Composing Multiple Variability Artifacts to Assemble Coherent Workflows

Mathieu Acher, Philippe Collet, Alban Gaignard, Philippe Lahire, Johan Montagnat, Robert France

Software Quality Journal manuscript No.
(will be inserted by the editor)

Composing Multiple Variability Artifacts to Assemble Coherent Workflows

Mathieu Acher · Philippe Collet · Alban Gaignard · Philippe Lahire · Johan Montagnat · Robert B. France

Received: date / Accepted: date

Abstract The development of scientific workflows is evolving towards the systematic use of service oriented architectures, enabling the composition of dedicated and highly parameterized software services into processing pipelines. Building consistent workflows then becomes a cumbersome and error-prone activity as users cannot manage such large scale variability. This paper presents a rigorous and tooled approach in which techniques from Software Product Line (SPL) engineering are reused and extended to manage variability in service and workflow descriptions. Composition can be facilitated while ensuring consistency. Services are organized in a rich catalog which is organized as a SPL and structured according to the common and variable concerns captured for all services. By relying on sound merging techniques on the feature models that make up the catalog, reasoning about the compatibility between connected services is made possible. Moreover, an entire workflow is then seen as a multiple SPL (i.e., a composition of several SPLs). When services are configured within, the propagation of variability choices is then automated with appropriate techniques and the user is assisted in obtaining a consistent workflow. The approach proposed is completely supported by a combination of dedicated tools and languages. Illustrations and experimental validations are provided using medical imaging pipelines, which are representative of current scientific workflows in many domains.

Mathieu Acher · Philippe Collet · Alban Gaignard · Philippe Lahire · Johan Montagnat
I3S Laboratory (CNRS - UNSA)
Les Algorithmes - Bât Euclide B
2000 route des Lucioles – B.P. 121
F-06903 Sophia Antipolis Cedex
E-mail: {acher, collet, lahire, johan, gaignard}@i3s.unice.fr

Robert France
Computer Science Department
Colorado State University, USA
E-mail: france@cs.colostate.edu

1 Introduction

The goal of *Software Product Line* (SPL) engineering is to produce reusable artifacts that can be used to efficiently build members of a software product family [Pohl et al., 2005]. Similar to mass customization in traditional industry, SPL engineering aims to industrially develop and evolve quality software systems with minimal development effort and time-to-market. The general idea is that the reusable artifacts encapsulate common and variable aspects of a family of software systems in a manner that facilitates planned and systematic reuse. An important concern is thus the management of *variability*, i.e., the ability of an artifact to be efficiently extended, changed, customized or configured for use in a particular context [Svahnberg et al., 2005].

Traditionally, variability management assumed that all artifact variants of a software system were provided by a single source. But now, in many SPL environments, including software systems, the amount of functionality that needs to be developed to satisfy customer needs is far larger than what can be built from scratch in a reasonable amount of time. To solve this problem and facilitate mass customization, it is necessary to take into account externally developed components or applications, themselves being highly variable. The reuse of software components between different product lines in the consumer electronics domain is an example [van Ommering, 2002]. Some of these SPLs may be developed and maintained by external suppliers, some of which may compete to deliver similar products. The same observation can be made in the semiconductor industry where a set of components from several suppliers has to be integrated into a product [Hartmann and Trew, 2008]. In this context, variability in requirements is driven by several different dimensions, e.g., different product types and different geographic regions. Van der Storm considered not only variability at the level of one software product, but also each variable component as an entry-point for a certain software product (obtained through component composition) [van der Storm, 2004].

Some SPL engineering approaches now support defining and managing variability across different SPLs so that they can be *composed* [Pohl et al., 2005, Buhne et al., 2005, Reiser and Weber, 2007]. This "shift from variation to composition" and support for managing *multiple SPLs* (a.k.a. product populations [van Ommering and Bosch, 2002] or software ecosystems [Bosch, 2009]) are increasingly needed. Managing variabilities across these multiple SPLs is especially challenging when the SPLs are owned by different third-parties [Hartmann and Trew, 2008, Hartmann et al., 2009]. In practice there are many configurations to consider and hence automatic methods and techniques are needed to guarantee consistency properties on compositions of selected functionality. A recurrent activity is, for example, to determine which SPLs are able to provide a specific (combination of) feature(s) or not. Support for composing multiple variability descriptions can help domain engineers to produce coherent characterizations of valid combinations of features. Product (application) engineers also need support for producing valid product configurations that belong to one or several SPLs.

Interestingly, similar issues occur during the building of workflow, and in our experience, particularly during the construction of *scientific workflows*. With the ongoing evolution of such workflows towards the use of service oriented architectures, each task of them is a software service. Many different kinds of highly parameterized software services exist, introducing a very important variability at all levels. The tasks of identifying, tailoring and composing those services become tedious and error-prone. There is thus a strong need to manage this variability so that developers can more easily choose, configure and compose those services with automated consistency guarantees.

To tackle this problem, our approach is to consider services as SPLs, as they are provided by different researchers or scientific teams, while the entire workflow is then seen as a multiple SPL in which the different service SPLs are composed. Throughout this paper, our work is illustrated in the medical imaging area, in which such scientific workflows are built to analyze large medical data sets.

Analyzing the problem, we consider it as two-fold. The first challenge is to provide mechanisms that enable service providers (e.g., researchers, workflow or platform experts) to capture the commonalities and variabilities in parameterized services that are provisioned. The second challenge is concerned with providing support for tailoring and composing services in a way that service consumers can ensure the consistency of resulting workflows with well-defined properties. In preliminary work, we rely on feature models merging techniques [Acher et al., 2009, Acher et al., 2010a] to check the compatibility between connected services [Acher et al., 2010a] inside a workflow. But several limitations appeared in this solution. First, users had to specify variability of services from scratch as no catalog of services were available. Second, constraints in and between the variability descriptions were not taken into account, whereas they are highly represented in real cases. Finally, the checking process was not automated, even if some programming facilities were provided.

In this article, we notably raise all these limitations and present an extended and rigorous approach in which compositional techniques allow users to select among sets of existing services organized in a catalog (seen as a SPL), while reasoning on the compatibility between connected services to ensure consistency of an entire workflow (seen as a multiple SPL). After giving some background on medical imaging workflows and analyzing associated requirements (Section 2), we introduce feature models and associated compositional techniques (Section 3). Using them, the approach to organize services as multiple SPLs within a variability-driven catalog is briefly described. In Section 4, a typical usage scenario is unfolded from design to configuration of a workflow. We show how to specify variability requirements over the workflow and how consistency is soundly checked when available services in the catalog are composed, notably with constraints between services and in the workflow. We also describe how this part of the process can be fully automated in order to incrementally assist the user until deriving a consistent workflow product. Section 5 describes the realization, detailing how workflows are analyzed and how variability is described and reasoning made possible by generating appropriate code in a powerful Domain-Specific Language for Feature Models named FAMILIAR [Acher et al., 2011a]. Section 6 summarizes our contribution and studies its user assistance, degree of automation and applicability. We also report our first experiences on workflow building and discuss related and future work.

2 Variability in Scientific Workflows

Scientific workflows are increasingly used for the integration of existing, legacy tools and algorithms to build large and complex applications such as scientific data analysis pipelines or computational science experiments. Despite the growing interest observed in scientific workflow technologies in recent years, workflow design remains a challenging, tedious, and often error-prone process, that slows down the adoption of scientific workflows [Gil et al., 2007, McPhillips et al., 2009]. In particular, although catalogs of domain-specific data processing services are common, the low-level interface represen-

tations used (*e.g.* Web Services) usually only provides information suited to assess the technical consistency of different services connected within a workflow. There is absolutely no guarantee regarding the coherency of the process composed, nor its validity from an application point of view. The use of software product lines techniques to gain potential advantages in terms of time, effort, cost, efficiency and agility may alleviate these problems. Our work is illustrated through the medical imaging area, which is typical of the usage of scientific workflows executed over compute-intensive distributed infrastructures such as grids. In this domain, grids help in building patient-specific models and in reducing computing time for meeting time constraints from clinical practice. The rest of this section will introduce our example and identify its sources of variability to determine associated requirements.

2.1 Medical Imaging Workflows

In the medical image analysis area, distributed computing capabilities are used for many purposes, ranging from validation and optimization processes of specific algorithms to overall reduction of computing time. Besides, image analysis pipelines are scientific, data-driven workflows which are undergoing homogenization nowadays, strongly motivated by the need for mutualizing software development and easily comparing results. This homogenization is conducted through the usage of common data formats and means to reuse algorithms.

In order to facilitate it, Service-oriented architectures (SOAs) [Foster et al., 2002] are increasingly used and aim at *i)* producing reusable self-contained, distributed imaging services, decoupled and abstracted from technical grid platforms ; *ii)* providing standardized interfaces for invoking wrapped application codes as well as information exchange protocols and *iii)* composing these atomic services to describe processing pipelines as complex workflows. Using SOAs, medical experts essentially compose different kinds of processing on images, each algorithm being provided by a service.

Running example. We use as an illustration an existing service-oriented workflow designed to conduct experiments on Alzheimer's disease [Lorenzi et al., 2010]. This disease is a neurodegenerative pathology which can be characterized by an atrophy of the brain. The workflow illustrated in Fig. 1 is based on several image processing activities aimed at tracking the evolution of Alzheimer disease through a longitudinal study. The disease follow up consists in comparing several MRIs from the same patient acquired over time, to detect changes in the volume of the brain and compute a brain tissue atrophy coefficient. In order to be compared in the last steps of the workflow, source MRIs first need to be homogenized both in terms of intensity biases and space alignment. It must be noted that, as in many similar scientific workflows, the complexity lies in the correct pre-processing of data, which is generally frustrating for the scientist end-users.

In Fig. 1, the blue boxes at the top represent the input images: the *Image sequence* box represents MRIs acquired at a given time (T_0+6 months and T_0+12 months), the *Reference image* represents the MRI acquisition at T_0, considered as the patient's reference. This configuration of the workflow will lead to two invocations, giving two estimations of brain atrophy, at time T_0+6 and T_0+12. The first image processing activity, *Bias correction* is a general restoration procedure which involves removing voxel inhomogeneities in the magnetic field of the MRI equipment, used to improve

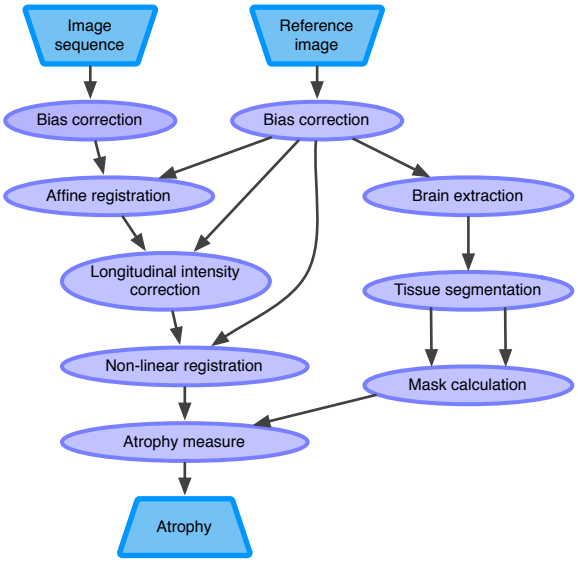


Fig. 1 Image processing activities involved in the calculation of brain atrophy.

the result of image post-processing algorithms. Then an *Affine registration* process is performed in which a spatial alignment is estimated so that the MRI considered is translated, reoriented and scaled to be superimposed on the patient's reference MRI. The next step *Longitudinal intensity correction* is another intensity homogenization procedure that normalizes intensities between the different images acquired over time. At the same time, the right branch of the workflow aims at identifying brain tissues (grey and white matter) through the *Brain extraction* and *Tissue segmentation* activities to finally create a mask (*Mask calculation*) delineating grey and white matter of the patient's brain at T_0. Finally, the comparison of MRIs starts by estimating the deformation field, which corresponds to the *Non-linear registration* activity, between the reference and the "moving" MRIs. The last step consists in applying the deformation field to the mask of the brain tissues in order to estimate the changes in the volume of brain tissues, and estimate, as the final result of the workflow, a potential atrophy.

2.2 Sources of Variability

The pre-processing involved in this workflow are based on three typical categories of image processing activities: *Restoration*, *Registration* and *Segmentation*. There are various ways in which the goals of these categories of activities can be accomplished. At the workflow level, end-users must address the variability when realizing these activities.

Functional variability usually appears at workflow design time. For instance, the two activities *Bias correction* and *Longitudinal intensity correction* realize the same coarse-grained activity, *Restoration*, but their fine-grained functionality varies because removing magnetic inhomogeneities is different than normalizing intensities from two different MR images. Another example comes from the *Registration* activities in which

the same kind of *functional variability* can be observed as there are significant differences between linear and non-rigid transformations.

At workflow runtime, end-users must cope with non functional concerns such as constraints related to the computing infrastructure or restricted access control. For example, to operationalize the *Brain extraction* activity, one may choose the BET tool from the FSL toolbox because it is fast at removing non-brain tissues and can be relocated to the end-user desktop. But if the accuracy of the segmentation is preferred to its computation time, another processing tool might be chosen with different constraints on the infrastructure. Using BET also introduces a deployment constraint as it depends on the full installation of another toolbox (FSL) and needs appropriate environment configuration.

More generally, for each process of a pipeline, numerous services are available on the grid and vary from different perspectives: the support of image formats (DICOM, Nifti, Analyze, etc.) and modality acquisition (MR, CT, PET, etc.), the support of network protocols, the algorithm method used to process an image, anatomical structures for which services are supposed to efficiently perform (Brain, Kidney, Breast, etc.), quality of service (QoS) provided in different contexts, etc. It must be noted that not only imaging but nearly all scientific services have a large number of input ports, parameters, data specificities, and dependencies at all levels, functional, non-functional and deployment related. The overall issue for scientific workflow users is thus to deal with services and their dependencies in their workflows while addressing a large amount of concerns. In our medical imaging illustration, from the workflow design time to run time, both domain-specific and technical knowledge are needed to resolve different forms of variability. This is typically accomplished by manually setting, among others, the choice of tools and the choice in their configuration. This type of manual variability management requires a considerable amount of time and effort, and can be a tedious and error-prone task.

2.3 Requirements

To tackle the above issues, our work addresses the following challenges. The first challenge is to capture commonalities and variabilities across a family of services in reusable parameterized services, e.g. identifying and organizing similar and recurrent imaging tasks such as registrations and corrections. The second challenge is related to providing support for tailoring and composing services to realize consistent workflows.

There are two categories of users for the NeuroLOG platform: (i) image analysis specialist create and deploy image analysis tools that are of interest for neuroscientists; and (ii) neuroscientists design data analysis experiments by composing such tools within specialized workflows. Rather than providing services and hoping that opportunities for reuse will arise during the design of a workflow, a proactive strategy is to plan which characteristics or features of a service are likely to be systematically reused. The ability to efficiently create many variations of a service and capitalize on its commonalities can improve its composability and increase the extent to which service logic is sufficiently generic so that it can be effectively reused. As discussed in previous papers, the difficulty of provisioning and composing parameterized services stems from the lack of mechanisms for managing variabilities within and across services [Acher et al., 2008, Acher et al., 2010b].

The goal of the SPL approach promoted in this article is to manage not only the variability of the services but also the variability of the resulting composed services. For structuring and managing variability information across a large amount of services, we identified the following requirements, emerging from the needs of both the image analysts and the neuroscientists:

- Mechanisms that enable service providers (image analysts) to capture the commonalities and variabilities in parameterized (imaging) services ;
- Assistance to the neuroscientists in selecting the appropriate service from among sets of existing services: they may want to search services matching several criteria to determine if at least one service can fulfill a specific feature or a set of features ;
- Ensuring that services are consistently composed in the resulting workflow. For example, connected services should inter-operate while exchanging medical images and support compatible formats. A sound formal basis together with tools are needed to support rigorous reasoning on a large number of services for ensuring that important properties are preserved ;
- Evolving services as the variability of services can evolve during time. Similarly, new services from new suppliers and scientists can be proposed. Consequently, there is a need to consistently maintain the set of existing services and favor the integration of new services.

In the remainder of this paper, we describe an approach that meets these requirements.

3 Engineering Services as Software Product Lines

A software product line (SPL) can be defined as "*a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way*" [Pohl et al., 2005]. SPL engineering is concerned with developing reusable artifacts that can be used extensively during the development of final products [Clements and Northrop, 2001, Pohl et al., 2005].

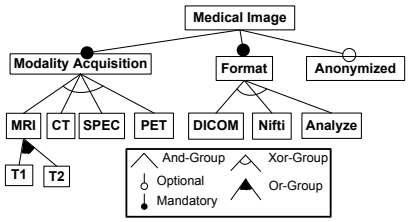
In our work, the goal is to develop reusable services that scientists can tailor and use to build customized workflows. A central activity is then the modeling and management of service variability. It must be noted that there are other important activities, such as testing services and workflow taking into account their variability, but this is out of the scope of the approach presented in this paper. First, we explain how variability of services is documented and represented with the feature modeling formalism. Second, the different services are organized (e.g., grouped together) and managed in a variability-aware catalog.

