Modeling and Execution of Scientific Workflows with the jABC Framework

Anna-Lena Lamprecht[1], Tiziana Margaria[1], and Bernhard Steffen[2]

[1] Chair for Service and Software Engineering, Potsdam University, Potsdam, D-14482, Germany
{lamprecht,margaria}@cs.uni-potsdam.de
[2] Chair for Programming Systems, Dortmund University of Technology, Dortmund, D-44227, Germany
bernhard.steffen@cs.tu-dortmund.de

Abstract. We summarize here the main characteristics and features of the jABC framework, used in the case studies as a graphical tool for modeling scientific processes and workflows. As a comprehensive environment for service-oriented modeling and design according to the XMDD (eXtreme Model-Driven Design) paradigm, the jABC offers much more than the pure modeling capability. Associated technologies and plugins provide in fact means for a rich variety of supporting functionality, such as remote service integration, taxonomical service classification, model execution, model verification, model synthesis, and model compilation. We describe here in short both the essential jABC features and the service integration philosophy followed in the environment. In our work over the last years we have seen that this kind of service definition and provisioning platform has the potential to become a core technology in interdisciplinary service orchestration and technology transfer: Domain experts, like scientists not specially trained in computer science, directly define complex service orchestrations as process models and use efficient and complex domain-specific tools in a simple and intuitive way.

Keywords: scientific processes and workflows, service integration, service orchestration, model-driven development, XMDD, jABC, jETI.

1 Introduction

As a general-purpose modeling framework for graphical process coordination and verification, the jABC [48] fully implements the concepts of service oriented computing [13]. It is a meanwhile mature service engineering environment that follows the eXtreme Model-Driven Development (XMDD) paradigm [37] and it has been used over the past two decades for business process and service logic modeling in several application domains, including telecommunications [43], supply chain management [14], e-commerce [20], and collaborative decision support systems [30,41], as well as scientific domains like bioinformatics [23].

The jABC uniformly supports all abstraction levels, ranging from the requirements analysis and conceptual design with non-IT experts (like in the requirements

A.-L. Lamprecht et al. (Eds.): Process Design for Natural Scientists, CCIS 500, pp. 14–29, 2014.
DOI: 10.1007/978-3-662-45006-2_2

and specification case study in Supply Chain Management described in [14]), over user-level design of scientific workflows [25,23] and the application to the construction of a family of retargetable compilers in Genesys [18,16], a framework for the high-level engineering of code generators in XMDD fashion [34], to the application design in the SWS Challenge Mediation Scenario [20] and middleware-level configurations in the MaTRICS [4,5] for the remote configuration and fault tolerance of the Online Conference Service [30].

From an end-user point of view, all the user interaction happens within an intuitive graphical environment, hardly requiring any classical programming skills. Users are able to model their analysis processes based on libraries of basic services in a graphical and intuitive way, combining functionality of services of different providers, and even from different application domains to solve complex problems that a single tool never would be able to tackle. Orchestration of services happens on the basis of the processes they realize in the respective application domain. These processes embody the business logic, and are expressed themselves as (executable) process models. The orchestrated processes can be hierarchical, allowing an easy reuse of subprocesses. Users are enabled to design and execute the orchestrated services, verify logical specifications using an embedded model checker or even generate stand-alone source code for the independent and repeating execution of a process. Services can also be grouped and classified according to domain-specific criteria, using taxonomies and ontologies.

Largely a rewrite and synthesis of [31,24,22,19,23], but also referring to some other articles on the addressed topics [48,32], the following introduces the principles of process and workflow modeling with the jABC in a hands-on fashion, from the point of view of a jABC user, not of a developer (Section 2), and briefly addresses the integration of domain-specific (remote) services into the framework (Section 3).

2 Process and Workflow Modeling in the jABC

The jABC [48,35] is a framework for service-oriented design and development that allows users to develop services and applications easily by composing reusable building blocks into (flow-) graph-like structures that are both formally sound and easy to read and to build. These building blocks are called *Service Independent Building Blocks* (SIBs) in analogy to the telecommunication terminology [43], and in the spirit of the Service-oriented Computing paradigm [38,28]. Their user-oriented nature is central for the *One-Thing Approach* [36], an evolution of the model-based lightweight coordination approach of [34] specifically applied to services. The SIBs are parameterizable, so that their behavior can be adapted depending on the current context of use. Furthermore, each SIB has one or more outgoing branches, which specify the successor(s) of the SIB. Which branch of the SIB is used is determined at runtime.

