VIP ANR-09-COSI-03
Virtual Imaging Platform

D1.2.2: Simulation Workflow Designer

Nadia Cerezo	MODALIS (I3S)	cerezo@i3s.unice.fr
Johan Montagnat	MODALIS (I3S)	johan.montagant@cnrs.fr

Abstract

This deliverable describes the workflow designer developed in the context of the VIP project ANR-09-COSI-03 (http://www.creatis.insa-lyon.fr/vip). Scientific workflows design is difficult for end users as they usually are not specialists of scientific workflow programming languages and these languages tackle rather low-level and complex features of distributed computing infrastructures on which workflows are enacted. The aim of the VIP workflow designer is to provide a high-level graphical interface and a workflow abstraction model that ease workflow design by focussing on domain concepts and automatising as much as possible technical code generation.

Contents

1 Introduction

Workflow formalisms provide means to formally describe complex experiments that execute on distributed computing infrastructures. Yet, most existing scientific workflow formalisms remain complex to use for scientists who are not experts in *Distributed Computing*. One issue with scientific workflow languages is that they assume a rather close relationship between the description of a scientific process and its implementation. 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).

The approach adopted in the VIP project is to model simulations at an *abstraction level* that is closer to the end-user's domain. Simulations are modelled at three distinct levels of abstraction that align with those defined in the *Model Driven Architecture*[1] (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 Models* (PDM) in the MDA.

[1] MDA: http://www.omg.org/mda/

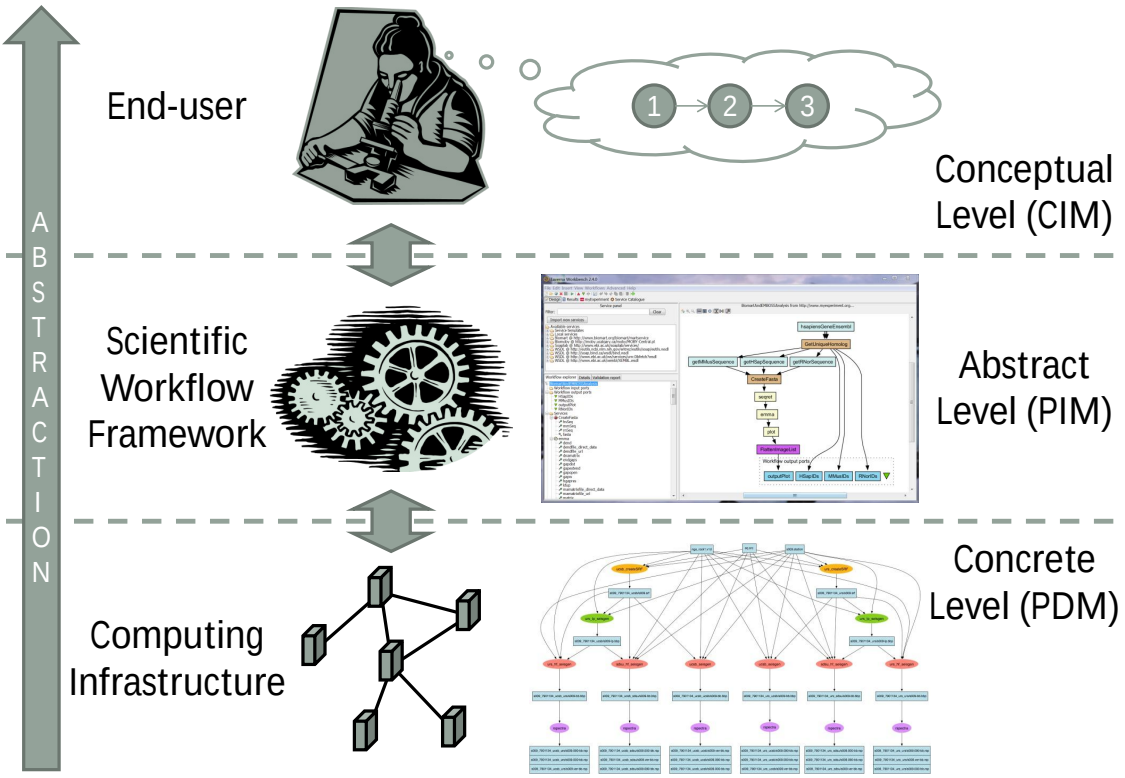


Figure 1: Scientific Workflow Abstraction Levels

- 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 Models* (PIM) in the MDA. In practice though, frameworks end up tied to target infrastructures enough to warrant interoperability projects like SHIWA[2].

- 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 Models* (CIM) in the MDA, for they remain independent from how the system is or will be implemented.

This deliverable describes the conceptual workflow model defined in the VIP project to formalize simulations at a higher level of abstraction, the transformation process used to generate executable workflows in the GWENDIA language [6], and the graphical workflow designer developed. Our 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.

