Scientific Workflows with the jABC Framework

A Review after a Decade in the Field

Anna-Lena Lamprecht¹, Bernhard Steffen², Tiziana Margaria¹

1. Lero - The Irish Software Research Centre, University of Limerick, Ireland, anna-lena.lamprecht@lero.ie, tiziana.margaria@lero.ie
2. Chair of Programming Systems, TU Dortmund University, Germany, steffen@cs.tu-dortmund.de

Received: date / Revised version: date

Abstract The jABC is a framework for process modelling and execution according to the XMDD (eXtreme Model-Driven Design) paradigm, which advocates the rigorous use of user-level models in the software development process and software life cycle. We have used the jABC in the domain of scientific workflows for more than a decade now – an occasion to look back and take stock of our experiences in the field. On the one hand, we discuss results from the analysis of a sample of nearly 100 scientific workflow applications that have been implemented with the jABC. On the other hand, we reflect on our experiences and observations regarding the workflow development process with the framework. We then derive and discuss ongoing further developments and future perspectives for the framework, all with an emphasis on simplicity for end-users through increased domain specificity. Concretely, we describe how the use of the PROPHETS synthesis plugin can enable a semantics-based simplification of the workflow design process, how with the jABC4 and DyWA frameworks more attention is paid to the ease of data management, and how the Cinco SCCE Meta Tooling Suite can be used to generate tailored workflow management tools.

1 Introduction

A major part of the scientific experiments carried out today require thorough computational support. While database and algorithm providers face the problem of bundling resources to create and sustain powerful computation nodes, scientists have to deal with combining sets of tools and (remote) services into specific data analysis and transformation processes. In the last decade, especially workflow-based approaches have enjoyed great popularity for dealing with complex processes for data gathering, cleansing, and analysis. A plethora of workflow systems is available and used in the scientific domain today, and we can distinguish two basic kinds of systems among them:

1. Systems targeted at high-throughput workflows and their execution on supercomputers, such as computational grids or clusters. Typically, the design of the workflow and its actual execution take place on different machines, hence a deployment step is needed between the two phases. Since efficient execution is crucial, the workflow design environments usually provide perspectives on the workflow that facilitate optimisation, such as diagrams that represent data flow and communication. Popular scientific workflows systems that can be assigned to this category include, for example, the ASKALON Cloud and Grid Application Development and Computing Environment [98], the Pegasus Workflow Management System [27], the Triana “problem solving environment” [116] and the Ptolemy II-based Kepler system [30,68].

2. Systems targeted at comparatively lightweight workflows, which can be executed directly (typically by interpretation) in the workflow design environment. This provides the user with feedback about the execution and with results immediately, and thus allows for a very agile and user-centric way of developing rapidly-evolving workflows. Such workflows can also contain calls to remote services, such as grid or web services, and thus make use of external computational power, but these calls are usually transparently encapsulated by workflow building blocks, so that they appear just as local components in the workflow model. Examples for systems of this kind include the jOrcia desktop client [48,84], the jOpera service composition environment [8], and the Vis-Trails scientific workflow management system [22].
Of course these categories are not strictly disjoint: many systems that are actually assigned to the first category can also be used for workflows of the second, and several intermediate and hybrid forms exist as well. The well-known Taverna Workflow Management System, for example, had in the beginning been promoted as a “tool for building and running workflows of services” [39] in the sense of the second category, but meanwhile explicitly also supports execution on larger infrastructures with the Taverna Server [132]. Web-based systems like Galaxy [18] or OnlineHPC [3] can be assigned to the above categories as well, based on whether they execute the workflows immediately, in the same environment in which they are designed, or if they need to be deployed to some other environment for execution.

In our work on scientific workflows with the jABC framework [78,113], we strive for an extreme incarnation of a system of the second kind, guided by the principles of simplicity [72,81] and the eXtreme Model-Driven Design (XMDD) paradigm [78,79,80,81,82,115]. XMDD is a software development methodology that advocates the rigorous use of user-level models and refinement throughout the software development process and software life cycle. Towards this aim, XMDD combines the decisive traits of several modern software engineering schools into a coherent paradigm that efficiently and effectively leverages their characteristic contributions. Concretely, it is based on ideas taken from:

- **eXtreme Programming** (XP) [15] and more recently **User Experience** (UX) [16], to work in a user-driven fashion and provide immediate feedback through requirement and design validation by means of model tracing, simulation and early testing,
- **Service Orientation** [38,83], to abstract from the implementation of functionality,
- **Aspect Orientation** [49], to treat crosscutting as well as role-specific concerns modularly, and
- **Model-Driven Design** [101], to control the overall development at the modelling level.

Aiming at an easy uptake by practitioners, simplicity [81] is the key guiding principle in XMDD. According to the motto “easy for the many, difficult for the few”, the XMDD approach systematically builds upon the intuition of the users through a workflow design framework that enables a graphical service composition (the “easy” part), and keeps as much technicality and formality as possible under the hood (the “difficult” part, which is normally only accessible to IT experts, and challenging even for the average programmer and software engineer). In this spirit, our aim is to support end-users in their daily work as producers and users of scientific workflows, and specifically to boost the independence and productivity of application experts highly skilled in a specific domain, but without specific IT or coding knowledge. And thus, in accordance with the XMDD paradigm, the jABC framework explicitly strives for an adequate end-user-level granularity, and for maintaining the full power of general-purpose model-driven design when adapting the framework to a particular scientific domain. This enables a very high level of agility in the workflow design process, which is still an uncommon characteristic for state-of-the-art scientific workflow management tools.

In this paper, we look back at a decade of using the jABC framework in the domain of scientific workflows, analyse existing applications and reflect on our experience and observations regarding workflow development with this framework. We derive and discuss ongoing further developments and future perspectives for the framework, all with an emphasis on simplicity for end-users through increased domain specificity. The paper is structured as follows. Section 2 introduces the jABC project in greater detail. Then, Section 3 is devoted to the analysis of scientific workflow applications implemented with the framework and the discussion of experiences with the use of the framework in the domain. Section 4 is devoted to ongoing further developments and future perspectives, and Section 5 concludes the paper.

### 2 The jABC Project

In its quality as the reference implementation of the XMDD paradigm, the jABC aims at providing a multipurpose and domain-independent modelling framework for service-oriented design and development that allows users to develop services and applications easily by composing reusable building blocks into (flow) graph structures that are both formally sound and easy to read and to build. From an end-user point of view, all user interaction happens within an intuitive graphical environment, hardly requiring any classical programming skills for the workflow modelling. This way, the jABC supports a particularly agile and model-driven approach to manage processes and workflows.

