification, conv : List(Fragment), target : Specification)$ is detailed by Algorithm 4 and it itself uses:
  - $subscore(source : Specification, target : Specification)$, which compares source and target types semantically and scores the conversion so that $exact > broader > narrower$.
- $nonConvert(converter : List(Fragment))$ returns the total number of Functions and Concerns unrelated to conversion among the specifications of Fragments that compose the Conversion chain.

Algorithm 4 Scoring converters

```
1: function SCORE(s : Specification, f : List(Fragment), t : Specification)
2:     score : Float ← 1
3:     score ← score * max_{it∈inputTypes(f[first])}(subscore(s, it))
4:     for all i ∈ (0..|f| - 2) do
5:         score ← score * max_{ot∈outputTypes(f[i]),it∈inputTypes(f[i+1])}(subscore(ot, it))
6:     end for
7:     score ← score * max_{ot∈outputTypes(f[last])}(subscore(it, t))
8:     return score
9: end function
```

6 Use Case

To illustrate the modeling and mapping of Conceptual Workflows, let us take a closer look at Sorteo, the PET simulator, one of the simulators included in the Virtual Imaging Platform.

All four simulators seminally included in the Virtual Imaging Platform fit the same high-level Conceptual Workflow, shown on Figure 13, with slightly different Requirements for the main Conceptual Function: `medical-image-simulation` in

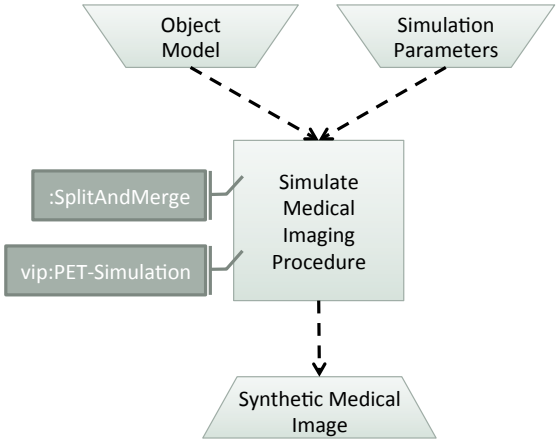


Fig. 13 VIP Simulator Core Template (color online)

general and `PET-simulation` in the case of Sorteo. No matter the simulator, it always bears the `SplitAndMerge` Requirement.

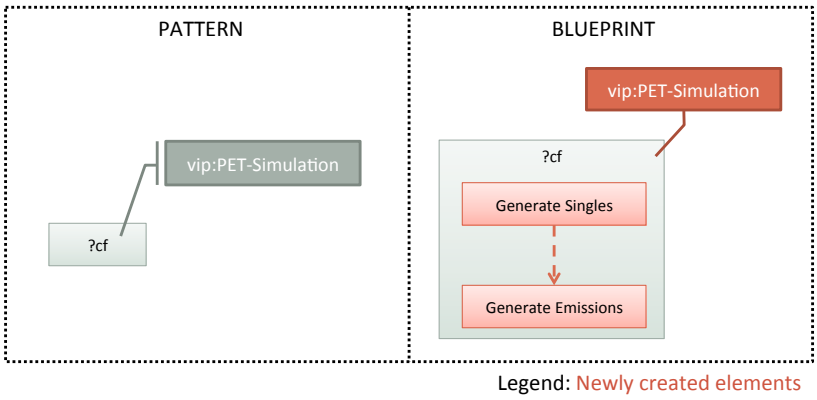
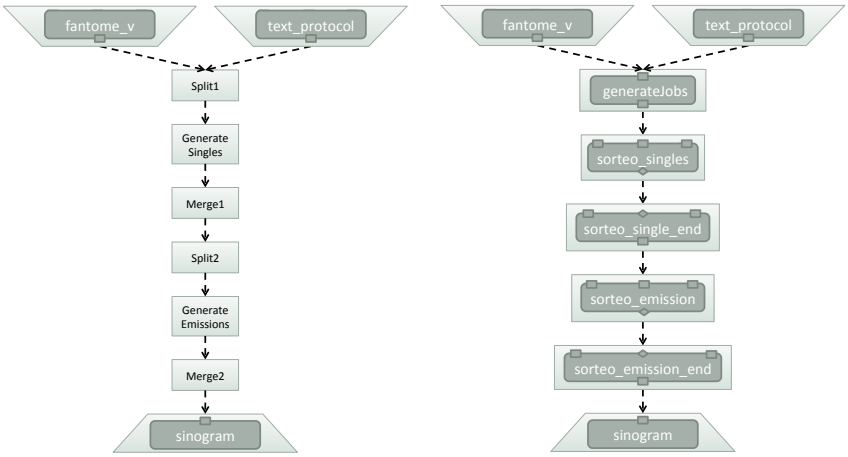


