Lessons Learned

Anna-Lena Lamprecht, Alexander Wickert, and Tiziana Margaria

Chair for Service and Software Engineering, Potsdam University,
Potsdam, D-14482, Germany
{lamprecht,awickert,margaria}@cs.uni-potsdam.de

Abstract. This chapter summarizes the experience and the lessons we learned concerning the application of the jABC as a framework for design and execution of scientific workflows. It reports experiences from the domain modeling (especially service integration) and workflow design phases and evaluates the resulting models statistically with respect to the SIB library and hierarchy levels.

Keywords: domain modeling, service integration, workflow design, service usage statistics.

1 Introduction

The gallery of scientific workflow applications in this book is so far the largest project where non-IT users (in this case Master students with biology and geography backgrounds) autonomously designed complete running workflows for their particular purposes with the jABC. The workflows in previous studies, also predominantly in the bioinformatics and geoinformatics domains (such as [36,19,29,27,12,8]), were in contrast discussed and modeled together with project partners from the field of application. These partners provided valuable input and knowledge, but hardly took part in the actual implementation. For instance, GeneFisher-P [29] and Flux-P [12] have been developed in close collaboration with their later users and strongly according to their needs, resulting in a set of readily configured workflows for the identified processes. The users sometimes built variants of existing workflows, but usually did not design entirely new workflows on their own, as it is the case for the projects presented here.

In general, we can distinguish three roles when working with scientific workflows:

1. The **domain modeler** prepares the initially domain-independent workflow framework for use in a particular application domain. Mostly this concerns the definition of the domain-specific workflow building blocks, deciding which service libraries should be integrated, which SIBs should be available, and how they should be integrated and organized within the jABC's SIB palettes.
2. The **service integrator** actually provides the defined SIBs by integrating the corresponding services into the framework. The service integrator can be the same person as the domain modeler, but also someone else. Service integration is a quite technical task, often requiring a good amount of programming skills, so that we recommend this role to be filled by a person with correspondingly solid IT expertise.

3. The **workflow designer** then works with the prepared domain-specific workflow framework and designs the workflows for his needs. This is mainly done by selecting, configuring and connecting adequate workflow building blocks so that they perform the intended computations. This kind of workflow design requires extensive knowledge about the application domain, but no classical programming skills, so that this role is filled by the application experts, in our case the students.

In fact, we observed this clear separation between domain modeling, service integration and workflow design in the scope of the course:

1. In the beginning, the **domain modeling** was performed as a joint effort by lecturers and students of the course. While the lecturers provided ("pushed") a number of SIBs from previous projects and services they considered useful also for the new projects, the students analyzed the functional requirements of their projects and asked ("pulled") for new SIB libraries.
2. The lecturers and tutors of the course took then care of **integrating** the identified services and functionality, and provided the missing SIBs. The few students who actually had the required technical expertise integrated the services they needed themselves.
3. The SIB libraries could then simply be used by **workflow designers**, i.e. the students. They realized their workflows without having to deal with the technical details of the underlying services implementations.

Following the above distinction of roles and phases, in Sections 2, 3 and 4 this chapter reports on lessons learned and experiences gained during the domain modeling, service integration and workflow design in the scope of this course, respectively. Focusing on the actual results of the project work, Section 5 evaluates the created workflow applications of this book in terms of some usage metrics (used SIBs, reused services) and workflow structures. Finally, Section 6 concludes this chapter and also the introductory part of this book.

2 Domain Modeling

In the context of the applications described in this book, domain modeling comprises the choice and integration of domain-specific services into the jABC framework in order to prepare it for being used for workflow design. We start with a closer look at two challenges the domain modeler typically faces: finding services providing the required functionality, and finding services suitable for being accessed from a workflow environment.

2.1 Shopping for Service Functionality

Finding tools and services that provide the "right" functionality required for realizing the intended workflows is a sometimes quite time-consuming research task, which is typically performed by web search and literature study. Sometimes one comes across curated service directories for particular scientific areas (like, e.g., the BioCatalogue [10] for life science web services), but they cover only a small part of the actually available functionality, so that other sources of information need to be considered to find adequate services. Accordingly, web search and literature study to find services with functionality adequate for the envisaged applications was a major research task in the beginning of the students' project work.

2.2 Service Access

In addition to finding services with adequate functionality, finding services that are also technically suitable for workflow integration means in the first place to find services that allow for some kind of programmatic access to their functionality. This can be a web service interface, an application programming interface (API), or command line options that execute a tool in "headless" operation mode (i.e. without requiring user interaction via a graphical or other user interface). Many available tools and programs, however, do not fulfill this crucial requirement. Desktop programs and web applications are unfortunately often implemented to require interaction with a human user for their execution. Usually this means that there is no (easy) way to strip the GUI and use them as embedded services in a workflow.

For example, Henriette Sens, author of "Web-Based Map Generalization Tools Put to the Test" [55], was searching for a map generalization tool to include as a service in her workflow, but could not find a single one that could be operated in headless mode as described above. The Flash-based "Mapshaper" web application[1] that she finally used can be started in a browser from the command line (and thus with the `ExecuteCommand` SIB), but is from then on interactive, so that the user can only produce and store a result manually, and then resume the automatic workflow at the respective point.

In general, also licensing issues and availability constraints can be relevant here. That is, services might only be available after signing particular agreements, if using the resources for particular purposes, after paying some access fee, or if being member of a particular organization. While we have experienced obstacles in this regard in other courses, they were fortunately not present for the services used in the projects described in this book.

3 Service Integration

The service integration is done by someone with adequate technical and programming skills. It concerns implementing the access to services, their encapsulation into SIBs and their basic testing, while the development of more complex applications is subsequently done by the application experts that design the actual

[1] `http://www.mapshaper.org`

workflows. Major issues here are usually the technicalities of the service interface, the lack of proper service documentation, and changing service interfaces, as explained and illustrated in Sections 3.1, 3.2 and 3.3, respectively.