The jABC is not one single tool, but rather a family of frameworks that has been developed and evolved since the beginning of the 1990s. In the following we explain its fundamental principles (Section 2.1) and give an overview of the evolution of the jABC project (Section 2.2), before introducing in greater detail the jABC version that was used for the vast majority of the scientific workflow applications (Section 2.3).

#### 2.1 Principles

Central to the jABC project is the workflow modelling formalism that is applied, the **Service Logic Graph** (or **SLG**) [109]. Broadly speaking, SLGs are directed graphs whose nodes represent basic services and whose edges define the flow of control.

The basic services from which the SLGs are constructed are called **Service-Independent Building Blocks**,
or SIBs by analogy with the telecommunication terminology [107], and in the spirit of the Service-oriented Computing (SOC) paradigm [69,83]. SIBs can refer to any programatically accessible piece of software functionality such as APIs, Web or REST Services, and command-line programs with scriptable interfaces. The SIBs are parameterisable, so that their behaviour can be adapted depending on the current context of use. Data exchange between the individual SIBs is done via a shared memory where the SIBs can put and access data objects. Each SIB has one or more outgoing branches that specify its possible successor(s): at runtime, the execution of a SIB determines which branch has to be taken to continue the computation. To form SLGs, the SIBs are connected by directed edges that carry one or more branch labels to define the flow of control.

The SLGs are furthermore hierarchical structures, that is, a SIB in an SLG may represent a single functionality (atomic SIB) or a whole subgraph (i.e. another SLG), thus serving as a macro that hides more detailed process models. This feature grants a high reusability not only of components, but also of whole models, within larger applications.

In our experience, the control flow-oriented approach to service design and analysis is intuitive and adequate for the work on scientific workflows as described above, as it makes it easy to model sophisticated processes, for instance with different execution traces depending on the kind of input data, or with iterations or recursions over sets of data. The data itself is managed within a shared memory, and uses identifiers similar to variables to refer to particular data items. This mechanism can be seen as a variant of the blackboard-oriented coordination spaces in the style of Linda [23] and, more generally, tuple-spaces, that have proven to be a simple and powerful model for loosely coupled distributed systems. As discussed in Section 3.1.2, control-flow structures like conditional branchings and loops are indeed frequently used in the applications and hence essential to have. They are also at the basis of the computational thinking capabilities advocated by Wing [130] and now increasingly widely adopted in education at primary and secondary level and for basic coding curricula, e.g. in Scratch [99] and similar approaches like Pencil Code [14] and BlueJ [13]. We are convinced that their clear graphical representation in the jABC, reinforcing the intuition of the users, greatly simplifies the understanding of the processes represented by the models in comparison to code-based approaches. At the same time, the data dependencies do not clog the representation, and even large processes with complex data flows are easily readable.

In contrast, many popular scientific workflow systems (such as Taverna [39,132], Triana [116], and Kepler [30,68]) are inherently data flow-oriented, which is indeed the most apparent conceptual difference to the control flow-oriented jABC. Both approaches are often considered to be capable of expressing the same processes, although when using the data-flow approach there are clear limitations with respect to the inclusion of elaborate control structures (see, e.g. [54]). These have to be treated by the respective systems by providing additional features like, for instance, the control links and iteration configurations that can be used in Taverna for specifying the execution order of the services in its data-flow workflow models further. Petri nets on the other hand, on which for instance the YAWL workflow language and system [118] is based, are very powerful and expressive, but in our opinion and experience too complex and not intuitive enough to be adequate for the users and lightweight workflow applications that we target with our XMDD approach as described above.

The SLGs can furthermore be interpreted with different foci for different uses. In a brainstorming phase for a project, they are seen as just drawings of the workflows (intuitive models). When all SIBs are implemented they are immediately executable (live models), and the execution plugin will interpret them as control flow graphs with fork/join parallelism. And they can also be interpreted as Kripke Transition Systems [87] and are thus directly amenable to formal analysis methods like model checking [24,87].

The jABC has in fact been designed with a focus on formal verification capability (cf. [44,110,114]) in order to be able to validate the workflow logic as early in the design process and as precisely as possible. The built-in model checker has for instance been used for verifying the behaviour of telecommunication services [107,109,110], the ExoMars Rover [12], the OCS conference system [95], and the framework’s code generator [46], and we demonstrated on a few examples how they could be applied to scientific workflow models [59]. However, while the methods for formal verification have been developed for decades and their use is increasing in many application domains, especially in the areas of embedded and critical systems, it has not received significant attention in the scientific workflow community so far. (To the best of our knowledge, only for the pioneering but meanwhile discontinued DiscoveryNet [33] scientific workflow system has a specific approach been described for the formal verification of the models based on temporal logic [26].) The reasons for this are certainly manifold and interesting to assess, ranging from a lack of awareness of such methods in the scientific workflow community to unclear effort and benefits of their application, but such a discussion would go beyond the scope of this paper.

2.2 Evolution

Figure 1 surveys the evolution of the jABC project from its beginnings. It all started in 1993 with the idea to better organise the combinatorial explosion of potential tool chains as they were implemented in the Concurrency Workbench [25] in a compositional fashion. The DaCapo
and later the META-Frame Tool can be regarded as first service-oriented environments where tool building blocks could be graphically composed into workflows [77,111,121]. After an industrial project with Siemens Nixdorf in 1995/96, where this tool was applied to construct a service definition environment for value-added services that has subsequently been sold to over 30 Telecoms world-wide [19,107], the META-Frame tool was again renamed, now to Application Building Center (ABC), and later on re-implemented as the jABC tool in Java since 2003 and applied in numerous industrial and scientific projects (such as [9,10,40,52,73,95]). Driving forces for the further development were the scientific workflow projects reported in more detail in the course of this paper. In parallel, the framework formed the backbone for numerous extensions, like the Electronic Tool Integration platform ETI [108] and its variants jETI [76] and Bio-jETI [60,75], the Genesys [45,47] and PROPHETS (Process Realization and Optimization Platform using Human-readable Expressions of Temporal-logic Synthesis) [63,88] plugins for code generation and model synthesis, and finally the DyWa tool for developing dynamic Web applications [94]. Currently, the jABC heritage of designing domain-specific modelling tools is taken over by the Cinco Meta-tooling suite [89], which lifts domain modelling to the meta level.