Fig. 14 Sorteo Use Case - PET Fragment (color online)

Matching the main Function highlights a modeling issue: the way Sorteo simulates PET is by first generating single photons, then generating the corresponding emissions. The two steps simulate PET only when combined, which means the corresponding Activities will not be annotated with `PET-simulation`. This is where purely conceptual Fragments can be useful: the Fragment shown on Figure 14 is a computer-legible assertion that combining those two steps amounts to simulating a PET procedure. Weaving it in the template and then erasing the left-over Conceptual Function (`Simulate Medical Imaging Procedure`) results in the base workflow (Figure 10) used as example in Section 5.1 to illustrate the Weaving of

the `SplitAndMerge` Fragment (Figure 9). The result of that Weaving is shown on Figure 11(b).



(a) Input/Output provided (b) Conceptual Functions Mapped

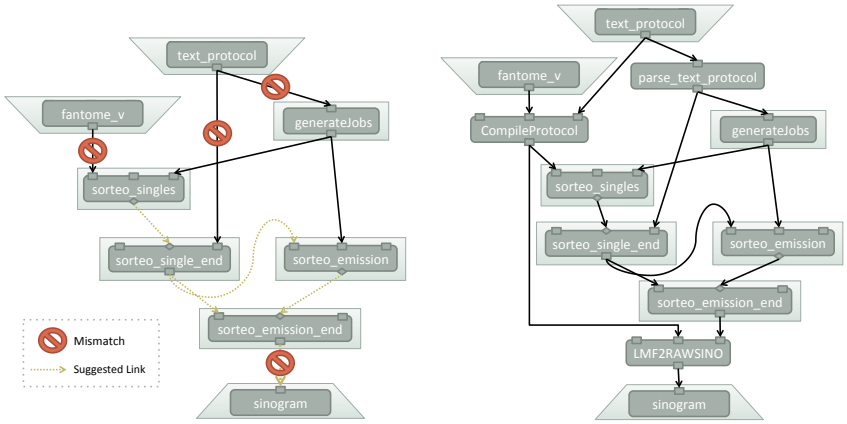
Fig. 15 Sorteo Use Case - Discovery and Weaving (color online)

After the user has provided input and output files (*i.e.* `fantome_v` for the Object Model, `text_protocol` for the Simulation Parameters and `sinogram` for the final output), the workflow becomes the one shown on Figure 15(a).

Matching `Split1` and `Split2` finds the same Activity `generateJobs` in both cases. The user may keep both instances (in which case the execution will be duplicated as well), but here there is only one instance needed, therefore the user erases the `Split2` Conceptual Function. There is no need to insert additional Conceptual Links, since that relationship is transitive. The composition will consider the Output Port of `generateJobs` to bind unattached ports in all Conceptual Functions after it, even if there is no direct Conceptual Link between, for instance, `Split1` and `Generate Emissions`. Figure 15(b) shows the workflow after all Conceptual Functions have been matched.

Figure 16(a) shows the links chosen by the user among links the system can suggest based on semantic types, links created by the user and the following mismatches:

- The link between `fantome_v` and `sorteo_singles` is a mismatch, because the latter requires a combination of an object model and simulation parameters: the Activity `CompileProtocol` must be inserted to produce that combination.
- Even though the link between `sorteo_emission_end` and `sinogram` is suggested, since both ports share the annotation `PET-sinogram`, there is a format mismatch (LMF versus raw).
  That format problem should be detected and the appropriate converter (here the Activity `LMF2RAWSINO`) suggested, provided the Activities are appropriately annotated and the converter is featured in the Knowledge Base.



(a) Suggested Links and Mismatches (b) Converters Added and Composed

Fig. 16 Sorteo Use Case - Composition (color online)

– The mismatch between `text_protocol` and `generateJobs` is the hardest to detect: the latter only needs one number (*i.e.* the number of jobs to generate) and that number must be extracted from the protocol, which can be done with a simple script like the Activity `parse_text_protocol`.
In most cases, technical details like this will fall in the gap between the physical and semantic layers and go undetected. Users would then have to come up with a fix without computer-assistance, much like they do with existing Abstract scientific workflow frameworks.

Figure 16(b) shows the result of inserting the needed converters and adding links from `text_protocol` to `CompileProtocol` and from the latter to `LMF2RAWSINO`.

After Matching, Weaving and Composition, there are 6 unattached ports left in the workflow. All of them expect the path to the folder where all the simulation files are put in and retrieved from. There is a convention on the VIP platform as to how this folder name should be constructed: by appending the current date to a simulation name, through the Activity `appendDate`.

To insert this Sub-workflow and bind it in the 6 places where it is needed would make the workflow extremely hard to read. We thus introduce the following graphical convention: hexagons with the same name represent the same instance of a Sub-workflow, so that it may be bound in multiple places in a workflow without cluttering it with edges. The Sub-workflow `DN/Directory Name` on Figure 17 is duplicated only graphically.