The transformation from a Conceptual to an Abstract workflow involves two steps as

[2] http://www.shiwa-workflow.eu/

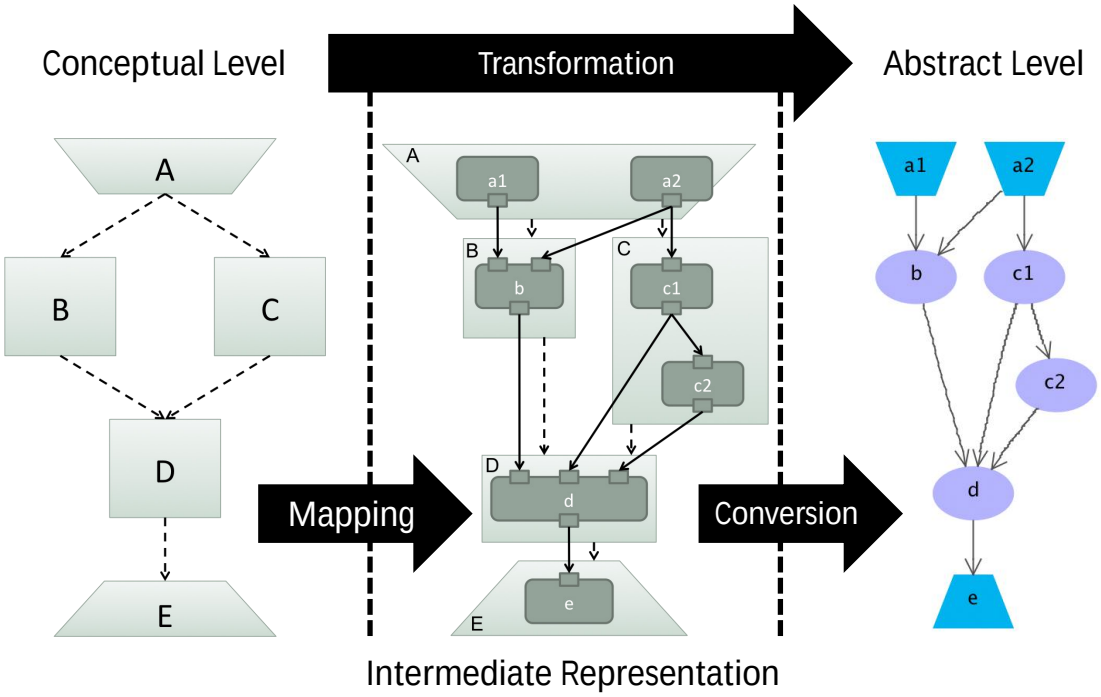


Figure 2: Transformation Process

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 the GWENDIA target Abstract Workflow language.

The **Conceptual Workflow Meta-model** is shown in *Unified Modeling Language*[3] (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 applied to the Sorteo simulator and Section 7 illustrates the usage of the VIP workflow designer to generate this simulation workflow.

2 Semantics

The **Conceptual Workflow Model** separates workflow elements into two abstraction layers: a high-level one 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