3.1 Service Interfaces

Unfortunately, even services which principally allow for programmatic access as described in Section 2.2 cannot always easily be integrated (cf., e.g. [21,20,34]). Different technical obstacles may be identified by the SIB programmer when working on the integration. Then it has to be decided whether the problems can be solved and if solving them is worth the effort, or if an alternative service is available. For example, when we created the twitter-sibs (for sending Twitter [5] messages), we first tried to use the Twitter REST-API [6] directly. After spending much time unsuccessfully trying to handle its authentication protocol, we dropped this plan and decided to use instead the simpler Twitter4J [7] library, although this meant to have no image upload functionality available in our SIBs.

More common is, however, that the programmer simply faces difficult-to-use service interfaces. For instance, the EBI web services [48,23] define several own, complex data types that have to be used by the client application, and furthermore operate most services in asynchronous mode, which requires active polling of the service results. The CSISS services [1] use URLs instead of standard file names for defining input and output data. This makes data transfer difficult, as all involved data has to be accessible by the URLs. All these examples do not prevent the service from being accessed by a SIB, but hamper the actually straightforward integration process.

3.2 Incomplete or Missing Documentation

When working with services of all kinds, one soon realizes that exact descriptions of parameters are very important. If one has incomplete lists of possible values of parameters and comprehensible explanations, it is usually difficult for an external user to integrate an unknown service (cf. also [34]). In the first place, the lack of proper documentation makes life difficult for the programmer who creates the SIBs. However, it is often a problem also for the workflow designer, who usually reads pieces of documentation that are propagated from the underlying levels. Typically, the programmer does not write new domain-specific documentation.

Among the SIB collections created for the students' projects, the CSISS and EBI web services (used in [55,47,50] and [49,11,53,33], respectively) are two examples of service collections with documentation in need of improvement. In fact, we often executed these services with different test data sets (which in the case of CSISS were provided by the providers on request) in order to learn from experience how they actually work, since this was not completely evident from the available documentation alone.

3.3 Changing Interfaces

As discussed elaborately in [25], scientific application domains are characterized by the long-term availability of their basic computational components. At the same time their concrete service interfaces are subject to frequent changes. While it is clear that this is a typical and unavoidable phenomenon when working with public third-party services, it is still a recurring source of interruption during workflow execution and requires code repairs.

For instance, currently many major service providers in bioinformatics are abandoning their SOAP-based web service interfaces and follow the general trend towards using REST-style interfaces. Consequently, SIBs that have been implemented for accessing, e.g., the DDBJ and EBI web services [2,48,23] have to be changed accordingly at some point, to follow the technology shift on the provider's side. Similarly, we experienced different changes of the Google Maps API for location-based mapping [3], so that the corresponding SIBs have been already updated several times. Luckily, as also discussed in detail in [25], the SIB interfaces typically do not change during these updates, so that on the workflow level this change of underlying technology is usually not perceptible at all.

More severe is the case where services simply stop operating. An example here are the EBI's SoapLab [4] services. They used to provide web service access to the tools of the EMBOSS [51] suite, but were terminated in February 2013. For several of the EMBOSS tools, alternative services are provided in the "standard" EBI web service collection [48,23], but do not cover the entire functionality previously available. Another example from the bioinformatics domain is the DDBJ Web API [2], which used to provide access to a number of molecular biology databases and standard query and analysis tools. Its operation has been "temporarily suspended" in February 2012, and not been resumed since then. Consequently, contrary to what was anticipated when we prepared the course, the DDBJ services could not be used in the students' projects and alternatives had to be identified.

4 Workflow Design: The jABC Framework

The experiences with the students from our course and also with project partners using the jABC framework, suggest that it provides an adequate level of abstraction from classical programming: after only a short introduction, non-IT users are quickly able to use it to design and manage scientific workflows according to their needs. In fact, the workflows presented in this book were built by the students mostly autonomously with only little support by the lecturers. As for other examples, the GeneFisher-P [29] and FiatFlux-P [12] workflows have been frequently adapted by their users (biology diploma and bio-engineering PhD students without specific computer science education) according to changing experimental setups.

The feedback on the jABC framework provided by some students gave us more detailed insights about the user's perception of working with the platform. First and foremost, it confirmed our expectation and impression that handling the jABC as a tool is or quickly becomes intuitive. Exemplarily, Christine Schütt, author of "Identification of differentially expressed genes" [54], reports: *"At the beginning it took some time to become familiar with its operating principle. This was of course not such a big surprise for it takes always a bit of work to learn the principles of a program or programming language. [...] Apart from this point no real difficulties occurred during the implementation. [...]"*

In fact, our experience with new jABC users shows that the graphical workflow modeling as such hardly requires any explanation. Dragging and dropping SIBs from the SIB browser to the canvas, and connecting them with labeled branches according to the flow of control, as well as the configuration of simple parameters and the adaptation of existing workflows can usually be learned in less than 30 minutes. We have seen this many times in 45-minute workshops for high school students that we gave at open days at our universities, where the students were shown how to use jABC models to develop strategies for the well-known Connect-Four game [9]. Similarly, enriching the models with custom SIB icons and draw elements and making use of plugins like the LocalChecker and the Tracer can be typically be learned within another hour of instruction.

Some particular concepts and plugins have to be explained and practiced more elaborately before they become useful. This concerns the ExecutionContext with its different scopes and the ContextKeys and ContextExpressions that are required for controlling the flow of data. For instance, Judith Reso, author of "Protein classification workflow" [49], reports in this regard that: *"[...] In the beginning it was difficult to find out how to access local variables and use them as input for another tool. [...]"*. As another example, to make proper use of the model checking plugin, the users need to have at least a basic idea about propositions and logics before they can formulate constraints and use these capabilities of the framework.

The feedback by the students also shows that intuition is subjective: some features that appear intuitive to one user can be quite confusing for another. For instance, with regard to debugging with the Tracer plugin, users' opinions differed considerably. Christine Schütt, for instance, thinks that *"A real good feature of jABC is the animated execution of the workflow model with theTracer plugin. With this it is possible to follow each single step in a debugging style, to see what single SIBs do and to find potential sources of errors even faster."*

Similarly, Monika Lis, author of "Workflow for phylogenetic tree construction" [33], notes on the Tracer: *"A very good option is the Tracer and its execution controller, which enables you to go through your workflow stepwise and look if every single step of your workflow works well and which results it provides."*, but then adds: *"However, the more SIBs have been executed, the more confusing the status overviews including the error messages. That is, in my opinion, the major disadvantage of jABC: The error messages which are, especially for beginners, not easy to interpret. [...]"*.