On the basis of a large library of such SIBs, the user builds models for the desired system in terms of hierarchical graphs called *Service Logic Graphs* (SLGs) [45]. In an SLG a SIB may represent a single functionality or a whole subgraph (i.e., another SLG), thus serving as a macro that hides more detailed

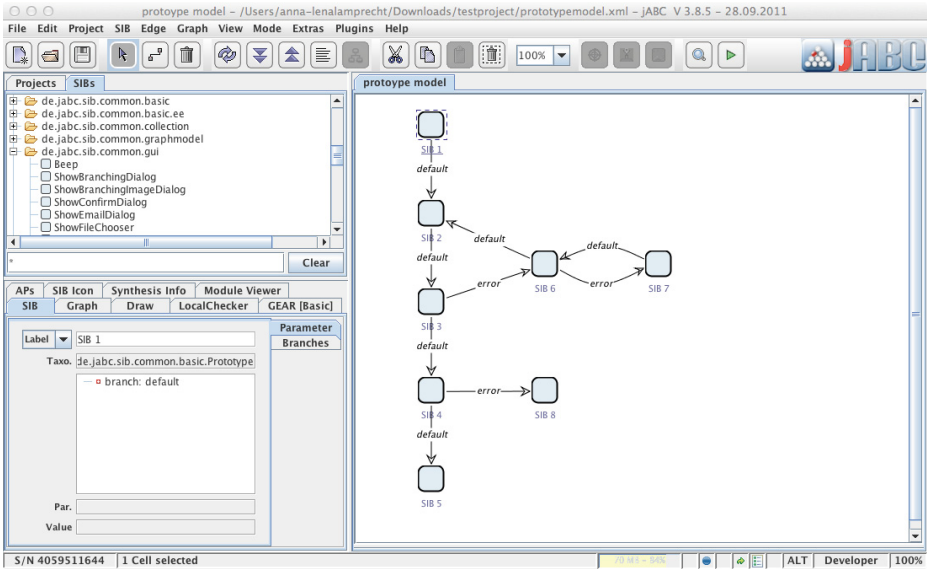

Fig. 1. Workflow modeling GUI of the jABC framework

process models. This feature grants a high reusability not only of components, but also of whole (sub-)models, within larger applications.

Figure 1 gives an impression of the user interface of the jABC framework: it provides a graphical high-level programming layer where no classical programming skills are required for workflow modeling. The available SIBs are listed in a browser (upper left), from where they can be dragged onto the drawing area (right), where the construction of the service compositions in terms of SLGs takes place. SIBs are connected by directed edges, which carry one or more labels (branches) to define the flow of control: the execution of a SIB determines which branch has to be taken to continue the computation at runtime. Data exchange between the individual SIBs is done via the so-called "ExecutionContext", a kind of shared memory where the SIBs can put and access data objects. Different inspectors (lower left) can be used for the detailed configuration of component and model parameters.

Semantically, SLGs are control flow graphs with fork/join parallelism, internally interpreted as Kripke Transition Systems [39]. The advantages of the SLG representation are in fact manifold: being a control-flow-oriented service definition formalism, it is adequate to support complex control structures as primitives. For example, iterations over lists or matrices are provided as SIBs in the environment. At the same time the data dependencies (which are secondary to the control flow) do not clog the representation: even large processes with complex data flows are still easily readable. Moreover, SLGs are at the same time mathematically analyzable objects: they are directed graphs, whose nodes (the SIBs) represent basic services and whose edges (their branches) define the flow

of control. They are thus amenable to the sophisticated formal analysis methods provided by modern computer science, like model checking [9,39,46] and model synthesis [33,47,49].

As such, the SLGs also provide a sound semantical basis for the representation of other workflow description formalisms (like, e.g., BPMN, BPEL, UML activity diagrams, or data flow graphs) and constitutes a lingua franca adequate for workflow design, analysis and verification. In fact, alternative formalisms are considered to be just different syntactic (visual) means for representing jABC models tailored for specific communities of users. In this context, we chose to privilege the abstract semantic view of the executable models over 'syntactic' sugar, and therefore use only the jABC notation.

The remainder of this section introduces different features of the jABC more concretely. Section 2.1 describes the "CommonSIBs", a set of SIBs for frequently occurring workflow steps that are provided with every jABC installation. The basic jABC is furthermore equipped with several plugins that exploit different semantic interpretations of the SLGs. Most useful when working with scientific processes and workflows (cf. [23]) are the plugins for definition of custom SIB taxonomies (Section 2.2), workflow execution (Section 2.3), the validation and verification of models (Section 2.4), and the model compilation and deployment (Section 2.5). The jETI plugin, which provides specific functionality for the integration and execution of remote services, is described in the next Section 3.

2.1 Common SIBs

The *Common SIBs* provide a collection of SIBs implementing functionality that is frequently needed during workflow development. This comprises, for instance, data handling, file management, basic user interaction, and different means for coordinating activities. Features like these are highly generic and thus provided by default in each jABC installation.

The following four CommonSIB subcollections are particularly relevant for the case studies of this book:

- *Basic SIBs* help work with the execution context, jABC's mechanism for data passing and communication between SIBs. SIBs for putting objects into the execution context, for accessing objects from the contexts, for evaluating conditions and for basic operations on character sequences are examples of commonly used Basic SIBs.
- *Collection SIBs* help deal with arrays, collections and maps. In addition to adding and removing objects from different kinds of collections, this library offers, for instance, SIBs that iterate over a collection object or perform specific operations like sorting a list.
- *GUI SIBs* support the user communication at runtime, e.g. showing dialogs for user interaction. This library contains predefined common dialogs for user interactions, such as displaying a message, selecting a file, and the input of login data.