There are no Requirements and no unattached ports left and the Conceptual Workflow is now mapped. There is nothing more we can do inside the Conceptual Workflow Model. What is left to do is to automatically convert this workflow into a target Abstract Workflow language. If we target GWENDIA [5], we obtain the executable workflow shown on Figure 18, which is a screenshot of the MOTEUR [10] client.

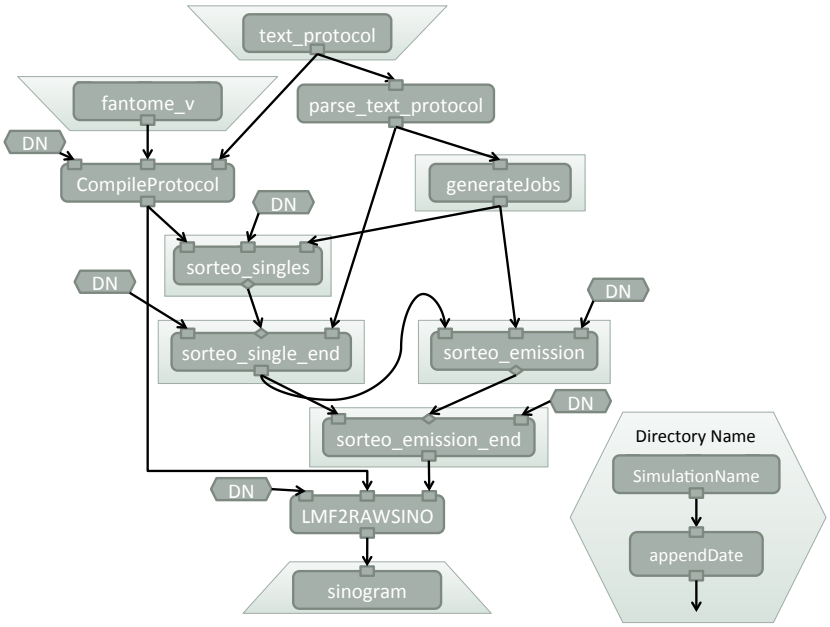


Fig. 17 Sorteo Use Case - Intermediary Representation (color online)

7 Discussion

The Mapping process described here assumes that a Knowledge Base containing Fragments is available. That base is queried with SPARQL to find Fragments matching Requirements and those are woven into the Conceptual Workflow whose transformation is ongoing. Admittedly, such a setup suffers from what is commonly referred to as the "*cold-start problem*": if there are no relevant Fragments in the Knowledge Base, then the system will provide very little assistance to the user.

While it is certainly a limitation in our knowledge-based approach, it is not a flaw per se: the situation is not better with existing Abstract Workflows frameworks, which cannot assist design even in common cases that have been seen countless times before. It should also be noted that our system allows a progressive enrichment of the Knowledge Base, as users contribute new workflows and elements.

Concerning discovery and selection, there are two main types of approaches to this problem depending on whether one relies on meta-data or on semantic data.

– In the former case, user-provided information is leveraged, such as workflow descriptions, workflow keywords, user associations (*e.g.* scientists A and B work in the same team), and content correlations (*e.g.* users who access workflow A tend to access workflow B). Based on such meta-data, queryable repositories, social platforms and recommendation engines can help users find content that is relevant to their queries, their profile or their history. The $^{my}Experiment$ project [38] is a great example of such meta-data-based sharing platforms.