Problems like this are common, however not caused the by actual jABC framework, but rather by SIBs that are poorly documented or erroneous, or simply due to not knowing which SIBs are available. For instance, Monika Lis describes problems similar to those sketched in Section 3.2 (Incomplete or missing documentation) that she experienced when trying to use new SIBs in her workflow: *"In my case there have been some difficulties while constructing the workflow. First, it was not always possible to use all parameters, because some of them seemed to be faulty and led to breaking up the workflow. Sometimes it was necessary to mess around the parameters if one wanted to see which of them causes in errors, because this was not always clear."*

Janine Vierheller, author of "Exploratory data analysis" [60] describes problems with finding adequate SIBs for a workflow: *"The implementation of the workflow with jABC was not always intuitive, because you start with the functions you already know, which are some Common SIBs. If you search for a specific SIB of which you think it might exist, you have to have a clear idea how this could be named."*

The last two comments point to a common problem that users of workflow management systems face: while the graphical workflow modeling facilities make it *syntactically* easy for the user to design the application, they do not help the user *semantically.* That is, the user is still responsible for identifying services with adequate functionality, and for understanding the technical details of the inputs and outputs so that he/she can connect them correctly. Semantics-based approaches to service discovery that make use of domain-specific ontologies for the description of services and data types, can help in this regard by reducing the gap between the domain language of the user and the technical language of the service infrastructure.

In fact the jABC provides technologies for using such methods, for instance with the plugin for semantically supported service selection described by [61] and with the PROPHETS plugin for semantically supported (semi-) automatic service composition [30,39]. Since their application crucially depends on adequate domain models (consisting of domain-specific ontologies, service and type descriptions in terms of these ontologies, and possibly also different kinds of constraints), which were not available and which could not be built in the time we had, we did not use them in the course. We plan, however, to include this topic in future editions of this course.

Finally, some comments pointed to usage details that sometimes cause confusion in the beginning and can be quite annoying, but are not related to the modeling approach of the jABC and are actually caused by details of the current implementation. For example, Monika Lis reports: *"In general, jABC is a good tool for connecting different services, but the use of jABC takes some getting used to. For example it is strange to need to press enter, if you want to change parameters. You need to forget it a hundred times, until you remember to press it. [...]"*

The overall impression of the jABC as a modeling tool was very positive. The students were able to get productive within hours and did not need to acquire classical programming skills, yet were successful in creating running workflows of medium complexity largely on their own. The encountered difficulties had mostly to do with the more programming-like characteristics, e.g. the management of variables, the use of the context and context expressions, and the issue of poor documentation of the inside error messages delivered by services and routines they used as SIBs. The idea of a typed variable as a placeholder for a value, that can be put there and subsequently looked up is hard to eliminate or simplify further. However, the other two issues can be tackled, although at different levels. The new jABC4 [41] already used in a workflow project in the field of Cachexia research [35] and the jABC5 [40,42] that is in preparation include an integrated and much simplified handling of data, not requiring an explicit management of the context anymore. The poor documentation and error treatment will gradually improve with the more widespread reuse of such components and routines as services, under the pressure of the crowd of workflow developers.

5 Workflow Metrics

In this book, we present 15 scientific workflow projects. 6 in the area of bioinformatics and 9 in the area of geovisualization. To evaluate the scientific jABC workflow models of these students' projects, we created *jABCstats*, a library of jABC workflows that computes metrics of jABC workflows. It is completely modeled with the Common SIBs that come with every jABC distribution. For a detailed explanation of jABCstats the reader is referred to [62]. In the following, Section 5.1 explains briefly the analyses carried out by jABCstats, before Section 5.2 discusses the actual findings for the projects presented in this book.

5.1 jABCstats: Empirical Analysis of jABC Workflows

jABCstats currently provides functionality for the analysis of jABC workflow models with regard to workflow and project sizes, hierarchy levels, control-flow workflow patterns and service usage. For the workflow and project sizes jABCstats simply counts the total number of SIBs in the workflows and the number of workflow files in the projects, respectively. To determine the number of hierarchy levels it uses a simple recursive counting mechanism. For the analysis of the modeling patterns used in the workflows, jABCstats evaluates the control-flow model structure of the workflows and counts the sequence, conditional branching, simple merge, fork, join and loop structures (as described in [59]).

For assessing the service usage, jABCstats analyzes the models with regard to the occurrences of individual SIBs and for each hierarchical level of its complete namespace, extracting this information from the XML file that stores the jABC model. For example, let the namespace be `de.jabc.ExampleSIB`. For this specific SIB the counters for `de.jabc.ExampleSIB`, `de.jabc.*`, `de.*`, and `#allSIBs` are increased by one. This happens for every SIB in the model. The result can be displayed on the screen and/or saved as a CSV file. As shown exemplarily in Figure 1, the resulting CSV file consists of two rows. The first row is labeled with the complete SIB names and all partial namespaces of them. The second row records its occurrences.

The service usage analysis can be applied to single models or to whole directories containing several files. In addition to that, it is also possible to sum

Table 1. Exemplary SIB usage numbers

#allSIBs	de.*	de.jabc.*	de.jabc.ExampleSIB	de.jabc.OtherSIB	...
10	10	6	1	2	...

the SIB counters from different CSV files into one single CSV file and to merge the metrics of different models into one single CSV file, achieving a project-level statistics aggregation.

5.2 Results

Here we describe and discuss the most interesting outcomes that we obtained by applying jABCstats to the 15 workflow projects of this book.

Workflow and Project Sizes: Designed and developed in the scope of a one-semester course, most of the projects are in fact quite small. Most of them (12) comprise only one model, one comprises two, and two projects comprise eight models. The models themselves also vary considerably in their size, being composed from 6 to 156 SIBs, with an average of 26.8 and a median of 18.5 (standard deviation 29.4).