– *IO SIBs* help perform file-related tasks. Reading and writing files are probably the most important and most frequently used functions in this library. Additionally, it provides functionality for browsing directories, executing console commands, and zipping and un-zipping files.

For concrete applications, it is typically necessary to have further SIBs that provide functionality required for the specific domain. Section 3 describes how to achieve this.

2.2 SIB Taxonomies

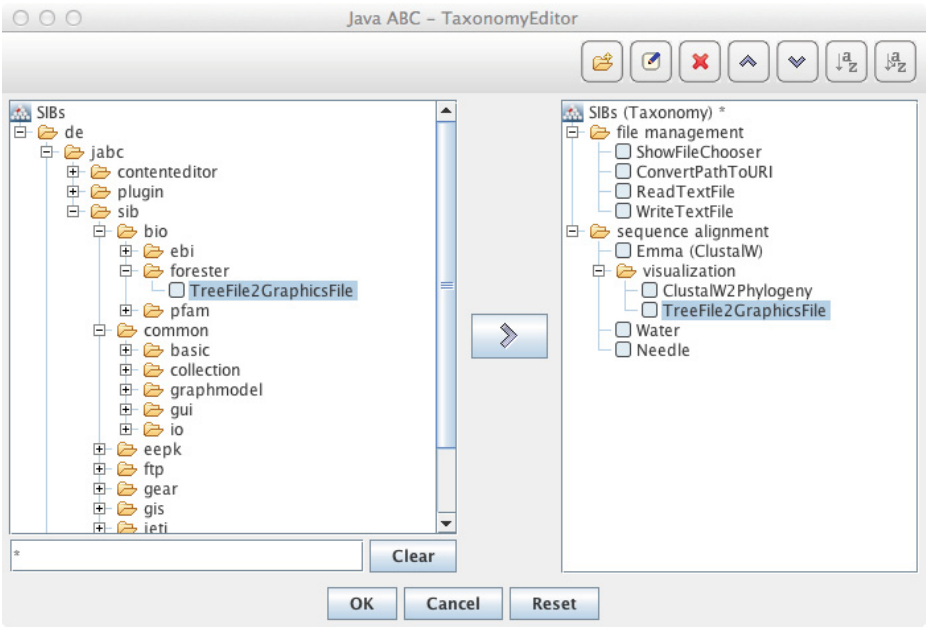

Fig. 2. Categorization of SIBs with the Taxonomy Editor

By default, the SIBs displayed in the SIB browser are sorted according to their location in the file system. The user can choose between a hierarchichal layout view (i.e. with the complete folder structure) and a flat layout view i.e. a package view). With the *Taxonomy Editor* plugin, a collection of SIBs can be (re-) arranged in arbitrary hierarchical categories. The users can introduce their own specific categories, for instance classifying them according to the area of functionality, and associate SIBs with these categories. This is exemplarily shown in Figure 2 for a group of file management and bioinformatics (sequence alignment) services. A SIB can potentially appear at various places in the taxonomy in case it belongs to different categories. Furthermore it does not have to be assigned with its real class name, but can get self-defined aliases.

2.3 SLG Execution

SLGs in the jABC are immediately executable when they consist of SIBs that implement the interfaces of the jABC's *ExecutionEnvironment*, which provides an API for starting, observing, and controlling the execution of models. It also provides *ExecutionContexts*, i.e. areas of shared memory which hold the data that is produced and consumed by the SIBs. The *Tracer* plugin is a comfortable graphical user interface to the ExecutionEnvironment, for instance offering means for model enactment and step-wise execution within the jABC.

As most workflow models are composed of Common SIBs (that are fully implemented and thus executable) plus a few extra SIBs that use algorithms or external available services, we informally talk about SLGs as "living models", because they most often can be directly executed after their composition.

Technically, the Tracer plugin is an interpreter for SLGs that uses the ExecutionEnvironment for enactment and the execution contexts as a means to implement the communication of data and information between the SIBs.