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

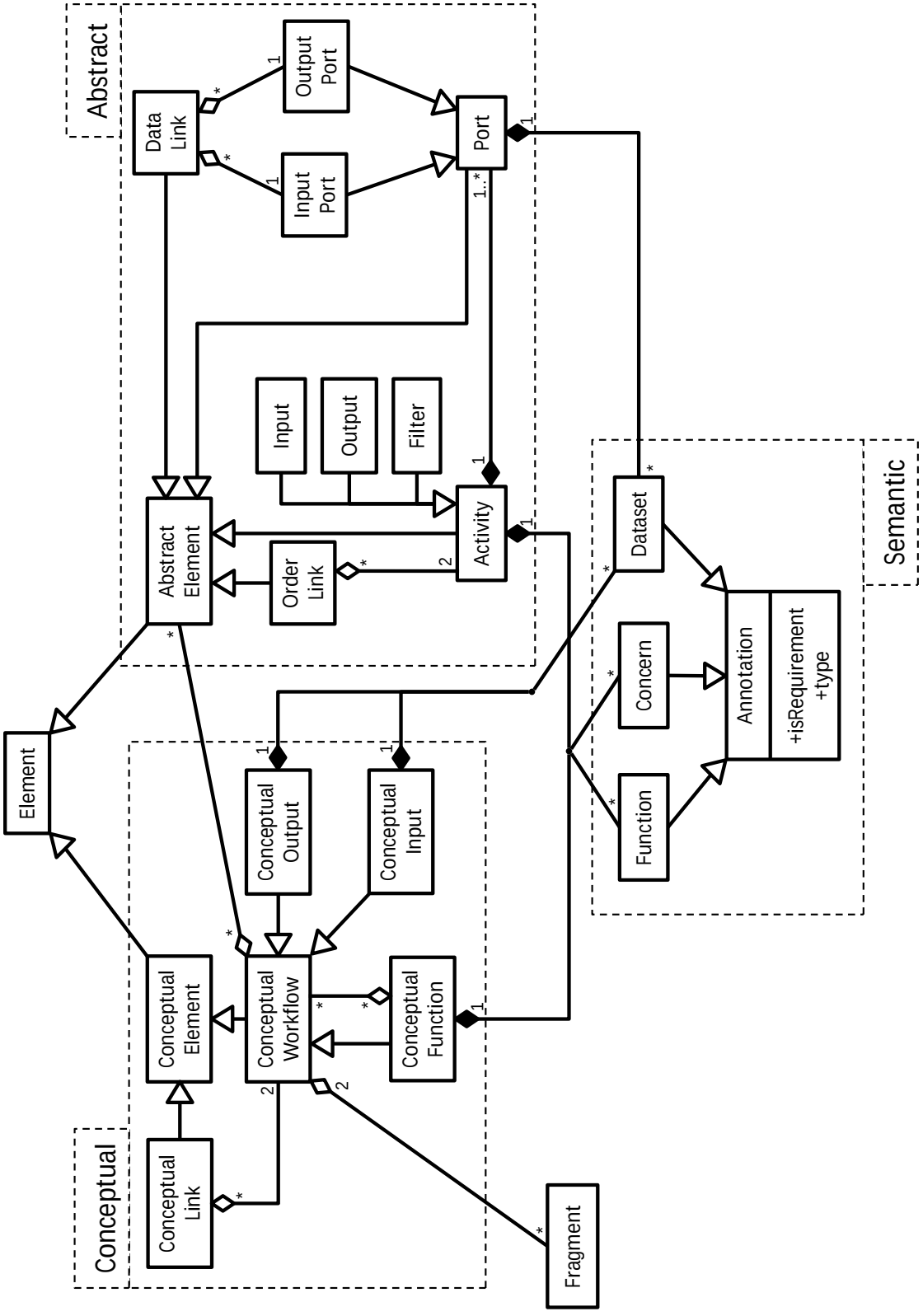


Figure 3: Conceptual Workflow Meta-model

Abstract Workflow formalisms.

To capture the scientific process and assist the user in implementing it, we leverages *Semantic Web* technologies [4] that can handle knowledge and know-how from multiple domains in a computer-legible and yet accessible way. Resources are identified by *Uniform Resource Identifier* (URI) which are described and connected through *Resource Description Framework*[4] (RDF) "`subject predicate object`" triples into a *knowledge graph*. The *SPARQL Protocol and RDF Query Language*[5] (SPARQL) query language is 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.

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:

- the Conceptual Workflow Model itself is described in an RDFS-based *COnceptual WORKflow* (COWORK) ontology[6] developed in the context of this work. Through COWORK, 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; 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**. 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:

- 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

[4]RDF: http://www.w3.org/TR/rdf-primer/
[5]SPARQL: http://www.w3.org/TR/rdf-sparql-query/
[6]COWORK ontology: http://www.i3s.unice.fr/~cerezo/cowork/latest/cowork.rdfs

- 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.

Figure 4: Graphical Convention - Semantic elements

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

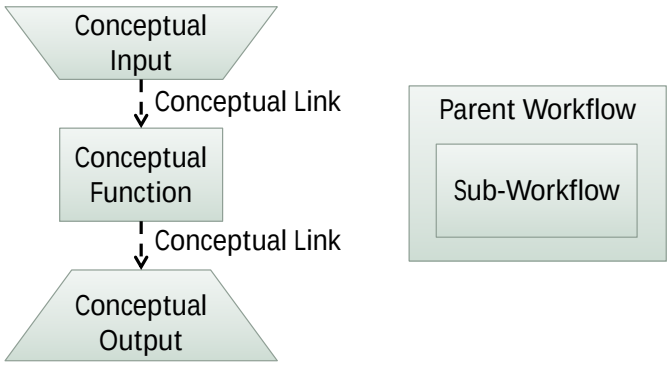


Figure 5: Conceptual elements graphical representation used in the VIP workflow designer GUI.

The *Conceptual* part of the Conceptual Workflow Meta-model, shown on Figure 3, pertains to simulation modeling at the highest level of abstraction. Figure 5 illustrates the graphical convention adopted in the VIP workflow designer GUI for all Conceptual elements of the Conceptual Workflow Model.

Conceptual Workflows are modeled through *Nested Directed Cyclic Graphs* whose vertices represent operations to be performed and edges represent dependencies between operations. 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 relation 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**.

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

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.