First applications in the field of scientific (bioinformatics) workflows emerged in 2005, when the second generation jABC then available was used in cooperation projects with biomedical partners to model workflows for the retrieval of orthologous\(^1\) gene structures among a selection of higher organisms [74] and for the preprocessing and analysis of LC-MS data (liquid chromatography - mass spectrometry) [51]. These projects demonstrated how hierarchical bioinformatics workflows that use external tools and resources can be modelled in the jABC framework (service orchestration), as well as how to represent and embed such external resources in the jABC (service integration).

In the following years an increasing number of scientific workflow applications was implemented with jABC3, which had been released in 2006. The projects often also made use of the associated jETI (Electronic Tool Integration) platform [76] to ease the integration of third-party tools as services for use within the jABC workflow framework. Initially predominantly adopted in the field of bioinformatics, where a domain-specific incarnation of the framework called Bio-jETI was established, applications of jABC soon emerged also in further scientific domains, such as geo-informatics and medical research (see, e.g. [7,54,56]).

Since 2008, the meanwhile mature jABC3 had also been an education tool. We used it to teach a variety of university courses on process modelling and service-based software development in the field of scientific workflows to Bachelor and Master students of Computer Science and related disciplines, such as Business Informatics, Geovisualization, and Bioinformatics. Although the students come with extremely different levels of prior knowledge in programming and software engineering – ranging from complete novices to experienced developers – they can even-handedly participate in the interdisciplinary oriented courses. By the end of 2015, more than 200 students had attended the courses, and accordingly we feel confident in saying that the approach has been validated in both research and education environments. Detailed experience reports about the courses are available in [56,57].

2.3 jABC3 Standard Release

jABC3, first released in 2006, provided a very stable platform for a lot of projects, and also a wealth of further features and capabilities for supporting workflow development that became available over the years. The latest jABC3 standard release, also known as jABC Basic 3.8.5, can be downloaded from the jABC homepage\(^2\) and comprises:

- the jABC core framework and the corresponding jABC editor as graphical user interface,
- the GEAR model checking plugin [12] for model-wide verification of properties,
- the Genesys code generation plugin [45,47].

\(^1\) An orthologous gene is a gene present in two or more species that has evolved from a common ancestor.

– the IconEditor for easy manipulation of SIB icons within the jABC GUI,
– the Layouter plugin, which supports the automatic layout of larger SLGs,
– the LocalChecker plugin that can check defined properties of individual SIBs,
– the SIBCreator plugin that can be used by workflow designers to create code skeletons for new SIBs which are implemented later,
– the TaxonomyEditor plugin, with which a collection of SIBs can be renamed and (re-) arranged in a hierarchy of categories in order to optimally fit the domain structure,
– the Tracer plugin as the inbuilt interpreter for SLGs that allows for their immediate execution,
– and two demo projects (a simplified web shop and a Turtle Graphics process project) designed to provide easy-to-follow examples and starting points for novice users.

Also the workflow synthesis plugin PROPHETS [63,88] that is discussed later in this paper, and a framework for learning-based quality assurance [41,106,129] are based on the jABC3 framework, but have for different reasons not (yet) become part of the jABC3 standard release.

Figure 2 gives an impression of jABC3 in action, using an example workflow from the aforementioned project on LC-MS data preprocessing and analysis [51]. The SLG on the canvas has been created using SIBs from the service library (displayed in the upper left of the window) in a drag&drop fashion, and connecting them with labeled branches representing the flow of control. As soon as the parameters of the SIBs have been configured (in the SIB inspector at the lower left), the workflow is ready for execution. The small window in the upper right corner of the figure is the control panel of the Tracer plugin. In the picture, the Tracer window indicates that it is currently executing a SIB, and the green-coloured branches of the model on the canvas visualise the current execution point and the path it followed. The third window in the figure shows an (intermediate) result from the workflow execution. It has been opened by the currently executed SIB.

3 Scientific Workflows with jABC3: Analysis and Experiences

Since the first applications of jABC in the field of scientific workflows a decade ago, many more projects have been carried out with it, addressing not only bioinformatics, but also other domains like geo-informatics, medical research, and business analytics. Concerning the provenance of the implemented workflow models,

– some of these case studies were developed by members of our team to serve as demonstration or proof-of-concept applications (such as the study described in [58]),
– a large number of standard or representative workflows has been implemented by the students in the different editions of our “Process Modeling in Scientific Applications” courses [56,57],
− others emerged from research activities or cooperation projects with partners from a particular field of research (such as [7,29,51,62,74]).
− and some more projects have been carried out by external users (such as [21,103]).

This section aims at reflecting on these scientific workflow projects as well as their origination processes. Section 3.1 reports results from an initiative for the empirical evaluation of jABC3 workflow projects, and in Section 3.2 we summarise collected experiences on the work with the framework.

3.1 Empirical Analysis and Evaluation

Considering the enormous numbers of scientific workflow tools and applications that are around today, surprisingly little empirical evaluation and analysis of workflow applications and workflow systems has been carried out. In particular, we are not aware of any systematic usability studies that provide empirical evidence of a system’s adequacy for the targeted user groups. There are a couple of review-style articles available that compare scientific workflow systems with regard to their functionality, like the study “Pattern-Based Evaluation of Scientific Workflow Management Systems” [86] by Migliorini et al., who compare the Kepler, Taverna and Triana workflow management systems concerning their ability to realise the workflow patterns described by van der Aalst et al. [119], or the “Taxonomy of Workflow Management Systems for Grid Computing” [134] by Yu et. al, which compares the capabilities of more than ten systems. And based on the workflows available in the myExperiment scientific workflow repository [36], different studies [122, 67,32] have been carried out to analyse the scientific workflows themselves.

With jABCstats [124], we have recently started a metrics project to analyse the accrued collection of jABC workflows empirically. Besides a basic academic interest in such metrics, we expect that a better understanding of their characteristics will help us to determine the directions for future improvements of our workflow design and management framework. In the current version, jABCstats analyses:

− **Structural properties** like the number of models and SIBs, the number of submodel calls and hierarchy levels in the projects, the complexity of the models in terms of branching factor and McCabe number, and the control-flow structures used. Taken together, these measures describe the computational thinking concepts applied, both in small examples and in larger case studies. We are interested in profiling these characteristics per application domain as well as across application domains, and wish to investigate whether and to which extent this profile reflects the skills of the users in the application domain and their IT knowledge.

− **Service usage**, that is, the kind and number of SIBs used in the workflows, documenting the occurrence of reuse within and across domains.

− **Data usage**, in terms of the kind and scope of the parameters of the SIBs used in the workflows. This is important in the data (type) definition part of the scientific workflow analysis. In a later stage, we intend to use this data as a source of information for IT governance and design data protection, with the aim of also supporting audits and forensics.