Workflow Hierarchy: As indicated above, 12 of 15 projects only used one main model and are hence only working with one level of hierarchy. The three projects that used more models [11,47,52] have one main model with several submodels (but no further hierarchical cascade), which means that they make use of two levels of hierarchy.

Workflow Patterns: We let jABCstats count the most relevant control-flow workflow patterns in all workflow models of our sample. As Figure 1 shows, the sequence pattern (simple sequential execution of two services) is by far the most-used control structure, with 571 occurrences in the analyzed workflows. Exclusive choices (conditional branchings), simple merges (convergence of branches) and loops (repetitive behavior) also occur quite often, while parallel executions (fork/join) do in fact only play a minor role. Although these are only results for a very small sample of workflows, this substantiates our observation that conditional branchings and loops are required and used quite frequently. Hence it is important that workflow modeling systems provide the possibility to include them in their models.

SIB Usage: Using jABCstats, we created individual SIB usage statistics for all students' projects and merged them into one single CSV file. Then we used a spreadsheet software and GNU R to compute aggregated values and some charts for analyzed workflows. Tables 2, 3 and 4 summarize the created SIB usage

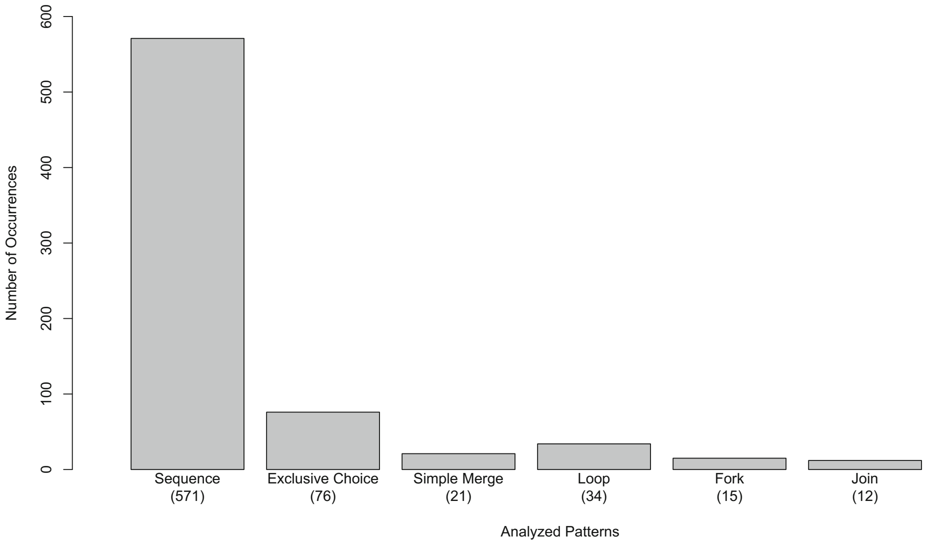


Fig. 1. Usage of workflow patterns

statistics with regard to all available SIBs, the SIBs provided by the jABC and the jABC's Common SIBs in particular, respectively. The upper parts of the tables contain the individual numbers for each project, while the lower parts report aggregated values for all analyzed workflows, namely:

1. the sum (total number) of SIBs from the package used in the projects,
2. the number of projects that used SIBs from the package,
3. the percentage of projects that used SIBs from the package,
4. the minimum number of SIBs from the package used in a project,
5. the average number of SIBs from the package used in the projects, and
6. the maximum number of SIBs from the package used in a project.

We were in particular interested in the distribution of the SIB usage, that is, how many of the individual SIBs were used in the workflows, and how much of the workflows use the individual SIBs. In Tables 2, 3 and 4 we give the numbers for selected hierarchy levels of the namespaces in order to be able to compare the usage of the most relevant groups of SIBs. Below each table, a pie chart additionally visualizes the distribution of the SIB packages used in the workflows, that is, the values contained in aggregation line 1.

Starting with the general overview of the SIB usage (see Table 2), we see, most interestingly, that more than 75% of the SIBs used by the students are included in every jABC distribution. Only about 23% of all SIBs used were created on demand by the lecturers and tutors of the course. This comprises, for instance, the REST SIBs, the FTP SIBs and the different bioinformatics and geovisualization SIBs described in [32]. They have been used by 12 projects (80%). 6 SIBs were created by the geovisualization student Daniel Teske [58]. He created some

Table 2. SIB usage overview

category		all SIBs	jABC SIBs	SIBs created for the students	SIBs created by the students
workflow analysis results per project	Christian Kuntzsch [22]	21	18	3	0
	Christine Schuett [54]	13	13	0	0
	Daniel Teske [58]	18	11	1	6
	Franziska Noack [47]	119	96	23	0
	Gunnar Schulze [53]	40	38	2	0
	Henriette Sens [55]	9	8	1	0
	Janine Vierheller [60]	28	28	0	0
	Josephine Kind [18]	67	48	19	0
	Judith Reso [49]	13	6	7	0
	Lasse Scheele [52]	142	76	66	0
	Leif Blaese [11]	38	38	0	0
	Marcel Hibbe [15]	79	77	2	0
	Monika Lis [33]	14	10	4	0
	Robin Holler [16]	46	45	1	0
	Tobias Respondek [50]	156	104	52	0
aggregation	1. total # of SIBs from package:	803	616	181	6
	2. # of projects using package:	15	15	12	1
	3. % of projects using package:	100	100	80	6.7
	4. min # SIBs from package:	9	6	0	0
	5. average # SIBs from package:	54	41	12	0.4
	6. max # SIBs from package:	156	104	66	6