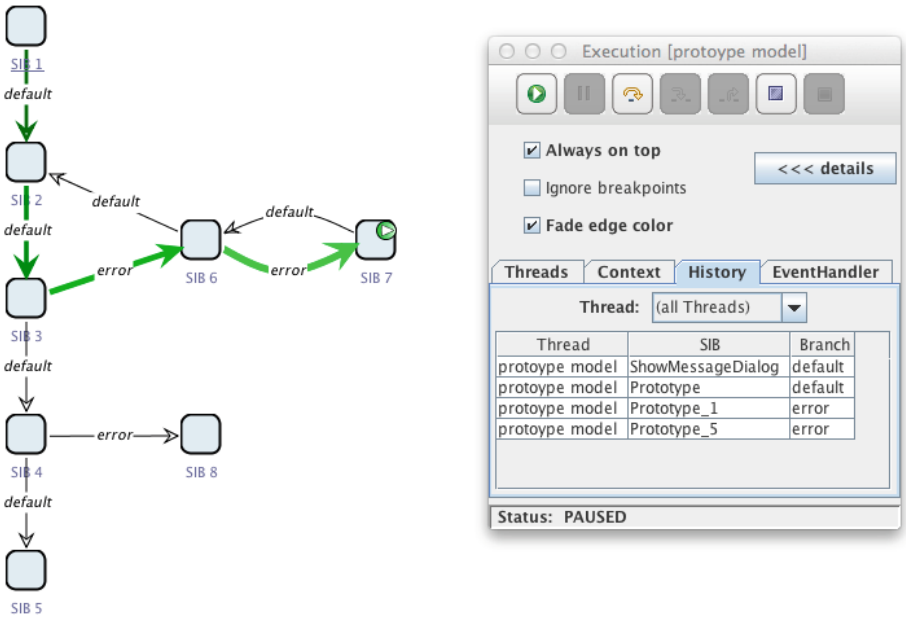

Fig. 3. Model Execution with the Tracer Plugin

Figure 3 shows an example model execution. It is steered by the buttons on the upper side of the Tracer window. The Tracer plugin allows for the overall or step-wise execution of the workflow, the latter being useful especially for debugging (breakpoints can also be set). The three tabs at the lower side of the Tracer window provide further information about the running threads, about

the data in the execution context, and, as shown in the figure, about the SIBs and branches visited so far in the current execution run.

2.4 SLG Validation and Verification

The *LocalChecker* and the *ModelChecker* verify constraints on single SIBs and on the whole model, respectively. Constraints embody knowledge about the application domain, the specific process under design, as well as well-formedness conditions. We talk here of incremental modeling of knowledge, as new properties or restrictions can be identified at any time, and added to the knowledge base of the verification environment (cf. [46,15]). Both local and model checking can be applied to the model continuously, enabling the process developer to recognize mistakes immediately, without need of model execution or code-level testing.

The LocalChecker plugin is responsible for testing the local well-formedness of SIBs. What is actually checked depends on the concrete implementation of the individual SIB. The standard checks provided by the plugin mainly deal with the configuration of the SIBs, such as that all the required branches must be present and labelled, and that all required parameters are configured. Type checking of the input parameters is another possible and useful extension that is often applied. Ideally, one could also use the local checking mechanism to implement design by contract and assume-guarantee styles of SIB and service descriptions.

Fig. 4. SIB markings according to the results of the LocalChecker

The plugin visualizes messages about the results of the checking in an inspector and additionally labels the SIBs on the canvas with icons indicating the local checking status (see Figure 4). If the configuration of a SIB passes all applied checks, a green circle with a checkmark is overlaid to its icon.

Otherwise, one of four labels denotes the severity of the problem:

- *informations* (blue circle with exclamation mark) denote that something might deviate from the normal usage, e.g., when no incoming branch is available and the SIB is not properly included into the process,
- *warnings* (yellow triangle with exclamation mark) refer to possible misconfigurations like unused branches,
- *errors* (red circle with a cross) can be caused by SIB configuration errors, like branches that have no target or required parameters that are not set,
- *fatal errors* (red circle with a green bar) are due to exceptions that occur in the SIB's implementation and cannot be handled by the user.