− The level of **documentation**, by assessing SIB naming and model documentation (“decoration”) with additional text, icons and draw elements. This is particularly interesting because to the best of our knowledge the jABC is the only design tool based on a formal semantics that allows this kind of customisation by the users. As it is in our tool semantically irrelevant, the “design skin” of the workflows and the canvas enrichments are indeed an independent layer, aimed at better user comprehension.

We have applied jABCstats to all scientific workflow projects that are currently maintained in our repositories. A complete discussion of all the findings would go beyond the scope of this paper. Hence, we focus here on selected findings concerning the analysis of the structural properties of workflows and the service usage statistics. They provide the basic figures that we consider most suitable to characterise and compare workflows from an abstract perspective, that is, across application domains. In the following, Section 3.1.1 describes the considered sample workflow collection and Section 3.1.2 discusses the analysis results.

3.1.1 Sample

The collection of scientific workflows that we analysed with jABCstats currently comprises 96 jABC3 projects. Each project corresponds to a specific workflow application and may consist of several workflow models. The most common case is that a project uses hierarchical structures with submodels, but applications may also comprise different workflow variants that exist in parallel and are not connected syntactically. To allow for a more differentiated analysis, we distinguish three groups of projects within the sample:

I. Demonstration and proof-of-concept projects (5):
− The **sequence alignment workflow** project used to demonstrate the agility of jABC modelling [58].
− The **DDBJ workflows**, realising abstract workflow descriptions provided at the DDBJ (DNA Data Bank of Japan) web site [2].
− The **Pfam workflows**, demonstrating the use of the REST web services provided by the Protein Families database (Pfam) [4].
− Some explorative workflows for **linguistic analyses** based on the Wortschatz web services [123].
– The SRTM workflows, demonstrating the use of the srtm-tools package [50] to process data from the NASA’s Shuttle Radar Topography Mission (SRTM) [43].

II. Student projects / coursework (86):
– The student projects from our process modelling courses, which we can not list individually here. At the time of this writing, this collection comprised 86 scientific workflow projects, 34 of which are bioinformatics applications, 37 are from the geoinformatics domain, and 15 realise other kinds of scientific analysis processes. The projects were not predefined by us as lecturers, but proposed by the students themselves. Having already graduated in a natural science discipline or being advanced undergraduates, they were totally able to identify and tackle a reasonable data analysis problem from their respective domain. Thus, even if the scope of these projects was limited due to the time and resources available, they provide representative examples of scientific workflows.

III. Projects from research and cooperation activities (5):
– The LC/MS analysis workflows [51] (one of which is visible in Figure 2) for preprocessing and statistical analysis of liquid chromatography/mass spectrometry (LC/MS) data using the statistics language GNU R [6] and in particular the XCMS package [104].
– GeneFisher-P [62], a workflow-based reimplementation of the GeneFisher [34,37] web application for PCR (Polymerase Chain Reaction) primer design. GeneFisher-P makes it possible to run the primer design process in a batch processing manner, and thus to design primers for large amounts of input sequences automatically.
– Flux-P [29], an approach to automate and standardise 13C-based metabolic flux analysis [125] based on the FiatFlux software [135].
– The Climate Impact Analysis workflow project described in [7], which aims to make customisable versions of the processes implemented in cigrasp (Climate Impacts: Global and Regional Adaptation Support Platform) [133] available to the scientific community.
– The jABCstats project [124] itself, which has been implemented as extensible library of jABC processes, and which is clearly also a scientific workflow application.

Table 1. Project size and complexity.

<table>
<thead>
<tr>
<th>Number of SIBs:</th>
<th>Min.</th>
<th>Max.</th>
<th>Mean</th>
<th>Median</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>all projects</td>
<td>9</td>
<td>797</td>
<td>91.2</td>
<td>55</td>
<td>119</td>
</tr>
<tr>
<td>group I</td>
<td>19</td>
<td>64</td>
<td>41</td>
<td>41</td>
<td>20.2</td>
</tr>
<tr>
<td>group II</td>
<td>9</td>
<td>797</td>
<td>83.8</td>
<td>54.5</td>
<td>101</td>
</tr>
<tr>
<td>group III</td>
<td>39</td>
<td>621</td>
<td>269</td>
<td>98</td>
<td>278</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of SLGs:</th>
<th>Min.</th>
<th>Max.</th>
<th>Mean</th>
<th>Median</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>all projects</td>
<td>1</td>
<td>83</td>
<td>7.14</td>
<td>4.5</td>
<td>11.7</td>
</tr>
<tr>
<td>group I</td>
<td>3</td>
<td>5</td>
<td>4.2</td>
<td>4</td>
<td>0.84</td>
</tr>
<tr>
<td>group II</td>
<td>1</td>
<td>56</td>
<td>5.92</td>
<td>4</td>
<td>7.89</td>
</tr>
<tr>
<td>group III</td>
<td>6</td>
<td>83</td>
<td>31</td>
<td>10</td>
<td>34.2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of hierarchy levels:</th>
<th>Min.</th>
<th>Max.</th>
<th>Mean</th>
<th>Median</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>all projects</td>
<td>1</td>
<td>7</td>
<td>2.11</td>
<td>2</td>
<td>1.32</td>
</tr>
<tr>
<td>group I</td>
<td>1</td>
<td>2</td>
<td>1.2</td>
<td>1</td>
<td>0.45</td>
</tr>
<tr>
<td>group II</td>
<td>1</td>
<td>7</td>
<td>2.08</td>
<td>2</td>
<td>1.28</td>
</tr>
<tr>
<td>group III</td>
<td>2</td>
<td>6</td>
<td>3.4</td>
<td>3</td>
<td>1.67</td>
</tr>
</tbody>
</table>

Note that the complete collection of jABC projects that have already been analysed with jABCstats currently comprises 180 examples from different application domains. Hence, in addition to the 96 scientific workflow projects targeted here, there are results available for another 84 projects from other application areas that we do not consider in this paper. They comprise, for instance, developing applications like the Genesys code generation framework [45], the processes of the Online Conference Service (OCS) [95], and also a number of student projects from Software Engineering courses at the University of Potsdam where the jABC was used for the verification of process models via model checking [20].

3.1.2 jABCstats results

The 96 scientific workflow projects in the sample described above comprise 679 models and 8,650 SIB instances in total. In the following, we take a closer look at the complexity and size of projects and individual models, and at the SIBs that the workflows coordinate. We also compare our results with those from related studies [32,67,122] as we go. Summarising, the results regarding service usage and workflow sizes we have obtained by analysing our scientific workflows with jABCstats do in fact largely comply with the findings described by those studies. However, with the other analyses carried out by jABCstats, such as the McCabe number and the controlflow workflow patterns, we obtain a more comprehensive and more detailed picture, in particular with regard to the complexity of the workflows.