3.1 Modeling Variability of Services

One of the most practical techniques for modeling variability is feature modeling which aims at representing the common and variable *features* of a product family. Several definitions of feature appear in the literature, ranging from "anything users or client programs might want to control about a concept" [Czarnecki and Eisenecker, 2000],

Service Identifier	Supplier	Medical Image Support			Method
		Modality Acquisition	Format	Anonymized	
$ServiceSegm_1$	$Supplier_1$	MRI T1	DICOM	Yes	...
$ServiceSegm_2$	$Supplier_2$	MRI T2	DICOM	No	...
$ServiceSegm_3$	$Supplier_2$	SPECT	DICOM	Yes	...
...					...
$ServiceSegm_n$	$Supplier_1$	CT	Analyze	No	...

(a) Services documentation



(b) Feature model

Fig. 2 Services' documentation and corresponding feature model.

"a prominent or distinctive user-visible aspect, quality or characteristic of a software system or systems" [Kang et al., 1990] to "an increment in product functionality" [Batory, 2005]. In our work, services are distinguished by features which are domain abstractions relevant to medical imaging stakeholders

Feature Modeling. Feature models (FMs) [Kang et al., 1998], [Batory, 2005], [Schobbens et al., 2007] are widely used to compactly represent product commonalities and variabilities in terms of optional, alternative and mandatory features. FMs hierarchically structure application features into multiple levels of increasing detail. When decomposing a feature into subfeatures, the subfeatures may be optional or mandatory or may form *And*, *Or*, or *Alternative*-groups[1]. Fig. 2 shows an example of an FM. An FM defines a set of valid feature *configurations*. The validity of a configuration is determined by the semantics of FMs (e.g., DICOM, Nifti and Analyze are mutually exclusive image formats and cannot be selected at the same time in Fig. 2).

A valid configuration is obtained by selecting features in a manner that respects the following rules: *i)* If a feature is selected, its parent must also be selected; *ii)* If a parent is selected, all the mandatory subfeatures in its And group, exactly one subfeature in each of its Alternative groups, and at least one of its subfeatures in each of its Or groups must also be selected; *iii)* Constraints relating features in different subtrees must hold. The set of configurations represented by an FM can be described by a propositional formula defined over a set of Boolean variables, where each variable corresponds to a feature [Batory, 2005, Czarnecki and Wąsowski, 2007]. In the remainder of the paper, a configuration is defined as a set of selected features, for example, {MedicalImage, ModalityAcquisition, Format, SPEC, Nifti} is a valid configuration of the FM shown in Fig. 2. In Fig. 2(b), the FM compactly represents all the valid combination of features supported by services, documented in a two-dimensional array of data shown

[1] In this paper, we consider only FMs in their basic form [Czarnecki et al., 2006]. We do not consider other notations nor richer formalisms (e.g., cardinality-based FMs [Czarnecki et al., 2005]).

in Fig. 2(a). Such an FM can be manually elaborated by service developers or automatically extracted from service documentation using merging techniques (as we will show in Section 3.3).

Definition 1 introduces the terms used in this paper and defines the well-known relationship between an SPL and an FM.

Definition 1 (SPL and Feature Model) *A software product line SPL_i is a set of products (e.g., software services) described by a feature model FM_i. Each product of SPL_i is a combination of features and corresponds to a valid configuration of FM_i. A configuration c of FM_i is defined as a set of features selected, i.e., $c = \{f_1, f_2, \ldots, f_m\}$ with f_1, f_2, ..., f_m features of FM_i. $[\![FM_i]\!]$ denotes the set of valid configurations of the feature model FM_i.*

3.2 Multiple Software Product Lines

Our approach is to consider services as SPLs and to reuse and combine a set of SPLs to form a workflow. We denote a *multiple SPL M_{SPL} an SPL that manages a set of constituent SPLs $\{SPL_1, SPL_2, \ldots, SPL_n\}$ and whose set of products is described by a feature model $FM_{M_{SPL}}$.*

For instance, the entire workflow can be seen as a multiple SPL in which the different service SPLs are combined. In terms of FMs, it simply consists in *aggregating* FMs into a global FM $FM_{M_{SPL}}$. The input FMs FM_1, FM_2, ..., FM_n of $SPL_1, SPL_2, \ldots, SPL_n$ are aggregated under a synthetic root, say $ft_{synthetic}$, so that the root features of input FMs are child-mandatory features of $ft_{synthetic}$ in the global FM. The aggregate operation also supports cross-tree constraints between features so that separated FMs can be inter-related (see Section 4 for examples).

An important form of multiple SPL is *competing* multiple SPLs and is the main focus of this section. In a competing multiple SPL, each constituent SPL describes a product that competes with products described in other constituent SPLs. For each category of process to be performed in the workflow (e.g., segmentation, registration), there are several competing SPLs. For example, the three SPLs in Fig. 3 provide competing Segmentation Services with different features and/or with different combinations of features. To formalize the concept of competing multiple SPLs, we define its semantics in terms of the relationship between $FM_{M_{SPL}}$ and the FMs of the constituent SPLs, FM_1, FM_2, ..., FM_n (see Definition 2).

Definition 2 (Competing Multiple SPL) *In a competing multiple SPL, M_{SPL}, any configuration of $FM_{M_{SPL}}$ should correspond to at least one valid configuration of FM_1, FM_2, ..., FM_n. Formally: $\forall c \in [\![FM_{M_{SPL}}]\!] : c \in [\![FM_1]\!] \vee c \in [\![FM_2]\!] \vee \ldots \vee c \in [\![FM_n]\!]$*

3.3 Merging Techniques

When competing multiple SPLs in a domain exist, FMs representing SPLs share several features. In this case, the merge operator can be used to *merge* the overlapping parts of the FMs and then to obtain an integrated FM of the set of SPLs. In prior works [Acher et al., 2009, Acher et al., 2010a], we introduced a merge operator that produces merged FMs with well-defined properties. The merge uses name-based matching: two features match if and only if they have the same name.

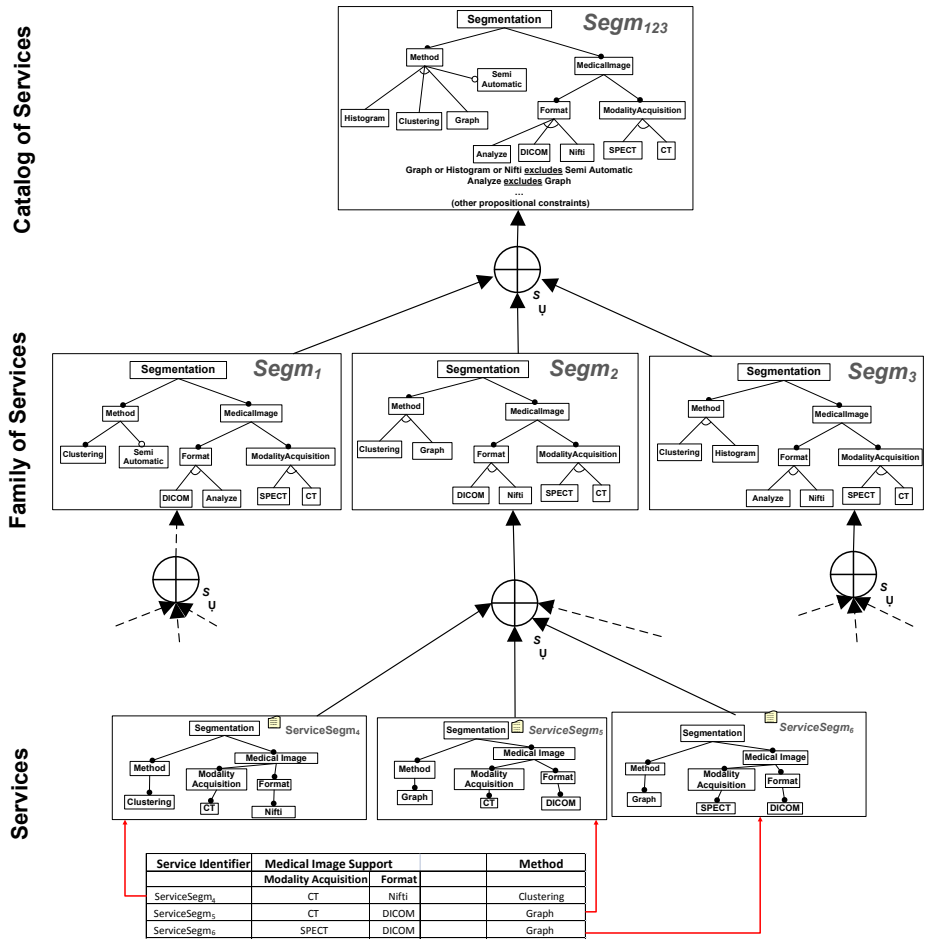


Fig. 3 Segmentation services are grouped together using merge in strict union mode.

Merge Operator Semantics. The properties of a merged FM produced by an application of the merge operator are formalized in terms of the sets of configurations of input FMs. Several modes are defined for the merge operator.

The *strict*[2] *union.* mode is the most inclusive option: the merged FM includes all the valid configuration defined by the input FMs and is defined as follows:

$$[\![FM_1]\!] \cup [\![FM_2]\!] = [\![Result]\!] \qquad (M_1)$$

The merge operator in the strict union mode is denoted $FM_1 \oplus_{\cup_s} FM_2 = Result$ and is typically used to synthetize an FM of a competing multiple SPL.

The *intersection* mode is the most restrictive option: the merged FM, FM_r, expresses the common valid configurations of FM_1 and FM_2. The merge operator in the

[2] In the literature [Segura et al., 2008, Acher et al., 2009], there exists another union mode, which is less restrictive in terms of sets of configurations expected from the resulting FM than the strict union mode defined in this paper.

intersection mode is denoted as follows: $FM_1 \oplus_{\cap} FM_2 = \ Result$. The relationship between a merged FM *Result* in intersection mode and two input FMs FM_1 and FM_2 can be expressed as follows:

$$[\![FM_1]\!] \cap [\![FM_2]\!] = \ [\![Result]\!] \qquad (M_2)$$

As we rely on set theory, the merge operators in strict union and intersection mode are associative and commutative. They can be applied to $n \geq 2$ input FMs.

Another merge operator, called *diff*, is denoted as $FM_1 \oplus_{\setminus} FM_2 = \ Result$. The following defines the semantics of this operator:

$$[\![FM_1]\!] \setminus [\![FM_2]\!] = \{x \in [\![FM_1]\!] \,|\, x \notin [\![FM_2]\!]\} = \ [\![Result]\!] \qquad (M_3)$$

Merge Implementation. In [Acher et al., 2010a], we compared different approaches to implement FM composition. We determined that an implementation based on propositional logic coupled with the algorithm proposed in [Czarnecki and Wąsowski, 2007] to construct a FM from propositional formula was efficient. In particular, other competing approaches, mostly based on *syntactical* strategies [Segura et al., 2008], [Schobbens et al., 2007], [Acher et al., 2009], have limitations to accurately represent the set of configurations expected, especially in the presence of cross-tree constraints. An approach based on propositional logic has the advantage of reasoning directly at the *semantic* level, i.e., in terms of sets of configuration.

The set of configurations represented by a FM can be compactly described by a propositional formula defined over a set of Boolean variables, where each variable corresponds to a feature [Batory, 2005, Czarnecki and Wąsowski, 2007]. The overall idea is to encode each input FMs involved in the merging operation as propositional formulas and, depending on the merging mode, applying some Boolean operations over these formulas. For instance, the strict union of two sets of configurations represented by two FMs, FM_1, and FM_2, can be computed as follows. First, FM_1 (resp. FM_2) FMs are encoded into a propositional formula ϕ_{FM_1} (resp. ϕ_{FM_2}). Then, the following formula is obtained:

$\phi_{Result} = \ (\phi_{FM_1} \wedge \ not(\mathcal{F}_{FM_2} \setminus \mathcal{F}_{FM_1})) \ \vee \ (\phi_{FM_2} \wedge \ not(\mathcal{F}_{FM_1} \setminus \mathcal{F}_{FM_2}))$

where $\mathcal{F}_{FM_1}$ (resp. $\mathcal{F}_{FM_2}$) is the set of features of FM_1 (resp. FM_2) and, $\mathcal{F}_{FM_2} \setminus \mathcal{F}_{FM_1}$ denotes the complement (or difference) of $\mathcal{F}_{FM_2}$ with respect to $\mathcal{F}_{FM_1}$, *not* is a function that, given a non-empty set of features, returns the Boolean conjunction of all negated variables corresponding to features:

$not(\{f_1, f_2, ..., f_n\}) = \ \bigwedge_{i=1..n} \neg f_i$

Computing the intersection of two sets of configurations represented by two FMs, FM_1, and FM_2, follows the same principles and we obtain:

$\phi_{Result} = \ (\phi_{FM_1} \wedge \ not(\mathcal{F}_{FM_2} \setminus \mathcal{F}_{FM_1})) \ \wedge \ (\phi_{FM_2} \wedge \ not(\mathcal{F}_{FM_1} \setminus \mathcal{F}_{FM_2}))$

Finally, the algorithm presented in [Czarnecki and Wąsowski, 2007] transforms ϕ_{Result} to an FM. It builds a tree with additional nodes for feature groups that can be translated into a basic FM. In particular, the algorithm can restore the hierarchy of input FMs by indicating parent-child relationships (mandatory or optional features) and Xor- or Or-groups.

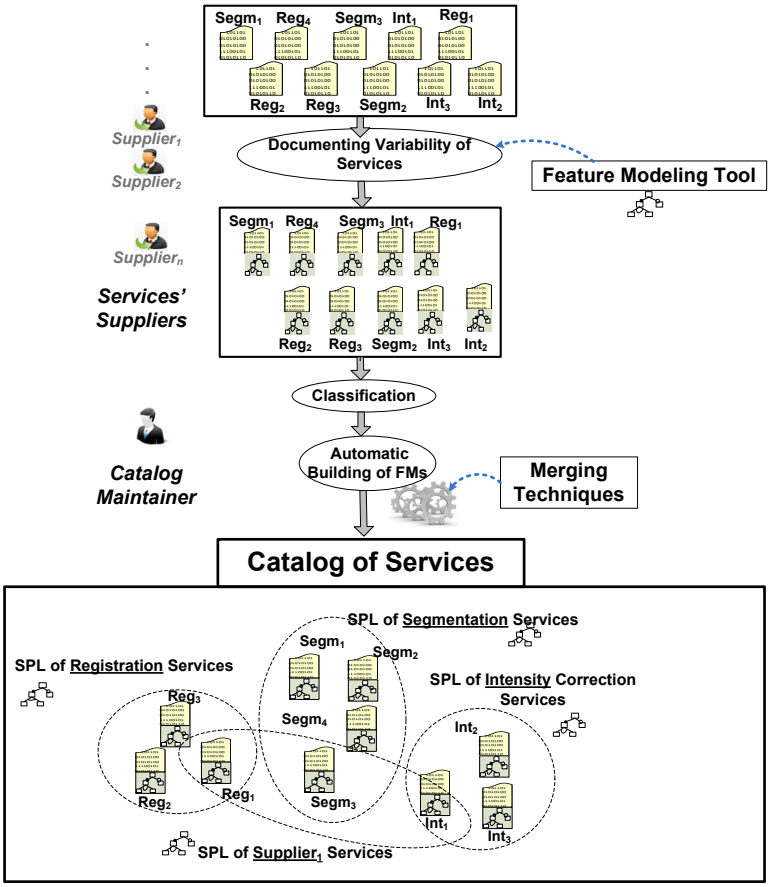


Fig. 4 From services to a catalog of services.

3.4 Building a Catalog of Services using Merging Techniques

We now provide some practical applications of the merging techniques. We consider the construction of a catalog of services from which several suppliers (e.g., research teams around the world) provide access to a set of (legacy) services implementing diverse medical imaging algorithms (see Fig. 4). The purpose is to provide a catalog of FMs describing the features of the services offered to workflow designers. This catalog is built in a bottom-up way, starting from the existing service documentations. In the rest of the paper, the term *catalog of services* is used to refer to a *catalog of FMs* describing the variability of a set of services.

From Services' Documentation to FM. In Fig. 2(a), each line of the array documents the variability of one service. Each service describes its variability with an FM. If there is no variability, the corresponding FM contains only *core*[3] features, i.e., all features are mandatory. We represent, in the bottom part of Fig. 3, the FMs corresponding to the

[3] A *core* feature is a feature that appears in every configuration of an FM.

variability of $ServiceSegm_4$, $ServiceSegm_5$ and $ServiceSegm_6$. The merge operator in strict union can be applied to organize a set of services, eventually with no variability, as a family of services. For example, it produces a new FM that represents a family of services $Segm_2$, including $ServiceSegm_4$, $ServiceSegm_5$ and $ServiceSegm_6$. The merging operations can be applied iteratively to array contents. (Similarly, that is how we obtain the FM of Fig. 2(b).)

From Services to Family of Services. Due to the large number of service features, there are various ways to classify services. Developers identify services that are to be managed through a unique SPL. Such an SPL should preserve the combinations of features provided by each service. This classification activity involves building an SPL which manages a set of services corresponding to the classifications that has been retained. How the classification is chosen is out of our scope here, but this is an important activity as there are different possible ways to organize functional and non-functional properties. Besides services can have interactions and constraints between them, and independently of the classification, we will show in Section 4 how our approach handle this issue.