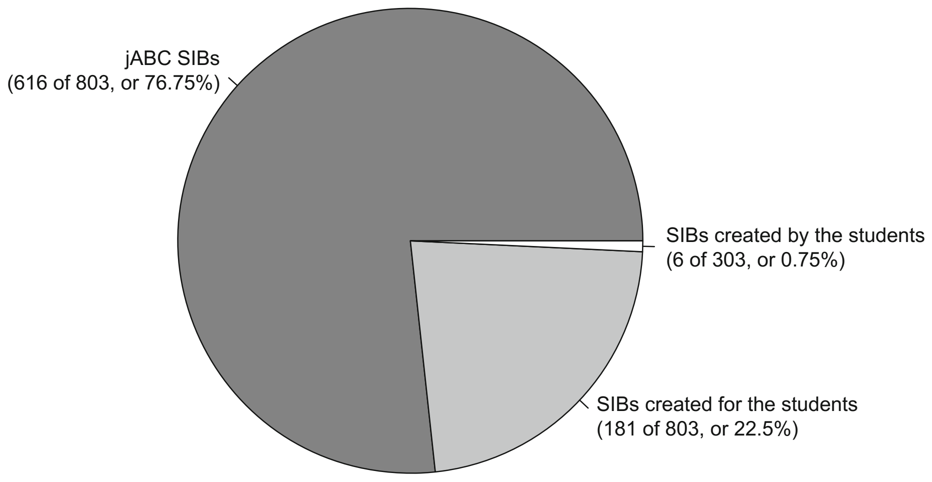


Table 3. Usage statistics of the SIBs provided by the jABC

category		jABC SIBs	Common SIBs	jETI SIBs	Control SIBs	Macro SIBs
workflow analysis results per project	Christian Kuntzsch [22]	18	18	0	0	0
	Christine Schuett [54]	13	10	0	3	0
	Daniel Teske [58]	11	7	0	4	0
	Franziska Noack [47]	96	89	0	0	7
	Gunnar Schulze [53]	38	35	0	3	0
	Henriette Sens [55]	8	8	0	0	0
	Janine Vierheller [60]	28	28	0	0	0
	Josephine Kind [18]	48	16	32	0	0
	Judith Reso [49]	6	6	0	0	0
	Lasse Scheele [52]	76	22	47	0	7
	Leif Blaese [11]	38	33	0	4	1
	Marcel Hibbe [15]	77	77	0	0	0
	Monika Lis [33]	10	10	0	0	0
	Robin Holler [16]	45	43	0	2	0
	Tobias Respondek [50]	104	93	0	11	0
aggregation	1. total # of SIBs from package:	616	495	79	27	15
	2. # of projects using package:	15	15	2	6	3
	3. % of projects using package:	100	100	13	40	20
	4. min # SIBs from package:	6	6	0	0	0
	5. average # SIBs from package:	41	33	5	1.8	1
	6. max # SIBs from package:	104	93	47	11	7

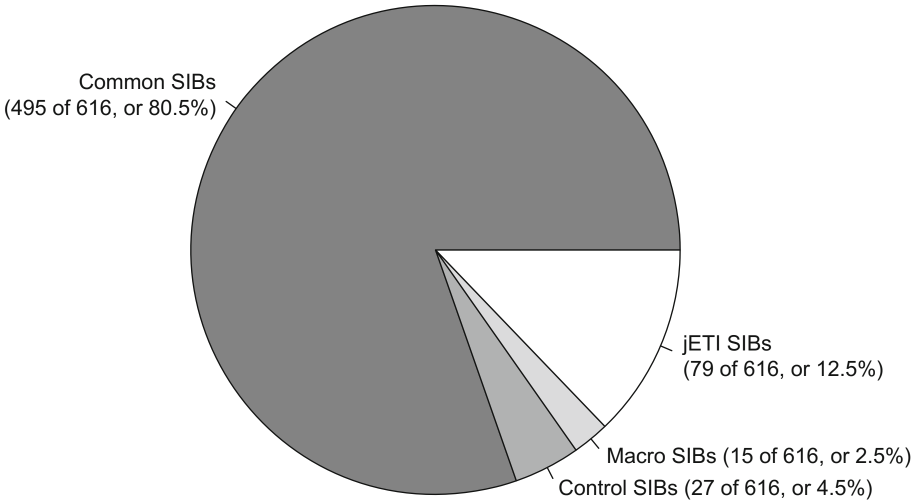
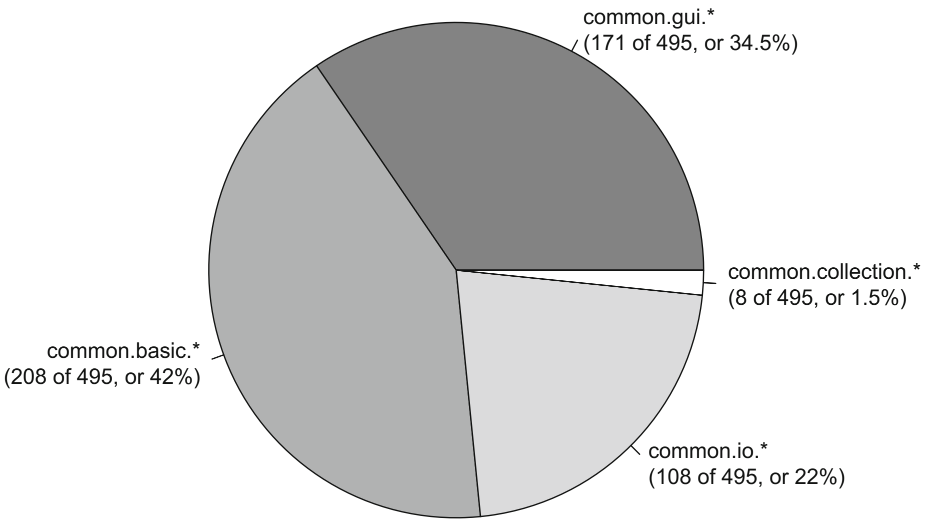


