Scientific Workflows: Eternal Components, Changing Interfaces, Varying Compositions

Anna-Lena Lamprecht and Tiziana Margaria

Chair for Service and Software Engineering
Potsdam University, Germany
{lamprecht,margaria}@cs.uni-potsdam.de

Abstract. We describe how scientific application domains are characterized by the long-term availability of the basic computational components, and how software systems for managing the actual scientific workflows must deal with changing service interfaces and varying service compositions. In this light, we explain how rigorous technical and semantic abstraction, which is key to dealing with huge and heterogeneous application domains in an "extreme model driven design" framework like the jABC, supports the management of workflow evolution. We illustrate the different aspects by means of examples and experiences from the application of the framework in different scientific application domains.

1 Introduction

Evolution of software systems and long-lived applications are currently intensively researched topics under many points of view [1, 2]. In the new field of e-science, workflow management for scientific applications is a key application domain that combines artifacts with very different timelines and life cycles. The basic algorithmic components that perform the individual analysis steps are in fact very long-lived: Many of the popular algorithms, tools, and databases have been available for over a decade and remained mainly unchanged. Their concrete use and composition, however, varies considerably from case to case, according to the current scientific analysis process and the involved data. In fact, progress and novelty in "in silico" experimentation, where experiments and analyses are carried out in computers on the basis of preexisting data and knowledge, thus largely happens "ex aliquo", and not "ex nihilo", i.e. from scratch. Therefore we need to distinguish two fields of progress and evolution: in the *application domain*, that originates from fast-paced evolution of analysis and simulation processes that use preexisting resources, and the progress and evolution of the *IT ingredients*, that makes the first one possible and is itself much less frequent. Process evolution occurs in the frequent case that an existing analysis process has to be adapted to new experimental requirements or data. Standard software evolution on the contrary occurs mainly when existing algorithms, tools and databases are equipped with new interfaces, which happens relatively seldom. The overall setting therefore yields a fairly stable basis of software artifacts, that are combined and recombined at fast pace to try out new analyses in an orchestration or coordination-oriented composition.

In this paper we describe how an extreme model driven approach [3, 4] supports the agile management of process/workflow evolution in the light of such really huge and truly heterogeneous application domains. The core concept is that here different levels of abstraction make a completely symbolic treatment of the involved entities possible. A combination of these abstractions then allows for handling application-level processes and their evolution at a semantic level, within the application domain.

The past decade has seen a lot of research on scientific workflow management in general (see, e.g, [5, 6]) and on the use of semantics-based methods for supporting service composition in particular (see, e.g., [7–10]). We are not aware, however, of any work that looks at the evolution of scientific workflows from the broad service-engineering perspective that we describe in this paper, which is based on the jABC framework [11, 12]. We draw here on the experience of several years of usage for the management of scientific workflows both in teaching and in research projects (cf., e.g., [13–18]),

The paper is structured as follows. Section 2 introduces the abstraction concepts needed to enable and support the fast-paced workflow evolution needed for "in-silico" e-science; Section 3 illustrates the specific jABC-based approach by reporting some exemplary experiences from the bioinformatics and geo-visualization application domains, and Section 4 concludes the paper.

2 Abstraction for Fast-Paced Workflow Evolution

The conceptual framework of *eXtreme Model-Driven Design* (XMDD) [3, 4, 19] aims at an agile yet service-oriented modeling, design, and development of process-style applications within a domain-specific setting. E-science is such a domain, with specific incarnations e.g. for bioinformatics, or geo-visualisation.

The central assets are here a collection of services (implemented by means of software artifacts) and a domain knowledge representation (implemented via taxonomies/ontologies) that can be easily used by domain experts and that are supported by a sophisticated framework that helps the user in the selection, composition, validation, and execution of the resulting workflows. While it is possible to achieve parts of this goal by means of traditional approaches, that use heterogeneous technologies to cover different aspects and subproblems[1], the *jABC* [11, 12] is a concrete framework that supports high automation and consistency in working with processes and workflows by offering a number of special-purpose plugins in an extreme model-driven setting, where the user only works at the model level and the necessary compositions and transformations are largely taken care of by the framework itself via specific plugins.

[1] For example, using the standard technologies from the different sub-communities of software engineering, one could have components modeled in UML, wrapped as services in a WSDL, with orchestrations expresses in BPEL or BPMN or Petri nets and domain knowledge expressed in WSMO or OWL ontologies.

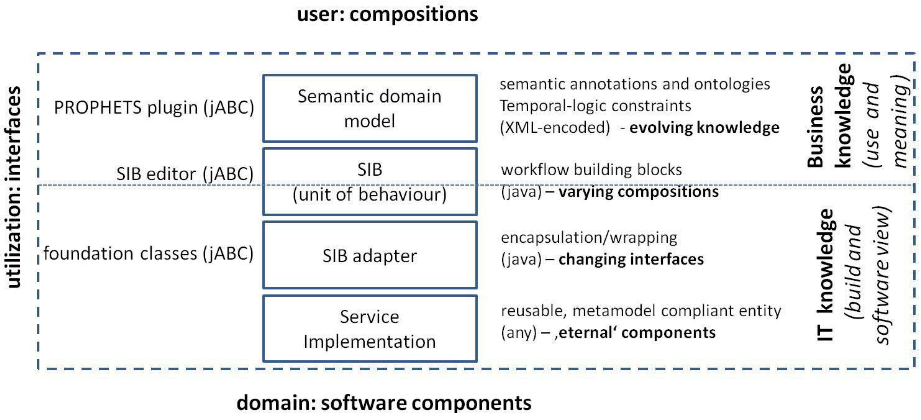
user: compositions
utilization: interfaces
PROPHETS plugin (jABC)
SIB editor (jABC)
foundation classes (jABC)
Semantic domain model
SIB (unit of behaviour)
SIB adapter
Service Implementation
semantic annotations and ontologies Temporal-logic constraints (XML-encoded) - evolving knowledge
workflow building blocks (java) – varying compositions
encapsulation/wrapping (java) – changing interfaces
reusable, metamodel compliant entity (any) – ‚eternal' components
Business knowledge (use and meaning)
IT knowledge (build and software view)
domain: software components