4 Abstract Elements

The end goal of Conceptual Workflow design is to produce an executable workflow executable by the MOTEUR workflow engine. The *Abstract* part of the Conceptual Workflow Meta-model shown on Figure 3 pertains to Abstract Elements (illustrated in Figure 6) which are used to describe the abstract workflow to be converted into GWENDIA.

An Activity represents an "atomic" task in the target workflow. 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.

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

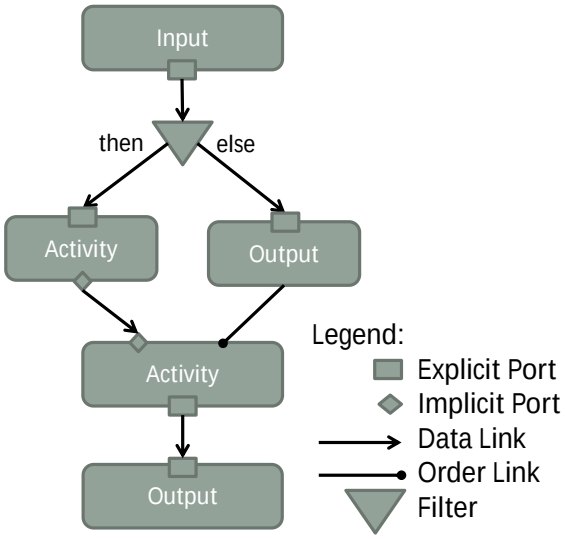


Figure 6: Abstract element graphical representation used in the VIP workflow designer GUI.

- **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.

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**. 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 7 (left) 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.

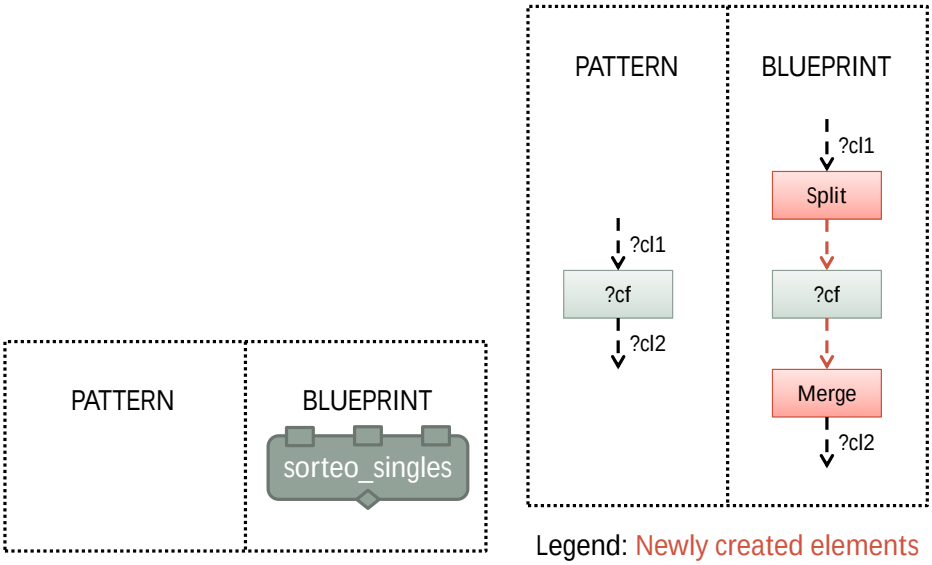


Figure 7: Left: Typical Activity Fragment. Right: Split and Merge Fragment.

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.

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 7 (right) 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.