Regarding the classification, the merge operator in strict union can be similarly applied to produce a compact representation of all valid combinations of features supported by a set of service families. For example, in Fig. 3, three families of segmentation services are grouped together into an unique SPL. Their FMs are merged in strict union mode: $Segm_1 \oplus_{\cup_s} Segm_2 \oplus_{\cup_s} Segm_3 = \ Segm_{123}$ As a result, all valid combinations of features of $Segm_{123}$ are supported by segmentation services that belong to $Segm_1$, $Segm_2$ and/or $Segm_3$.

When using the merge operator, it may be useful to keep the traceability/provenance of the features of the resulting merged FM ($Segm_{123}$) regarding the original input FMs ($Segm_1$, $Segm_2$ and $Segm_3$). This issue is addressed in Section 5.

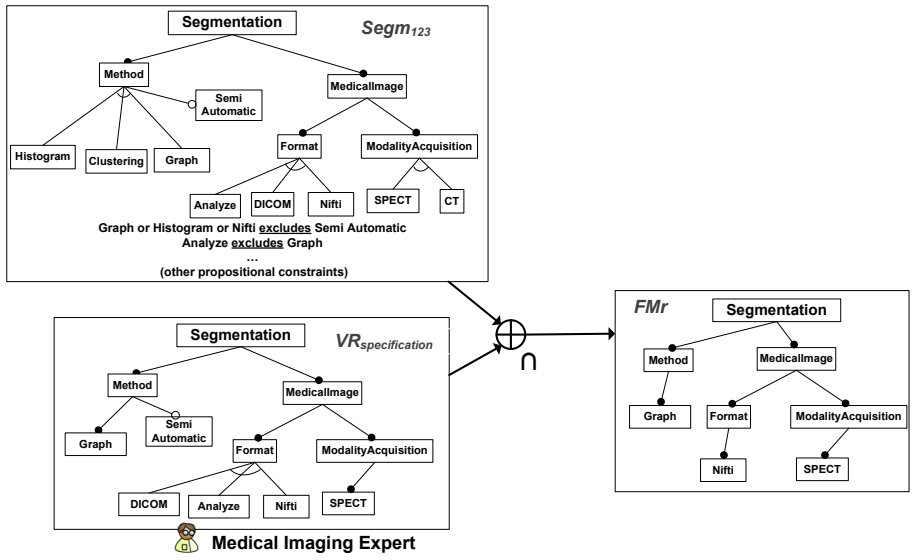


Fig. 5 Availability of services and selection of services using merge in intersection mode.

Mapping Variability Requirements to Family of Services. The merge in intersection mode can be applied, for example, when we want to check that the variability requirements specified by a user can be fulfilled by at least one service in the catalog. For example, in Fig. 5, the medical imaging expert specifies that the method of segmentation he/she wants to apply is Graph and the acquisition of the medical image to process is SPECT ; no choice is made for the Format of medical image used (DICOM, Analyze, Nifti are still valid alternative choices) ; eventually, a Semi Automatic method can be chosen. The merge in intersection mode is applied on $Segm_{123}$ (previously computed) and the FM $VR_{specification}$ representing the variability requirements of the medical imaging expert. In the example of Fig. 5, we are sure there exists at least one service in the catalog since $[\![Segm_{123}]\!] \cap [\![VR_{specification}]\!]$ is not empty. The resulting FM FM_r can then be used to select an effective service of the catalog. In the example, it corresponds to only one service, $Segm_2$, since $[\![Segm_1]\!] \cap [\![FM_r]\!]$ and $[\![Segm_3]\!] \cap [\![FM_r]\!]$ gives the empty set.

The variability requirements specified by the medical imaging expert may express some combination of features that cannot be entirely provided by the catalog (and vice-versa). Performing a merge diff operation assists users in understanding which set of configurations is missing.

4 From Design to Configuration of Workflow

We now describe how the merging techniques proposed in Section 3 can be used and complemented with others to facilitate and automate workflow design, using the domain of medical imaging as an illustration. Fig. 6 gives an overview of the proposed multi-step process described in this section. The overall goal of the process is to derive, from an high-level description augmented with variability requirements, a consistent workflow product composed of services offered by the catalog.

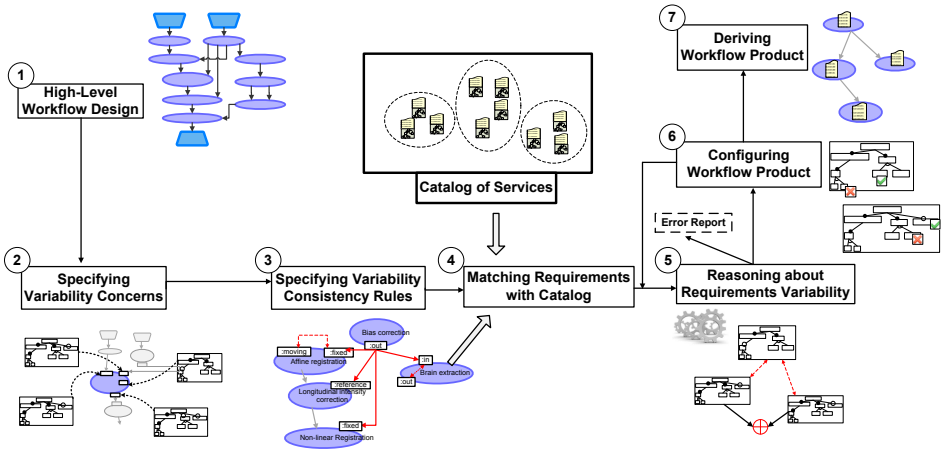


Fig. 6 Overview of the approach: From design to configuration of the workflow.

In step ① of the process, the workflow designer first develops a high-level description of the workflow that defines the computational steps (e.g., data analyses) that

should take place as well as the dependencies between them. We introduce the workflow description language (GWENDIA) we used in this study in Section 4.1.

The workflow description is then augmented with rich representation of requirements in order to support discovery, creation and execution of services used to realize the computational steps. In step ②, the workflow designer identifies the variable concerns (e.g., medical image format, algorithm method) for each process of the scientific workflow. An FM can be associated to a concern of a process, so that the variability of this concern is represented by it (Section 5 discusses how this is implemented). Hence several FMs are woven at different, well-located places in each process (e.g., dataport, interface) for specifying the variability of application-specific requirements. We present in Section 4.2 mechanisms to achieve separation of concerns and to reuse sub-parts of the catalog of FMs (rather than developing from scratch FMs).

In the general case, some features of a concern may interact with one or more features of other concern(s). In step ③, some application-specific constraints within or across services are typically specified by the workflow designer to not permit some combinations of features. Similarly, some compatibility constraints (e.g., between dataports) can be deduced from the workflow structure and be activated or not by the workflow designer. We described in Section 4.3 the kinds of constraints that can be specified or deduced from the workflow structure.

In order to ensure that the variability requirements do match the combination of features offered by the catalog, the workflow designer compares, in step ④, the FMs woven in the workflow with the FMs in the catalog of legacy services. In Section 4.4.2, we explained how the matching verification is performed for all services of the workflow and reduces the set of features to consider.

In step ⑤, we automatically reason about FMs and constraints specified by the workflow designer in step ① and ②. Constraints propagation and merging techniques are combined to reason about FMs and their compositions (see Section 4.4.3). This provides assistance to the workflow designer (detection of errors, automatic selection of features, etc.)

To complete the workflow configuration (step ⑥), the workflow designer has to resolve concern FMs where some variability still remains, and to perform select/deselect operations. The step ⑥ may be repeated as much as needed in order to allow the workflow designer to proceed incrementally (see Section 4.4.4). ⑤ should also be repeated in order to ensure workflow consistency is maintained. In step ⑦, the workflow designer uses the final workflow configuration to identify the services in the catalog that support the combination of features. If more than one service is identified for a given configuration, the workflow designer examines all candidate services to chose a best fit or an arbitrary legacy service.

4.1 Workflow Design

To support the workflow modeling activity, we use the GWENDIA language [Montagnat et al., 2009]. GWENDIA specifically focuses on coarse grain data-intensive scientific applications and enables the description of massively data-parallel applications. Some workflow engines (e.g., MOTEUR [Glatard et al., 2008]) use the GWENDIA language to describe and deploy applications on grid infrastructures. A GWENDIA workflow is notably composed of a series of processors connected to each other through their input and output ports. For the purpose of the paper, we consider that

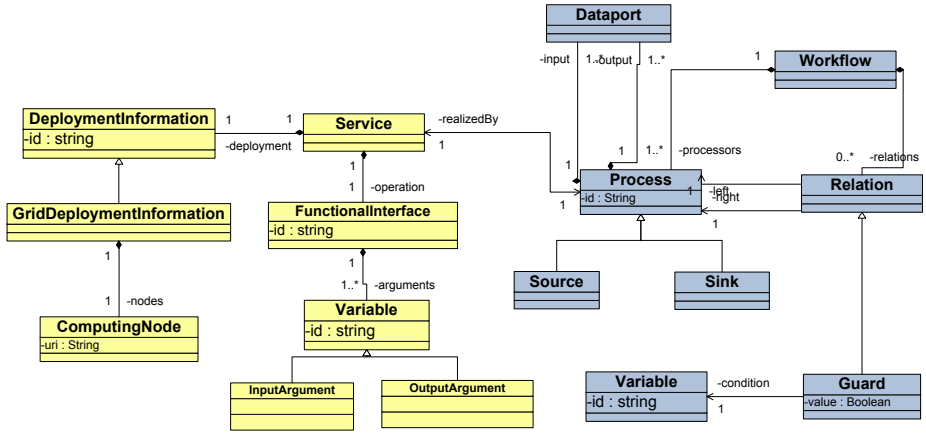


Fig. 7 Excerpt of workflow and service metamodel.

GWENDIA workflows can be represented using the metamodel described in Fig. 7, referred hereafter as the GWENDIA metamodel. There are two main parts: the general description of a workflow (elements in blue color) and the specification of a service (elements in yellow color).

A workflow is a set of process which are connected by directed links *relation* through input and or output *dataports*. These links may correspond to operators for *i)* executing services in sequence, *ii)* parallel computations and *iii)* branching through if-then-else constructs. Some processes do not have inputs (*source*) while others do not have outputs (*sink*). The workflow services' can be detailed from different levels of description that could then be (automatically) exploited in the workflow. For example, with descriptions of the data format, it is possible to incorporate automatic reasoning that could automatically check data interoperability between services connected in the workflow. In the GWENDIA metamodel of Fig. 7, we have identified some abstraction capabilities that can be used to augment services' description. This includes the functional part of the service, in particular its input and output parameters, as well as extra-functional information that can be related to the platform in which the service is deployed. In our context, some variability information can be attached to services' elements, for example, to describe the variety of medical image formats supported as an input parameter. With regard to the metamodel, an instance of a service element is a joinpoint in which an FM can be woven.

4.2 Separation of Concerns while Specifying Variability

There are at least two approaches that a workflow designer can use to define workflow service variabilities.

One approach is to create from scratch FMs for each service with variable concerns (as in [Acher et al., 2010b]). This solution has two major drawbacks. First, the mod-

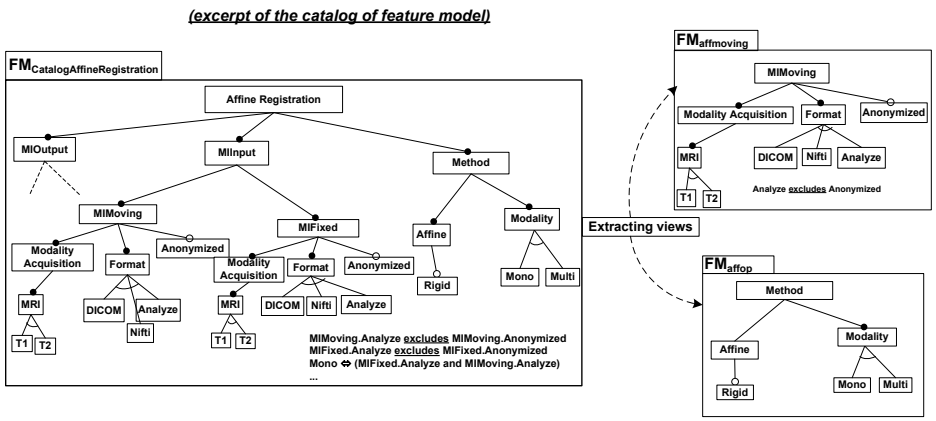


Fig. 8 Extracting views from catalog of affine registration.

eling effort tends to be important and time-consuming. Second, when the workflow designer wants to determine whether the specified combination of features is realized in the catalog, the FMs developed have to be compared with catalog feature models. There is a risk that vocabulary terms used for features' names, hierarchies to structure features as well as granularity detail largely differ, thus requiring an important *alignment* effort.

Another approach is to build FMs that are modified versions of FMs of the catalog, that is, closely matched FMs of the catalog are reused and modified so that they include only the features that are needed in the workflow. Hence the modeling effort as well as the alignment effort are reduced through reuse.

Unfortunately, an FM may represent the variability of *all* concerns within a service, including the features' constraints between concerns, whereas the workflow designer wants to focus on some specific views of the catalog of feature models. For example, $FM_{CatalogAffineRegistration}$ integrates the description of four concerns, the two input medical images, the output medical image and the algorithm method, while there is a relationship between the feature Mono and the features Analyze[4].

To resolve this issue, extractions, based on the *slicing* mechanism (see Definition 3 or [Acher et al., 2011b] for more details), can be performed and the original FM can be split into smaller FMs also called *variability concerns* in the remainder of the paper. Once extracted, the workflow designer can weave the smaller FMs into specific places of a service to document its variability requirements.

Definition 3 (Slicing) *We define slicing as an operation on FM, denoted* $\Pi_{\mathcal{F}_{slice}}(fm)$ *where* $\mathcal{F}_{slice} = \{ft_1, ft_2, ..., ft_n\} \subseteq \mathcal{F}$ *is a subset of the set of features of* fm*. The result of the slicing operation is a new FM,* fm_{slice}*, such that:*

$[\![fm_{slice}]\!] = \{\ x \in [\![fm]\!] \mid x\ \cap\ \mathcal{F}_{slice}\ \}$ *(called the projected set of configurations).*

[4] The two features Analyze have the same name but are different entities. To avoid ambiguity, we use a qualified feature name including the root feature when needs be (e.g., to distinguish the Analyze feature of MIMoving from the Analyze feature of MIFixed. In this specific case, the merge operator described in Section 3.3 makes internally the distinction such that MIMoving.Analyze does not match with MIMoving.Analyze.

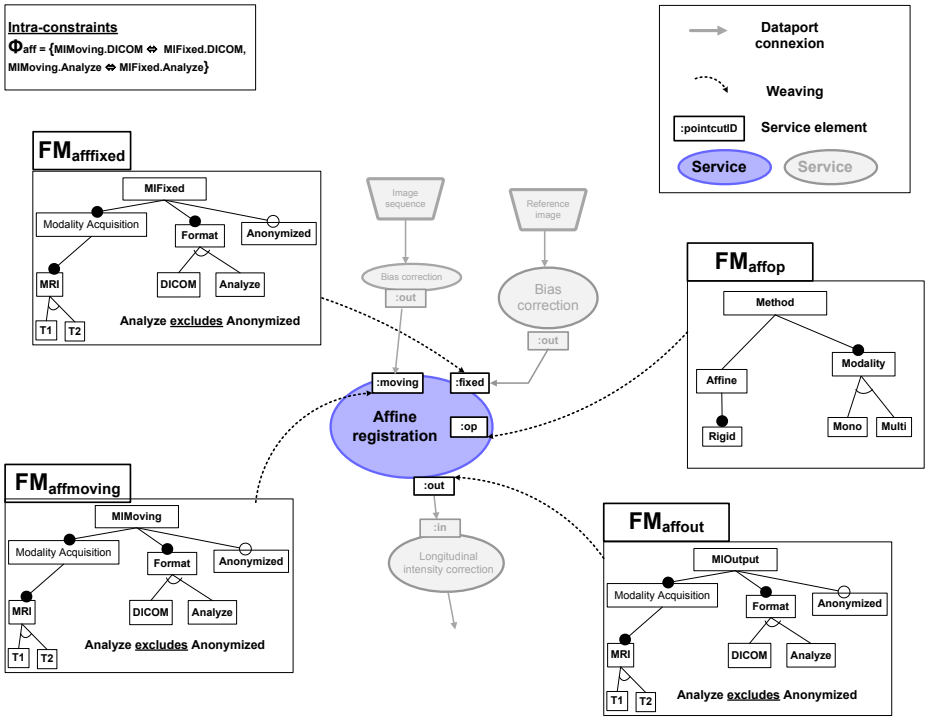


Fig. 9 Weaving variability concerns into services.

For example, in Fig. 8, two FMs $FM_{affmoving}$ and FM_{affop} are extracted from the FM $FM_{CatalogAffineRegistration}$. These two FMs are then specialized (feature Rigid becomes mandatory in FM_{affop} and feature Nifti is no longer present in $FM_{affmoving}$) and finally woven into two joinpoints of an *Affine registration* service (see Fig. 9). Four joinpoints are defined in *Affine registration*: *:moving* and *:fixed* are instances of type *InputArgument*, *:out* is an instance of type *OutputArgument* and *:op* is an instance of type *FunctionalInterface*. Four FMs, including the two FMs $FM_{affmoving}$ and FM_{affop}, are woven into the four joinpoints: three of them deal with medical image formats whereas the fourth FM deals with the kind of algorithm used for processing the images (see Fig. 9).