Fig. 1. From the domain knowledge to the IT: Abstraction layers in the jABC framework

In particular, the PROPHETS plugin [20] bundles functionality for semantic annotations, model checking, and automatic synthesis of workflows according to the *Loose Programming* paradigm of [21], which offers abstraction from the concrete workflow (see Sect. 3.4). The basis for the models in jABC are libraries of semantically annotated, behavioral workflow components, called *SIBs* (*Service Independent Building Blocks*). SIBs provide access to encapsulated units of functionality, which can be freely combined into flowchart-like workflow structures called *Service Logic Graphs* (SLGs) that are technically service orchestrations. The concretization of SLGs towards running systems typically happens in a hierarchical fashion via several refinement steps (cf. [22]).

The key to an agile dealing with software evolution is rigorous virtualization/abstraction: striving for a completely symbolic treatment of the involved entities allows one to handle application-level processes at a semantic level, as illustrated in Figure 1. Concretely, given the usual service or component implementation customary in component-based design or in service-oriented architectures, the jABC works with three principal layers of abstraction, which address different challenges of software evolution:

1. The *SIBs* are the actual behavioral entities of the workflow building blocks as defined from the application's/user's perspective. They tailor the single workflow building blocks to the specific needs of the application/user. In essence, a SIB defines an adequate interface for a workflow building block, and connects it to the services that are provided by the SIB adapters. Thus, if something in the SIB adapters changes, the corresponding SIB implementations can be adapted accordingly, but the workflow models themselves are not affected. In the sense of [23], the concept of a SIB is that of a *behavioral metamodel*: it presents all sorts of needed components in a uniform manner to the user and makes them really work in a simple, easy-to-use and composite fashion.

2. The *SIB adapters* absorb the change management of evolving technologies and evolving components by abstracting from the technical details of the underlying platforms and service implementations, which are typically implemented in heterogeneous technologies. Thus, if something in the service implementation changes, the SIB adapters can be changed accordingly, but the SIBs, as workflow building blocks, are themselves not affected: Workflows use only the SIB-level information.
3. *Semantic domain models* on top of the SIB libraries drive the abstraction even further, allowing in particular for the completely symbolic description of the SIBs and their parameters in terms of the application's/user's domain language. As shown in Fig. 1, the user's composition of workflows happens within the "business domain" knowledge, that includes SIB selection and composition, while the IT issues are taken care of within the lower layers [24, 25]. In particular, this organization facilitates "loose" programming of workflows, where parts of the workflow model can be left underspecified, and are only concretized upon request. Thus, if anything in the SIB library changes, the semantic domain model can be independently adapted, and the (loose) workflow models themselves are not affected.

In the following sections we describe these notions of technical and semantic abstraction in greater detail. Together, they lead to two separate levels in workflow development: a *story level*, that takes place between the SIB and the semantic knowledge layers (the upper two in Fig. 1), where one designs and communicates the spirit of the analysis/experiment, and the actual *executable level* (the lower two layers in Fig. 1), where a concretized version takes care of the coherence and consistency of all details.

2.1 Handling Technical Abstraction: The SIBs

The SIBs in the jABC are services in the proper sense, encapsulating units of functionality that are defined from the application's/user's perspective. Instead of being architectural components, as most service component models like in the standard SCA [26], they are *units of behavior* oriented to their use within processes or workflows. As discussed for instance in [25], granularity decisions are essential for the design of SIBs that are adequate for the envisaged applications. While standard services are remote software units that offer a collection of behavior to their users, SIBs only have one behavior and are thus easier to understand, manage, and compose. A single standard (web) service can thus correspond to a collection of SIBs. This explains why they are easy to use for domain experts, much easier than standard services.

Additionally, SIBs provide homogeneous service interfaces that truly abstract from the technical details of the underlying implementation. Their provisioning involves integrating distributed services that are provided via heterogeneous technologies (such as SOAP and REST web services, legacy tools, and specific APIs) into homogeneous libraries of workflow building blocks.

The SIB structure is depicted in the lower three layers of Figure 1:

– The actual *SIB*[2] provides the service interface within the workflow environment. In addition to service documentation and other usability information, it defines the service parameters to be configured by the calling application/workflow.
– The *SIB adapter* handles the service call, using the information from the SIB class that is relevant for its execution (esp. parameters).
– The *service implementation* defines the actual execution behavior. It can be arbitrary functionality in any programming language.

Thus, as the underlying implementation changes, a SIB's adapter has to be exchanged or adapted too, but the SIB class - which is the actual interface to the workflow environment - mostly remains unchanged. In practice, this largely decouples the workflow development in the jABC framework from the evolution of the underlying platforms and of the concrete algorithms.

2.2 Handling Semantic Abstraction: Loose Programming

Loose programming [21] supports a form of model-based *declarative* software development that enables workflow developers to design their application-specific workflows in an intuitive (graphical) style driven by their domain knowledge, rather than from the technicalities of composition and composability. In particular, it aims at making highly heterogeneous collections of services accessible to application experts who have no classical programming skills, but who need to design and manage complex workflows. After an adequate domain modeling, application experts should ultimately be able to profitably and efficiently work with a world-wide distributed collection of services and data, using their own domain language and understanding services at the behavioral metamodel level. In particular, the semantic domain models abstract from the SIB interfaces, making everything in the workflow environment even more symbolic and intuitively comprehensible to the scientist.

Concretely, the semantic domain models are defined on top of the SIB libraries and comprise:

– service and data type taxonomies that provide semantic categories and relations for the involved entities, building the *domain vocabulary*,
– behavioral *interface descriptions*, i.e. input and output annotations, in terms of the domain vocabulary, and
– temporal-logic *constraints* that express additional knowledge about the application domain in general or about the intended workflows in particular.