Project Size and Complexity We assess the size and complexity of the scientific workflow projects in the sample by means of the number of SIBs, the number of models, and the number of hierarchy levels they contain. Table 1 and Figure 3 list and visualise the results.

The number of SIBs per project ranges from 9 to 797, with a mean of 91.2, a median of 55 and a standard deviation of 119. There is a significant difference in the average number of SIBs between the projects of the three
groups. Not too surprisingly, the demo and proof-of-concept projects (group I) typically comprise the smallest number of SIBs, with 41 on average. The student projects (group II), already tackling more complex problems, comprise 83.8 SIBs on average. Finally, the research and cooperation projects (group III), which address even more complex computational problems, have an average SIB count of 269. The first histogram in Figure 3 illustrates the frequencies of the number of SIBs per project. It shows that in fact the majority of projects contain only comparatively few SIBs (up to 50 or 100), and that only very few projects actually consist of hundreds of SIBs.

The number of models per project ranges from 1 to 83, with a mean of 7.14, a median of 4.5 and a standard deviation 11.7. There is no big difference between groups I and II here, but the projects of group III clearly comprise larger amounts of models. The second histogram in Figure 3 visualises the frequencies of particular project sizes, measured as the number of models in the project. It shows again that the majority of projects consists of only up to five different models, some comprise between 5 and 20, and only very few projects have more than 20, 40 or even 80 models.

The number of hierarchy levels in the projects ranges from 1 to 7, with a mean of 2.11, a median of 2 and a standard deviation of 1.32. The third histogram in Figure 3 shows the distribution. More than half of the projects exploit the possibility of hierarchical structuring and introduce new levels of sub-workflows. Indeed, we observed that most workflow designers prefer to keep the size of the individual models quite small (see also the evaluation of the model size and complexity in the following), and rather organise several models hierarchically when more functionality is needed. This separates different levels of abstraction, prevents the individual models from becoming unmanageably large (this starts to be the case when the workflow model does not properly fit onto the modelling canvas anymore), and results in small and easily reusable units of compound functionality. This is in good alignment with the results reported by Garijo et al. in their study “Common motifs in scientific workflows: An empirical analysis” [32], where they analysed a sample of 177 Taverna and Wings workflows from eight different scientific application domains and found, among other things, that around 50% of the workflows make use of some kind of hierarchical modelling.

In many workflows in our sample there is only one additional modelling level. Such cases often consist of a SIB that wraps the call to an external service (a data repository or a compute node), surrounded by a workflow that prepares the data and postprocesses the results. However, a number of scientific workflow applications in our collection comprise three or more hierarchy levels. This usually happens in larger workflow projects, which typically realise more comprehensive analyses, where differ-
ent levels of granularity are distinguished and modelled separately. In these projects, the fine-granular models at the lower levels of the hierarchy generally implement quite technical but often also reusable functionalities like, e.g. operations on domain-specific data structures that are not readily provided by the available SIB libraries.

An illustrative example for a project with six hierarchy levels is jABCstats; its call graph is depicted in Figure 4. The “Master” workflow at the top level is only responsible for the selection of the jABC projects to be analysed and for the execution of the different analyses for the chosen inputs. The different analyses are carried out by separate models, which are submodels of the Master workflow. They again comprise submodels that perform the different individual analyses, such as counting patterns, analysing the model hierarchy, or calculating the McCabe number. The leaves of the tree, as for instance the set operations (fifth level) used by the loop detection and the filename retrieval operations (sixth level) for the SIB usage analysis, do indeed implement quite technical but reusable functionality. Note that “Get Model Depth” and “Print Model Hierarchy” (which created this call graph) work recursively, as indicated by the self-referencing edge.

Model Size and Complexity

To get a picture of the size and complexity of the individual models, we consider the number of SIBs per model and their cyclomatic complexity in terms of the McCabe number. Tables 2 and Figure 5 list and visualise the results.

The number of SIBs per model ranges from 1 to 156, with a mean of 12.7, a median of 10 and a standard deviation of 12.9. As discussed above, users typically distribute the functionality to individual models of manageable sizes, and organise them hierarchically along different levels of granularity. Accordingly, the first histogram in Figure 5 shows that models with up to only 10 SIBs are indeed most frequently present, models with 10 to 40 SIBs can still be considered to be quite common, but larger models are rather rare phenomena. The models of the projects of group II are typically the largest, while the project sizes in group III are clearly leading. This can simply be explained by the fact that the developers of the projects of group III, who are typically more advanced users with solid programming experience, more often distribute the functionality across different models and hierarchy levels (as shown above), and consequently the individual models become smaller.

In their study “Analysing Scientific Workflows: Why Workflows Not Only Connect Web Services” [122] Wassink et al. also analyse the size of Taverna workflows models. Similar to our results regarding the size of the individual models, they find that most workflows in their sample consist of less than ten building blocks, workflows with up to 20 can still be considered common, but larger sizes occur comparatively seldom. They report an average workflow size of 8.8 (standard deviation 11.7), which is smaller than our average of 12.7 (standard deviation 12.9). Whereas their range of workflow sizes is from 1 to 70, we have a range of 1 to 156 in our sample, which explains the higher averages despite the similar distribution for the smaller workflow sizes.

Some time later, with more workflows available in myExperiment, Littauer et al. again described typical sizes of Taverna workflows and evaluate how the workflows in the repository are used in their paper “Trends in Use of Scientific Workflows: Insights from a Public Repository and Recommendations for Best Practices” [67]. The average workflow size has almost tripled compared to the previous study, and is now reported to be at an average of 24.6 building blocks (standard deviation 26.6), within a range of 1 to 250. An explanation might be that with users becoming more familiar with the tool and with the tool becoming more mature, larger workflow projects are undertaken. However, histograms show that still most of the workflow are of comparably small size, and large ones are much less frequent.