4.3 Variability Consistency Rules

The variability information attached to services authorizes numerous combinations of features (configurations) so that a final workflow product can be derived. Nevertheless, not all configurations are valid since variability concerns, documented as separated FMs, can be dependent on other variability concerns *within* services and *across* services. Constraints may be specified by the workflow designer to restrict the set of valid configurations.

We define a classification of the various types of constraints and then we present how the specification and the verification of these constraints are integrated within the process shown in Fig. 6.

4.3.1 Constraint Classification

We identify four kinds of constraints:

Intra-services constraints. Variability concerns within a service may interact.

Catalog constraints. An example was given in Section 3.4 where some Method child-features are related to some Format child-features (e.g., Analyze excludes Graph) in catalog FMs. The variability specification of a service in the workflow should be mapped to at least one service in the catalog. Hence, the variability concerns attached to a workflow service should be coherent with the constraints imposed by the catalog.

Application specific constraints. Intra-constraints may be specific to an application, for example, a user can require that the imaging formats supported as inputs should be the same for each input data port of the service. As a result, the FMs representing the different input images supported by a service are related to each other through constraints between features. For example, a user specifies in Fig. 9 that the feature DICOM (resp. Analyze) of $FM_{affixed}$ implies the feature DICOM (resp. Analyze) of FM_{moving}.

Inter-services constraints. Variability concerns are likely to interact across services. We identify two kinds of situations where the sets of valid combination of services' features may be further constrained:

Workflow Compatibility constraints. Due to the interconnection of services in the workflow, elements of services may be dependent. As a result, concerns attached to these elements may, in turn, be dependent on each other. This typically occurs when concerns are attached to input/output data port. For instance, the medical image output format of the service *Bias correction* is considered to be *compatible* with the medical image input format of the other connected services, i.e., *Affine registration*, *Longitudinal intensity correction*, *Brain extraction* and *Non-linear registration* in the workflow of Fig. 10. The compatibility relation restricts the set of valid combination of features in each of those services (see next Section).

Application specific constraints. Two (resp. more than two) FMs attached to two (resp. more than two) different services may be related to each other in some workflow applications. The user should have the ability to specify some specific constraints when he/she considers that services are tightly coupled. For example, it is required in the medical image domain that *registration* and *unbias* services, that are directly connected in the workflow, are using the same algorithm method.

4.3.2 Integration of Constraints within the Process

Some constraints are manually specified by the user (e.g., cross-FM constraints specific to an application) whereas some others can be detected from the workflow analysis (see Table 1). In particular, compatibility constraints between data ports can be deduced and then constraints are applied on FMs attached to data ports. Nevertheless, if the workflow designer is developing the workflow in an incremental manner, he/she may want to deactivate part of the compatibility checks according to the service and/or the concerns he/she focused on.

	Constraint Classification	Constraint Specification		Contraint Checking
		By who and how?	When	
Intra-Service Constraints	Catalog constraints	The specification is implicit. But the association catalog/concern is performed explicitely by the workflow designer	Association is performed at step 3 of the process	It is performed at the step 4 of the process *i.e.* when the mapping catalog/specification is done (*See Section 4.4.2*)
	Application-specific constraints	The workflow designer specifies it explicitely	At step 3 of the process	It is performed at the step 5 of the process *i.e.* when the consistency checking and variability propagation is done (*See Section 4.4.3*)
Inter-Service Constraints	Application-specific constraints			
	Workflow compatibility constraints	The specification is built-in. But the desactivation may be performed explicitely by the workflow designer	Possible deactivation is performed at step 3 of the process	

Table 1 Specification and checking of constraints within the process.

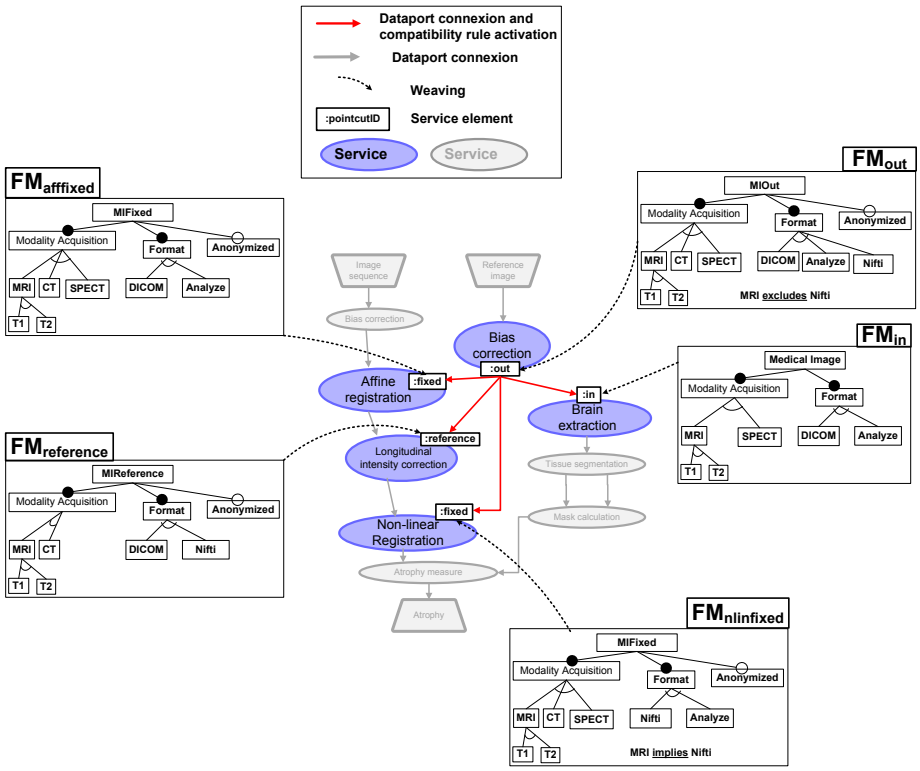


Fig. 10 Data compatibility between services.

Dataport compatibility. The compatibility relation can be defined, at the FM level, as follows: For at least one valid configuration of the FM associated to the medical image output there is an equal configuration valid in the FM(s) associated to the medical image input(s) (and vice-versa).

As shown in [Acher et al., 2010b], when n services are concurrently executed, it is not sufficient to reason about *pairs* of services independently when checking dataport compatibility:

$$(FM_{out} \oplus_{\cap} FM_{affixe} \neq nil) \wedge (FM_{out} \oplus_{\cap} FM_{reference} \neq nil) \wedge$$
$$(FM_{out} \oplus_{\cap} FM_{in} \neq nil) \wedge (FM_{out} \oplus_{\cap} FM_{nlinfixed} \neq nil)$$

since the merging (e.g., $(FM_{out} \oplus_{\cap} FM_{affixe})$) has side effects on input feature models (e.g., FM_{out} and FM_{affixe}).

We thus need to reason about *all* services at the same time:

$$(FM_{out} \oplus_{\cap} FM_{affixe} \oplus_{\cap} FM_{reference} \oplus_{\cap} FM_{in} \oplus_{\cap} FM_{nlinfixed}) \neq nil \quad (Cmp_1)$$

It can be generalized as follows: When the output dataport of a service $FService_1$ is connected to input data ports of a set of services $FService_2, \ldots, FService_n$, services are consistent according to feature models attached to dataports if the following relation holds:

$FM_{o1} \oplus_{\cap} FM_{i2} \oplus_{\cap} FM_{i3} \ldots \oplus_{\cap} FM_{in} \neq nil$

when FM_{o1} is the feature model attached to the output dataport of $FService1$ and $FM_{i2} \ldots FM_{in}$ are feature models attached to the input dataports of resp. $FService_2$, $\ldots$, $FService_n$.

For other workflow constructs (e.g., if-then-else), properties in terms of sets of configuration have also been defined (see [Acher et al., 2010b] for more details).

4.4 Reasoning about Catalog and Requirements Variability

We have described and illustrated how a user can specify variability at different places as well as the kinds of constraints that may occur in a scientific workflow. We now show how to perform automated reasoning about the FMs and constraints.

4.4.1 Formalization

We first formalize the relationship between FMs, services and workflows as well as the notion of *validity* at the service and workflow level. The formalization is used afterwards to describe the algorithms that realize reasoning operations.

Definition 4 (Service and Feature Models) *A service $FService_i$ is described as*

- *a set of feature models, $VC_i = \{FM_{i,0}, FM_{i,1}, \ldots, FM_{i,n}\}$.*
- *a set of intra-constraints, Φ_i where each $\gamma \in \Phi_i$ is an arbitrary propositional constraint over the set of features of any FM belonging to VC_i.*

Definition 5 (Service and Validity) *Let Γ_{agg_i} be the aggregated FM of $FService_i$ obtained by placing the FMs of VC_i under an And-decomposed synthetic root r and adding the conjunction of each constraint that belongs to Φ_i.*

A configuration c of a service $FService_i$ is a set of features selected where each feature of c is either a feature of $FM_{i,0}, FM_{i,1}, \ldots,$ or $FM_{i,n}$. The configuration c is valid iff $c \in (\llbracket \Gamma_{agg_i} \rrbracket \setminus r)$

For example, the service *Affine Registration* of Fig. 9 is composed of four FMs, $FM_{affixed}$, $FM_{affmoving}$, FM_{affout} and FM_{affop}, and two intra-constraints. An example of a valid configuration of this service is given below:

{ MIFixed, MIFixed.Modality Acquisition, MIFixed.MRI, MIFixed.T1, MIFixed.Format, MIFixed.Analyze MIMoving, MIMoving.Modality Acquisition, MIMoving.MRI, MIMoving.T1, MIMoving.Format, MIMoving.Analyze MIOutput, MIOutput.Modality Acquisition, MIOutput.MRI, MIOutput.T1, MIOutput.Format, MIOutput.DICOM, MIOutput.Anonymized Method, Affine, Rigid, Modality, Mono }

Definition 6 (Workflow and Feature Models) *A workflow is described as*

- *a set of services, $\epsilon_{services} = \{FService_0, FService_1, \ldots, FService_n\}$;*
- *a set of connections between those services, $\mathcal{C} \subseteq \epsilon_{services} \times \epsilon_{services}$;*
- *a set of inter-constraints, ζ where each $\eta \in \zeta$ is an arbitrary propositional constraint over the set of features of any FM of $\epsilon_{services}$, i.e., $FM_{0,0}, FM_{0,1}, \ldots, FM_{0,m_0}, \ldots, FM_{i,0}, FM_{i,1}, \ldots,$ or $FM_{i,m_i}, \ldots, FM_{n,m_n}$.*
- *a set of compatibility constraints μ over the set of FM configurations of the workflow. The set can be deduced from the connections between workflow services or be deactivated/specified by a workflow designer.*

Definition 7 (Workflow and Validity) *A configuration cw of a workflow is a set of features selected where each feature of cw is either a feature of $FM_{0,0}, FM_{0,1}, \ldots,$ or FM_{n,m_n}.*

Let Δ_{agg} be the aggregated FM of a workflow obtained by placing the aggregated FMs of each service under an And-decomposed synthetic root r and adding the set of constraints ζ and μ.

The configuration c is valid iff $c \in (\llbracket \Delta_{agg} \rrbracket \setminus r)$

The approach we developed provides automated support for *i)* ensuring for each FMs associated with a service of the workflow, that only valid and consistent select/deselect decisions are made, *ii)* propagating the decisions so that the workflow designer is only required to answer questions needing human intervention (the answers to the other questions are inferred automatically).

We illustrate how we can automate consistency checking and reduction of variability using *Affine registration* as an example. According to the semantics defined above, the following three conditions should not be violated in the *Affine registration* service:

- **(a)** at least one configuration of the workflow service should correspond to another configuration of an existing service in the catalog. Formally:
  Let $\Gamma_{agg_{aff}}$ be the aggregated FM of service *Affine registration* and $\Gamma_{catalog_{aff}}$ be the FM of the corresponding family of service in the catalog. Then, the following relation holds: $\llbracket \Gamma_{agg_{aff}} \rrbracket \cap \llbracket \Gamma_{catalog_{aff}} \rrbracket \neq \emptyset$;
- **(b)** the compatibility constraints between *Affine registration* and other connected services are enforced. Formally: the relation (Cmp_1) holds ;

– **(c)** FM_{affout}, FM_{op}, $FM_{affixed}$ and $FM_{affmoving}$ are to be consistent. Formally:
Let $\Gamma_{agg_{aff}}$ be the aggregated FM of service *Affine registration*.
$\exists c_{out} \in [\![FM_{affout}]\!], c_{op} \in [\![FM_{op}]\!], c_{fixed} \in [\![FM_{affixed}]\!]$,
$c_{moving} \in [\![FM_{affmoving}]\!]$ *s.t.* $(c_{out} \cup c_{op} \cup c_{fixed} \cup c_{moving}) \in [\![\Gamma_{agg_{aff}}]\!]$;

4.4.2 Catalog Mapping

The reasoning process starts by ensuring that the catalog can provide, for all services in the workflow, at least one corresponding FM that matches its variability requirements. We consider that each workflow service may be mapped to a catalog FM. The mapping between a service of the workflow and the catalog is specified by the workflow designer using a domain-specific language (see Section 5). The availability is checked for all services that are mapped to a catalog. The reasoning process has also the capability to identify variability choices that are no longer available in the catalog FM. In Fig. 11, we illustrate how the mapping between *Affine registration* and the catalog of Fig. 8 is realized. Some variability choices have been inferred, for example, feature Multi is no longer present and thus the feature Mono is now a core feature. The intra-constraints have been reinforced. For example, configurations of the catalog that include the feature Nifti are not considered because the variability requirements of the service *Affine registration* do not include the feature Nifti. Such reasoning can be automated using the merging techniques described in Section 3.3. The key idea is to assemble all FMs of the service into an aggregated FM and then queries the catalog FM. We use the Algorithm 1 that describes how the catalog is queried. First, the FMs of each service are aggregated together with their intra-constraints. The merge in intersection mode is then performed[5] and FMs of the workflow services as well as the intra-constraints are updated after *decomposing* the aggregated FM using the slicing mechanism (see Definition 3). This is the same decomposition mechanism as the one described in Section 4.2 which consists in extracting a set of FMs from the aggregated FM. For instance, $FM_{affixed}$ at the bottom of Fig. 11 is extracted from the merged FM and includes the constraints that involve its features Anonymized and Analyze.

In this step, every workflow service is consistently mapped to a catalog FM. There is no longer need to query again the catalog. The restrictions on the sets of configurations, compactly represented by the merged FM, guarantee the existence of at least one corresponding service in the catalog.

4.4.3 Consistency Checking and Variability Propagation

Dataport compatibility. The reasoning process continues by ensuring that the compatibility constraints between dataports are enforced (see ① of Fig. 12). At the FM level, the merge intersection is performed between FM_{out}, $FM_{affixed}$, $FM_{reference}$, $FM_{linfixed}$ and FM_{in}. The root features of the different input FMs fed to the merge

[5] Some alignment issues may occur when merging two FMs. For example, a naive aggregation of FMs can lead to an aggregated FM without the structuring feature MIInput, and thus disturbs the merging process. We provide to the user the ability to specify some pre-directives before merging FMs. The FM alignment problem is more general and further discussed in Section 6.