An example for such models will be discussed in Sect. 3. These semantic domain models are entirely based on symbolic names for services and data types,

[2] Technically, the SIB class, which we associate with the part of the SIB that belongs to the IT world in contrast to the view on the SIB from the business domain (from above), which corresponds to concepts in the domain knowledge model.

therefore the user-level semantic service descriptions are completely decoupled from the SIBs that implement the actual functionality. Accordingly, the semantic description(s) provided for a service can be freely defined: it is possible to use custom terminology, use the same service for different purposes, or simply omit unnecessary details in the interface description.

With loose programming users specify their intentions about a workflow in a very sparse way, by just giving an intuitive, high-level rough process flow in terms of ontologically defined semantic entities from the domain model, without caring about types, precise knowledge about the available workflow components or the availability of resources. A synthesis mechanism in the background automatically completes this sketch into a correctly running workflow by inserting missing details. This is achieved by means of a combination of different formal methodologies: Data-flow analysis provides information on available and required resources, which is used by a temporal-logic synthesis algorithm [27] to find sequences of services that are suitable to concretize the loose parts. Additionally, model checking is used to monitor global process constraints continuously.

In loose programming there is thus abstraction from the concrete workflow that implements a particular analysis process: if the set of available SIBs/services changes, the framework can automatically find another suitable composition of services that solves the problem.

In the next section we report some exemplary experiences with software evolution from the application of the jABC framework in the bioinformatics application domain.

3 Examples and Experiences

To show the spread of possible applications and techniques, we focus now on four different aspects: Dealing with the wealth of command line tools (Sect. 3.1), dealing with a technology migration for entire collections of widely used services (Sect. 3.2), dealing with the high volatility of ad-hoc workflow design and evolution (Sect. 3.3), and dealing with a new, declarative way of describing the intentions of a workflow that is automatically synthesized in one or more variants (Sect. 3.4). In the following, we concentrate mainly on examples from bioinformatics, but experiences from the geo-visualization domain are analogous.

3.1 SIBs for "Good Old" Command Line Tools

In scientific application domains, algorithms are often implemented as "small" special-purpose tools that can simply be invoked from the command line, without requiring to cope with "unnecessary" stuff (like fancy GUIs etc.) that is intended to help the user but often hampers programmatic, systematic access to the underlying functionality. Thus, classical command line tools are usually well suited to provide collections of basic building blocks for service compositions. Conveniently, classical command line tools are in fact very popular in the bioinformatics and geo-visualization domains.

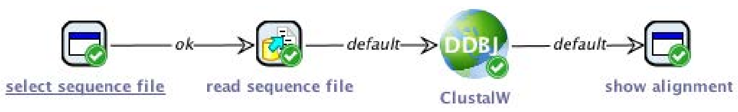

Fig. 2. Basic multiple sequence alignment workflow

For example, the European Molecular Biology Open Software Suite (EMBOSS) [28][3] is a rich collection of freely available tools for the molecular biology user community dealing with proteins and amino acids. It contains a number of small and large programs for a wide range of tasks, such as sequence alignment, nucleotide sequence pattern analysis, and codon usage analysis as well as the preparation of data for presentation and publication.

This and similar collections have been used in the respective user community for quite a long time: the first version of EMBOSS was released around 2000, and their functionalities are still useful as a basis for new workflow applications. This is an example of the longevity of basic domain-specific "eternal" components that support several generations of scientists and serve unchanged the communities for decades.[4]

Sometimes such service collections are repackaged and provided as "modern" web services. For example, the EMBOSS tools are provided as web services in the scope of the EBI's SoapLab project [30, 31]. Often, however, communication with the web services via these interfaces happens at a quite technical level: the sheer operations of the web services are not adequate for direct integration as workflow building blocks.

The jETI technology [32] specifically supports such direct integration for command line tools in the jABC framework. In fact, command line tools are typically designed to execute specific well-defined tasks, and usually all inputs and configuration options can be provided upon invocation, so that their execution runs completely autonomous (headless, in bioinformatics terminology). They also typically work on files in a pipe-and-filter transformer fashion, which is per se closer to the user-level than the programming language entities (such as Java objects) that are required for the communication with, e.g., Web Service APIs. Accordingly, jETI services, which are designed to provide convenient (remote) access to file-based command line tools as SIBs, are inherently closer to the user-level than web services. Entire collections of services can be made available to end-users this way.

[3] `http://emboss.sourceforge.net/`

[4] As an example from the geo-visualization domain: the Generic Mapping Tools (GMT) collection [29] (`http://gmt.soest.hawaii.edu/`) was released around the year 1990 and is in heavy use since then, also clearly deserving the denomination of *eternal components* used in the title.

3.2 SIBs for Bioinformatics Web Services: From SOAP to REST

Large bioinformatics institutions like the DDBJ (DNA Data Bank of Japan), EBI (European Bioinformatics Institute), and NCBI (National Center for Biotechnology Information) have been providing publicly available web services to access databases and computational tools already for a quite long time (that is, since web services became popular).

Currently, many major service providers are abandoning their SOAP-based web service interfaces and follow the general trend towards using REST interfaces. Consequently, the SIBs that had been implemented for accessing the DDBJ and EBI web services [33–35] had to be changed accordingly at some point, to follow this technology shift on the provider's side. Luckily, as discussed in Sect. 2, it was indeed only required to change in the SIB adapters the portion of code that executes the actual service calls. The SIB classes on the contrary were not touched at all by the transition: the user/application-level parameters of the services did not change. Likewise, on the workflow level this change of underlying technology was not perceptible at all.

This is an example of how the "changing interfaces" due to technology migration only locally impact the provisioning of the services. As il lustrated by Fig. 1, technology agnosticism at the behavioral metamodel level guarantees the stable fruition of the SIB for the end-users.