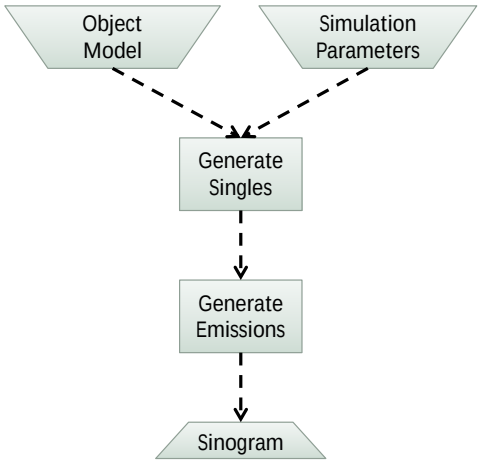
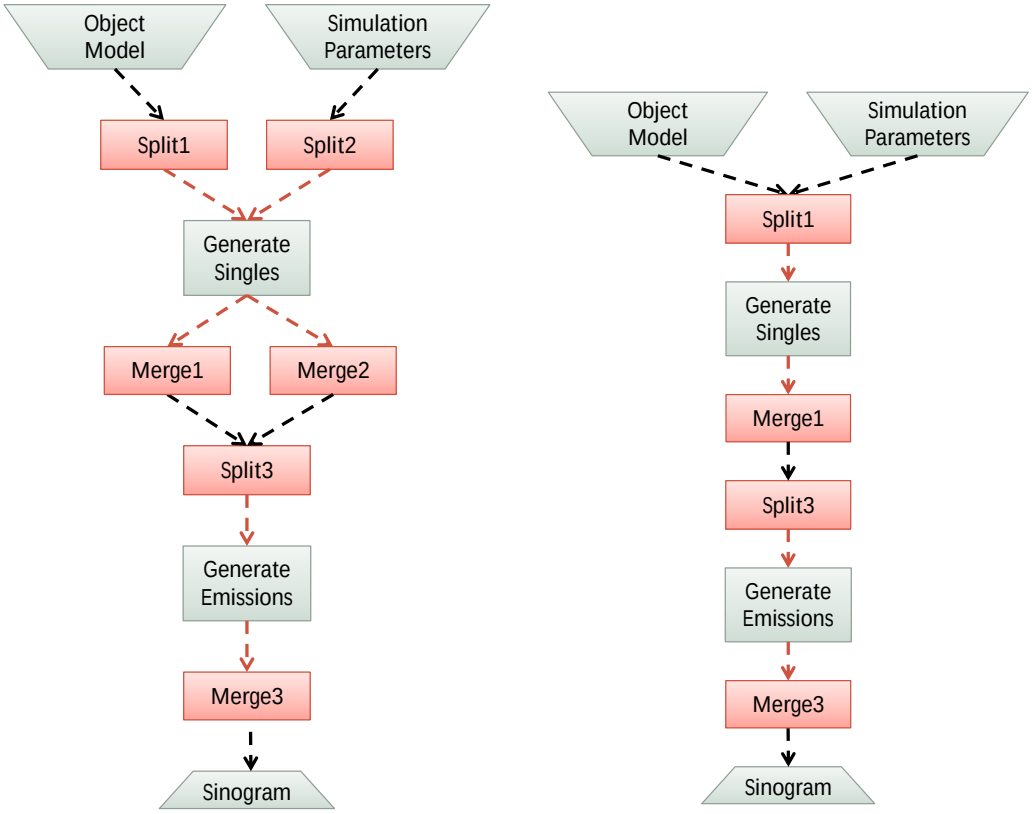


Figure 8: Weaving Split and Merge - Base Workflow

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. Let us consider the simple workflow shown on Figure 8, 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 9(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 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 9(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-



(a) Query Result (b) Final Result

Figure 9: Weaving Split and Merge

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.