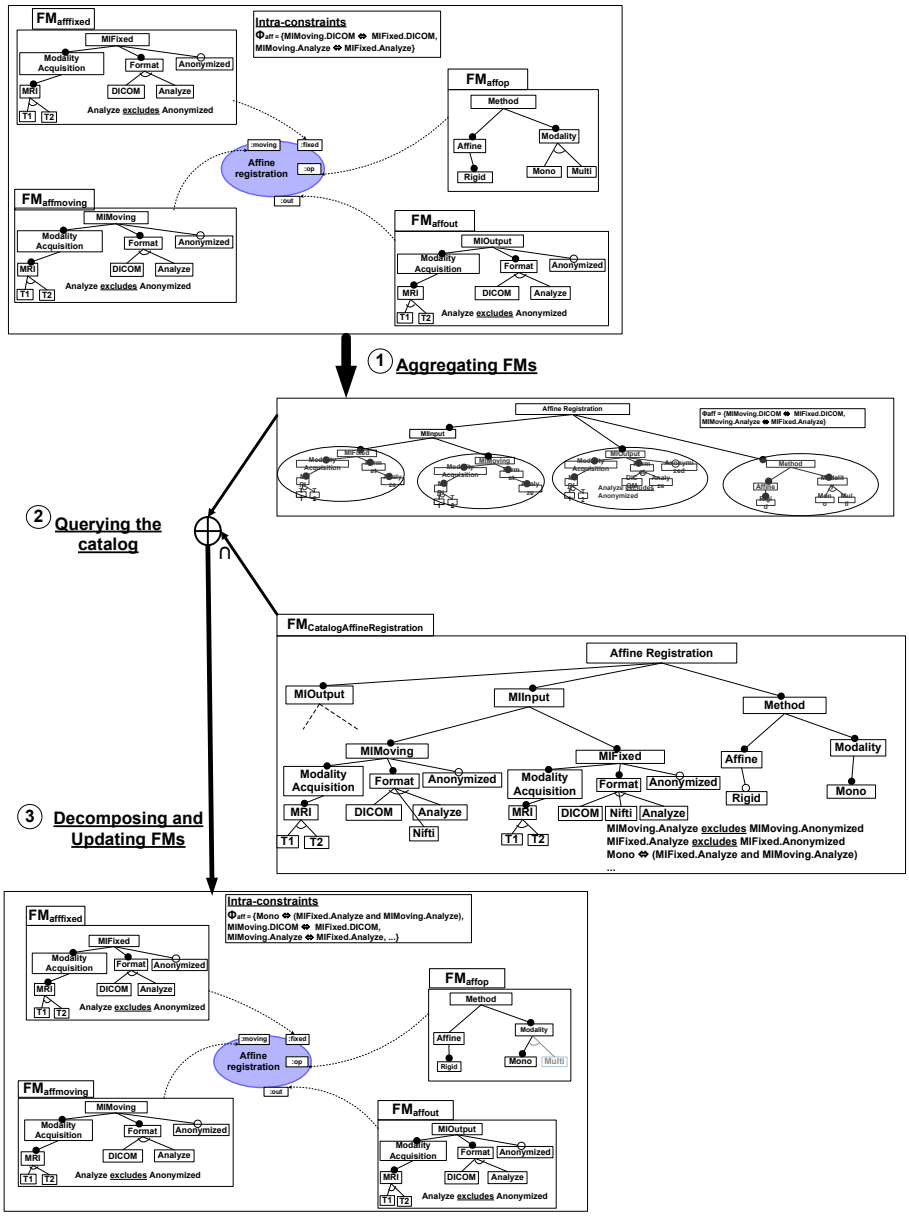


Fig. 11 *Affine registration* is mapped to the catalog of Fig. 8 and its four FMs are updated.

operator may have different names[6]. Such features disturb the merging process so that, theoretically, $[\![FM_{merged}]\!]$ is empty. For practical reasons, we automatically re-

[6] It may be for practical reasons (e.g., convention) or for better characterizing the high-level concept for which the FM applies. In the example, rather than always using Medical Image, users prefer to be more precise for describing the kind of medical image associated to an FM. This issue is an instance of the FM alignment problem which is further discussed in Section 6

Algorithm 1 Querying the catalog

Require: the set of services, $FServices$, of the workflow that are mapped to a catalog
Ensure: services requirements match at least one service in the catalog and are updated.

```
for all FService_i ∈ FServices do
  Γ_i ← build the aggregated FM of FService_i
  FM_mapped ← the associated FM in the catalog
  if [[Γ_i]] ∩ [[FM_mapped]] = ∅ then
    print "Unable to find a corresponding service in the catalog"
  else
    FM_merged ← Γ_i ⊕∩ FM_mapped
    fms_decomposed ← decompose FM_merged
    for all FM_vc ∈ fms_decomposed do
      update FService_i with FM_vc
    end for
    Φ_new ← extract intra-constraints from fms_decomposed
    Φ_i ← Φ_new {update intra-constraints}
  end if
end for
```

name each root feature of the FMs involved in the merging with the same temporary name (e.g., Medical Image), if needs be. When the relation (Cmp_1) defined in Section 4.4.1) does not hold, the sources of the error (i.e., the identification of the services that are not compatible to each other) are reported to the user. Otherwise, a valid FM is computed: FM_{merged}.

The resulting FM produced by the merge operator, FM_{merged}, is then used to update (i.e., replace) *all* the FMs involved in the compatibility relation. For example, a new FM, called $FM_{affixed'}$, is now associated to the pointcut *:fixed* of *Affine registration* (see ② of Fig. 12) and is equal to FM_{merged}. Hence the features DICOM and Anonymized of $FM_{affixed}$ are no longer present. Algorithm 2 recaps the situation.

Propagating constraints within a service. The intra-constraints of the service *Affine registration* may further reduce the set of valid combinations of features in other FMs of the service. When the FM involved in the dataport compatibility has been modified, as it is the case for $FM_{affixed}$, intra-constraints have to be considered for checking validity or propagating choices within a service. It should be noted that Algorithm 2 does propagate constraints only for services whose FMs have been modified during the compatibility checking. It may happen that the compatibility relation involving $FM_{affixed}$ truly holds but that the service is not valid due to intra-constraints. In addition, intra-constraints can be used to propagate variability choices.

The approach consists in aggregating the four feature models $FM_{affixed'}$, $FM_{affmoving}$, FM_{affout}, FM_{op}, FM_{aff} together with the constraints Φ_{aff} of the service *Affine registration*. The resulting feature model is denoted FM_{all}. FM_{all} is then being analyzed for various purposes:

- consistency of the service *Affine registration* can be decided by checking the satisfiability of FM_{all} ;
- we can detect new dead and core features and report back to the workflow designer ;
- the corrective capabilities of the slicing technique can be applied to simplify and update the different feature models of the service *Affine registration* (removal of

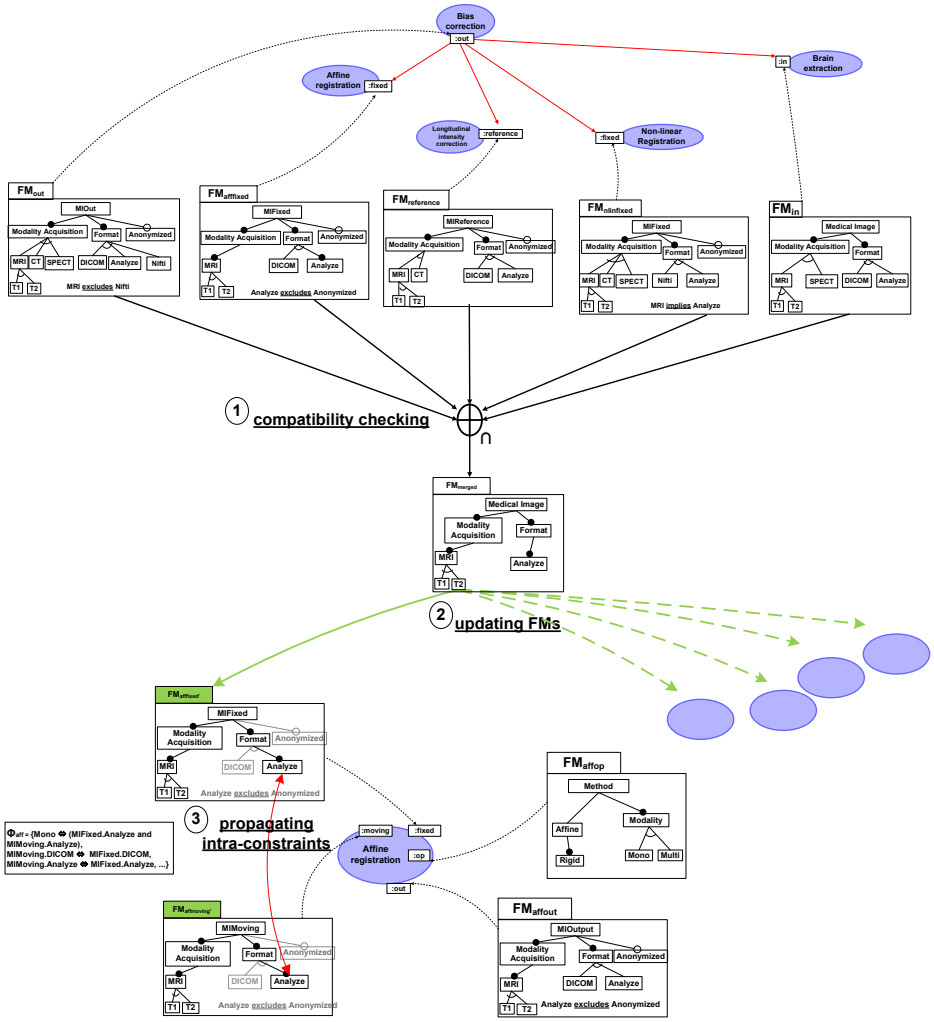


Fig. 12 Reasoning process: for each connected dataports in the workflow, we propagate variability choices within each service involved in the compatibility checking.

features when features are known to be dead, setting the mandatory status to some core features, etc.) ;

Reiterating the reasoning process. It may happen that the inference of variability choices through intra-constraints propagation leads to the modification of an FM involved in a dataport compatibility. (It is not the case for *Affine registration.*)

Let us consider the example given in Fig. 13 where services *FService1* and *FService2* are sequentially connected. Compatibility checking between their dataports *:out* and *:in* is first performed (see ①) such that features E, D are no longer present in FM_{output} of *FService1* while feature F is no longer present in FM_{input} of *FService2*. Then, constraint propagation is performed in *FService1* and *FService2* (see ②). No variability

Algorithm 2 Updating FMs after compatibility checking

Require: a set of dataport connections, $connection_{dps}$, where a dataport connection is represented as a set of dataports. $connection_{dps}$ is typically obtained through workflow analysis.

for all $connection \in connection_{dps}$ **do**
  $fmsToMerge \leftarrow \{\}$
  for all $dp \in connection$ **do**
    if dp has no FM attached **then**
      print "Warning: unable to find an FM in dataport"
    else
      $fmDP \leftarrow$ retrieve the FM attached to dp
      $fmsToMerge \leftarrow fmsToMerge \cup fmDP$
    end if
  end for
  $mergedFM \leftarrow fms_1 \oplus_{\cap} fms_2 \oplus_{\cap} \ldots fms_n$ where $fms_1, fms_2, \ldots, fms_n \in fmsToMerge$

  if $mergedFM$ not valid **then**
    print "Error: dataports of $connection$ are not compatible" {diff operations can be performed to provide fine-grained explanations}
  else
    $services_{modified} \leftarrow \{\}$ {services that have been impacted}
    for all $dp \in connection$ **do**
      $fm_{dp} \leftarrow$ retrieve the FM attached to dp
      if $mergedFM$ is a specialization of fm_{dp} **then**
        $service_{dp} \leftarrow$ retrieve the service of dp
        $services_{modified} \leftarrow services_{modified} \cup service_{dp}$ {propagation is needed}
      end if
      $fm_{dp} \leftarrow mergedFM$
    end for
  end if
end for
propagating choices on $services_{modified}$

Algorithm 3 Consistency checking and constraint propagation

Require: a set of services, $FServices$
Ensure: requirements variability are consistent and updated within each service
for all $FService_i \in FServices$ **do**
  $\Gamma_i \leftarrow$ build the aggregated FM of $FService_i$
  if $[\![\Gamma_i]\!] = \emptyset$ **then**
    print "The service $FService_i$ is not consistent"
  else
    propagate choices in Γ_i
    $fms_{decomposed} \leftarrow$ decompose Γ_i
    for all $FM_{vc} \in fms_{decomposed}$ **do**
      $FM_{corr} \leftarrow$ retrieve the original FM that corresponds to FM_{vc} in $FService_i$
      update $FService_i$ with FM_{vc}
      if FM_{vc} is attached to a dataport and FM_{vc} is a specialization of FM_{corr} **then**
        mark FM_{vc} {compatibility checking should be reiterated}
      end if
    end for
  end if
end for

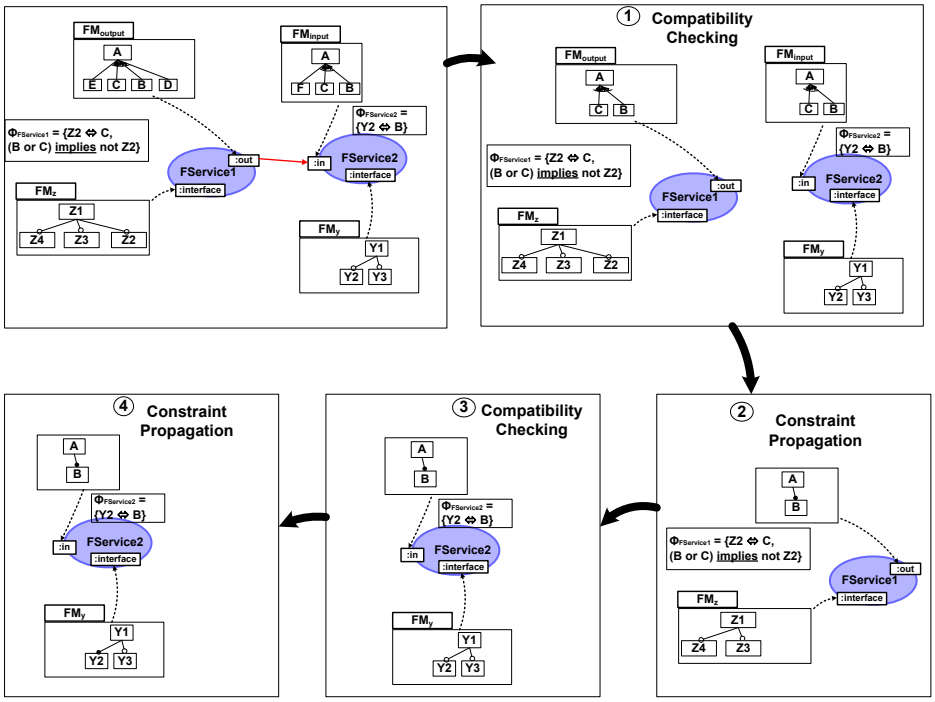


Fig. 13 An example: reiterating compatibility checking and constraints propagation.

choices can be inferred in *FService2*. In *FService1*, feature Z2 has been removed due to the intra-constraint $B \vee C \Rightarrow \neg Z2$. In turn, feature C has been removed due to the intra-constraint $Z2 \iff C$. Hence, the FM FM_{output} involved in a compatibility relation between dataports has been modified and compatibility checking should be reiterated (see ③). It modifies FM_{input} and, after constraint propagation, FM_Y in *FService2* (see ④). The reasoning process stops since no variability choices can be inferred in *FService1* and *FService2* and the compatibility checking between FM_{output} and FM_{input} has no effect.

Algorithm 3 and Algorithm 4 describe the reasoning process. Compatibility checking is performed if and only if an FM attached to a dataport connected to other dataports has been modified during constraint propagation (and thus *marked* during Algorithm 3 execution) . If the set of configuration of an FM remains the same, the algorithm terminates. An important property of the merging operator in intersection mode is that each input FM is either a *refactoring* or a *generalization* of the merged FM. (We rely on the terminology used in [Thüm et al., 2009]. Let f and g be FMs. f is a *specialization* of g if $[\![f]\!] \subset [\![g]\!]$; f is a *generalization* of g if $[\![g]\!] \subset [\![f]\!]$; f is a *refactoring* of g if $[\![g]\!] = [\![f]\!]$.). As a result, each time the compatibility checking is performed, FMs involved are either specialized or not impacted. Hence, the set of configurations is either the same or is *decreased* until being a singleton. This property guarantees that the algorithm necessary terminates when no variability choices can be deduced.

Algorithm 4 Reiterating the reasoning process

Require: the set of services, $FServices$, of the workflow ; the set of dataport connections, $connection_{dps}$, of the workflow
  for all $FService_i \in FServices$ **do**
    for all $vc \in VC_i$ **do**
      if vc is marked **then**
        print "Info: compatibility checking should be reiterated"
        $dp_{vc} \leftarrow$ retrieve dataport of vc
        $connection_{dp_{vc}} \leftarrow \{conn \in connection_{dps} \wedge dp_{vc} \in conn\}$
        unmark vc
        perform compatibility checking on $connection_{dp_{vc}}$
      end if
    end for
  end for

4.4.4 Multi-step Configuration of the Workflow

All FMs of the workflow can be partially configured or *specialized* [Czarnecki et al., 2005, Thüm et al., 2009]. The specialization activity includes the selection/removal of some features, the adding of some constraints within an FM, etc. Reasoning operations, as described above, can be similarly performed at each step to ensure consistency of the whole workflow and propagate variability choices (see ⑥ of Fig. 6).

Once the configurations of all FMs are complete, we know by construction that it corresponds to services in the catalog. It may happen that given a configuration of a service (see Definition 5), there is more than one service that corresponds in the catalog (since there exists services that support the same combinations of features). In this case, one possible and our current solution is that the user has to arbitrary choose which services he/she wants to include in the final workflow product (see further discussions in Section 6.2.3).

5 Realization and Tool Support

The approach proposed is comprehensively supported by a combination of dedicated tools and domain-specific languages (DSLs). The goal is to assist users at each step of the process – from workflow design to configuration of each of its constituent parts – described in Fig. 6.

5.1 Workflow Modeling

The first activity is to design a workflow (see ① of Fig. 6). We rely on the GWENDIA language (see Section 4.1), which proposes two concrete syntax, a graphical representation and a textual representation, and supports all the workflow constructions mentioned in Fig. 7. Using GWENDIA, scientists can specify a workflow including all data connections. In the left upper part of Fig. 14, the XML representation of an excerpt of a GWENDIA workflow is depicted.

Once designed, the workflow description can be augmented by specifying the variability requirements associated to services. To do so, different techniques and tools that are centered on features could be reused. Several systems have been developed, and concern all kinds of artifacts, from code or models to documentation [Apel and Kästner, 2009].