3.3 Agile Models for Variable/Evolving Scientific Workflows

Contrary to the stability of the single services offered, scientific workflows are characterized by being variant-rich and having to be adapted frequently to varying experimental setups and analysis objectives. Actually, in most cases the scientist is even searching for the optimal workflow in a cumbersome cycle of modification, test, analysis, and adaptation. The point of the research is often to find a data analysis or data processing workflow, that is itself the central result of the quest. Working by approximation, the volatility of such workflows is high, yielding series of "varying" compositions.

XMDD as evolution-oriented paradigm explicitly supports these kinds of application evolution and adaptation at a user-accessible level. In jABC, workflows can easily be modified, adapted, customized and tested in its graphical user interface, and (parts of) workflows can be prepared and flexibly (re-)combined according to current analysis objectives.

Example: Multiple Sequence Alignment. In the bioinformatics application domain, the multiple sequence alignment is an example that is particularly suited to illustrate the agility of workflow design (cf. [17] for further details). Figure 2 shows a simple workflow for this computation. In terms of algorithmic computations, it consists of just one SIB that calls an alignment service, here ClustalW. Just this step would however not suffice: it also needs some SIBs that take care of handling the input and output data. The SIB select sequence file (at the left, with the underlined name) lets the user select a file from the local file system and the SIB read sequence file puts the file's content into the execution context.

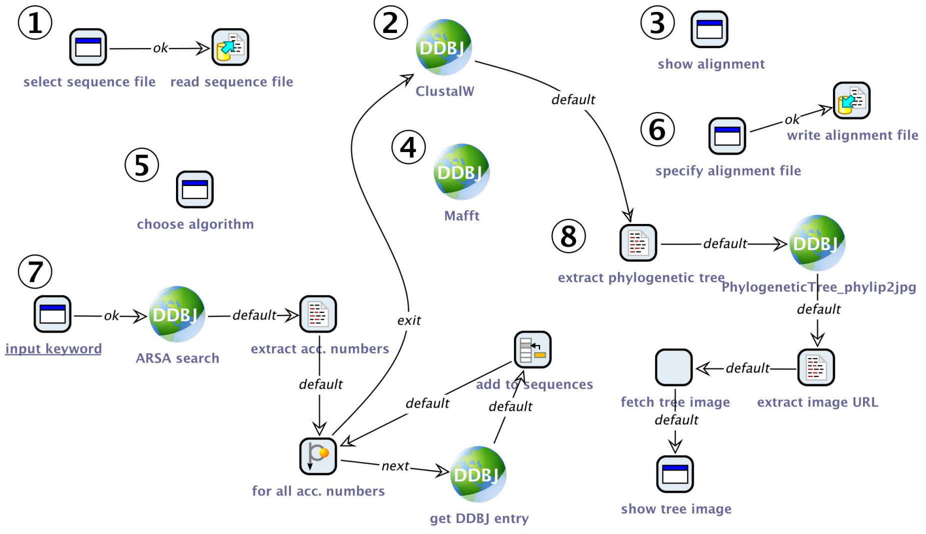

Fig. 3. Flexibly variable multiple sequence alignment workflow

This data is then sent to the DDBJ's ClustalW web service for the alignment computation, and finally show alignment displays the result to the user.

This is the simplest, but certainly not the only way an alignment can be computed. It can be useful to use other algorithms and to manage the input and output in different ways. Figure 3 shows an orchestration of SIBs (a Service Logic Graph) with several preconfigured workflow snippets that in detail provide the following functionalities:

1. Select and read a sequence file from the local file system.
2. Call the DDBJ's `ClustalW` alignment service.
3. Show an alignment in a simple text dialog window.
4. Call the DDBJ's `Mafft` alignment service.
5. Let the user choose the service.
6. Save the alignment to the local file system.
7. Let the user enter a keyword, which is used for a DDBJ database search (via the ARSA system). This results in a list of accession numbers (i.e. identifiers) for which the corresponding sequences are fetched from the DDBJ database.
8. Extract the phylogenetic tree that is part of a ClustalW result (using a regular expression) and call the `phylip2jpg` service of the DDBJ that converts the textual tree representation into an image, followed by retrieving and displaying the image.

These snippets might have arisen from the work of the same scientist in different contexts and stored for reusal, or have been designed by different community members and shared within the community. No matter their origin, they can now be put together to form various alignment workflows simply by connecting them appropriately. For instance, connecting the snippets 1, 2 and 3 results in the basic alignment workflow of Fig. 2. Connecting the snippets 2, 7, and 8 forms a more complex workflow (as depicted in Fig. 3), comprising database search by keyword, sequence retrieval, alignment computation, and visualization of the implied phylogenetic tree.

We see here that variability and a large selection of alternative subprocesses arise naturally in this highly dynamic domain and that artifacts are naturally shared within the community. Variability and reuse at this level, corresponding to the upper two layers of Fig. 1 are the norm in this kind of scientific applications.