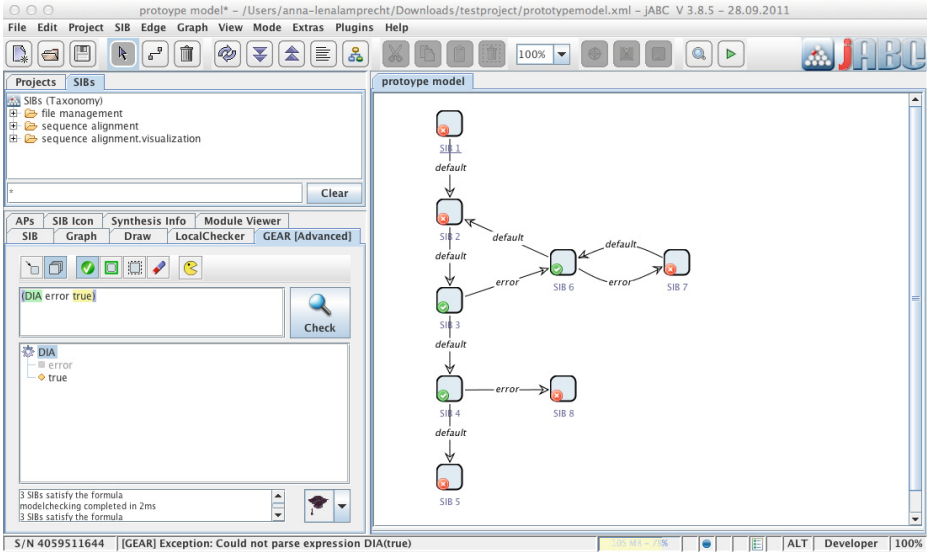

Fig. 5. Global Verification with the GEAR Model Checking Plugin

Complementary to the SIB-level checks of the LocalChecker, the *GEAR* model checking plugin [6,7] is used for the formal verification of global properties of an entire SLG. When the GEAR plugin is loaded, two additional tabs appear in the inspector pane (lower left corner of the GUI) where the properties of the SIBs and the constraints for the model, respectively, are created and managed. The constraints concern the correctness and well-formedness of the workflows. They are expressed as temporal formulae of one of several temporal logics, most frequently in CTL [9, Chapter 4] or the modal μ-calculus [9, Chapter 7]. Users can specify sets of rules that the workflow has to obey, and may concern policies, or best practices, or constraints. When a formula is checked, the satisfying SIBs in the SLG are marked with a green checkmark, the others with red crosses (exemplarily shown in Figure 5).

Frequently, model constraints are safety properties that should invariantly hold for the whole process, that is, which should be satisfied at all SIBs. In these cases every unsatisfying SIB indicates a constraint violation, that is, a situation, where the model must usually be corrected/improved. However, there are often also cases in which the model is correct and the constraints need some refinement or some reformulation, in order to better reflect the knowledge about the specifics of the application and its domain [46,15].

2.5 SLG Compilation and Deployment

The jABC is conceived not only as a process modeling tool, but as a complete software development framework, hence it provides different means for model compilation to program code or to a variety of other representations. This way,

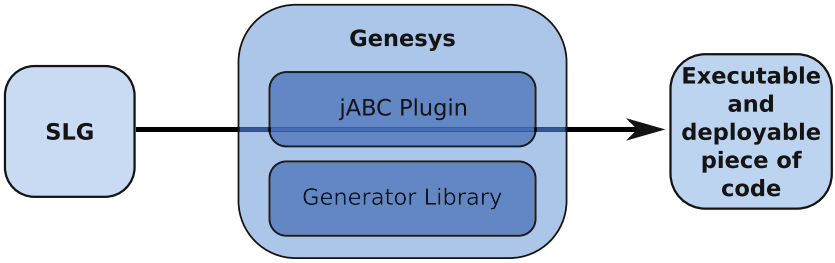

Fig. 6. Model compilation with Genesys

the created processes can be stored, distributed, and executed also in formats and environments independent from the actual SLGs and the jABC framework. As sketched in Figure 6, the *Genesys* Framework [19,17,16] offers an extensible set of code generators for different target languages, which can be used to compile any executable SLG models into an executable and deployable piece of code that can be run independently of the jABC.

Currently, the Genesys code generator library contains predefined generators for Java, Java HTTP Servlets, and complex domain-specific SIBs. The predefined code generators are themselves available both as compiled Java classes and in the form of SLGs, since they have been assembled from an appropriate set of code generation SIBs provided by the library themselves. In this spirit, specialized code generators for further target formats can easily be defined by building appropriate jABC models, mostly starting with a preexisting one and modifying and enhancing it to suit the new needs. The Genesys plugin to the jABC then manages the access to the underlying code generation library by providing inspectors and dialogs for using and creating code generators within the jABC.

3 Service Integration: jETI, REST and Web Services

While the basic jABC, the Common SIBs and the plugins introduced in the previous section essentially provide domain-independent functionality, domain-specific workflow building blocks are typically required when realizing concrete processes. Tailoring the framework to a particular domain involves in fact the integration of (remote) specialist services that are required in the course of the particular processes and workflows. Obviously, the extent and richness of the tool repository plays a crucial role in the success of the platform: the benefit gained from our experimentation and coordination facilities grows with the amount and variety of integrated algorithms, services, and software tools.

While the case studies reported in this book concern scientific workflows, the jABC framework is not limited to this particular application domain: by providing an appropriate set of building blocks, any application domain can be covered. Examples are the user-level definition of telephony services [43], model-based code generation [19,17,16], and the automated completion of model

sketches [26,40]. Thus easy and powerful integration facilities are essential to the capability of jABC to cover a large selection of domains.

In the context of the jABC framework, integration of services mostly means to provide SIBs that encapsulate the (remote) invocation of a particular service or the access to a specific resource. Technically, SIBs use just a few lines of Java code to encapsulate the desired functionality. This functionality can either be available locally, such as in case of access to local services (in the easiest case in the form of Java APIs) as well as be provided by remote tools, for instance via web services. In many cases, appropriate web service interfaces are already available and can be used by the components as described in Section 3.1. Often, however, tool developers lack the time or technical knowledge to set up proper web services for their programs. In many of these cases, the jETI technology described in Section 3.2 can help to integrate them as services into the jABC.