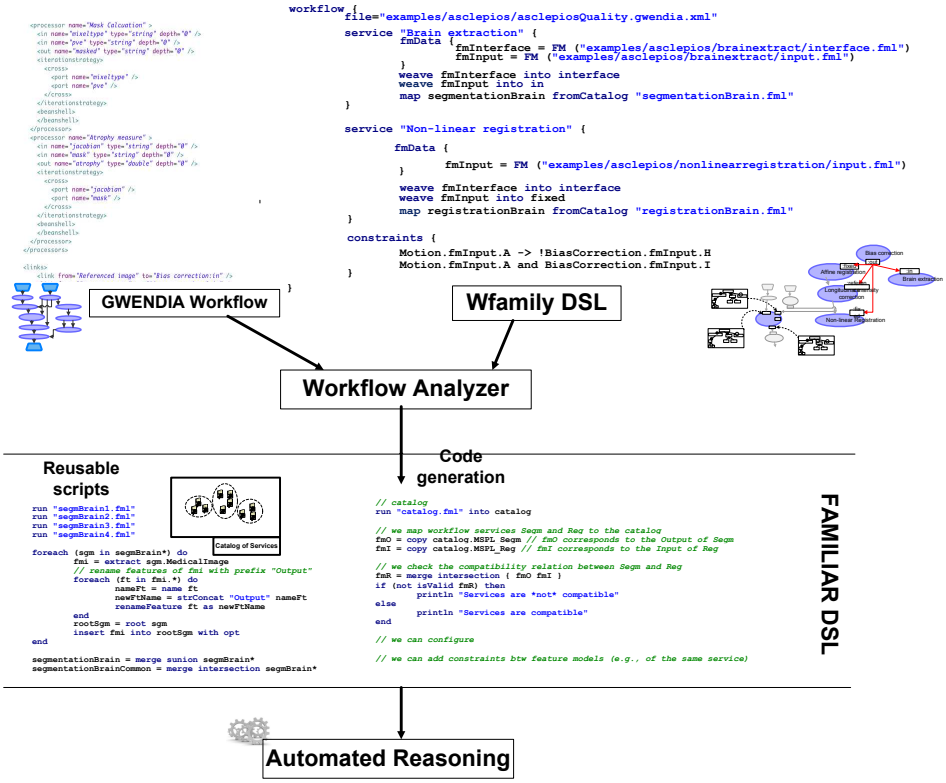


Fig. 14 Tool support and Domain-Specific Languages.

They notably include different software engineering techniques, i.e. aspect orientation [Voelter and Groher, 2007], components [van der Storm, 2004, Beuche et al., 2004], feature-oriented software development [Apel and Kästner, 2009] [Czarnecki and Pietroszek, 2006, Czarnecki and Antkiewicz, 2005]. Such techniques could be applied in the context of our contribution. Nevertheless this paper focuses on the coherent assembly of service-oriented workflow, rather than the development of the service themselves. Consequently, to specify the variability requirements, we choose to develop a simple and dedicated formalism to relate FMs to services and workflows. We propose an DSL, called *Wfamily* (see right upper part of Fig. 14). It enables one to:

- *import* FMs from external files while performing some high-level operations (extraction, renaming/removal of features, etc.). For example, the user can load an existing FM from a catalog, then extracts the sub-parts that are of interest and finally specialize the different FMs ;
- *weave* FMs to specific places of the workflow. We reify the concept of *pointcut*, which have an unique identifier within a service. Hence users can specify for which specific pointcut of a service an FM is attached to.
- *constrain* FMs within and across services by specifying propositional constraints. Each FM that have been woven has an unique identifier and can be related each other through cross-tree constraints.

5.2 FAMILIAR

Managing SPLs on a large scale requires users to perform complex tasks: FMs to be composed are usually located in different files ; it is necessary to reason about FMs (e.g., to check validity, to compute dead features) after performing several composition operations. More generally, it is important to provide SPL developers with facilities that allow them to capture and reuse a sequence of SPL management operations. As shown in this paper, similar requirements are encountered when managing variability in scientific workflows.

We thus designed and implemented FAMILIAR (FeAture Model scrIpt Language for manIpulation and Automatic Reasoning) [Acher et al., 2011a], a DSL dedicated to the management of FMs. It is an executable language that supports manipulating and reasoning about multiple FMs. It provides high-level operators for *i)* splitting and composing FMs (several merge, aggregate or extract operators), *ii)* aligning FMs (insertion, renaming and removal of features) and *iii)* reasoning about FMs and their configurations (validity of an FM, comparison of FMs, number of valid configurations in an FM, etc.). FAMILIAR also has constructs for describing iterations and conditions and to write and run customizable scripts.

FAMILIAR supports multiple notations for specifying FMs (GUIDSL/FeatureIDE [Batory, 2005, Kästner et al., 2009], SPLOT [Mendonca et al., 2009], a subset of TVL [Boucher et al., 2010], etc.). Off-the-shelf SAT solvers (i.e., SAT4J) and BDD library (i.e., JavaBDD) are internally used to perform some FAMILIAR operations (e.g., merging of FMs, configuration operations). The tool also allows users to import FMs or configurations from their own environments. Outputs generated by FAMILIAR can be processed by other tools, for example, in order to relate FMs to other artefacts (e.g., code, models) [Czarnecki and Antkiewicz, 2005, Heidenreich et al., 2010].

In terms of tool support, we provide an Eclipse text editor and an interpreter that executes the various scripts. The interpreter can also be used in an interactive mode to ease prototyping. Moreover the language was also integrated with the FeatureIDE [Kästner et al., 2009] graphical editor so that all graphical edits are synchronized with variables environment and all interactive commands are synchronized with the graphical editors.

5.3 Reasoning about Variability with FAMILIAR

Considering the approach defended in the paper, FAMILIAR is used:

- to specify variability requirements within services (cf. ② and ③ of Fig. 6): FAMILIAR is embedded into the DSL *Wfamily* described above. As a result, extracting a sub-FM from an FM in a catalog essentially consists in reusing FAMILIAR operators ;
- to build catalogs of services, organized as reusable scripts: FMs that document variability of services are merged together ; similarly, querying a catalog of services is realized using FAMILIAR scripts (cf. ④ of Fig. 6) ;
- as a target language. FAMILIAR code is *generated* from the workflow analysis, for example, to reason about dataport compatibility (cf. ⑤ of Fig. 6).

The FAMILIAR code is then interpreted to check the consistency of the whole workflow, report errors to users as well as automatically propagating choices (cf. ②

and ③ of Fig. 6). Users can incrementally configure, using graphical facilities provided by FeatureIDE editors, the various FMs of the workflow (cf. ⑥ of Fig. 6). Finally, in order to derive a final workflow product, competing services can be chosen from among sets of services in the catalog using FAMILIAR reusable scripts (cf. ⑦ of Fig. 6).

The three different uses of FAMILIAR mentioned above rely on the facilities provided to the stakeholders for separating and aggregating/merging the various variability concerns (i.e., to decompose and to recompose several FMs) they may want to address. Then the traceability among the original FMs and the resulting composed or decomposed FM may become an issue. In different situations, it may be useful to trace the provenance of a given feature. Not providing any specific means would oblige the stakeholders to look at the variable identifiers of a FAMILIAR script that store intermediate FMs produced by FAMILIAR operators. In order to keep the traceability, we associate meta-information to each feature of an FM that reccords the list of FMs where the feature comes from. This list of meta-informations is internally updated when needs be by operators of FAMILIAR (e.g., the merge operator). Additional accessors allowing to query or modify the contents of the list of meta-informations are provided in order to address situations where the provenance of a given feature is needed.

6 Discussion

6.1 Summary

Building service-oriented scientific workflows mainly consists in first selecting some appropriate services from all available parameterized services, then configuring and assembling them in a consistent way. In many application domains, these activities are cumbersome and error-prone, and this hampers current development efforts in computational science. In this article, we introduced a rigorous and tooled approach that extends current SPL engineering techniques to facilitate and automate consistent workflow construction. The following contributions were presented:

- The organization of a catalog of services as a SPL was described. It enables service providers to capture in a structured way the commonalities and variabilities present in the parameterized services. Relying on FMs to structure SPLs, we have defined merging techniques so that a SPL can be automatically generated from all service descriptions, following an user-chosen classification.
- A multi-step process to obtain a consistent workflow was also detailed. Taking a basic description of the workflow, it consists in specifying different variable concerns (ranging from functional parameters to non-functional properties or deployment specificities) on one or more services. Constraints within or between concerns can be added and all these elements are incrementally checked for consistency against the service catalog. The workflow is then seen as a multiple SPL which composes the SPLs of services. Configuration is assisted, consistency checking and propagation are incremental and automated (see Section 6.2.1 below), so that a consistent workflow product is obtained. Evolution of both the services and their variable parts is also supported by the approach. Moreover this process completely rests on a sound formal basis realized by FM management operators and a Domain Specific Language, so that generic parts of the process can be more easily reused in other contexts (see Section 6.2.2 below).

– An overview of the implementation and user-oriented tool support has also been given. Besides illustrations were provided using a non trivial example in the representative domain of medical image analysis. Additionally, first experimental results are discussed below (see Section 6.2.3).

6.2 Assessment

6.2.1 User Assistance and Degree of Automation

Organizing the workflow construction with SPL engineering techniques leads to a shift in the process. The activities are then well targeted for each kind of stakeholder. The service provider documents variability once and for all, the catalog maintainer handles all available services over time and the workflow designer can focus on its construction activity. It must also be noted that our proposed approach is heavily relying on the service catalog, so that the effort of building it is compensated. The catalog is indeed used both to assist the expert, when determining his relevant concerns — which is more error-prone when specified from scratch —, and to incrementally enforce consistency.

A first assistance to the user is provided when he/she can select an appropriate service from among sets of existing services. He/she may also want to search services matching several criteria to determine whether at least one service can fulfill a specific feature or a set of features. During the process, if the variability manipulated by the user leads to some inconsistency but is considered to be more important than the workflow structure, the user has to correct the workflow itself. Using our approach, such inconsistencies are automatically and systematically detected and several correction strategies can be applied. The separation of concerns provides the ability to precisely locate the source of errors and to give information to assist users in correcting the workflow: choose an other service, correct an applied concern, relax some variability description, configure differently some services.

Properties of the merge operator can then be exploited. The various compositions of FMs may be performed in any order because of the associativity property of the merge operator. Heuristics, such as merging larger FMs first, can thus be planned to detect an earlier source of errors. The merge between FMs contributes to decrease the number of remaining variability choices presented to the user. Indeed an additional property of the merge in intersection mode is that the number of features of the resulting FM is lesser than or equal to the number of features commonly shared by input FMs. This property can dramatically reduce the set of configurations to be considered by the user during workflow configuration. As a result, it is likely that the amount of time and effort needed during the configuration process can be reduced (see experimental results below).

As for the process automation, it ranges from the catalog building to the resulting workflow product. First, taking all service variability descriptions, the catalog of services is automatically generated. During the workflow construction and configuration, all assisted steps discussed before are coupled with incremental and automatic consistency checking. The specified concerns over services are extracted from the catalog with the guarantee to be consistent subsets. After having been woven to services, their consistency according the workflow is automatically checked. When the resulting workflow is configured, the automatic propagation of constraints among feature models representing the concerns is conducted, ensuring again consistency while reducing

the user burden. Finally, as the variability of services may evolve over time, the complete process can be easily replayed to check again the consistency with additional or modified concerns.

6.2.2 Applicability

In our case study, a medical imaging service can be seen as an SPL provided by different researchers or scientific teams. The entire workflow is then a *multiple* SPL in which different SPLs are composed. Applying the same approach to another domain is possible and many automatic parts of the process can be reused.

Let us consider that the other domain would manipulate connected components, possibly hierarchically composed so that one would face different software artifacts with a different composition techniques. This case is comparable to the variable components proposed by Van der Storm in [van der Storm, 2004]. The process would then remain similar to the one we propose. The variability would have to be extracted from components and expressed as FMs, then organized in a new catalog, reusing the FAMILIAR script (see Section 5.2). A new DSL for weaving concerns on relevant point-cuts of components would have to be designed. Interpretation for this DSL (the "Workflow Analyzer" block in Fig. 14) would need to be developed, so that either reusable FAMILIAR scripts can be called or FAMILIAR code can be generated in order to provide automate propagation and checking as in our workflow illustration. Consequently, the application of our approach would only necessitate to focus on the definition and weaving of FM concerns, whereas the main difficulties of handling FM composition would be automated.

We believe this is a significant improvement, as managing multiple SPLs is identified as a complex issue in the literature. For example, Hartmann and Trew dealt with multiple SPLs and identified several compositional issues in the context of software supply chains. They notably recognized that "merging FMs, especially when they are overlapping, requires a significant engineering activity" [Hartmann and Trew, 2008]. They did not provide a set of operators, a semantics nor a mechanism to automate this task. Hartmann et al. also introduced the Supplier Independent Feature Model (SIFM) in order to select products among the set of products described by several Supplier Specific Feature Models (SSFM) [Hartmann et al., 2009]. Intuitively, the SIFM references several SSFMs thanks to constraints between features. Our merging techniques produce more compact FMs and thus reduce the number of features to be considered.

6.2.3 Experiments

Application to Three Real Workflows. Using the tool support described in Section 5, we have applied the proposed approach to three real medical imaging workflows, the Alzheimer's disease workflow [Lorenzi et al., 2010] used as a running example in this paper, a cardiac analysis workflow [Maheshwari et al., 2009], and a workflow for determining the quality of a segmentation algorithm [Pernod et al., 2008]. The number of services that constitute the three workflows varies from 9 to 24 (see #services in Table 2), so that experiments are conducted on different scales[7]. We consider scenarios in which the workflow designer augments the workflow description with FMs and

[7] The size of *scientific* workflows varies depending on the domain (e.g., bioinformatics, medical imaging). In the medical imaging domain, the presence of 24 services can be considered as a large workflow, even though larger workflows have been developed.

constraints. As we want to determine the ability of our approach to handle compatibility constraints between services, we count the number of *active* dependencies (see #active). A dependency between two connected services is active when there are FMs related to the same concern on both sides, so that these FMs have to be merged. This number is lesser or equal than the number of data dependencies (see #dependencies) since FMs are not necessary attached to all dataports.

In Table 2, the total number of FMs (see #FMs), features (see #features), core features (see #cores), variation points (see #VP) and configurations[8] (see #configurations) in the initial workflow description are reported for the three workflows. Core features refer to features necessary included in any configuration whereas variation points refer to features whose selection/deselection still needs to be fixed. In order to determine how the proposed automated reasoning reduces the number of variation points (thus the number of valid configurations) and possibly facilitates the decision-making process, the same metrics are reported after the reasoning mechanism.

The first experiment concerns the workflow used throughout the paper [Lorenzi et al., 2010]. This workflow is rather small (composed of 9 services) and 16 dependencies between data ports are present. We wove 12 FMs (using the catalog of FMs) into workflow services but not into *Atrophy measure* and *Mask Calculation* services. As a result, 9 compatibility constraints were detected (see #active). We did not edit the FMs and we only specified some intra-constraints services. Applying the reasoning mechanisms significantly reduced the number of variation points (from 79 to 32) and the number of configurations (from 10^{12} to 10^{4}).

For the second experiment, we used the cardiac analysis workflow described in [Maheshwari et al., 2009]. The management of variability was focused on data pre-treatments so that we only wove 8 FMs and we deliberately did not consider other parts of the workflow. We specialized the format of the image sources before propagating choices. Again we observe a noticeable reduction of variability points.

For the third experiment, we used a larger workflow (cf. [Pernod et al., 2008]) in which 24 services are combined to evaluate segmentation. A noticeable property of this workflow is that 6 registration services and 5 normalization services are used. We thus made an extensive use of the catalog. Again, we did not edit the FMs and we only specified some intra-constraints services. The reduction of variation points is even more significant (from 176 to 67), mainly because of the large number of data dependencies (41) that are automatically handled by our approach.

As a result, these experiments show that the reasoning mechanisms developed for supporting consistent composition of multiple SPLs significantly reduces the high complexity to be managed by the workflow designer. Larger experimental validations should confirm these first results (see Section 6.4).

Practical Experience. We design an experiment in which three different users from the NeuroLOG project had to design and configure a workflow *without* our techniques and then, for the sake of comparison, *with* our techniques. These users were mainly PhD students or software engineers with a good knowledge of scientific workflows, but practically no skill in FM or SPL engineering. The input of the experiment is as follows:

– a medium-size, GWENDIA workflow (see Fig. 1) that consists in 9 processes (only 5 processes, *Affine Registration*, *Brain Extraction*, *Longitudinal intensity correc-*

[8] The number of initial configurations is computed by considering FMs without inter-/compatibility constraints.

		1. Alzheimer's disease	2. Cardiac analysis	3. Segmentation evaluation
Input workflow	#services	9	14	24
	#dependencies	16	20	41
	#active	9	6	19
Initial specification	#FMs	12	8	25
	#features	131	97	286
	#cores	52	43	110
	#VPs	79	54	176
	#configurations	10^{12}	10^{9}	10^{25}
After reasoning	#features	104	79	213
	#cores	72	48	146
	#VPs	32	31	67
	#configurations	10^{4}	10^{5}	10^{9}

Table 2 Experimental results on three scientific workflows.

tion, *Tissue segmentation* and *Non-linear registration*, have to be configured in the experiment) ;
- a description of about 80 existing services according to different criteria. We consider two categories of criteria: the first category concerns *medical images* (e.g., medical image format supported as input/output), the second category is about *medical imaging algorithm* (e.g., affine or non-affine transformation). For this experiment, the average number of features to consider per service is around 30. Semi-structured, tabular data are used for the description of variability services in terms of features and stored in CSV (comma-separated values) format, as it is the case in Fig. 3. To facilitate the identification of services, we group together similar services that are candidate to implement a process ;
- a document describing in natural language the requirements of the application and the constraints of the workflow. Three scenarios are described in the document: the first scenario simply consists in selecting five services while ensuring data compatibility ; the second scenario is similar to the first scenario except that some requirements on the anonymization of images are added ; the third scenario involves more constraints (e.g., formats of images that can be used are restricted to two predefined alternatives and no interactive algorithm can be used).