The McCabe number of the models in our sample ranges from 1 to 55, with a mean of 4.25, a median of 3 and a standard deviation of 5.21. The McCabe number [85] gives the number of linearly independent paths through a process model and is a commonly used quantitative complexity measure for software. Usually, a McCabe number of 10 is considered to be the upper acceptable limit of complexity for programs to remain understandable and maintainable. The second histogram in Figure 5 shows that for the vast majority of models the McCabe number is clearly below this threshold, indicating that the models are usually at a very comprehensible level, and that the model with the McCabe number of 55 is a clear outlier. The models of the projects in group II typically have a higher cyclomatic complexity than those of the other two groups. Presumably, this is related to the previous observation that the model sizes in this group are typically larger, whereas in the other

<table>
<thead>
<tr>
<th>Number of SIBs:</th>
<th>Min.</th>
<th>Max.</th>
<th>Mean</th>
<th>Median</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>all projects</td>
<td>1</td>
<td>156</td>
<td>12.7</td>
<td>10</td>
<td>12.9</td>
</tr>
<tr>
<td>group I</td>
<td>3</td>
<td>23</td>
<td>9.76</td>
<td>8</td>
<td>5.82</td>
</tr>
<tr>
<td>group II</td>
<td>1</td>
<td>156</td>
<td>14.2</td>
<td>11</td>
<td>14.1</td>
</tr>
<tr>
<td>group III</td>
<td>1</td>
<td>44</td>
<td>8.31</td>
<td>6</td>
<td>7.24</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cyclomatic complexity (McCabe number):</th>
<th>Min.</th>
<th>Max.</th>
<th>Mean</th>
<th>Median</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>all projects</td>
<td>1</td>
<td>55</td>
<td>4.25</td>
<td>3</td>
<td>5.21</td>
</tr>
<tr>
<td>group I</td>
<td>1</td>
<td>11</td>
<td>3</td>
<td>2</td>
<td>3.18</td>
</tr>
<tr>
<td>group II</td>
<td>1</td>
<td>35</td>
<td>4.65</td>
<td>3</td>
<td>5.23</td>
</tr>
<tr>
<td>group III</td>
<td>1</td>
<td>44</td>
<td>8.31</td>
<td>6</td>
<td>7.24</td>
</tr>
</tbody>
</table>
groups also the functionality and hence also complexity is more often distributed over several models.

**Control-Flow Structure** In order to get a better understanding of the control-flow structure of the workflows, jABCstats identifies and counts occurrences of the essential **control-flow patterns**: sequence (simple sequential execution of two services), exclusive choice (conditional branchings), simple merge (convergence of branches), loop (repetitive behaviour), fork (split into parallel execution threads), and join (synchronisation of parallel execution threads). Together with the recursions counted by jABCstats during the analysis of the model hierarchy, this corresponds to the basic control-flow patterns and the iteration control patterns described in [119] and at the Workflow Patterns website\(^3\), which comprise 43 control patterns in total. In addition to the essential patterns listed above, some more of these 43 can in principle be found in jABC, but they are currently not (explicitly) considered by jABCstats. For example, the transient trigger pattern (signalling of pre-defined events that initiate another tasks) can be implemented using the **FireEvent** SIB in one SLG and a corresponding Listener SIB in another. In the sample analysed for this paper, the SIB count of these is in fact 0, meaning that this pattern has not been used by any of the scientific workflow projects.

Without going into the details of the usage of the analysed patterns at the individual model level and differences between the groups, the pie chart Figure 6 shows the overall distribution of the analysed patterns in the sample. Not surprisingly, the sequence is by far the most frequently used pattern, accounting for almost 70% of all counted control-flow patterns. As Table 3 shows, it is in fact in 100% of the projects, that is, not a single project does without it. However, in contrast to what is sometimes suggested by the use of mere process chains in the scientific workflow community, it is by no means the only pattern. Exclusive choices, loops, and simple merges also occur quite often, with together almost 30% of the patterns. Only the parallel executions (fork/join) with their 1.3% of the usage are really rare. This shows that scientific workflows are indeed not merely (linear) pipelines of service executions, but rather complex processes whose implementation requires proper control structures.

For instance, intermediate results or user inputs are often used to determine which of several possible processing options or analysis variants to choose, which requires the possibility to evaluate conditions dynamically at runtime. As another example, many workflows in our sample comprise database searches that return a list of results or entries, which then must be iteratively processed in a loop in order to aggregate the relevant content before the next analysis steps start. Thus, workflow systems should provide easy means to include such control structures in their workflows. While this is nat-

---

\(^3\) http://www.workflowpatterns.com/patterns/control/
Table 4. SIB usage.

<table>
<thead>
<tr>
<th>SIB package</th>
<th>% of models using it</th>
<th>% of projects using it</th>
</tr>
</thead>
<tbody>
<tr>
<td>de.metaframe.jabc.*</td>
<td>46.24 %</td>
<td>70.83 %</td>
</tr>
<tr>
<td>de.jabc.sib.common.*</td>
<td>96 %</td>
<td>100 %</td>
</tr>
<tr>
<td>other SIBs</td>
<td>48.8 %</td>
<td>86.46 %</td>
</tr>
</tbody>
</table>

Figure 7. SIB usage.

Figure 5. Model size and complexity.

Figure 6. Usage of control-flow patterns.

In control flow-based systems like the jABC, it is a principal limitation for most data flow-oriented workflow management systems (see, e.g. [54,61,86]).

The usage of the control-flow patterns in the three groups of projects is in general quite similar, but there are also some noticeable differences. For example, the shares of the exclusive choice and simple merge pattern are clearly higher for the student projects (group II) compared to groups I and III. Accordingly, the share of the sequence pattern is somewhat lower. There is no significant difference in the share of loops, forks and joins in the projects. Interestingly, in groups I and III the use of the forks and joins seems to be completely balanced, that is, each fork has a corresponding join, while in group II there are apparently some parallel splits that are not (explicitly) joined again.

Service Usage For the service usage analysis, jABCstats counts the occurrences of all individual SIBs in the workflow models, and also directly computes the usage numbers for superordinate groups of SIBs defined by the package structure. For example, the SIB de.jabc.sib.common.basic.EvaluateCondition will not only count for itself, but also for the packages de.jabc.sib.common.* and de.jabc.sib.common.*, and so on. This allows us to assess SIB usage group-wise, at a higher-level, which is more convenient and also more meaningful than only the basic, fine-granular individual occurrence count.
Table 4 and the pie chart in Figure 7 give a picture of the overall usage of SIBs in our sample. They show that 9.32% of the SIBs in the analysed workflows stem from the `de.metaframe.jabc.*` package. This package is part of every JABC release and comprises special SIBs like the GraphSIBs and MacroSIBs for hierarchical modelling and the Fork and Join SIBs for modelling parallel execution threads. 76.09% of the SIBs belong to the package `de.jabc.sib.common.*`, that is, from the CommonSIB collection for frequently needed, domain-independent functionality that is provided with the JABC standard release described above. That is, roughly 85% of the workflow building blocks in the analysed applications have been taken from the domain-independent SIB libraries that are provided by the framework, and only about 15% of the SIBs (the “other” SIBs) are domain-specific implementations. Many of these domain-specific SIBs act as clients to publicly available web services. They provide, for instance, access to databases for queries and data retrieval, or interfaces for launching remote computational tasks.