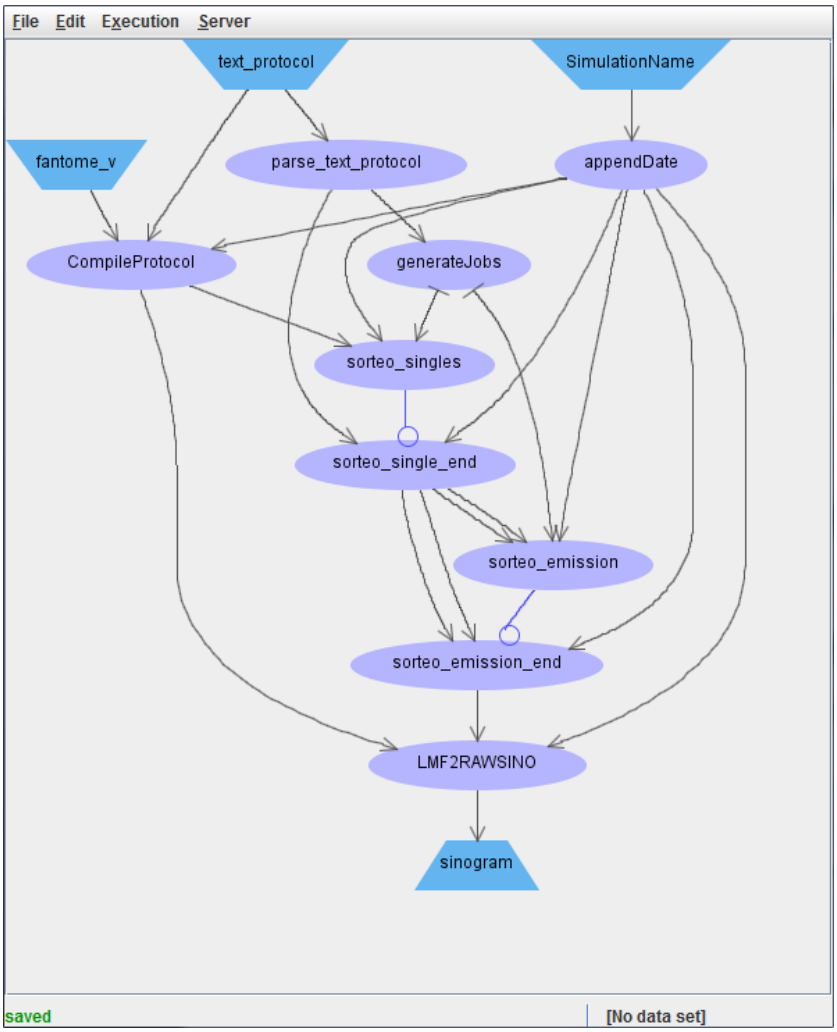


Fig. 18 Sorteo Use Case - GWENDIA Abstract Workflow (color online)

– In the latter case, semantic data is built right into the workflow model and this information is used to suggest candidates whose semantic data at least partially matches the needs.

We adopted the latter approach, not only to leverage semantic annotations that are an integral part of our model, but also because our aim is to assist the user during the transformation process which is workflow-centered and not user-centered like meta-data-based systems.

8 Related Works

The ^{my}Grid[6] team, which developed and maintains ^{my}*Experiment* [38] as well as Taverna [24], did work on automated annotation of services [39] and used semantic annotations to improve service discovery [40] and metadata management [41]. This latter work is "*investigating potential uses of metadata, e.g., discovering workflow entities and guiding their composition*". Bechhofer *et al* [42] also advocate for "*semantically rich aggregations of resources, that possess some scientific intent or support some research objective*". In [43], Missier *et al* present the notion of Functional Units, which are fairly close to Conceptual Workflows but significantly different in intent: Functional Units are high-level service descriptions and, accordingly, they support polymorphism (generic services that can perform multiple functions) and invocation patterns (e.g. three distinct operations needed to use a given service), but are restricted to one service. The Conceptual Workflow Model is more akin to what Missier et al. call an "*ideal Taverna workflow*" and the Transformation Process means to assist the user in converting that "*ideal*" workflow into an actual executable workflow. Their discussion about eliciting Functional Units (i.e. deriving service annotations from existing workflows) is definitely worth consideration as a tentative solution to the Knowledge Base cold start problem and whether Functional Units can be leveraged as-is or converted for use with Conceptual Workflows is worth investigating.

The team behind the WINGS [30] framework (itself based on Pegasus [7]), tackles problematics very similar to ours. Gil et al. [29] describe a framework built on top of WINGS to automatically transform user queries into Scientific Workflows. Garijo and Gil [14] describe an approach to publishing "*abstract workflows*" (*i.e.* workflow templates with undetermined Activities) and "*executable workflows*" (*i.e.* what we call Abstract Workflows) as Open Linked Data through an extension of the Open Provenance Model[7] (OPM). Hauder *et al* [37], focus their framework on the state of the art in data mining pipelines and show convincing results on automated composition of Scientific Workflows and increased accessibility. In all three cases, they use Semantic Web technologies and multiple abstraction levels to ease reuse, improve accessibility and provide computer-assisted workflow design.

The major difference between our approach and that of WINGS is that our proposal is top-down, from the highest level of abstraction of scientific experiments and domains to the level of executable Scientific Workflows, whereas the WINGS framework is bottom-up, from working Scientific Workflows to more flexible and accessible templates. It might be a consequence of this bottom-up perspective that their workflow templates have a rigid structure, assuming a one-to-one mapping between "*abstract*" and "*concrete*" components. While this assumption does not hold in general, it does hold very well in fields like data mining for which the framework is tailored. There is clearly a trade-off between structural flexibility and maximum achievable automation.