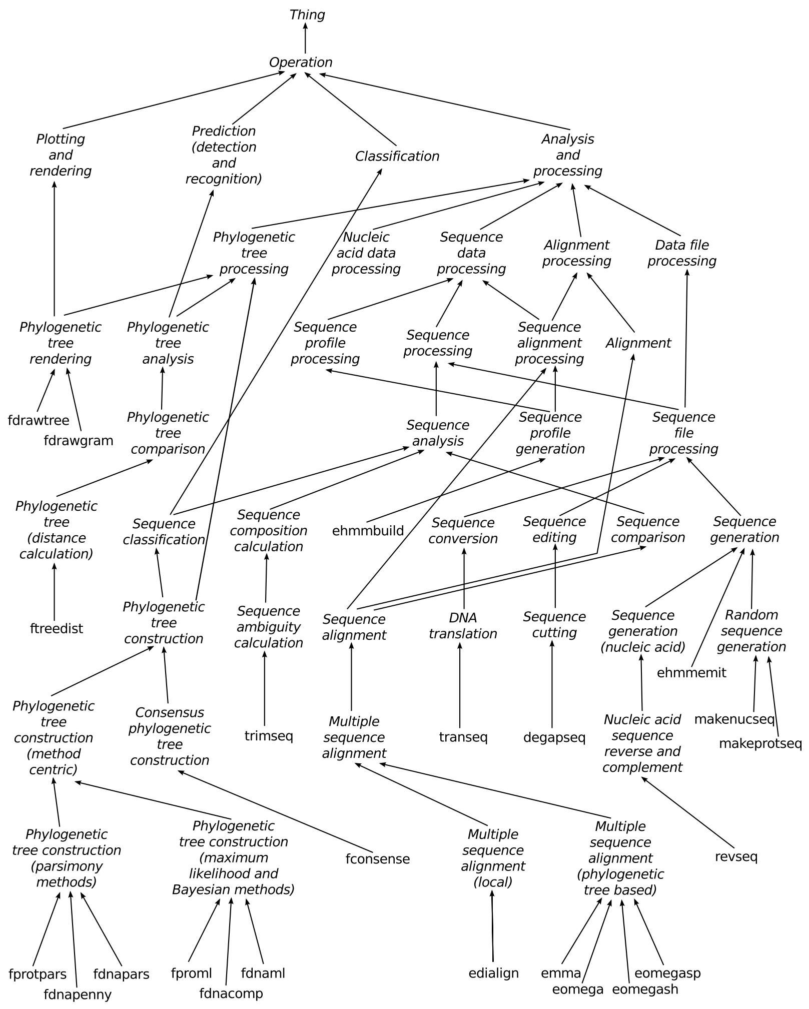
Thing
Operation
Plotting and rendering
Prediction (detection and recognition)
Classification
Analysis and processing
Phylogenetic tree processing
Nucleic acid data processing
Sequence data processing
Alignment processing
Data file processing
Phylogenetic tree rendering
Phylogenetic tree analysis
Sequence profile processing
Sequence processing
Sequence alignment processing
Alignment
fdrawtree
fdrawgram
Phylogenetic tree comparison
Sequence analysis
Sequence profile generation
Sequence file processing
Phylogenetic tree (distance calculation)
Sequence classification
Sequence composition calculation
ehmmbuild
Sequence conversion
Sequence editing
Sequence comparison
Sequence generation
ftreedist
Phylogenetic tree construction
Sequence ambiguity calculation
Sequence alignment
DNA translation
Sequence cutting
Sequence generation (nucleic acid)
Random sequence generation
ehmmemit
Phylogenetic tree construction (method centric)
Consensus phylogenetic tree construction
trimseq
Multiple sequence alignment
transeq
degapseq
Nucleic acid sequence reverse and complement
makenucseq
makeprotseq
Phylogenetic tree construction (parsimony methods)
Phylogenetic tree construction (maximum likelihood and Bayesian methods)
fconsense
Multiple sequence alignment (local)
Multiple sequence alignment (phylogenetic tree based)
revseq
fprotpars
fdnapenny
fdnapars
fproml
fdnacomp
fdnaml
edialign
emma
eomega
eomegash
eomegasp

Fig. 4. Part of the service taxonomy of the EMBOSS domain model

3.4 Loose Models for Variable and Evolving Scientific Workflows

While the previous section described how the jABC framework supports building variations of preconfigured workflow snippets consisting of known concrete SIBs, this section demonstrates how the loose programming provided by the jABC framework facilitates creating workflow variants in an exploratory way, without concrete knowledge about the available workflow building blocks. The envisaged workflow is only modeled declaratively, and the framework takes care of translating the specification into a concrete executable workflow based on the available collection of workflow building blocks.

With loose programming, workflow design is not only flexible with regard to changing experimental setups and analysis objectives as described above, but also with regard to evolving service libraries, as the synthesis framework automatically takes into account all changes and extensions of the domain model. This enables even more agile workflow development, as shown in [36] it is not required to pre-define the possible variants. As the underlying constraint language allows fully describing the intended solution space without imposing any overspecification, neither on the structure, nor on the artifacts, our approach may in particular be regarded as a step from the today typical settings with closed-world assumption to one with an *open-world assumption*, where new artifacts are automatically and seamlessly integrated in the domain description and thus in the loose programming solutions as soon as they are available.

Example: Phylogenetic Workflows Based on the EMBOSS Tool Suite. As an example from the field of bioinformatics, we take a look at loose programming of phylogenetic workflows with the EMBOSS tool suite (cf. [37, 38] for further details). Conveniently, as of release 6.4.0 from July 2011, the more than 400 tools of the EMBOSS suite and their parameters are annotated in terms of the EMBRACE Data and Methods Ontology (EDAM) [39], which allows for automatic generation of the semantic domain models that are required for loose programming (cf. [38]).

Figure 4 shows excerpts from the service taxonomy of the domain model. The OWL class *Thing* is always the root of the taxonomies, below which EDAM terms provide groups for concrete and abstract service and type representations. The (part of the) service taxonomy shown there comprises a number of service categories for different *Operation*s. Note that the type and service taxonomies comprise 565 and 1425 terms, respectively, directly after being derived from EDAM. They are then automatically reduced to those parts that are relevant for the services and data that appear in the domain model in order to avoid overhead, still covering 236 and 207 terms, respectively. To facilitate the printed presentation, the figure includes only the parts of the service taxonomy relevant for this example.