Table 4. Usage statistics of the Common SIBs

package		de.jabc.sib.common.*	de.jabc.sib.common.basic.*	de.jabc.sib.common.collection.*	de.jabc.sib.common.gui.*	de.jabc.sib.common.io.*
workflow analysis results per project	Christian Kuntzsch [22]	18	11	1	3	3
	Christine Schuett [54]	10	0	0	5	5
	Daniel Teske [58]	7	1	0	6	0
	Franziska Noack [47]	89	23	0	60	6
	Gunnar Schulze [53]	35	1	1	14	19
	Henriette Sens [55]	8	0	0	4	4
	Janine Vierheller [60]	28	0	0	19	9
	Josephine Kind [18]	16	16	0	0	0
	Judith Reso [49]	6	0	0	6	0
	Lasse Scheele [52]	22	7	0	15	0
	Leif Blaese [11]	33	13	0	6	14
	Marcel Hibbe [15]	77	49	2	15	11
	Monika Lis [33]	10	5	0	3	2
	Robin Holler [16]	43	13	0	15	15
	Tobias Respondek [50]	93	69	4	0	20
aggregation	1. total # of SIBs from package:	495	208	8	171	108
	2. # of projects using package:	15	11	4	13	11
	3. % of projects using package:	100	73	27	87	73
	4. min # SIBs from package:	6	0	0	0	0
	5. average # SIBs from package:	33	14	0.5	11	7
	6. max # SIBs from package:	93	69	4	60	20



SIBs on his own because he already had background knowledge in Java programming, thus it was easy for him to implement his own SIBs. Not visible from the statistics shown in the tables, the `common.io.ExecuteCommand` SIB is also used quite often to integrate external functionality in the workflows. In fact, 8 projects (53%) made use of this SIB for executing some functionality, such as own scripts, on the local machine.

Going top-down in the namespace hierarchy and having a closer look at the distribution of the jABC SIBs used (see Table 3), we see that the Common SIBs are by far the most-used SIBs from those provided by the jABC (with a share of around 80%). In fact, each project makes use of the Common SIBs. As described in [32], they are necessary for defining values on the context (Basic SIBs), working with collections (Collection SIBs), retrieving some input from the user of the application and showing results (GUI SIBs), and for reading and writing files on the file system (IO SIBs). 13 students (87%) used GUI SIBs and 11 students (73%) used basic and IO SIBs. The outcome of 11 project workflows (73%) depended on some user input (`common.gui.ShowInputDialog`), and 8 projects (53%) used SIBs to read from and write to text files.

Table 4 details further on the usage of the Common SIBs. It shows that most of the Common SIBs used in the projects are from the `common.basic.*` (42%) and `common.gui.*` (34.5%) packages. The IO SIBs make up for 22%, while the Collection SIBs do only account for 1.5% of the Common SIBs that were used in the projects.

Small Comparison of the Bioinformatics and Geovisualization Projects: The total amount of SIBs used by geovisualization students is considerably higher than by the students with bioinformatics-related workflows. Especially the usage difference between the usage of Basic and GUI SIBs is noticeable. jETI SIBs are used by only two geovisualization projects. The diversity of the available SIBs is generally more exploited by the geovisualization projects.

Looking at the details of the percentage of SIB usage, we see that all bioinformatics and all geovisualization students used the Common SIBs, which reinforces our impression that they are a basis for every workflow. Geovisualization students made more use of `common.gui.ShowBranching[/Image]Dialog` SIBs and bioinformatics students used more `ShowInputDialog` and `ExecuteCommand` SIBs. Additionally, geovisualization students read (`ReadTextFile`) and wrote (`WriteTextFile`) three times more often text files (31 times) than bioinformatics students (11 times). REST SIBs have only been used by geovisualization students, who made more use of submodels (`MacroSIB`).

6 Conclusion

The results of this course were very satisfying for us. We had previously used the jABC in many courses in Computer Science (Software Engineering, Formal Methods in System Design, Service-Oriented Architectures, Foundations of

Service Engineering, and many more), but not so systematically with non-IT students. The student feedback provided in this course and meanwhile also in similar courses gives a good impression of the wide usability of the jABC framework. The experiences with student users showed that the framework is syntactically easy to handle, but the central remaining challenge concerns how to make the services more easily accessible. A systematic usability study is subject of future work. Features that would improve the handling and are already available in jABC, or jABC4, but were not used in the students' workflows concern design for simplicity, processes as data, and variability and evolution.

Design for Simplicity

Design for Simplicity aims at fighting the often unexpectedly high longer-term costs for maintenance, adaptation, migration, etc.. Guided by the questions, 'what is really required', 'what is provided in terms of standards solutions', and 'which parts really need to be newly developed' (cf. [37,38]), a number of projects have been realized which comprise, for instance, a plug-in framework enhancing the functionality of the jABC framework [41], a synthesis-based approach solution to service-oriented programming [30], and a learning-based testing approach [57].

Processes as Data

A graphical and dynamic framework for binding and execution of (business) process models has been developed that is tailored to integrate

- ad hoc processes modelled graphically,
- third party services discovered in the (Inter)net, and
- (dynamically) synthesized process chains that solve situation-specific tasks, with the synthesis taking place not only at design time, but also at runtime.

Key to this approach [44,45,46,43] is the introduction of type-safe stacked second-order execution contexts that allow for higher-order process modeling. Tamed by our underlying strict service-oriented notion of abstraction, this approach is tailored also to be used by application experts with little technical knowledge: users can select, modify, construct and then pass (component) processes during process execution as if they were data. The most advanced feature of this framework is the combination of online synthesis with the integration of the synthesized process into the running application [46]. This ability leads to a particularly flexible way of implementing self-adaption, and to a particularly concise and powerful way of achieving variability not only at design time, but also at runtime.

Variability and Evolution

Constraint-based variability modeling is a flexible, declarative approach to managing solution-space variability [17,31]. Product variants are defined in a top-down manner by successively restricting the admissible combinations of product artifacts until a specific product variant is determined. Methods range from applying model checking to manually designed variants to applying synthesis technology [56,13,14] for the fully automatic generation of product variants that satisfy all given constraints [26]. This technology underlies the the loose programming approach, where incomplete specifications are turned into running programs via synthesis [30]. Moreover, it can be even combined with the above mentioned concept of higher-orderedness to realize a very flexible form of runtime adaptability [46].