The challenge for users is to have, at the end of the experiment, a workflow in which appropriate services are consistently combined. We report below our observations and lessons learned.

Effort and Time Needed. In the experiment, users have to consider a large number of candidate services (80) for a large number of features per service (30), so that the total number of distinct features[9] to consider is more than 200. At this scale, the configuration process turned out to be impractical without adequate support. In particular, a manual configuration process (e.g., based on "trial and error" strategy) should be avoided as it is both error-prone, laborious and time-consuming. The observation applies to two specific tasks of the workflow design. Firstly, when users have to select a service from among the set of existing services, the main difficulty comes from the fact that some features from different concerns interact (e.g., the selection of the format DICOM may imply the selection of an interactive algorithm), which is

[9] Two features are distinct if their names are distinct.

not straightforward to identify. In addition, users complain from the lack of *querying* operations, for example, to filter the set of services that fulfills specific requirements. Secondly, when users have to ensure that services are data compatible, multiple variability descriptions are to be considered for resolving complex features' interactions. The difficulties we observed are not surprising. The satisfiability of an FM is known to be a difficult computational problem, i.e., NP-complete [Schobbens et al., 2007] and in our case, not only one FM has to be considered. Several authors claim that in real software projects, there can be thousands of features whose legal combinations are governed by many and often complex rules [Mendonça, 2009, Mendonca and Cowan, 2010, Hubaux et al., 2010, Janota, 2010] – the design of scientific workflows exhibits similar complexity. As argued by the same authors, it is thus of crucial importance to be able to simplify and automate the decision-making process as much as possible [Mendonça, 2009, Mendonca and Cowan, 2010, Hubaux et al., 2010, Janota, 2010]. Our observations and case study reinforce this statement. Our tool-supported approach do assist the user when he/she can select an appropriate service from among sets of existing services (thanks to the construction of catalog of FMs) and propagate variability choices (thanks to automated reasoning techniques). Once the catalog of FMs has been built and the variability has been specified at the workflow level, the time and effort needed to complete the configuration process is manageable.

In addition, the three experimental scenarios can be realized in a similar fashion. We just *reused* the catalog of FMs and modified the original script *Wfamily* developed for the first scenario to fulfill the new requirements. The costly process (in time and user effort) is related to the construction of the catalog of FMs and development of *Wfamily* scripts. For this experiment, the catalog construction was highly facilitated by *i)* the identification and grouping of similar services, *ii)* the use of a common terminology and hierarchy of features to described services. The development of the *Wfamily* scripts was more laborious due to the costs of *i)* learning a new language and *ii)* understanding the concepts behind the approach.

User Assistance and Correctness. Manual attempts for configuring the workflow all led to several errors that should have been corrected. Additionally, without adequate assistance, the process would have been reiterated several times. Moreover a manual checking that determines whether a selection/deselection of features is correct (i.e., does not violate any constraint of the workflow and corresponds to at least one existing service) was proved to be not satisfying (i.e., confidence about the solution was too low). An important benefit of using automated techniques based on a sound basis is that we can assist users at each step of the configuration process while guaranteeing properties of the designed workflow. FAMILIAR (and thus the underlying implementation of the approach) rely on SAT solvers or BDD. It has been shown that both SAT solvers and BDD can be used to implement an *interactive* configuration process, where the computer provides information about validity of choices each time the user makes a new choice – this feedback typically takes the form of graying out the choices that are no longer valid [Mendonça, 2009, Janota, 2010]. Moreover a configurator can *infer* which choices are valid and which are not at each step of the process.

Flexibility. Another drawback of a non tooled-approach was the lack of *flexibility* in selecting an appropriate service. Although finding one appropriate service can be sufficient, it is more preferable to have the choice between several candidate services, for example, to favor services that have been developed by a specific research team.

Advanced querying operations were thus identified as important when designing the workflow. Our tool-supported approach supports a proper management of variability and the ability to infer which existing services are no longer able to fulfill the requirements.

Inadequate Support. Our experience with the tool-supported approach reveals that there is room for improvements and further research and developments:

- *user interface*: an advanced user interface should be developed to facilitate the modeling of variability and the configuration process. Indeed the *Wfamily* DSL suffers from a lack of integration with the workflow editor of GWENDIA, such that it is difficult and error-prone for users to specify the weaving of FMs and the mapping with the catalog. The development of new visualization techniques, for example, to facilitate the connection understanding between different FMs of the workflow, is an interesting perspective to consider ;
- *explanations*: a lot of variability choices are inferred to speed up the configuration process. In some cases, the user wants to know why a certain feature was automatically selected or eliminated, i.e. he/she wants explanations. Though some techniques exist (e.g., see [White et al., 2010, Janota, 2010]), it should be integrated and adapted in our context since several FMs are potentially impacted ;
- *modeling*: there exists some services that support the same combination of features (e.g., same format, same algorithm method), so that even at the end of the configuration process, more than two services are still adequate. In that case, more information could be included to describe and select services, including quality attributes (e.g., see [Benavides et al., 2010]) attached to features.

Threats to Validity. Threats to external validity are conditions that limit our ability to generalize the results of our experiment to scientific workflow design practice. In our experiments, we use three real scientific workflows, already developed and used by scientists. Workflows are on different scales (the presence of 24 processes can be considered as a large workflow in the medical imaging domain, even though larger workflows exist.). However, further empirical studies need to be conducted on more complex workflows and catalogs of FMs as well as more participants.

A threat to internal validity may be related to the tools that were used to work with the workflow and the FMs. In our experiments, we used our own tools (*MOTEUR*, *Wfamily*, FAMILIAR), then the results might be affected by their usability. We report above the need of a more comprehensive and mature environment. Nevertheless, even if the provided environment does not have all required skills, we already observe benefits in terms of effort and time needed, user assistance and correctness.

Another internal threat concerns the correctness of the reasoning techniques implementation. For instance, the merge operators are supposed to guarantee that some semantic properties are preserved when building the catalog of FMs. Our implementation is currently checked by a comprehensive set of unit tests, complemented by cross-checked testing with other operations provided by FAMILIAR. We also manually verified a large number of slice and merge examples as well as their specific uses in the context of our case study.

6.3 Related Work

Feature Models. A few other approaches use multiple FMs during the SPL development. Kang et al. defined four layers, each containing a number of FMs that are interrelated with constraints between features [Kang et al., 1998]. These layers and their FMs are rather used on a structural level and mainly provides guidelines for building FMs, whereas our contribution is to propose mechanisms to attach any FM to a system and to reason about the relations between FMs. In [Czarnecki et al., 2005], separate FMs are used to model decisions taken by different stakeholders or suppliers. The authors recognize the need to compose and merge FMs during multi-stage and multi-step configuration process, but do not achieve it. In [Tun et al., 2009], several FMs are used to separate feature descriptions related to requirements, problem world context and software specifications. Constraints then inter-relate features of FMs. Metzger et al. proposed a formal approach for separating product lines variability (e.g., economical-oriented variability) and software variability, thereby enabling automatic analysis [Metzger et al., 2007]. The two kinds of variability can be considered as concerns of an SPL. Previous contributions do not consider FMs or concerns that are sharing some features. This can happen when concerns along the same dimension interact, when multiple perspectives on a concern needs to be managed or when SPLs are composed with SPLs. A few work [Schobbens et al., 2007, Alves et al., 2006, Segura et al., 2008, Hartmann and Trew, 2008] suggested the use of a merge operator. Schobbens et al. identified three operations to merge FMs – intersection, union (a.k.a. disjunction) or reduced product of two FMs [Schobbens et al., 2007]. Alves et al. and Segura et al. proposed a catalogue of rules to merge FMs [Alves et al., 2006, Segura et al., 2008]. Our proposal goes further in this direction as it clarifies the semantics of the merge and, most importantly, shows how this operator can be used in practice. In addition, we already compared the different approaches [Segura et al., 2008, Schobbens et al., 2007, Acher et al., 2009] to implement merge operators in [Acher et al., 2010a].

Several languages support features and their composition by superimposition (see, e.g., AHEAD [Batory et al., 2003]). The work presented in [Apel et al., 2008] presents an algebra that unifies all these languages. A feature is represented by a FST (Feature Structure Tree), roughly a stripped-down abstract syntax tree and can be seen as an FM without variability information. The superimposition mechanism and the underlying theory do not consider the case where *variability* (e.g., optional, alternatives) mismatches have to be resolved, as we have done with the merge operator.

In [Hubaux et al., 2009, Mendonca and Cowan, 2010], the configuration process is represented as a workflow and different stakeholders are configuring the same FM. The first difference with our work is that the term workflow used in the approach does not refer to a processing pipeline, but to the activities completed during configuration. The second difference is that only a single FM is considered during the whole configuration process. In [Hubaux et al., 2010], techniques are proposed to extract and configure *views* (or concerns) from an FM, which come with three alternative visualisations. The extraction operator we use in this paper (e.g., to split a catalog FM) may be complemented by these techniques. In a staged or parallel configuration process, invalid configurations may be created. Techniques have been developed to automate the diagnosis of the errors and specify the minimal set of feature selections and deselections to remove the errors [White et al., 2010]. They can be applied in the context of our work during the configuration process (see ⑥ in Fig. 6, page 14).

Beyond Feature Models. In SPL engineering, reusable software artefacts (e.g. requirements model) must be composed to derive specific products. In [Pohl et al., 2005], Orthogonal Variability Models (OVMs) are proposed to document the variability in all artefacts of an SPL. In the same way, FMs can be used to specify the variability and then to relate FMs to code [Apel and Kästner, 2009], design models [Perrouin et al., 2008, Voelter and Groher, 2007, Czarnecki and Antkiewicz, 2005, Ziadi and Jézéquel, 2006] [Jayaraman et al., 2007] or domain-specific modeling languages [White et al., 2009]. Our work focuses strictly on the composition of the variability models, i.e., FMs. Our proposal is not incompatible with these approaches, as each FM can be related to other code/model elements and thus be used during the product derivation process. Besides relating our approach to final code is one of our envisaged future work (see Section 6.4).

Separation of Concerns. Various composition approaches exist in aspect-oriented software development. Our contribution is largely inspired by previous work on viewpoints (e.g., [Nuseibeh et al., 1993]), subject-oriented approaches (e.g., [Ossher et al., 1996, Baniassad and Clarke, 2004]), and multidimensional separation of concerns (MDSoC) (e.g., [Tarr et al., 1999, Batory et al., 2003, France et al., 2003]). MDSoC deals with software systems containing overlapping concerns which lie along multiple concern dimensions. MDSoC is also an issue in requirements engineering [Moreira et al., 2005]. Work on *early aspects* focuses on systematically identifying, modularizing, and analyzing concerns in requirement analysis, domain analysis and architecture design [Baniassad et al., 2006]. We precisely propose techniques to manage variability of those concerns along multiple dimensions and perspectives. Currently, the merge process dedicated to FMs does not require user intervention and automatically guarantees some properties according to the sets of configurations of input FMs. We plan to extend the merge operator by providing to the user techniques to drive the composition process (see next section).

Service-Orientation, Workflows and SPLs. A large amount of work exists in (automatic) service composition (e.g., see [Dustdar and Schreiner, 2005]). To the best of our knowledge, there is no specific approach combining separation of concerns while managing variability in the same kind of context. In [Charfi and Mezini, 2007], AO4BPEL promotes a well-modularized specification of concerns and dynamic strategy for web service composition. Our work focuses on how to ensure in a processing chain, at design time, consistency between concerns with respect to variability. In [Wada et al., 2007, Fantinato et al., 2008], a framework is proposed to model the constraints between non-functional aspects (e.g., quality of service properties) in SOAs. Through the notion of feature modeling, the proposed framework allows developers to validate non-functional constraints. The feature modeling formalism used in their approach is more expressive than in ours. Nevertheless, it turns out that the semantics of FMs needs to be revised in order to reason about multiple variability sources described within each service and for the entire workflow. A few work investigate the relationship between SOAs, business process models (BPMs) and SPLs (see, e.g., [Boffoli et al., 2008, Lee and Kotonya, 2010]). Work in [Schnieders and Puhlmann, 2007] focused on how to map a FM to a business process model described in BPEL; each feature of a FM corresponds to a business process. Similarly, an approach to service identification methods is proposed in [Kang and Baik, 2010] that bridges the FMs of SPLs and the BPMs in SOA. In [Gottschalk et al., 2008], a general approach is proposed

to configure workflow models expressed in YAWL. The authors developed an extension of YAWL, called CYAWL, to configure the elements of a workflow model such that the behavior represented by the model is restricted. The motivation of our work is rather to describe the variability *within* a process. In the context of our case study (scientific workflows), we do not observe structural variability. We thus consider that the processing chain is fixed and does not vary structurally, i.e., there is no optional process or variant relationship in the workflow. Nevertheless, the approaches proposed in [Schnieders and Puhlmann, 2007, Ogasawara et al., 2009, Kang and Baik, 2010] might be combined with the techniques we proposed in this paper.

6.4 Future Work

Tool Support. In the proposed approach, a set of domain-specific (textual) languages is currently provided to the workflow designer. The activities he/she performs includes the specification of the workflow, the FMs weaving instructions, possibly the specification of constraints and the mapping with catalog FMs. An extension of this work is to build, on top of these languages, a user interface in which graphical facilities are provided for modeling the workflow, weaving FMs, etc. into an *integrated* environment. As revealed by some experiences we have with our current tool, this should facilitate and accelerate the process.

Alignment of Feature Models. An key element in the *automation* of the proposed approach is the merge operator. It is extensively used to realize catalogs building and querying, compatibility checking, etc. All along the paper, we have identified some situations in which the intervention of the workflow designer may be necessary, for example, when he/she creates an FM from scratch and then merges the developed FM with a catalog FM. A major problem is that the FMs to be merged are not aligned. It occurs when the FMs do not share the same vocabulary for describing features' name ; the FMs have different level of granularity (e.g., much more details in one of the FMs) ; the hierarchy of features differs ; the features refer to different concepts. In our case study, the alignment effort is currently not significant since suppliers rely on a common ontology [Temal et al., 2008] while FMs are views on such an ontology [Fagereng Johansen et al., 2010]. Though we provide guidelines to prevent the problem (e.g., to opt for the reuse of a catalog FM rather than the development of a new FM), the need to integrate several FMs from different, independent sources may still be persistent. To the best of our knowledge, there is no (semi-)automated techniques to align FMs. To handle such situations, we are currently working on extensions of our merging techniques that go beyond manually restructuring the hierarchy and renaming or removing features.

Validation. Validation on our medical imaging use case currently continues on a larger scale. We are planning the construction of a catalog containing hundreds of legacy services, so that valuable feedback can be obtained on both the approach and the user process. In addition, based on the set of features that are selected and mapped to implementation-specific elements, services descriptors are planned to be automatically generated for their deployments and executions on the grid. Besides, we are also developing a second case in the video surveillance domain in which configurable components

are composed into a processing chain to be deployed in various contexts while being adaptable at runtime [Acher et al., 2011c]. On the long term, our objective is to provide a model-based, end-to-end approach for generating the targeted software systems. We also expect to identify complementarity ways of managing FMs and to develop specific methodological guidelines.

References

Acher et al., 2009. Acher, M., Collet, P., Lahire, P., and France, R. (2009). Composing Feature Models. In *2nd International Conference on Software Language Engineering (SLE'09)*, LNCS, pages 62–81. Springer.

Acher et al., 2010a. Acher, M., Collet, P., Lahire, P., and France, R. (2010a). Comparing Approaches to Implement Feature Model Composition. In *6th European Conference on Modelling Foundations and Applications (ECMFA)*, volume 6138 of *LNCS*, pages 3–19. Springer.

Acher et al., 2010b. Acher, M., Collet, P., Lahire, P., and France, R. (2010b). Managing Variability in Worklow with Feature Model Composition Operators. In *9th International Conference on Software Composition (SC'10)*, volume 6144 of *LNCS*, pages 17–33. Springer.

Acher et al., 2011a. Acher, M., Collet, P., Lahire, P., and France, R. (2011a). A Domain-Specific Language for Managing Feature Models. In *26th International Symposium on Applied Computing (SAC'11)*, pages 1333–1340, Taiwan. Programming Languages Track, ACM.

Acher et al., 2011b. Acher, M., Collet, P., Lahire, P., and France, R. (2011b). Slicing Feature Models. In *26th IEEE/ACM International Conference On Automated Software Engineering (ASE'11), short paper*, , Lawrence, USA. IEEE/ACM.

Acher et al., 2011c. Acher, M., Collet, P., Lahire, P., Moisan, S., and Rigault, J.-P. (2011c). Modeling Variability from Requirements to Runtime. In *16th International Conference on Engineering of Complex Computer Systems (ICECCS'11)*, pages 77–86, Las Vegas. IEEE.