Table 1 lists the services that are relevant for the following examples, along with their input and output data types. It comprises only 23 of the more than 430 services in the complete domain model. The set of input types contains all mandatory inputs (i.e., optional inputs are not considered), while the set of output types contains all possible outputs. The service interface definitions only

Table 1. Selection of services from the EMBOSS domain model

Service	**Input types**	**Output types**
degapseq	*Sequence record*	*Sequence record*
edialign	*Sequence record*	*Sequence alignment*, *Sequence record*
ehmmbuild	*Sequence record (protein)*	*Hidden Markov Model*, *Sequence alignment (protein)*
ehmmemit	*Hidden Markov Model*	*Sequence record (protein)*
emma	*Sequence record*	*Phylogenetic tree*, *Sequence record*
eomega	*Sequence record*	*Phylogenetic tree*, *Sequence record*
eomegash	*Sequence record*, *Sequence-profile alignment (HMM)*	*Phylogenetic tree*, *Sequence record*
eomegasp	*Sequence record*, *Sequence-profile*	*Phylogenetic tree*, *Sequence record*, *Sequence distance matrix*
fconsense	*Phylogenetic tree*	*Phylogenetic tree*
fdnacomp	*Sequence record (nucleic acid)*	*Phylogenetic tree*
fdnaml	*Sequence alignment (nucleic acid)*	*Phylogenetic tree*
fdnapars	*Sequence alignment (nucleic acid)*	*Phylogenetic tree*
fdnapenny	*Sequence alignment (nucleic acid)*	*Phylogenetic tree*
fdrawgram	*Phylogenetic tree*	*Phylogenetic tree*
fdrawtree	*Phylogenetic tree*	*Phylogenetic tree*
fproml	*Sequence alignment (protein)*	*Phylogenetic tree*
fprotpars	*Sequence alignment (protein)*	*Phylogenetic tree*
ftreedist	*Phylogenetic tree*	*Phylogenetic report (tree distances)*
makenucseq	-	*Sequence record*
makeprotseq	-	*Sequence record (protein)*
revseq	*Sequence record*	*Sequence record (nucleic acid)*
transeq	*Sequence record*	*Sequence record (protein)*
trimseq	*Sequence record*	*Sequence record*

consider the data that is actually passed between the individual services, that is, input parameters that are used for configuration purposes are not regarded as service inputs.

Figure 5 (top) shows a simple loosely specified phylogenetic analysis workflow: it begins by generating a set of random nucleotide sequences (using the EMBOSS service `makenucseq`) and ends by drawing and displaying a tree image (using `fdrawtree` and the `viewer` SIB of the jETI plugin), respectively. The first two SIBs are connected by a loosely specified branch (colored red and labeled with a question mark). This loose branch constitutes a synthesis query to the PROPHETS plugin.

The lower part of the figure shows three of the millions of possible service sequences that solve this synthesis problem: The first, which is also one of the shortest solutions, is a single call to `emma` (an interface to ClustalW), which produces a phylogenetic tree in addition to a multiple sequence alignment. In the second, the reverse complement of the input sequence is built (`revseq`) and

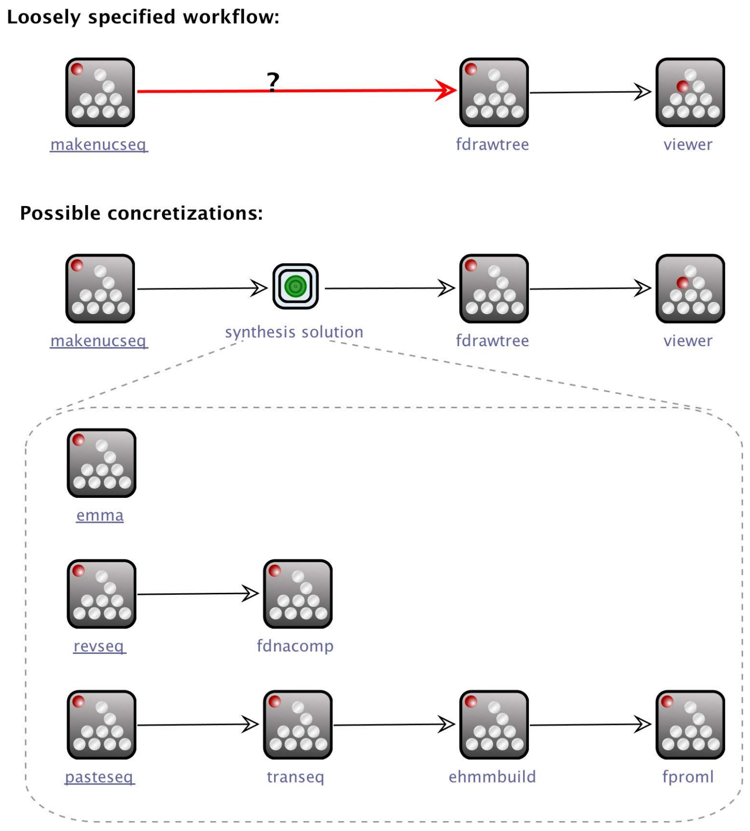

Fig. 5. Loosely specified phylogenetic analysis workflow and possible concretizations

then used for phylogenetic tree construction with `fdnacomp`. In the third, the sequences are translated into protein sequences (`transeq`), which are then aligned via `ehmmbuild` and used for phylogenetic tree estimation with `fprotpars`. The last solution is a four-step workflow where an additional sequence is pasted into the input sequences (`pasteseq`), which are then translated into protein sequences (`transeq`) and aligned via `ehmmbuild` before `fproml` is used for the tree construction.

Since EMBOSS provides various tools for phylogenetic tree construction as well as for the different sequence processing tasks, the solutions contained in the figure are by far not the only possible ones. In fact, millions of logically correct solutions are easily *possible* with the described domain model already when searching only for comparatively short solutions up to length 4. However, they comprise a lot of solutions that are not *desired* or *adequate*. Hence, it is desirable to influence the synthesis process so that it is more focused, returning less solutions that are more adequate. Here we see the advantages of the declarative approach to the problem formulation: we can simply provide temporal-logic constraints that express the corresponding intents by describing more precisely the wished solutions in terms of properties. Conveniently, PROPHETS provides natural-language templates for frequently occurring constraints, so that the workflow designer does not need to be trained in temporal logics. As an example, consider the following three constraints:

– *Do not use services that have no inputs.* Excludes services that distract from the actual synthesis problem: such services require no input but provide new data that is planted into the workflow.)
– *Do not use Sequence editing and alignment services.* (Avoids particular operations that are not wanted for some reason.)
– *Enforce the use of Phylogenetic tree construction (parsimony methods).* (Includes a particular kind of operation.)

With these constraints, manageable sets of adequate solutions are now obtained: there are two solutions of length three and 268 of length four.