The Kepler Project [9] has always considered SoC a top priority, going as far as isolating the Model of Computation (*i.e.* the exact same workflow can be executed in entirely different ways depending on which "*Director*" is selected, for instance in sequence or in parallel). Both [44] and [45] introduce templates

[6] ^{my}Grid: http://www.mygrid.org.uk/

[7] OPM: http://openprovenance.org/

meant to enhance reuse by shielding the user from technicalities. Neither work emphasizes the scientific experiment: the former discusses the use of reasoning to automate "*wiring*" (*i.e.* composition) and the latter the use of semantic types to automate discovery. Kepler Workflow templates comparable to that of WINGS were introduced in [46] with the added notion of context-awareness.

In [47], Sonntag et al. present compelling reasons for a workflow system to provide the user with different views depending on the current phase in the workflow lifecycle. Most of the views they propose deal with execution/monitoring and are thus unrelated to this work, but the views they call "*Aggregation of Complex Workflow Logic*" and "*Phases in Simulation Workflows*" are definitely relevant for Scientific Workflow modeling. We argue that both can be achieved through Conceptual Workflow nesting: the user can easily aggregate parts of a workflow into a sub-workflow and high-level Conceptual Functions can model simulation phases. There is also the rather interesting case of the "*Data Flow Visualization*" view, which needs only exist as a view in a control-driven model, and not in a data-driven or a hybrid one like Conceptual Workflows.

9 Conclusion and Future Works

This work introduces a high-level Scientific Workflow model, called Conceptual Workflow, that aims at capturing scientific experiments inside the user domain(s). A semi-automatic Transformation Process leveraging Model Driven Engineering (MDE) and Knowledge Engineering techniques is proposed to generate executable workflows. The Mapping part of the transformation process is computer-assisted, using the domain knowledge formally captured.

The Conceptual Workflow model is itself fully represented as a knowledge graph and the Transformation Process takes advantage of that by leveraging the advanced pattern matching and graph construction capabilities of SPARQL in order to (1) automate the weaving of graph Fragments into the Conceptual Workflow during its transformation; and (2) ease the identification of partial matches to help the user fully exploit the knowledge database at hand.

We are now focusing on the development and testing of a user interface to design and map Conceptual Workflows as well as converters to other languages besides GWENDIA [5], notably the SHIWA interoperability project's language: the Interoperable Workflow Intermediate Representation (IWIR) [48].

We are investigating how we can feed back annotations from Conceptual Workflows to the Knowledge Base automatically, *i.e.* deduce annotations for Activities from the Conceptual Workflows they are embedded into, as well as how to detect and deal with inconsistencies in the annotations and/or ontologies.

Acknowledgements This work was funded by the French National Agency for Research under grant ANR-09-COSI-03 VIP and the European RI ER-Flow project under contract number 312579.

Glossary

Abstract Level: Intermediary level of abstraction between the Conceptual Level and the Concrete Level. Most scientific workflow frameworks handle Scientific Workflows at that level. It aligns with the level of PIMs in the MDA.

Business process: Structured set of tasks meant to achieve a business goal such as providing a service or making a product.

CIM: (Computation-Independent Model) Class of models defined by the MDA and based on the abstraction level: computation-independent models are so high-level as to not be tied to a specific implementation method or infrastructure. They are designed to be easy for domain experts to understand, design and manipulate. However, they must be transformed into lower-level models to be used in practice.

Conceptual Level: Level of abstraction at which simulations are conceived by scientists, in their own domain(s). It aligns with the level of CIMs in the MDA.

Concrete Level: Level of abstraction at which Scientific Workflows are enacted. It aligns with the level of PDMs in the MDA.

Enactor: Program deploying Scientific Workflows on DCIs and managing their execution.

MDA: (Model Driven Architecture) MDE approach launched by the Object Management Group which, among many things, classifies software models into three abstraction levels: PDM, PIM and CIM.
→ `http://www.omg.org/mda/`

MDE: (Model Driven Engineering) Approach to software development that focuses on the creation and management of high-level domain models on which development is based, in order to leverage domain experts' knowledge and ease communication between system designers.

OWL: (Web Ontology Language) Language defined and maintained by the W3C to formally define RDF-based ontologies.
→ `http://www.w3.org/TR/owl-overview/`

PDM: (Platform-Dependent Model) Class of models defined by the MDA and based on the level of abstraction: platform-dependent models are tightly-coupled with the infrastructure they run on and, as a result, are difficult to reuse.

PIM: (Platform-Independent Model) Class of models defined by the MDA and based on the abstraction level: platform-independent models are loosely-coupled with the infrastructure they run on, but algorithmically defined and thus somewhat inflexible.

RDF: (Resource Description Framework) According to the W3C, "*a standard model for data interchange on the web*". It is most notably used as a basis for most Semantic Web technologies.
→ `http://www.w3.org/TR/rdf-primer/`

RDFS: (RDF Schema) W3C recommendation specifying how to use RDF to define vocabularies.
→ `http://www.w3.org/TR/rdf-schema/`

Scientific Workflow: A workflow meant to formalize and/or enact a simulation.