Acher et al., 2008. Acher, M., Collet, P., Lahire, P., and Montagnat, J. (2008). Imaging Services on the Grid as a Product Line: Requirements and Architecture. In *Service-Oriented Architectures and Software Product Lines - Putting Both Together (SOAPL'08)*. (associated workshop issue of SPLC 2008), IEEE Computer Society.

Alves et al., 2006. Alves, V., Gheyi, R., Massoni, T., Kulesza, U., Borba, P., and Lucena, C. (2006). Refactoring product lines. In *Proc. of GPCE'2006*, pages 201–210. ACM.

Apel and Kästner, 2009. Apel, S. and Kästner, C. (2009). An overview of feature-oriented software development. *Journal of Object Technology (JOT)*, 8(5):49–84.

Apel et al., 2008. Apel, S., Lengauer, C., Moller, B., and Kastner, C. (2008). An algebra for features and feature composition. In *12th International Conference on Algebraic Methodology and Software Technology (AMAST '08)*, volume 5140 of *LNCS*, pages 36–50. Springer.

Baniassad and Clarke, 2004. Baniassad, E. and Clarke, S. (2004). Theme: An approach for aspect-oriented analysis and design. In *ICSE'04*, pages 158–167. IEEE.

Baniassad et al., 2006. Baniassad, E., Clements, P. C., Araujo, J., Moreira, A., Rashid, A., and Tekinerdogan, B. (2006). Discovering early aspects. *IEEE Softw.*, 23(1):61–70.

Batory et al., 2003. Batory, D., Liu, J., and Sarvela, J. N. (2003). Refinements and multi-dimensional separation of concerns. In *ESEC'03: Proceedings of the 9th European software engineering conference*, pages 48–57, Helsinki, Finland. ACM.

Batory, 2005. Batory, D. S. (2005). Feature models, grammars, and propositional formulas. In *9th International Software Product Line Conference (SPLC'05)*, volume 3714 of *LNCS*, pages 7–20.

Benavides et al., 2010. Benavides, D., Segura, S., and Ruiz-Cortés, A. (2010). Automated analysis of feature models 20 years later: A literature review. *Information Systems*, 35:615–636.

Beuche et al., 2004. Beuche, D., Papajewski, H., and Schrader-Preikschat, W. (2004). Variability management with feature models. *Science of Computer Programming*, 53(3):333–352.

Boffoli et al., 2008. Boffoli, N., Caivano, D., Castelluccia, D., Maggi, F. M., and Visaggio, G. (2008). Business process lines to develop service-oriented architectures through the software product lines paradigm. In Thiel, S. and Pohl, K., editors, *SPLC (2)*, pages 143–147. Limerick, Ireland.

Bosch, 2009. Bosch, J. (2009). From software product lines to software ecosystems. In *Proc. of SPLC'2009*, volume 446 of *ICPS*, pages 111–119. ACM.

Boucher et al., 2010. Boucher, Q., Classen, A., Faber, P., and Heymans, P. (2010). Introducing TVL, a text-based feature modelling language. In *4th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'10)*, pages 159–162.

Buhne et al., 2005. Buhne, S., Lauenroth, K., and Pohl, K. (2005). Modelling requirements variability across product lines. In *RE '05: Proceedings of the 13th IEEE International Conference on Requirements Engineering*, pages 41–52, Washington, DC, USA. IEEE Computer Society.

Charfi and Mezini, 2007. Charfi, A. and Mezini, M. (2007). AO4BPEL: an aspect-oriented extension to BPEL. *World Wide Web*, 10(3):309–344.

Clements and Northrop, 2001. Clements, P. and Northrop, L. M. (2001). *Software Product Lines : Practices and Patterns*. Addison-Wesley Professional.

Czarnecki and Antkiewicz, 2005. Czarnecki, K. and Antkiewicz, M. (2005). Mapping features to models: A template approach based on superimposed variants. In *GPCE'05*, volume 3676 of *LNCS*, pages 422–437.

Czarnecki and Eisenecker, 2000. Czarnecki, K. and Eisenecker, U. (2000). *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley.

Czarnecki et al., 2005. Czarnecki, K., Helsen, S., and Eisenecker, U. (2005). Staged Configuration through Specialization and Multilevel Configuration of Feature Models. *Software Process: Improvement and Practice*, 10(2):143–169.

Czarnecki et al., 2006. Czarnecki, K., Kim, C. H. P., and Kalleberg, K. T. (2006). Feature models are views on ontologies. In *SPLC '06: Proceedings of the 10th International on Software Product Line Conference*, pages 41–51, Washington, DC, USA. IEEE Computer Society.

Czarnecki and Pietroszek, 2006. Czarnecki, K. and Pietroszek, K. (2006). Verifying feature-based model templates against well-formedness ocl constraints. In *GPCE'06*, pages 211–220. ACM.

Czarnecki and Wąsowski, 2007. Czarnecki, K. and Wąsowski, A. (2007). Feature diagrams and logics: There and back again. In *11th International Software Product Line Conference (SPLC'07)*, pages 23–34. IEEE.

Dustdar and Schreiner, 2005. Dustdar, S. and Schreiner, W. (2005). A survey on web services composition. *Int. J. Web Grid Serv.*, 1(1):1–30.

Fagereng Johansen et al., 2010. Fagereng Johansen, M., Fleurey, F., Acher, M., Collet, P., and Lahire, P. (2010). Exploring the Synergies Between Feature Models and Ontologies. In *International Workshop on Model-driven Approaches in Software Product Line Engineering (MAPLE 2010)*, volume 2 of *SPLC '10*, pages 163–171, Jeju Island, South Korea. Lancester University.

Fantinato et al., 2008. Fantinato, M., de Toledo, M. B. F., and de Souza Gimenes, I. M. (2008). Ws-contract establishment with qos: an approach based on feature modeling. *Int. J. Cooperative Inf. Syst.*, 17(3):373–407.

Foster et al., 2002. Foster, I., Kesselman, C., Nick, J., and Tuecke, S. (2002). The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Technical report, Open Grid Service Infrastructure WG, GGF.

France et al., 2003. France, R., Georg, G., and Ray, I. (2003). Supporting multi-dimensional separation of design concerns. In *3rd workshop on AOM with UML at AOSD'03*.

Gil et al., 2007. Gil, Y., Deelman, E., Ellisman, M. H., Fahringer, T., Fox, G., Gannon, D., Goble, C. A., Livny, M., Moreau, L., and Myers, J. (2007). Examining the challenges of scientific workflows. *IEEE Computer*, 40(12):24–32.

Glatard et al., 2008. Glatard, T., Montagnat, J., Lingrand, D., and Pennec, X. (2008). Flexible and efficient workflow deployment of data-intensive applications on grids with moteur. *Int. J. High Perform. Comput. Appl.*, 22:347–360.

Gottschalk et al., 2008. Gottschalk, F., van der Aalst, W. M. P., Jansen-Vullers, M. H., and Rosa, M. L. (2008). Configurable workflow models. *Int. J. Cooperative Inf. Syst.*, 17(2):177–221.

Hartmann and Trew, 2008. Hartmann, H. and Trew, T. (2008). Using feature diagrams with context variability to model multiple product lines for software supply chains. In *12th International Software Product Line Conference (SPLC'08)*, pages 12–21, Washington, DC, USA. IEEE.

Hartmann et al., 2009. Hartmann, H., Trew, T., and Matsinger, A. (2009). Supplier independent feature modelling. In *13th International Software Product Line Conference (SPLC'09)*, pages 191–200. IEEE.

Heidenreich et al., 2010. Heidenreich, F., Sanchez, P., Santos, J., Zschaler, S., Alferez, M., Araujo, J., Fuentes, L., amd Ana Moreira, U. K., and Rashid, A. (2010). Relating feature models to other models of a software product line: A comparative study of featuremapper and vml*. *Transactions on Aspect-Oriented Software Development VII, Special Issue on A Common Case Study for Aspect-Oriented Modeling*, 6210:69–114.

Hubaux et al., 2009. Hubaux, A., Classen, A., and Heymans, P. (2009). Formal modelling of feature configuration workflows. In *13th International Software Product Line Conference (SPLC'09)*, pages 221–230. IEEE.

Hubaux et al., 2010. Hubaux, A., Heymans, P., Schobbens, P.-Y., and Deridder, D. (2010). Towards multi-view feature-based configuration. In Wieringa, R. and Persson, A., editors, *REFSQ*, volume 6182 of *Lecture Notes in Computer Science*, pages 106–112. Springer.

Janota, 2010. Janota, M. (2010). *SAT Solving in Interactive Configuration*. PhD thesis, Department of Computer Science at University College Dublin.

Jayaraman et al., 2007. Jayaraman, P. K., Whittle, J., Elkhodary, A. M., and Gomaa, H. (2007). Model composition in product lines and feature interaction detection using critical pair analysis. In *MODELS'07*, LNCS, Springer, pages 151–165.

Kang and Baik, 2010. Kang, D. and Baik, D.-K. (2010). Bridging software product lines and service-oriented architectures for service identification using bpm and fm. In *Computer and Information Science (ICIS), 2010 IEEE/ACIS 9th International Conference on*, pages 755 –759.

Kang et al., 1990. Kang, K., Cohen, S., Hess, J., Novak, W., and Peterson, S. (1990). Feature-Oriented Domain Analysis (FODA). Technical Report CMU/SEI-90-TR-21, SEI.

Kang et al., 1998. Kang, K., Kim, S., Lee, J., Kim, K., Shin, E., and Huh, M. (1998). FORM: a feature-oriented reuse method with domain-specific reference architectures. *Annals of Software Engineering*, 5(1):143–168.

Kästner et al., 2009. Kästner, C., Thüm, T., Saake, G., Feigenspan, J., Leich, T., Wielgorz, F., and Apel, S. (2009). Featureide: A tool framework for feature-oriented software development. In *31st International Conference on Software Engineering (ICSE'09), Tool demonstration*, pages 611–614.

Lee and Kotonya, 2010. Lee, J. and Kotonya, G. (2010). Combining service-orientation with product line engineering. *IEEE Software*, 27(3):35–41.

Lorenzi et al., 2010. Lorenzi, M., Ayache, N., Frisoni, G., and Pennec, X. (2010). 4d registration of serial brain mr's images: a robust measure of changes applied to alzheimer's disease. In *Miccai Workshop on Spatio-Temporal Image Analysis for Longitudinal and Time-Series Image Data. First prize award for the "Best Oral Presentation"*, Beijing, China.

Maheshwari et al., 2009. Maheshwari, K., Glatard, T., Schaerer, J., Delhay, B., Camarasu, S., Clarysse, P., and Montagnat, J. (2009). Towards Production-level Cardiac Image Analysis with Grids. In *HealthGrid'09*, , Berlin.

McPhillips et al., 2009. McPhillips, T., Bowers, S., Zinn, D., and Ludäscher, B. (2009). Scientific workflow design for mere mortals. *Future Gener. Comput. Syst.*, 25:541–551.

Mendonça, 2009. Mendonça, M. (2009). Efficient reasoning techniques for large scale feature models. Master's thesis, University of Waterloo, Waterloo.

Mendonca et al., 2009. Mendonca, M., Branco, M., and Cowan, D. (2009). S.p.l.o.t.: software product lines online tools. In *OOPSLA '09: Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, pages 761–762, New York, NY, USA. ACM.

Mendonca and Cowan, 2010. Mendonca, M. and Cowan, D. (2010). Decision-making coordination and efficient reasoning techniques for feature-based configuration. *Science of Computer Programming*, 75(5):311 – 332.

Metzger et al., 2007. Metzger, A., Pohl, K., Heymans, P., Schobbens, P.-Y., and Saval, G. (2007). Disambiguating the documentation of variability in software product lines: A separation of concerns, formalization and automated analysis. In *15th IEEE International Conference on Requirements Engineering (RE '07)*, pages 243–253.

Montagnat et al., 2009. Montagnat, J., Isnard, B., Glatard, T., Maheshwari, K., and Blay-Fornarino, M. (2009). 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. ACM.

Moreira et al., 2005. Moreira, A., Rashid, A., and Araujo, J. (2005). Multi-dimensional separation of concerns in requirements engineering. In *Requirements Engineering (RE '05)*, pages 285–296. IEEE.

Nuseibeh et al., 1993. Nuseibeh, B., Kramer, J., and Finkelstein, A. (1993). Expressing the relationships between multiple views in requirements specification. In *ICSE'93*, pages 187–196. IEEE.

Ogasawara et al., 2009. Ogasawara, E., Paulino, C., Murta, L., Werner, C., and Mattoso, M. (2009). Experiment line: Software reuse in scientific workflows. In *Proceedings of the 21st International Conference on Scientific and Statistical Database Management*, SSDBM 2009, pages 264–272, Berlin, Heidelberg. Springer-Verlag.

Ossher et al., 1996. Ossher, H., Kaplan, M., Katz, A., Harrison, W., and Kruskal, V. (1996). Specifying subject-oriented composition. *Theor. Pract. Object Syst.*, 2(3):179–202.

Pernod et al., 2008. Pernod, E., Souplet, J.-C., Rojas Balderrama, J., Lingrand, D., and Pennec, X. (2008). Multiple Sclerosis Brain MRI Segmentation Workflow Deployment On The EGEE Grid. pages 55–64.

Perrouin et al., 2008. Perrouin, G., Klein, J., Guelfi, N., and Jézéquel, J.-M. (2008). Reconciling automation and flexibility in product derivation. In *SPLC'08*, pages 339–348. IEEE.

Pohl et al., 2005. Pohl, K., Böckle, G., and van der Linden, F. J. (2005). *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Reiser and Weber, 2007. Reiser, M.-O. and Weber, M. (2007). Multi-level feature trees: A pragmatic approach to managing highly complex product families. *Requir. Eng.*, 12(2):57–75.

Schnieders and Puhlmann, 2007. Schnieders, A. and Puhlmann, F. (2007). Variability modeling and product derivation in e-business process families. In Abramowicz, W. and Mayr, H. C., editors, *Technologies for Business Information Systems*, pages 63–74. Springer Netherlands.

Schobbens et al., 2007. Schobbens, P.-Y., Heymans, P., Trigaux, J.-C., and Bontemps, Y. (2007). Generic semantics of feature diagrams. *Computer Networks*, 51(2):456–479.

Segura et al., 2008. Segura, S., Benavides, D., Ruiz-Cortes, A., and Trinidad, P. (2008). Automated merging of feature models using graph transformations. *Post-proceedings of the Second Summer School on GTTSE*, 5235:489–505.

Svahnberg et al., 2005. Svahnberg, M., van Gurp, J., and Bosch, J. (2005). A taxonomy of variability realization techniques: Research articles. *Software Practice and Experience*, 35(8):705–754.

Tarr et al., 1999. Tarr, P., Ossher, H., Harrison, W., and Sutton, Jr., S. M. (1999). N degrees of separation: multi-dimensional separation of concerns. In *21st International Conference on Software Engineering (ICSE'99)*, pages 107–119. ACM.

Temal et al., 2008. Temal, L., Dojat, M., Kassel, G., and Gibaud, B. (2008). Towards an ontology for sharing medical images and regions of interest in neuroimaging. *Journal of Biomedical Informatics*. To appear.

Thüm et al., 2009. Thüm, T., Batory, D., and Kästner, C. (2009). Reasoning about edits to feature models. In *31st International Conference on Software Engineering (ICSE'09)*, pages 254–264. ACM.

Tun et al., 2009. Tun, T. T., Boucher, Q., Classen, A., Hubaux, A., and Heymans, P. (2009). Relating requirements and feature configurations: A systematic approach. In *13th International Software Product Line Conference (SPLC'09)*, pages 201–210. IEEE.

van der Storm, 2004. van der Storm, T. (2004). Variability and component composition. In *International Conference on Software Reuse (ICSR'04)*, volume 3107 of *LNCS*, pages 157–166. Springer.

van Ommering, 2002. van Ommering, R. (2002). Building product populations with software components. In *22rd International Conference on Software Engineering (ICSE'02)*, pages 255–265. ACM.

van Ommering and Bosch, 2002. van Ommering, R. and Bosch, J. (2002). Widening the scope of software product lines - from variation to composition. In *Software Product Lines*, pages 31–52. LNCS.

Voelter and Groher, 2007. Voelter, M. and Groher, I. (2007). Product line implementation using aspect-oriented and model-driven software development. In *11th International Software Product Line Conference (SPLC'07)*, pages 233–242. IEEE Computer Society.

Wada et al., 2007. Wada, H., Suzuki, J., and Oba, K. (2007). A feature modeling support for Non-Functional constraints in service oriented architecture. In *SCC'07*, pages 187–195.

White et al., 2010. White, J., Benavides, D., Schmidt, D. C., Trinidad, P., Dougherty, B., and Cortés, A. R. (2010). Automated diagnosis of feature model configurations. *Journal of Systems and Software*, 83(7):1094–1107.

White et al., 2009. White, J., Hill, J. H., Gray, J., Tambe, S., Gokhale, A. S., and Schmidt, D. C. (2009). Improving domain-specific language reuse with software product line techniques. *IEEE Software*, 26:47–53.

Ziadi and Jézéquel, 2006. Ziadi, T. and Jézéquel, J.-M. (2006). *Product Line Engineering with the UML: Deriving Products*, chapter 15, pages 557–586. Number 978-3-540-33252-7 in Software Product Lines: Reasearch Issues in Engineering and Management. Springer Verlag.