3.1 Integrating Web Services

Web services [8], as defined by the World Wide Web Consortium (W3C), provide programmatic interfaces for application-to-application communication via the internet. Technically, web services are a platform- and language-independent technology for distributed computation useful to create client-server applications. This contrasts other RPC technologies (like CORBA, Java RMI, or EJB) that were not platform- and language-independent.

In the classical web service architecture, the application interfaces are characterized by means of an XML-based language, the *Web Service Description Language (WSDL)*, which provides elements for describing functions, data, data types and protocols of a web service. Essentially, the remotely accessible operations are defined along with their parameters and return values. From these WSDL documents, so-called *stubs* can be automatically generated for different target languages for both the client and the service provider side.

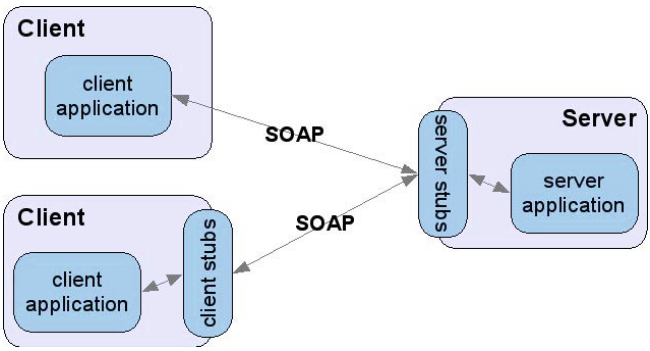

Fig. 7. Schema of client-server interaction using SOAP-based web services

As illustrated in Figure 7, server-side stubs are responsible at the provider's side for the communication with the clients via the network. Client applications usually send requests and receive responses in the form of SOAP messages that are transmitted via HTTP. For web services with simple input data types, these messages can be built directly by the client. In case of services with more complex input data types, as common in many scientific application domains, it is more convenient to generate client-side stubs from the WSDL description and let the client make use of these pre-defined communication interfaces.

Consequently, when creating clients for existing web services, the first step is usually the generation of the client-side stubs from the WSDL document. This can be done in many ways, for instance by the `wsimport` program included in current JDKs. This tool generates a source code package containing Java classes for all required data types and operations specified in the given WSDL document.

The next step is then to implement the client application itself. In the case of the jABC, the clients are the jABC's process building blocks, the SIBs. Different from other common service handling frameworks, the SIBs have an invocation-level granularity [29]. They encapsulate a certain functionality, which in this case is the invocation of a single web service operation. True to our philosophy of behavioral, workflow-oriented service consumption, no matter how complex the service itself is, with the SIBs we access it invocation by invocation. If we need different operations, we create different SIBs, one for each type of specific operation invocation. Therefore, the SIB's execution code simply has to:

1. initialize a connection to the web service by defining a new service object,
2. build the request object including all parameters,
3. call the service with the set parameters,
4. save the response in the execution environment.

As detailed in [27], for the case studies of this book we have integrated this way a number of external web service resources, namely from the EBI (European Bioinformatics Institute) [42,21,12] and from the the CSISS (Center for Spatial Information Science and Systems)[1].

In the last years, REST-style web services [10] have become increasingly popular. They enable the client application to easily request information and access functionality by simply calling a URL which includes all the necessary parameters. Such an URL typically looks something like:

```
http://[website-URL][service-name]?[parameter1]&[parameter2][...]
```

where the first part consist of the website's basic URL and the name of the concrete service, followed by a list of parameters to be set by the calling application. Accordingly, a SIB that calls a REST service just needs to construct an URL with the required parameters, retrieve the data that it points to using HTTP GET, and put them into the execution context.

As detailed in [27], for the case studies of this book we have integrated this way a number of external REST resources: the Pfam services [11], the GeoPlugin services [3] and the Gisgraphy services [2]. Furthermore (also described in [27]),

we provided a small collection of generic REST SIBs which can be configured to access arbitrary REST services simply by providing the complete URL.

3.2 Electronic Tool Integration

The jETI (Electronic Tool Integration) platform [44,32] provides means for making file-based Java or command line applications accessible via the internet. In contrast to other remote integration techniques, jETI has only few system requirements and is easy to set up. No further coding is needed for the integration, since all configurations can be done via the HTML interface of the jETI Tool Configurator. Integration of services by means of jETI is convenient especially in the case of legacy applications and whenever else the setup of a classical web or REST service is not adequate or feasible.

Fig. 8. Schema of Client-Server Interaction Using the ETI Technology

In jETI, as sketched in Figure 8, the service provider maintains a server that accesses (a collection of) applications on the one side, and on the other it provides an interface to the internet. At runtime, the server receives service requests from a client (in our case an SLG) and forwards them to the actual tools, then collects the results, builds adequate response messages and sends them back to the client. Similar to web services' WSDL descriptions, relevant request parameters as well as the actual calls used by the jETI server to execute the tools are defined in an XML file. This information is also used by the jETI server to automatically generate the SIBs for the defined services.