We see therefore the current basis as an excellent foundation for further studies, for instance more empirical analyses with jABCstats, as shown in [62], but also with regard to the automated synthesis of such workflows based on knowledge nuggets as made possible by the PROPHETS plugin [39]. While the domain models get more precise, and more work is needed to properly describe semantics, constraints, relations concerning services, their data and their embedded functionalities and quality of service, the subsequent workflow design requires less knowledge and less work due to (semi-) automatic composition and conformity to constraints by construction [28,26,25,24]. In this sense, in the further practical investigation we intend to include more semantics, and raise the availability and use of the domain knowledge during the workflow composition.

References

1. CSISS/GMU Geospatial Web Services, `http://geobrain.laits.gmu.edu/grassweb/manuals/index.html` (last accessed September 9, 2013)
2. DDBJ Web API for Biology, `http://xml.nig.ac.jp/workflow/` (temporarily suspended since February 15, 2012)
3. Google Maps, A.P.I.: - Google Developers, `https://developers.google.com/maps/` (last accessed March 21, 2013)
4. Soaplab, `http://soaplab.sourceforge.net/soaplab1/` (last accessed May 5, 2013)
5. Twitter, `https://twitter.com` (last accessed September 10, 2013)
6. Twitter Developers, `https://dev.twitter.com` (last accessed September 11, 2013)
7. Twitter4J - A Java Library for the Twitter API. `http://twitter4j.org/` (last accessed September 10, 2013)
8. Al-areqi, S., Kriewald, S., Lamprecht, A.-L., Reusser, D., Wrobel, M., Margaria, T.: Agile Workflows for Climate Impact Risk Assessment based on the ci:grasp Platform and the jABC Modeling Framework. In: International Environmental Modelling and Software Society (iEMSs) 7th Intl. Congress on Env. Modelling and Software (accepted, 2014)
9. Bakera, M., Jörges, S., Margaria, T.: Test your Strategy: Graphical Construction of Strategies for Connect-Four. In: Proceedings of the 2009 14th IEEE International Conference on Engineering of Complex Computer Systems, ICECCS 2009, pp. 172–181. IEEE Computer Society, Washington, DC (2009)
10. Bhagat, J., Tanoh, F., Nzuobontane, E., Laurent, T., Orlowski, J., Roos, M., Wolstencroft, K., Aleksejevs, S., Stevens, R., Pettifer, S., Lopez, R., Goble, C.A.: BioCatalogue: a universal catalogue of web services for the life sciences. Nucleic Acids Research 38(suppl. 2), 689–694 (2010)

11. Blaese, L.: Data Mining for Unidentified Protein Sequences. In: Lamprecht, A.-L., Margaria, T. (eds.) Process Design for Natural Scientists. CCIS, vol. 500, pp. 73–87. Springer, Heidelberg (2014)
12. Ebert, B.E., Lamprecht, A.-L., Steffen, B., Blank, L.M.: Flux-P: Automating Metabolic Flux Analysis. Metabolites 2(4), 872–890 (2012)
13. Freitag, B., Margaria, T., Steffen, B.: A Pragmatic Approach to Software Synthesis. In: Workshop on Interface Definition Languages, pp. 46–58 (1994)
14. Freitag, B., Steffen, B., Margaria, T., Zukowski, U.: An Approach to Intelligent Software Library Management. In: Proceedings of the 4th International Conference on Database Systems for Advanced Applications (DASFAA), pp. 71–78. World Scientific Press (1995)
15. Hibbe, M.: Spotlocator Project Documentation. In: Lamprecht, A.-L., Margaria, T. (eds.) Process Design for Natural Scientists. CCIS, vol. 500, pp. 149–158. Springer, Heidelberg (2014)
16. Holler, R.: GraffDok: A Graffiti Documentation Application. In: Lamprecht, A.-L., Margaria, T. (eds.) Process Design for Natural Scientists. CCIS, vol. 500, pp. 235–247. Springer, Heidelberg (2014)
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. Kind, J.: Creation of Topographic Maps. In: Lamprecht, A.-L., Margaria, T. (eds.) Process Design for Natural Scientists. CCIS, vol. 500, pp. 225–234. Springer, Heidelberg (2014)
19. 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 (2006)
20. Kubczak, C., Margaria, T., Steffen, B., Nagel, R.: Service-oriented Mediation with jABC/jETI (2008)
21. Kubczak, C., Margaria, T., Steffen, B., Naujokat, S.: Service-oriented Mediation with jETI/jABC: Verification and Export. In: Proceedings of the 2007 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, WI-IAT Workshop, Silicon Valley, California, USA, pp. 144–147. IEEE Computer Society Press (November 2007)
22. Kuntzsch, C.: Visualization of Data Transfer Paths. In: Lamprecht, A.-L., Margaria, T. (eds.) Process Design for Natural Scientists. CCIS, vol. 500, pp. 140–148. Springer, Heidelberg (2014)
23. 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)
24. Lamprecht, A.-L.: User-Level Workflow Design. LNCS, vol. 8311. Springer, Heidelberg (2013)
25. Lamprecht, A.-L., Margaria, T.: Scientific workflows: Eternal components, changing interfaces, varying compositions. In: Margaria, T., Steffen, B. (eds.) ISoLA 2012, Part I. LNCS, vol. 7609, pp. 47–63. Springer, Heidelberg (2012)
26. Lamprecht, A.-L., Margaria, T., Schaefer, I., Steffen, B.: Synthesis-based variability control: Correctness by construction. In: Beckert, B., Damiani, F., de Boer, F.S., Bonsangue, M.M. (eds.) FMCO 2011. LNCS, vol. 7542, pp. 69–88. Springer, Heidelberg (2012)