Table 4 also shows that while the general picture is again quite similar for all three groups of projects, group III appears to use a greater share of SIBs from the `de.metaframe.jabc.*` and a smaller share of other, domain-specific SIBs than the other two groups. One reason clearly contributing to this is that, as described earlier, these projects use hierarchical modelling more often, using the MacroSIB and the GraphSIB from the `de.metaframe.jabc.*` package.

Interestingly, 100% of the analysed projects use SIBs from the `de.jabc.sib.common.*` package, but only 70.83% use SIBs from `de.metaframe.jabc.*`, and also only 86.46% of the projects use other SIBs. That is, there are indeed several applications that apparently do not need any additional domain-specific SIBs. One major reason for this is presumably that in the student projects the `ExecuteCommand` SIB, which is also part of the CommonSIBs, is often used to call domain-specific tools or scripts within the workflows. The service behind the instance of `ExecuteCommand` would of course count as domain-specific (and would add another 5.14% to the 14.52% of “other”, domain-specific functionality used in the projects), but the SIB itself is domain-independent.

The `ExecuteCommand` SIB is indeed a special building block in JABC’s library. It has been introduced primarily as a means for rapid prototyping of workflows as a part of the domain modelling. It is not meant to be used in actual products or releases, because it typically executes external platform-specific commands that may easily introduce errors and impair the reusability of the respective workflows. For example, many projects use statistics functionality provided by GNU R [6]. The fastest way to use it in the workflows is to provide the functionality in the form of an R script that can be executed via the command line and hence also by the `ExecuteCommand` SIB. However, the paths to the Rscript executable and to the script file itself vary between different machines, requiring reconfigurations or dynamic adaptations to execute the workflow on another computer. The preferred and more sustainable way to go in such cases would actually be to provide the scripts as platform-independent remote services using the jETI technology, so that the workflows using them are independent from such issues. In fact, longer-term projects involving tools of this kind, like GeneFisher-P [62], Flux-P [29] and the climate impact analyses of [7], use jETI and avoid these problems. As the time constraints of the courses often do not allow for the provisioning of new thorough SIB libraries for all projects, we simply accept the use of the `ExecuteCommand` SIB in the student projects.

Table 5 lists the “Top Ten” SIBs over all projects. As the majority of the SIBs used in the analysed projects are from the Common SIBs package (see above), it is not surprising that also these “charts” are clearly dominated by the CommonSIBs. The `PutExpression` SIB, which evaluates an arbitrary context expression and puts its value into the execution context, is most used across all projects and groups. The overall second-most used SIB and the most-used SIB from the `de.metaframe.jabc.*` package is the MacroSIB, which encapsulates the call of a sub-workflow and thus facilitates hierarchical workflow modelling. As discussed before, most workflow projects do make use of hierarchical modelling, and this is also reflected by these numbers.

In their study “Analysing Scientific Workflows: Why Workflows Not Only Connect Web Services” [122] WasSink et al. analyse the service usage of Taverna workflows models. As the title of the paper suggests, they find that external (web) services only account for a comparatively small share building blocks used (22% in their sample), while local tasks, such data management, control-flow handling, or user interaction constitute most of the workflows’ building blocks (57%). Scripts (14%) and sub-workflows (7%) account for the rest. Although technically we evaluated the SIB usage in our sample according to other categories (namely using the SIB packages as a basis), we can see that we obtain similar results. The “other” SIBs, typically wrappers to domain-specific

### Table 5. SIB Usage: Top Ten SIBs.

<table>
<thead>
<tr>
<th>Rank</th>
<th>SIB</th>
<th>all projects</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>PutExpression</code> (Common SIBs)</td>
<td>849</td>
</tr>
<tr>
<td>2</td>
<td><code>MacroSIB</code> (Framework SIBs)</td>
<td>630</td>
</tr>
<tr>
<td>3</td>
<td><code>ShowMessageDialog</code> (Common SIBs)</td>
<td>526</td>
</tr>
<tr>
<td>4</td>
<td><code>ExecuteCommand</code> (Common SIBs)</td>
<td>445</td>
</tr>
<tr>
<td>5</td>
<td><code>ShowInputDialog</code> (Common SIBs)</td>
<td>380</td>
</tr>
<tr>
<td>6</td>
<td><code>ShowBranchingDialog</code> (Common SIBs)</td>
<td>360</td>
</tr>
<tr>
<td>7</td>
<td><code>PutString</code> (Common SIBs)</td>
<td>321</td>
</tr>
<tr>
<td>8</td>
<td><code>ExtractPattern</code> (Common SIBs)</td>
<td>318</td>
</tr>
<tr>
<td>9</td>
<td><code>ShowFileChooser</code> (Common SIBs)</td>
<td>230</td>
</tr>
<tr>
<td>10</td>
<td><code>ReplaceString</code> (Common SIBs)</td>
<td>225</td>
</tr>
</tbody>
</table>
functionality such as web services, cover 14.87% of the SIBs in our sample. The invocation of scripts via the ExecuteCommand SIB accounts for 5.14% of the SIBs in our sample, and the invocation of sub-workflows with the MacroSIB or GraphSIB make up for 8.03%. The remaining 71.96% are local services, mostly from one of the different Common SIB packages. Also the aforementioned study by Littauer et al. [67] observes that most of the building blocks carry out local tasks, and only a relatively small share calls external (web) services.

In “Common motifs in scientific workflows: An empirical analysis” [32] Garijo et al. describe how they identified and quantified typical motifs in a sample of 177 Taverna and Wings workflows from eight different scientific application domains. By motifs, in contrast to the more technical numbers and patterns discussed earlier, they refer to services or service combinations for specific purposes, such as, for example, data preparation or human interaction. Still, their results can be related to our findings and those of the studies mentioned before. For example, they report that the majority of building blocks in the workflows is not concerned with the real computations but rather with various kinds of data preparation, while the actual data analyses only make up for around 10% of the steps in the workflows.

3.2 Experiences

When working with workflow systems that, like jABC, have been designed in a domain-independent fashion, building blocks for the application domain to be worked on have to be provided at some point before the actual workflow execution becomes possible. Hence, we report here on our experiences with the service integration (Section 3.2.1) before we focus on the workflow modelling as such (Section 3.2.2). Further details and examples are available in [56].