Scientific workflow framework: Workflow framework designed to handle simulations and dedicated to one or multiple Scientific Workflow model(s).

Semantic annotation: In general, any annotation that defines a semantic concept or links part of a document with a semantic concept defined elsewhere.

Simulation: Scientific experiment carried out entirely or partially via computers.

SoC: (Separation of Concerns) Software design principle of ensuring that loosely-related aspects of a system are developed separately and/or easily untangled. For instance, decoupling log handling from the program whose activity is logged respects the principle, whereas indiscriminately mixing both aspects does not.

SPARQL: (SPARQL Protocol and RDF Query Language) Standard defined by the W3C to query and manipulate knowledge graphs.
See also RDF.
→ `http://www.w3.org/TR/rdf-sparql-query/`

WfMC: (Workflow Management Coalition) According to their website, "*a global organization of adopters, developers, consultants, analysts, as well as university and research groups engaged in workflow and [Business Process Management]*".
→ `http://www.wfmc.org/`

Workflow: According to the WfMC: "*the computerized facilitation or automation of a business process, in whole or part*".

Workflow framework: Set of tools to enable the use of workflows, most notably their creation, edition, enactment, deployment and monitoring.

References

1. Ian Taylor, Ewa Deelman, Dennis Gannon, and Matthew Shields. *Workflows for e-Science*. Springer-Verlag, 2007.
2. Roger Barga and Dennis Gannon. Scientific versus Business Workflows. In *Workflows for e-Science* [1], chapter 2, pages 9–16.
3. Katharina Görlach, Mirko Sonntag, Dimka Karastoyanova, Frank Leymann, and Michael Reiter. Conventional Workflow Technology for Scientific Simulation. In *Guide to e-Science*, pages 323–352. Springer London, 2011.
4. Mirko Sonntag, Dimka Karastoyanova, and Frank Leymann. The Missing Features of Workflow Systems for Scientific Computations. In *Proceedings of the 3rd Grid Workflow Workshop(GWW)*, pages 209–216, Paderborn, Germany, 2010. Gesellschaft für Informatik.
5. Johan Montagnat, Benjamin Isnard, Tristan Glatard, Ketan Maheshwari, and Mireille Blay-Fornarino. A data-driven workflow language for grids based on array programming principles. In *Workshop on Workflows in Support of Large-Scale Science(WORKS'09)*, pages 1–10, Portland, USA, November 2009. ACM.
6. Tom Oinn, Matthew Addis, Justin Ferris, Darren Marvin, Martin Senger, Mark Greenwood, Tim Carver, Kevin Glover, Matthew R. Pocock, Anil Wipat, and Peter Li. Taverna: A tool for the composition and enactment of bioinformatics workflows. *Bioinformatics journal*, 17(20):3045–3054, 2004.
7. Ewa Deelman, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, K. Vahi, G. Bruce Berriman, John Good, Anastasia Laity, Joseph C. Jacob, and Daniel S. Katz. Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems. *Scientific Programming Journal*, 13(3):219–237, 2005.
8. Péter Kacsuk and Gergely Sipos. Multi-Grid, Multi-User Workflows in the P-GRADE Grid Portal. *Journal of Grid Computing (JOGC)*, 3(3-4):221 – 238, September 2005.
9. Bertram Ludäscher, Ikay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A. Lee, Jing Tao, and Yang Zhao. Scientific Workflow Management and the Kepler System. *Concurrency and Computation: Practice & Experience (CCPE)*, 18(10):1039 – 1065, August 2006.
10. Tristan Glatard, Johan Montagnat, Diane Lingrand, and Xavier Pennec. Flexible and efficient workflow deployement of data-intensive applications on grids with MOTEUR.

International Journal of High Performance Computing Applications (IJHPCA) Special issue on Special Issue on Workflows Systems in Grid Environments, 22(3):347–360, August 2008.