As detailed in [27], for the case studies of this book we have integrated this way the Generic Mapping Tools (GMT) [51,50].

4 Conclusion

In this chapter, we have described the main features of the jABC framework for its use as a graphical tool for modeling scientific processes and workflows. Associated technologies and plugins provide means for a rich variety of supporting functionality, such as remote service integration, taxonomic service classification, model execution, model verification, and model compilation. As such, the jABC is a comprehensive environment for service-oriented modeling and design according to the XMDD (eXtreme Model-Driven Design) paradigm. With

the sophisticated execution context and hierarchy concepts, the graphical process definition in the jABC is in fact as powerful as application development in a classical programming language. Its way of "orchestration without programming" is, however, directly accessible by non-IT experts, unlike the scripting or programming-based approaches to workflow management that are still most common today.

References

1. CSISS/GMU Geospatial Web Services, `http://geobrain.laits.gmu.edu/grassweb/manuals/index.html` (last accessed September 9, 2013)
2. Free opensource geocoder and webservices for geonames and openstreetmap data, `http://www.gisgraphy.com/` (last accessed September 9, 2013)
3. geoPlugin to geolocate your visitors, `http://www.geoplugin.com/` (last accessed September 9, 2013)
4. Bajohr, M., Margaria, T.: MaTRICS: A Service-Based Management Tool for Remote Intelligent Configuration of Systems. Innovations in Systems and Software Engineering (ISSE) 2(2), 99–111 (2005)
5. Bajohr, M., Margaria, T.: High Service Availability in MaTRICS for the OCS. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation. CCIS, vol. 17, pp. 572–586. Springer, Heidelberg (2008)
6. Bakera, M., Margaria, T., Renner, C., Steffen, B.: Verification, Diagnosis and Adaptation: Tool-supported enhancement of the model-driven verification process. In: Revue des Nouvelles Technologies de l'Information (RNTI-SM-1), pp. 85–98 (December 2007)
7. Bakera, M., Margaria, T., Renner, C., Steffen, B.: Tool-supported enhancement of diagnosis in model-driven verification. Innovations in Systems and Software Engineering 5, 211–228 (2009)
8. Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., Orchard, D.: Web Services Architecture. W3C Working Group Note (February 2004), `http://www.w3.org/TR/ws-arch/` (last accessed May 5, 2013)
9. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. The MIT Press, Cambridge (1999)
10. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. PhD thesis, University of California, Irvine (2000)
11. Finn, R.D., Mistry, J., Tate, J., Coggill, P., Heger, A., Pollington, J.E., Gavin, O.L., Gunasekaran, P., Ceric, G., Forslund, K., Holm, L., Sonnhammer, E.L.L., Eddy, S.R., Bateman, A.: The Pfam protein families database. Nucleic Acids Research 38(suppl 1), D211–D222 (2010)
12. Goujon, M., McWilliam, H., Li, W., Valentin, F., Squizzato, S., Paern, J., Lopez, R.: A new bioinformatics analysis tools framework at EMBL-EBI. Nucleic Acids Research 38(Web Server issue), W695–W699 (2010)
13. Huhns, M.N., Singh, M.P.: Service-Oriented Computing: Key Concepts and Principles. IEEE Internet Computing 9, 75–81 (2005)
14. Hörmann, M., Margaria, T., Mender, T., Nagel, R., Steffen, B., Trinh, H.: The jABC Approach to Rigorous Collaborative Development of SCM Applications. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation. CCIS, vol. 17, pp. 724–737. Springer, Heidelberg (2008)