This example illustrates how with loose programming workflow models remain robust against evolution of the service infrastructure and the semantic domain model: Loose workflow models and constraints capture the essential properties of the envisaged workflow, and can be synthesized at need into a concrete, executable workflow based on the currently available components using as services the SIBs and in the constraints the concepts of the domain model shown in the upper two layers of Fig. 1.

4 Conclusion

In this paper we focused on a central observation concerning software evolution in scientific application domains: Their basic software components (databases, algorithms, tools) remain available in largely unchanged form for a very long time, even decades, once they have been introduced. New functionality is added to the pool of available components rather than replacing existing assets. Hence, it is the *periphery* of the concrete service interfaces that is subject to sudden changes, for example when an entire suite of algorithms migrates its provisioning from SOAP to REST, and their application-specific use and composition are subject to fast-paced evolution, as the data analysis processes are themselves part of the research work and object of experimentation. In fields like bioinformatics and geo-visualization, e-science seems to have a hard core of stable ingredients (repositories of data and algorithms) and a sizzling outer layer of process-oriented experimental work that yields the progress of the disciplines today.

We have shown how the rigorous abstraction concepts of the extreme model driven design paradigm facilitate dealing with changing service interfaces and varying service compositions and thus with workflow evolution in these application domains. The decoupling of concerns due to adequate abstractions and layers in the semantic service engineering approach we propose, together with the plugin-based tool support offered by the jABC framework is the key to a semantics- (or application domain knowledge-) driven workflow design, that enables scientists (our end-users) to largely work within their domains of competence, without the need of IT knowledge as required by scripting languages that are today considered necessary for any "do it yourself"-style of scientific workflow composition. In particular the declarative loose specification approach, coupled with the automatic synthesis of executable workflows, seems to us to be a promising path towards self-assembling and self-optimizing processes: the declarative top-down approach (plus synthesis) is knowledge-driven and specifies just as much as necessary/wanted (but not more), which leads to an *open-world* assumption, where new components or services or repositories automatically appear in the solutions as soon as they are made available. This contrasts traditional orchestration-based approaches that explicitly define variability as a configuration space, which typically leads to overspecification and a *closed-world* assumption, where one actively deselects from a predefined choice of options.

References

1. Mens, T., Demeyer, S. (eds.): Software Evolution. Springer (2008)
2. : EternalS: EU FET Coordination Action on Trustworthy Eternal Systems via Evolving Software, Data and Knowledge
3. Margaria, T., Steffen, B.: Thinking in User-Centric Models. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation. CCIS, vol. 17, pp. 490–502. Springer, Heidelberg (2009)
4. Margaria, T., Steffen, B.: Service-Orientation: Conquering Complexity with XMDD. In: Hinchey, M., Coyle, L. (eds.) Conquering Complexity, pp. 217–236. Springer, London (2012)
5. Oinn, T., Greenwood, M., Addis, M., Alpdemir, M.N., Ferris, J., Glover, K., Goble, C., Goderis, A., Hull, D., Marvin, D., Li, P., Lord, P., Pocock, M.R., Senger, M., Stevens, R., Wipat, A., Wroe, C.: Taverna: lessons in creating a workflow environment for the life sciences: Research Articles. Concurr. Comput.: Pract. Exper. 18(10), 1067–1100 (2006)
6. Taylor, I.: Workflows for E-Science: Scientific Workflows for Grids. Springer (2007)
7. Chen, L., Shadbolt, N.R., Goble, C., Tao, F., Cox, S.J., Puleston, C., Smart, P.R.: Towards a Knowledge-Based Approach to Semantic Service Composition. In: Fensel, D., Sycara, K., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, pp. 319–334. Springer, Heidelberg (2003)
8. Ludäscher, B., Altintas, I., Gupta, A.: Compiling Abstract Scientific Workflows into Web Service Workflows. In: International Conference on Scientific and Statistical Database Management, p. 251 (2003)
9. Potter, S., Aitken, S.: A Semantic Service Environment: A Case Study in Bioinformatics. In: Gómez-Pérez, A., Euzenat, J. (eds.) ESWC 2005. LNCS, vol. 3532, pp. 694–709. Springer, Heidelberg (2005)
10. Withers, D., Kawas, E., McCarthy, L., Vandervalk, B., Wilkinson, M.: Semantically-Guided Workflow Construction in Taverna: The SADI and BioMoby Plug-Ins. In: Margaria, T., Steffen, B. (eds.) ISoLA 2010, Part I. LNCS, vol. 6415, pp. 301–312. Springer, Heidelberg (2010)
11. Steffen, B., Margaria, T., Nagel, R., Jörges, S., Kubczak, C.: Model-Driven Development with the jABC. In: Bin, E., Ziv, A., Ur, S. (eds.) HVC 2006. LNCS, vol. 4383, pp. 92–108. Springer, Heidelberg (2007)