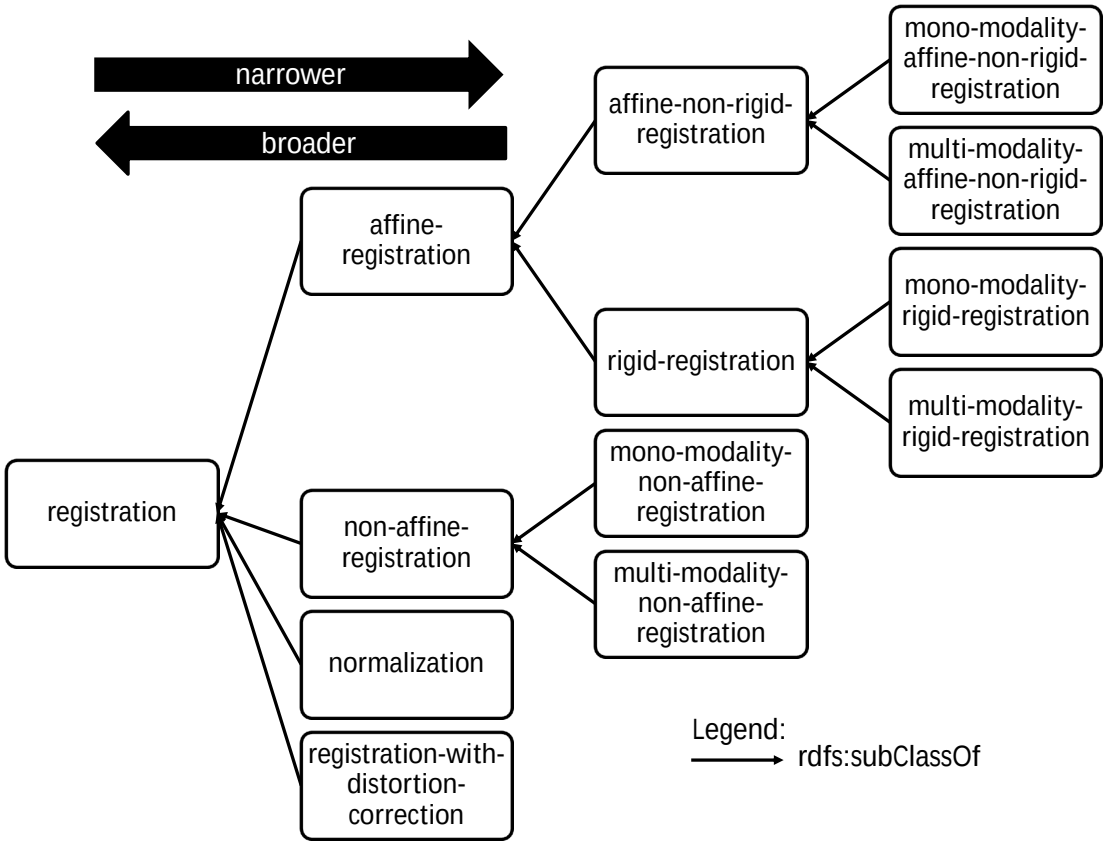


Figure 10: VIP Ontology - Registration Processes Taxonomy

To illustrate those matches, let us consider the small excerpt of the Virtual Imaging Platform[7] (VIP) ontology shown on Figure 10. 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 [2] (interested readers please refer to [1] for more details).

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

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

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

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.

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 [5] assist the user very thoroughly through the creation of workflows, but are restricted to curated application domains (data mining pipelines in the case of [5]). 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. 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.

6 Sorteo simulation workflow generation

To illustrate the modeling and mapping of Conceptual Workflows, let us take a closer look at the Sorteo PET simulator.

All four simulators seminally included in the Virtual Imaging Platform fit the same high-level Conceptual Workflow, shown on Figure 11, with slightly different Requirements for the main Conceptual Function: `medical-image-simulation` in general and `PET-simulation` in the case of Sorteo. No matter the simulator, it always bears the `SplitAndMerge` Requirement.

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 12 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 8) used as example in Section 5.1 to illustrate the