15. Jonsson, B., Margaria, T., Naeser, G., Nyström, J., Steffen, B.: Incremental requirement specification for evolving systems. Nordic J. of Computing 8, 65–87 (2001)
16. Jörges, S.: Construction and Evolution of Code Generators. LNCS, vol. 7747. Springer, Heidelberg (2013)
17. Jörges, S., Lamprecht, A.-L., Margaria, T., Schaefer, I., Steffen, B.: A Constraint-based Variability Modeling Framework. International Journal on Software Tools for Technology Transfer (STTT) 14(5), 511–530 (2012)
18. Jörges, S., Margaria, T., Steffen, B.: Genesys: service-oriented construction of property conform code generators. Innovations in Systems and Software Engineering 4(4), 361–384 (2008)
19. Jörges, S., Steffen, B., Margaria, T.: Building code generators with genesys: A tutorial introduction. In: Fernandes, J.M., Lämmel, R., Visser, J., Saraiva, J. (eds.) Generative and Transformational Techniques in Software Engineering III. LNCS, vol. 6491, pp. 364–385. Springer, Heidelberg (2011)
20. Kubczak, C., Margaria, T., Steffen, B., Nagel, R.: Service-oriented Mediation with jABC/jETI (2008)
21. 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)
22. Lamprecht, A.: Orchestration of Heterogeneous Bioinformatics Processes in Bio-jETI as a Service Integration, Design, and Provisioning Platform. Master's thesis, Georg-August-Universität Göttingen (August 2007)
23. Lamprecht, A.-L.: User-Level Workflow Design. LNCS, vol. 8311. Springer, Heidelberg (2013)
24. 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)
25. Lamprecht, A.-L., Margaria, T., Steffen, B.: Bio-jETI: a framework for semantics-based service composition. BMC Bioinformatics 10(Suppl 10), S8 (2009)
26. Lamprecht, A.-L., Naujokat, S., Margaria, T., Steffen, B.: Synthesis-Based Loose Programming. In: Proc. of the 7th Int. Conf. on the Quality of Information and Communications Technology (QUATIC 2010), Porto, Portugal, pp. 262–267 (September 2010)
27. Lamprecht, A.-L., Wickert, A.: The Course's SIB Libraries. In: Lamprecht, A.-L., Margaria, T. (eds.) Process Design for Natural Scientists. CCIS, vol. 500, pp. 30–44. Springer, Heidelberg (2014)
28. Margaria, T.: Service is in the Eyes of the Beholder. IEEE Computer (November 2007)
29. 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)
30. Margaria, T., Karusseit, M.: Community Usage of the Online Conference Service: an Experience Report from three CS Conferences. In: Proceedings of the IFIP Conference on Towards The Knowledge Society: E-Commerce, E-Business, E-Government, Deventer, The Netherlands, pp. 497–511. Kluwer, B.V (2002)
31. 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)

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. Margaria, T., Steffen, B.: Backtracking-free design planning by automatic synthesis in *METAFrame*. In: Astesiano, E. (ed.) ETAPS 1998 and FASE 1998. LNCS, vol. 1382, pp. 188–204. Springer, Heidelberg (1998)
34. Margaria, T., Steffen, B.: Lightweight coarse-grained coordination: a scalable system-level approach. Software Tools for Technology Transfer 5(2-3), 107–123 (2004)
35. Margaria, T., Steffen, B.: Agile IT: 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, Berlin (2009)
36. 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)
37. Margaria, T., Steffen, B.: Service-Orientation: Conquering Complexity with XMDD. In: Hinchey, M., Coyle, L. (eds.) Conquering Complexity, pp. 217–236. Springer, London (2012)
38. Margaria, T., Steffen, B., Reitenspieß, M.: Service-oriented design: The roots. In: Benatallah, B., Casati, F., Traverso, P. (eds.) ICSOC 2005. LNCS, vol. 3826, pp. 450–464. Springer, Heidelberg (2005)
39. Müller-Olm, M., Schmidt, D., Steffen, B.: Model-Checking - A Tutorial Introduction. In: Proceedings of the 6th International Symposium on Static Analysis (SAS 1999), pp. 330–354 (1999)
40. Naujokat, S., Lamprecht, A.-L., Steffen, B.: Loose programming with PROPHETS. In: de Lara, J., Zisman, A. (eds.) Fundamental Approaches to Software Engineering. LNCS, vol. 7212, pp. 94–98. Springer, Heidelberg (2012)
41. Neubauer, J., Margaria, T., Steffen, B.: Design for Verifiability: The OCS Case Study. In: Formal Methods for Industrial Critical Systems: A Survey of Applications, ch. 8, pp. 153–178. Wiley-IEEE Computer Society Press (March 2013)
42. 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 (2005)
43. Steffen, B., Margaria, T.: META*Frame* in practice: Design of intelligent network services. In: Olderog, E.-R., Steffen, B. (eds.) Correct System Design. LNCS, vol. 1710, pp. 390–415. Springer, Heidelberg (1999)
44. Steffen, B., Margaria, T., Braun, V.: The Electronic Tool Integration platform: concepts and design. International Journal on Software Tools for Technology Transfer (STTT) 1(1-2), 9–30 (1997)
45. Steffen, B., Margaria, T., Braun, V., Kalt, N.: Hierarchical Service Definition. Annual Review of Communications of the ACM 51, 847–856 (1997)
46. Steffen, B., Margaria, T., Claßen, A., Braun, V.: Incremental Formalization: A Key to Industrial Success. Software - Concepts and Tools 17(2), 78–95 (1996)
47. Steffen, B., Margaria, T., Freitag, B.: Module Configuration by Minimal Model Construction. Technical report, Fakultät für Mathematik und Informatik, Universität Passau (1993)
48. 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)

49. Steffen, B., Margaria, T., von der Beeck, M.: Automatic synthesis of linear process models from temporal constraints: An incremental approach. In: ACM/SIGPLAN International Workshop on Automated Analysis of Software, AAS 1997 (1997)
50. Wessel, P.: The GMT Home Page, `http://gmt.soest.hawaii.edu` (last accessed January 3, 2013)
51. Wessel, P., Smith, W.H.F.: Free software helps map and display data. EOS Trans. Amer. Geophys. U. 72(41) (1991)