12. Kubczak, C., Jörges, S., Margaria, T., Steffen, B.: eXtreme Model-Driven Design with jABC. In: CTIT Proc. of the Tools and Consultancy Track of the Fifth European Conference on Model-Driven Architecture Foundations and Applications (ECMDA-FA), vol. WP09-12, pp. 78–99 (2009)
13. Kubczak, C., Margaria, T., Fritsch, A., Steffen, B.: Biological LC/MS Preprocessing and Analysis with jABC, jETI and xcms. In: Proceedings of the 2nd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2006), Paphos, Cyprus, November 15-19, pp. 308–313. IEEE Computer Society, Paphos, Cyprus (2006)
14. Margaria, T., Kubczak, C., Njoku, M., Steffen, B.: Model-based Design of Distributed Collaborative Bioinformatics Processes in the jABC. In: Procedings of 11th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2006), Stanford, California, Los Alamitos, CA, USA, August 15-17, pp. 169–176. IEEE Computer Society (August 2006)
15. Margaria, T., Kubczak, C., Steffen, B.: Bio-jETI: a service integration, design, and provisioning platform for orchestrated bioinformatics processes. BMC Bioinformatics 9(suppl. 4), S12 (2008)
16. Lamprecht, A.L., Margaria, T., Steffen, B., Sczyrba, A., Hartmeier, S., Giegerich, R.: GeneFisher-P: variations of GeneFisher as processes in Bio-jETI. BMC Bioinformatics 9 (suppl. 4), S13 (2008)
17. Lamprecht, A.-L., Margaria, T., Steffen, B.: Seven Variations of an Alignment Workflow - An Illustration of Agile Process Design and Management in Bio-jETI. In: Măndoiu, I., Wang, S.-L., Zelikovsky, A. (eds.) ISBRA 2008. LNCS (LNBI), vol. 4983, pp. 445–456. Springer, Heidelberg (2008)
18. Lamprecht, A.L., Margaria, T., Steffen, B.: Bio-jETI: a framework for semantics-based service composition. BMC Bioinformatics 10(suppl. 10), S8 (2009)
19. Margaria, T., Steffen, B.: Business Process Modelling in the jABC: The One-Thing-Approach. In: Cardoso, J., van der Aalst, W. (eds.) Handbook of Research on Business Process Modeling. IGI Global (2009)
20. Naujokat, S., Lamprecht, A.-L., Steffen, B.: Loose Programming with PROPHETS. In: de Lara, J., Zisman, A. (eds.) FASE 2012. LNCS, vol. 7212, pp. 94–98. Springer, Heidelberg (2012)
21. Lamprecht, A.L., Naujokat, S., Margaria, T., Steffen, B.: Synthesis-Based Loose Programming. In: Proceedings of the 7th International Conference on the Quality of Information and Communications Technology, QUATIC (September 2010)
22. Steffen, B., Margaria, T., Braun, V., Kalt, N.: Hierarchical Service Definition. Annual Review of Communications of the ACM 51, 847–856 (1997)
23. Jung, G., Margaria, T., Nagel, R., Schubert, W., Steffen, B., Voigt, H.: SCA and jABC: Bringing a Service-Oriented Paradigm to Web-Service Construction. In: Margaria, T., Steffen, B. (eds.) ISoLA 2008. CCIS, vol. 17, pp. 139–154. Springer, Heidelberg (2008)
24. Margaria, T., Steffen, B.: Service Engineering: Linking Business and IT. Computer 39(10), 45–55 (2006)
25. Margaria, T., Bosselmann, S., Doedt, M., Floyd, B. D., Steffen, B.: Customer-Oriented Business Process Management: Visions and Obstacles. In: Hinchey, M., Coyle, L. (eds.) Conquering Complexity, pp. 407–429. Springer London (2012)
26. Service Component Architecture (SCA), `http://www.oasis-opencsa.org/sca/` (2012) (online; last accessed July 26, 2012)
27. Steffen, B., Margaria, T., Freitag, B.: Module Configuration by Minimal Model Construction. Technical report, Fakultät für Mathematik und Informatik, Universität Passau (1993)

28. Rice, P., Longden, I., Bleasby, A.: EMBOSS: the European Molecular Biology Open Software Suite. Trends in Genetics: TIG 16(6), 276–277 (2000)
29. Wessel, P., Smith, W. H. F.: Free software helps map and display data. EOS Trans. Amer. Geophys. U. 72(41) (1991)
30. Soaplab, `http://soaplab.sourceforge.net/soaplab1/` (online; last accessed June 25, 2012)
31. Soaplab2, `http://soaplab.sourceforge.net/soaplab2/` (online; last accessed June 25 2012)
32. Margaria, T., Nagel, R., Steffen, B.: jETI: A Tool for Remote Tool Integration. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 557–562. Springer, Heidelberg (2005)
33. DDBJ Web API for Biology, `http://xml.nig.ac.jp/workflow/` (online; temporarily suspended since February 15, 2012)
34. Pillai, S., Silventoinen, V., Kallio, K., Senger, M., Sobhany, S., Tate, J., Velankar, S., Golovin, A., Henrick, K., Rice, P., Stoehr, P., Lopez, R.: SOAP-based services provided by the European Bioinformatics Institute. Nucleic Acids Research 33(Web Server issue), W25–W28 (July 2005)
35. Labarga, A., Valentin, F., Anderson, M., Lopez, R.: Web services at the European bioinformatics institute. Nucleic Acids Research 35(Web Server issue), W6–W11 (2007)
36. Lamprecht, A., Margaria, T., Schaefer, I., Steffen, B.: Synthesis-based variability control: correctness by construction. In: Proceedings of FMCO 2011, Software Technologies Concertation Meeting on "Formal Methods for Components and Objects", Torino, Italy (October 2011)
37. Lamprecht, A.L., Naujokat, S., Margaria, T., Steffen, B.: Semantics-based composition of EMBOSS services. Journal of Biomedical Semantics 2(suppl. 1), S5 (2011)
38. Lamprecht, A.L., Naujokat, S., Steffen, B., Margaria, T.: Constraint-Guided Workflow Composition Based on the EDAM Ontology. In: Burger, A., Marshall, M.S., Romano, P., Paschke, A., Splendiani, A. (eds.) Proceedings of the 3rd Workshop on Semantic Web Applications and Tools for Life Sciences (SWAT4LS 2010). CEUR Workshop Proceedings, vol. 698 (December 2010)
39. Pettifer, S., Ison, J., Kalas, M., Thorne, D., McDermott, P., Jonassen, I., Liaquat, A., Fernandez, J.M., Rodriguez, J.M., Partners, I., Pisano, D.G., Blanchet, C., Uludag, M., Rice, P., Bartaseviciute, E., Rapacki, K., Hekkelman, M., Sand, O., Stockinger, H., Clegg, A.B., Bongcam-Rudloff, E., Salzemann, J., Breton, V., Attwood, T.K., Cameron, G., Vriend, G.: The EMBRACE web service collection. Nucl. Acids Res., gkq297 (May 2010)