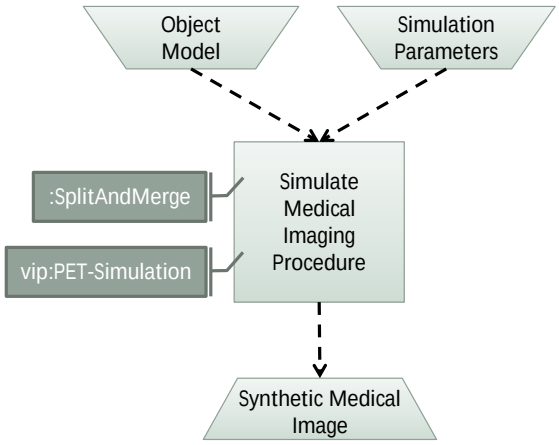


Figure 11: VIP Simulator Core Template

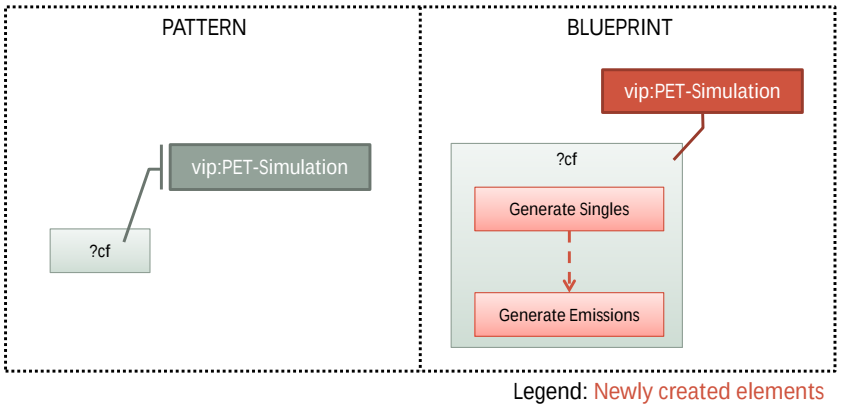


Figure 12: Sorteo simulator - PET Fragment

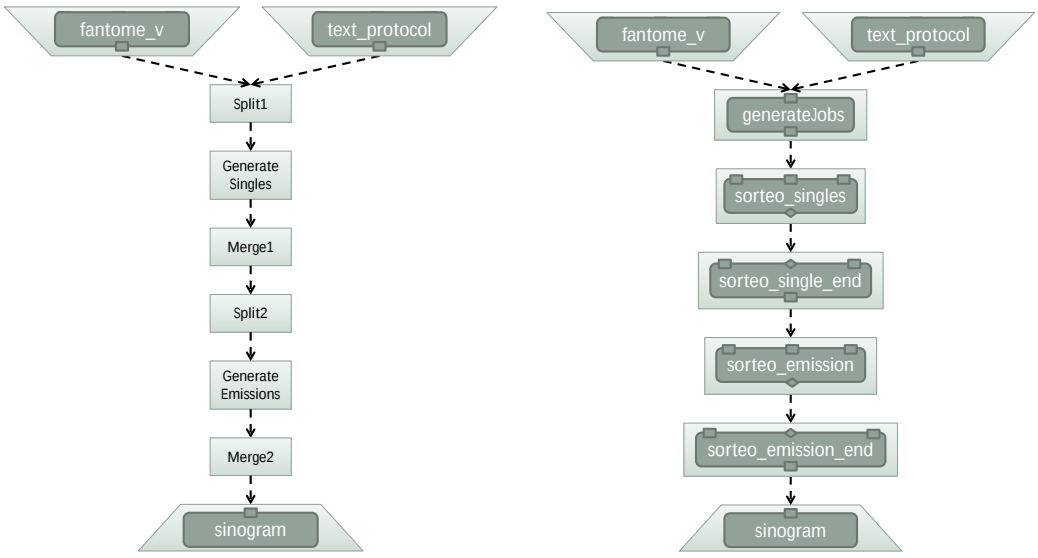
Weaving of the `SplitAndMerge` Fragment (Figure 7, right). The result of that Weaving is shown on Figure 9(b).

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 13(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 13(b) shows the workflow after all Conceptual Functions have been matched.

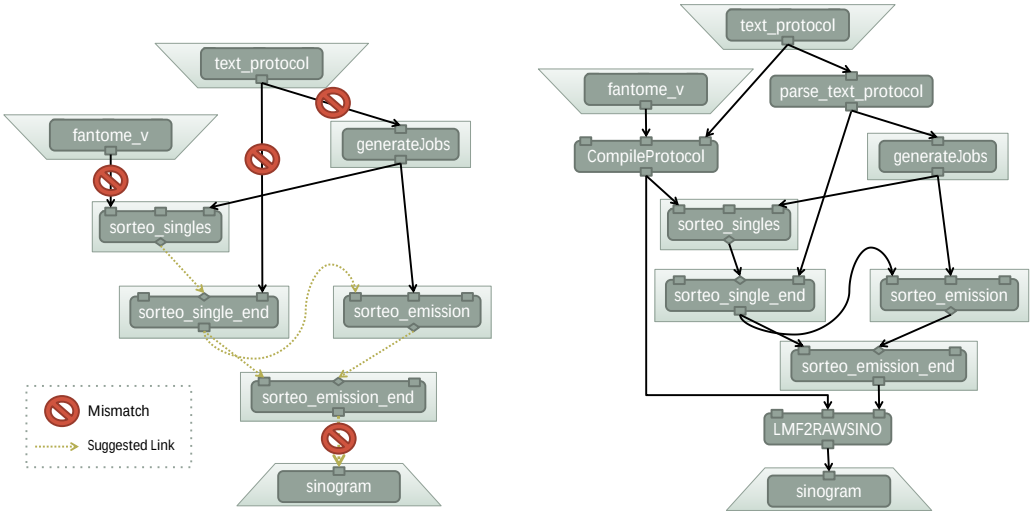
Figure 14(a) shows the links chosen by the user among links the system can suggest based on semantic types, links created by the user and the following mismatches:

- 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



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

Figure 13: Sorteo simulator - Discovery and Weaving



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

Figure 14: Sorteo simulator - Composition

`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.