27. 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)
28. Lamprecht, A.-L., Margaria, T., Steffen, B.: Bio-jETI: a framework for semantics-based service composition. BMC Bioinformatics 10(Suppl 10), S8 (2009)
29. 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)
30. 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)
31. Lamprecht, A.-L., Naujokat, S., Schaefer, I.: Variability Management Beyond Feature Models. IEEE Computer 46(11), 48–54 (2013)
32. 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)
33. Lis, M.: Constructing a Phylogenetic Tree. In: Lamprecht, A.-L., Margaria, T. (eds.) Process Design for Natural Scientists. CCIS, vol. 500, pp. 101–109. Springer, Heidelberg (2014)
34. 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)
35. Margaria, T., Floyd, B., Lamprecht, A.-L., Camargo, R.G., Neubauer, J., Seelaender, M.: Simple Management of High Assurance Data in Long-lived Interdisciplinary Healthcare Research: A Proposal. In: ISoLA 2014. LNCS, Springer, Heidelberg (to appear, 2014)
36. Margaria, T., Kubczak, C., Njoku, M., Steffen, B.: Model-based Design of Distributed Collaborative Bioinformatics Processes in the jABC. In: Proceedings of the 11th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2006), Los Alamitos, CA, USA, pp. 169–176. IEEE Computer Society (August 2006)
37. Margaria, T., Steffen, B.: Simplicity as a Driver for Agile Innovation. Computer 43(6), 90–92 (2010)
38. Merten, M., Steffen, B.: Simplicity driven application development. Journal of Integrated Design and Process Science (SDPS) 16 (2013)
39. 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)
40. Naujokat, S., Lybecait, M., Steffen, B., Kopetzki, D., Margaria, T.: Full Generation of Domain-Specific Graphical Modeling Tools: A Meta^{2}modeling Approach (under submission, 2014)
41. Naujokat, S., Neubauer, J., Lamprecht, A.-L., Steffen, B., Jürges, S., Margaria, T.: Simplicity-First Model-Based Plug-In Development. Software: Practice and Experience 44(3), 277–297 (2013) (first published online)
42. Naujokat, S., Traonouez, L.-M., Isberner, M., Steffen, B., Legay, A.: Custom Graphical Specification of Multi-Faceted Concurrent Systems: A Metamodeling Approach. In: Proc. of the 12th Int. Conf. on Software Engineering and Formal Methods (SEFM 2014) (2014) (under submission)

43. Neubauer, J.: Higher-Order Process Engineering. Phd thesis, Technische Universitüt Dortmund (2014)
44. Neubauer, J., Steffen, B.: Plug-and-Play Higher-Order Process Integration. IEEE Computer 46(11), 56–62 (2013)
45. Neubauer, J., Steffen, B.: Second-order servification. In: Herzwurm, G., Margaria, T. (eds.) ICSOB 2013. LNBIP, vol. 150, pp. 13–25. Springer, Heidelberg (2013)
46. Neubauer, J., Steffen, B., Margaria, T.: Higher-Order Process Modeling: Product-Lining, Variability Modeling and Beyond. Electronic Proceedings in Theoretical Computer Science 129, 259–283 (2013)
47. Noack, F.: CREADED: Coloured-Relief Application for Digital Elevation Data. In: Lamprecht, A.-L., Margaria, T. (eds.) Process Design for Natural Scientists. CCIS, vol. 500, pp. 182–195. Springer, Heidelberg (2014)
48. 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)
49. Reso, J.: Protein Classification Workflow. In: Lamprecht, A.-L., Margaria, T. (eds.) Process Design for Natural Scientists. CCIS, vol. 500, pp. 65–72. Springer, Heidelberg (2014)
50. Respondeck, T.: A Workflow for Computing Potential Areas for Wind Turbines. In: Lamprecht, A.-L., Margaria, T. (eds.) Process Design for Natural Scientists. CCIS, vol. 500, pp. 196–211. Springer, Heidelberg (2014)
51. Rice, P., Longden, I., Bleasby, A.: EMBOSS: the European Molecular Biology Open Software Suite. Trends in Genetics 16(6), 276–277 (2000)
52. Scheele, L.: Location Analysis for Placing Artificial Reefs. In: Lamprecht, A.-L., Margaria, T. (eds.) Process Design for Natural Scientists. CCIS, vol. 500, pp. 212–224. Springer, Heidelberg (2014)
53. Schulze, G.: Workflow for Rapid Metagenome Analysis. In: Lamprecht, A.-L., Margaria, T. (eds.) Process Design for Natural Scientists. CCIS, vol. 500, pp. 88–100. Springer, Heidelberg (2014)
54. Schütt, C.: Identification of Differentially Expressed Genes. In: Lamprecht, A.-L., Margaria, T. (eds.) Process Design for Natural Scientists. CCIS, vol. 500, pp. 127–139. Springer, Heidelberg (2014)
55. Sens, H.: Web-Based Map Generalization Tools Put to the Test: A jABC Workflow. In: Lamprecht, A.-L., Margaria, T. (eds.) Process Design for Natural Scientists. CCIS, vol. 500, pp. 171–181. Springer, Heidelberg (2014)
56. Steffen, B., Margaria, T., Freitag, B.: Module Configuration by Minimal Model Construction. Technical report, Fakultät für Mathematik und Informatik, Universität Passau (1993)
57. Steffen, B., Neubauer, J.: Simplified Validation of Emergent Systems through Automata Learning-Based Testing. In: 2011 34th IEEE Software Engineering Workshop (SEW), pp. 84–91 (June 2011)
58. Teske, D.: Geocoder Accuracy Ranking. In: Lamprecht, A.-L., Margaria, T. (eds.) Process Design for Natural Scientists. CCIS, vol. 500, pp. 159–170. Springer, Heidelberg (2014)
59. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow Patterns. Distributed and Parallel Databases 14(1), 5–51 (2003)
60. Vierheller, J.: Exploratory Data Analysis. In: Lamprecht, A.-L., Margaria, T. (eds.) Process Design for Natural Scientists. CCIS, vol. 500, pp. 110–126. Springer, Heidelberg (2014)

61. Wickert, A.: Semantische Dienstselektion für modellgetriebene Geschäftsprozesse in jABC: Analyse, Integration und Evaluation. Diploma thesis, Universität Potsdam (2010)
62. Wickert, A., Lamprecht, A.-L.: jABCstats: An Extensible Process Library for the Empirical Analysis of jABC Workflows. In: ISoLA 2014. LNCS. Springer (to appear, 2014)