11. Yong Zhao, Mihael Hategan, B. Clifford, Ian Foster, Gregor von Laszewski, I. Raicu, T. Stef-Praun, and M. Wilde. Swift: Fast, Reliable, Loosely Coupled Parallel Computation. In *IEEE International Workshop on Scientific Workflows*, Salt-Lake City, Utah, USA, July 2007.
12. Ketan Maheshwari and Johan Montagnat. Scientific workflows development using both visual-programming and scripted representations. In *International Workshop on Scientific Workflows(SWF'10)*, Miami, Florida, USA, July 2010. IEEE.
13. Zhijie Guan, Francisco Hernández, Purushotham Bangalore, Jeff Gray, Anthony Skjellum, Vijay Velusamy, and Yin Liu. Grid-Flow: a Grid-enabled scientific workflow system with a Petri-net-based interface. *Concurrency and Computation: Practice & Experience (CCPE)*, 18(10):1115–1140, 2006.
14. Daniel Garijo and Yolanda Gil. A new approach for publishing workflows: abstractions, standards, and linked data. In *Proceedings of the 6th workshop on Workflows in support of large-scale science(WORKS)*, pages 47–56, New York, NY, USA, 2011. ACM.
15. Chunhyeok Lim, Shiyong Lu, A. Chebotko, and F. Fotouhi. Prospective and Retrospective Provenance Collection in Scientific Workflow Environments. In *IEEE International Conference on Services Computing(SCC)*, pages 449–456, July 2010.
16. Yolanda Gil, Ewa Deelman, M.H. Ellisman, Thomas Fahringer, Geoffrey Fox, Dennis Gannon, Carole Goble, Miron Livny, Luc Moreau, and Jim Myers. Examining the Challenges of Scientific Workflows. *Computer*, 40:24–32, 2007.
17. T.M McPhillips, S. Bowers, Daniel Zinn, and Bertram Ludäscher. Scientific workflow design for mere mortals. *Future Generation Computer Systems (FGCS)*, 25(5):541–551, 2009.
18. S. Bowers. Scientific Workflow, Provenance, and Data Modeling Challenges and Approaches. *Journal on Data Semantics*, 1(1):19–30, 2012.
19. Jia Yu and Rajkumar Buyya. A taxonomy of scientific workflow systems for grid computing. *ACM SIGMOD records (SIGMOD)*, 34(3):44–49, September 2005.
20. Yashwant Singh and Manu Sood. The Impact of the Computational Independent Model for Enterprise Information System Development. *International Journal of Computer Applications*, 11(8):24–28, 2010.
21. Nadia Cerezo and Johan Montagnat. Scientific Workflow Reuse through Conceptual Workflows. In *6th Workshop on Workflows in Support of Large-Scale Science(WORKS'11)*, Seattle, WA, USA, November 2011. ACM.
22. Adrien Marion, Germain Forestier, Hervé Liebgott, Carole Lartizien, Hugues Benoit-Cattin, Sorina Camarasu-Pop, Tristan Glatard, Rafael Ferreira Da Silva, Patrick Clarysse, Sébastien Valette, Bernard Gibaud, Patrick Hugonnard, Joachim Tabary, and Denis Friboulet. Multi-modality image simulation of biological models within VIP. In *24th International Symposium on Computer-Based Medical Systems(CBMS)*, pages 1–6, Bristol, UK, June 2011.
23. Thomas Fahringer, Radu Prodan, Rubing Duan, Francesco Nerieri, Stefan Podlipnig, Jun Qin, Mumtaz Siddiqui, Hong-Linh Truong, Alex Villazon, and Marek Wieczorek. ASKALON: A Grid Application Development and Computing Environment. In *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing(GRID)*, pages 122–131, Washington, DC, USA, 2005. IEEE Computer Society.
24. Duncan Hull, Katy Wolstencroft, Robert Stevens, Carole Goble, Matthew R. Pocock, Peter Li, and Tom Oinn. Taverna: a tool for building and running workflows of services. *Nuclear Instruments and Methods in Physics Research A*, 34(Web Server issue):729–732, July 2006.
25. Ian Taylor, Matthew Shields, Ian Wang, and Andrew Harrison. The Triana Workflow Environment: Architecture and Applications. In *Workflows for e-Science* [1], chapter 20, pages 320–339.
26. Paolo Missier, Stian Soiland-Reyes, Stuart Owen, Wei Tan, Aleksandra Nenadic, Ian Dunlop, Alan Williams, Tom Oinn, and Carole Goble. Taverna, reloaded. In *SSDBM 2010*, Heidelberg, Germany, June 2010.
27. Antoon Goderis, Ulrike Sattler, Phillip Lord, and Carole Goble. Seven Bottlenecks to Workflow Reuse and Repurposing. In *The Semantic Web ? ISWC 2005(LNCS)*, pages 323–337. Springer, Heidelberg, Germany, 2005.

28. Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns: Abstraction and Reuse of Object-Oriented Design. *Medical Image Analysis (MedIA)*, 707:406–431, 1993.
29. Yolanda Gil, P.A. Gonzales-Calero, Jhie Kim, J. Moody, and V. Ratnakar. A semantic framework for automatic generation of computational workflows using distributed data and component catalogues. *Journal of Experimental & Theoretical Artificial Intelligence*, 23(4):389–467, 2011.
30. Yolanda Gil, V. Ratnakar, Kim Jihie, J. Moody, Ewa Deelman, P.A. Gonzales-Calero, and P. Groth. Wings: Intelligent Workflow-Based Design of Computational Experiments. *IEEE Intelligent System*, 26(1):62–72, January 2011.
31. Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-Oriented Programming. In *Proceedings of the European Conference on Object-Oriented Programming*, volume 1241, pages 220–242, 1997.
32. Tudor B. Ionescu, Andreas Piater, Walter Scheuermann, and Eckart Laurien. An Aspect-Oriented Approach for the Development of Complex Simulation Software. *Journal of Object Technology (ETH Zurich)*, 9(1):161–181, January 2010.
33. David Schumm, Dimka Karastoyanova, Oliver Kopp, Frank Leymann, Mirko Sonntag, and Steve Strauch. Process fragment libraries for easier and faster development of process-based applications. *Journal of Systems Integration*, 2(1):39–55, 2011.
34. David Schumm, Dimitrios Dentsas, Michael Hahn, Dimka Karastoyanova, Frank Leymann, and Mirko Sonntag. Web Service Composition Reuse through Shared Process Fragment Libraries. In *Web Engineering(ICWE)*, volume 7387 of *LNCS*, pages 498–501, Berlin, Germany, 2012. Springer Berlin Heidelberg.
35. Aldo Gangemi, Nicola Guarino, Claudio Masolo, Alessandro Oltramari, and Luc Schneider. Sweetening Ontologies with DOLCE. In *Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web(LNCS)*, pages 166–181. Springer Berlin Heidelberg, 2002.
36. Germain Forestier and Bernard Gibaud. Semantic models in VIP. Technical report, INSERM, Rennes, France, November 2010.
37. Matheus Hauder, Yolanda Gil, Yan Liu, Ricky Sethi, and Hyunjoon Jo. Making data analysis expertise broadly accessible through workflows. In *Proceedings of the 6th workshop on Workflows in support of large-scale science(WORKS)*, pages 77–86, New York, NY, USA, 2011. ACM.
38. D. De Roure, Carole Goble, and Robert Stevens. The Design and Realisation of the myExperiment Virtual Research Environment for Social Sharing of Workflows. *Future Generation Computer Systems (FGCS)*, 2008.
39. Khalid Belhajjame, Suzanne M. Embury, Norman W. Paton, Robert Stevens, and Carole Goble. Automatic annotation of Web services based on workflow definitions. *ACM Trans. Web*, 2(2), May 2008.
40. Katy Wolstencroft, Pinar Alper, Duncan Hull, Chris Wroe, Phillip Lord, Robert Stevens, and Carole Goble. The myGrid Ontology: Bioinformatics Service Discovery. *International Journal of Bioinformatics Research and Applications (IJBRA)*, 2007.
41. Khalid Belhajjame, Katy Wolstencroft, O. Corcho, Tom Oinn, F. Tanoh, A. William, and Carole Goble. Metadata Management in the Taverna Workflow System. In *Cluster Computing and the Grid, 2008. 8th IEEE International Symposium on(CCGRID)*, pages 651–656, May 2008.
42. Sean Bechhofer, David de Roure, Matthew Gamble, Carole Goble, and Iain Buchan. Research Objects: Towards Exchange and Reuse of Digital Knowledge. *The Future of the Web for Collaborative Science (FWCS)*, April 2010.
43. Paolo Missier, Katy Wolstencroft, F. Tanoh, Sean Bechhofer, Khalid Belhajjame, S. Pettifer, and Carole Goble. Functional Units: Abstractions for Web Service Annotations. In *6th World Congress on Services(SERVICES)*, pages 306–313, July 2010.
44. Ikay Altintas, Adam Birnbaum, Kim Baldridge, Wibke Sudholt, M. Miller, Celine Amoreira, Yohann Potier, and Bertram Ludäscher. A Framework for the Design and Reuse of Grid Workflows. In *Scientific Applications of Grid Computing(LNCS)*, pages 295–299. Springer, 2005.
45. S. Bowers, Bertram Ludäscher, A.H.H. Ngu, and T. Critchlow. Enabling Scientific Workflow Reuse through Structured Composition of Dataflow and Control-Flow. In *IEEE Workshop on Workflow and Data Flow for Scientific Applications(SciFlow)*, Atlanta, USA, April 2006.

46. George Chin, Chandrika Sivaramakrishnan, T. Critchlow, Karen Schuchardt, and A.H.H. Ngu. Scientist-Centered Workflow Abstractions via Generic Actors, Workflow Templates, and Context-Awareness for Groundwater Modeling and Analysis. In *IEEE World Congress on Services(SERVICES)*, pages 176–183, July 2011.
47. Mirko Sonntag, Katharina Görlach, Dimka Karastoyanova, Frank Leymann, Polina Malets, and David Schumm. Views on Scientific Workflows. In *Perspectives in Business Informatics Research(LNBIP)*, pages 321–335. Springer Berlin Heidelberg, 2011.
48. Kassian Plankensteiner, Johan Montagnat, and Radu Prodan. IWIR: A Language Enabling Portability Across Grid Workflow Systems. In *Workshop on Workflows in Support of Large-Scale Science(WORKS'11)*, Seattle, USA, November 2011.