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

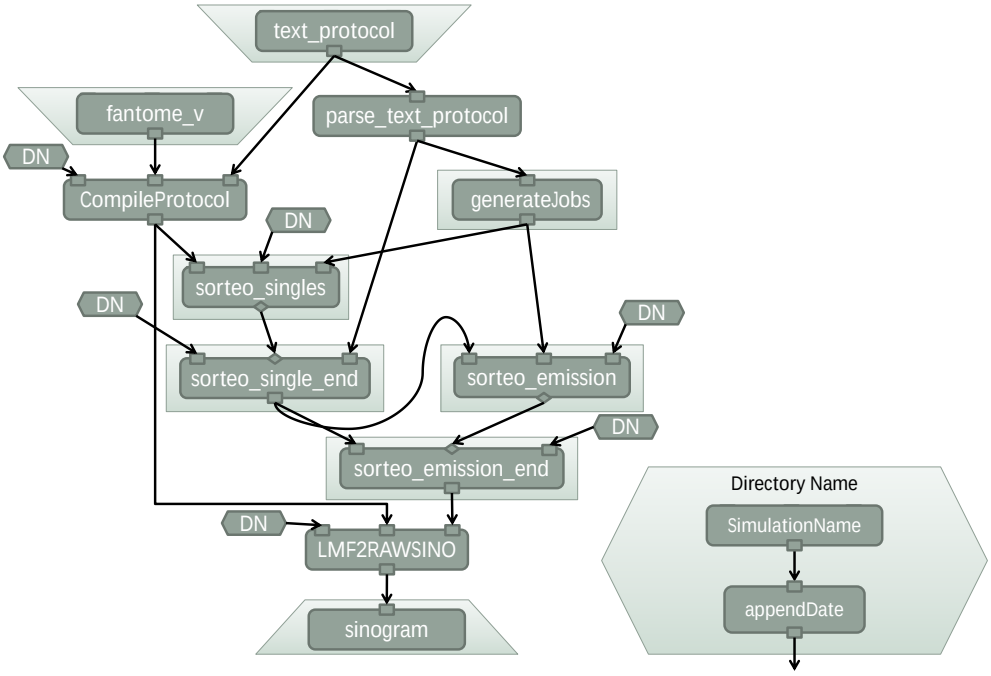


Figure 15: Sorteo simulator - Intermediary Representation

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 15 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.

7 VIP workflow designer

Quelques illustrations de la GUI lors de l'dition du workflow Sorteo .

What is left to do is to automatically convert this workflow into a target Abstract Workflow language. If we target GWENDIA [6], we obtain the executable workflow shown on Figure 16, which is a screenshot of the MOTEUR [3] client.

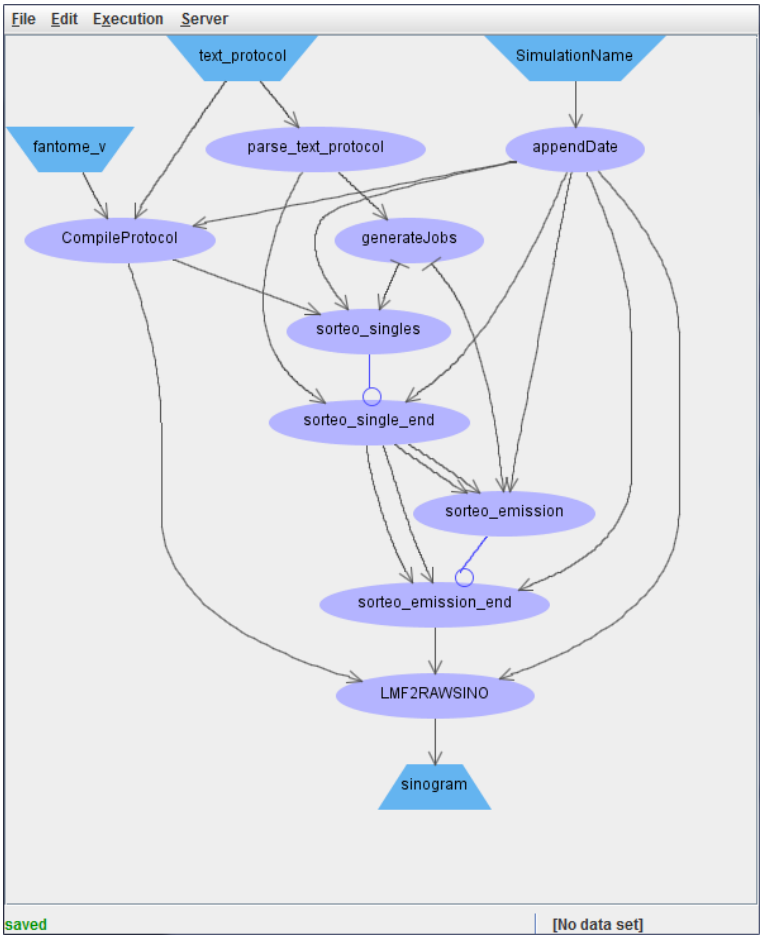


Figure 16: Sorteo simulator - Abstract Workflow (GWENDIA)

8 Conclusion

The VIP workflow designer is based on a Conceptual Workflow model and a user-assisted generative process. A transformation process was implemented to generate GWENDIA workflows executable by the MOTEUR workflow engine. The approach adopted is based on semantic Web technologies and it leverages the ontology designed in the context of the VIP project. It is generic and could be applied to other workflow engines (with a different language generator) or different domains (with a different domain ontology).

The work described in this deliverable is a generalisation of the task planed at the time of VIP proposal writing (in task T1.2.b, "assisted designer for simulation work-

flows"), where a simple workflow template customisation process had been anticipated. The workflow designer delivered is more ambitious and a step beyond the state-of-the-art as it implements a much more versatile workflow transformation process than workflow activity substitution. The counter part to this additional flexibility is the need for an knowledge base with semantically annotated workflow fragments. This knowledge base can be progressively enriched as the platform is being used for expertise capitalisation and improved workflow design assistance.

References

[1] Germain Forestier and Bernard Gibaud. Semantic models in VIP. Technical report, INSERM, Rennes, France, November 2010.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] Ian Taylor, Ewa Deelman, Dennis Gannon, and Matthew Shields. *Workflows for e-Science*. Springer-Verlag, 2007.