3.2.1 Service Integration

Preparing the jABC for concrete workflow projects comprises the definition of the required domain-specific building blocks (i.e. deciding which service libraries should be named and integrated, which SIBs should be available, and how they should be integrated and organised within the jABC’s SIB palettes) and the actual service integration (i.e. providing the defined SIBs by integrating the corresponding services into the framework). Classically, libraries of services would be provided before being used for modelling workflows. With the Prototype SIB and the ExecuteCommand SIB (see above) that are provided by the jABC standard SIB library, however, it is possible to model workflows also when the actually required functionality has not yet been integrated. They serve as placeholders during the early modelling, and are replaced by their thoroughly implemented counterparts once these are available. This way, the workflow design process as such is not held up by the unavailability of domain-specific building blocks. The service integrator can be the same person as the workflow modeller, but is typically someone else. Service integration is usually 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 adequate IT expertise.

Our experience with service integration for the different jABC scientific workflow projects suggests the following aspects as central in this context:

- **Service functionality:** Finding tools and services that provide the “right” functionality required for realising the intended workflows can already be quite time-consuming. Sometimes one comes across curated service directories for particular scientific areas (like, e.g. the BioCatalogue [17] 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.

- **Service access:** Finding services that are also technically suitable for workflow integration means finding 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 fulfil this essential 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. And also licensing issues and availability constraints can be of practical relevance. That is, services might only be available after signing particular agreements, if using the resources for particular purposes, after paying some access fee, or becoming a member of a particular organisation.

- **Service interfaces:** Unfortunately, even services which principally allow for programmatic access cannot always easily be integrated (see e.g. [52,53,70]), e.g. because of difficulty-to-use service interfaces. While this does not prevent the service from being accessed by a SIB, it hampers the integration process and often simply leads to re-programming the functionality.

- **Incomplete or missing documentation:** When working with services of all kinds, one soon realises that exact descriptions of parameters are very important. If one has incomplete lists of possible values of parameters and inadequate explanations, it is usually difficult for an external user to integrate an unknown service (see also [70]). 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.

- **Changing interfaces:**

  As discussed elaborately in [55], scientific application domains are characterised by the long-term availability of their basic computational components. At the same time their concrete service interfaces are subject to frequent changes. For instance, some service providers in bioinformatics are currently abandoning their SOAP-based web service interfaces and follow the general trend towards using REST-style interfaces. Luckily 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 and alternatives or replacements do not cover the entire functionality previously available. 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.

Summarising, service integration is a crucial but at the same time inherently complex and difficult part of workflow projects. While the integration of WSDL-based web services can be automated to a large extent, this is not the case in general. In our experience, in contrast to what is frequently suggested by literature on workflow management, web service interfaces are anyway only available for a small part of the required functionality. Hence, service integration is difficult to automate and remains a great challenge. Still, the availability of the functionality required by the user is of course crucial for the successful application of a workflow system. In fact, we think that the major reason that the Taverna workflow management system is so successful in the bioinformatics domain is because it integrates a number of popular (bioinformatics) Web and SoapLab services automatically, which lowers the cost of entry for users from this application domain [97]. While we see no sense in tailoring the jABC to a concrete application domain just for a potential user group, we see potential in developing ways to support and simplify the creation of domain-specific incarnations of the jABC product line when a concrete demand arises.

### 3.2.2 Workflow Development

The jABC editor provides its users with an intuitive graphical user interface, where the composition of SIBs into SLGs can easily be learned also by workflow developers without a classical programming education. In particular, the graphical process model frees users from having to deal with the syntactical details of a textual programming language. The experiences with the students from our courses and also with project partners using the jABC framework, suggest that it does indeed provide an adequate level of abstraction from classical programming: The graphical workflow modelling as such hardly requires any explanation and after only a short introduction, even non-IT users are quickly able to use it to design and manage scientific workflows according to their needs. Dragging and dropping SIBs from the SIB browser onto 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 [11]. Similarly, enriching the models with custom SIB icons and drawing elements and making use of plugins like the LocalChecker and the Tracer can typically be learned within another hour of instruction.

In fact, the workflows in the courses were built by the students mostly autonomously with only a little support by us lecturers. The feedback on the jABC framework provided by some of the students confirmed our expectation and impression that handling the jABC as a tool quickly becomes intuitive. As an example, the student who developed a workflow for the “Identification of differentially expressed genes” [102], 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. […]”

Also the cooperation projects greatly benefited from the provided level of abstraction. FiatFlux-P [29], for example, provides services that carry out the different flux analysis steps autonomously and makes it possible to assemble them into software workflows that perform automated, high-throughput intracellular flux analysis of high quality and reproducibility. Besides significant acceleration and standardisation of the data analysis, with this workflow-based realisation the user can easily change the analysis workflows, making it easy to perform custom analyses without having to code. As another example, GeneFisher-P [62] was designed to facilitate the user-level definition of workflow variants. Workflows can flexibly be customised by including alternative services for the individual steps of the design process. And indeed the GeneFisher-P and FiatFlux-P workflows have been frequently adapted by their users (biology diploma and bio-engineering PhD students without specific computer science education) according to changing experi-
mental setups. In a similar fashion, the Climate Impact Analysis workflow project [7] aims at enabling users (climatologists) to flexibly define and adapt the workflows for exploring climate change-related information in its geographical context according to their specific needs.

However, the experiences with the different scientific workflow projects with jABC and the feedback provided by the students and other users have revealed that two major needs had not yet been addressed sufficiently:

1. A need for (further) semantic simplification:
A common problem that users of workflow management systems face also exists in jABC3. While the graphical workflow modelling 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.

For instance, the student who developed an “Explainatory data analysis” workflow [120] in one of our courses, 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 which you think it might exist, you have to have a clear idea how this could be named.” Note that principally the SIB taxonomies provided by the TaxonomyEditor plugin are there to help identify services in such cases, but as they are created for each project separately and need to be set up and maintained by the workflow designers themselves, they are initially only of limited practical use and must continuously be adapted.

As another example, a student who created a “Workflow for phylogenetic tree construction” [66], described problems she experienced when trying to use new SIBs in her workflow as follows: “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 [play around with] the parameters if one wanted to see which of them causes an error, because this was not always clear.”

Hence, there is clearly a need for (further) semantic simplification. Domain modelling should not only comprise the integration of services, but also their annotation with meta-data in terms of an ontologically defined vocabulary, so that the technical knowledge about the application domain gets captured in a systematic way. This is the prerequisite for the application of semantics-based methods for service discovery and automatic workflow composition, which can provide higher-level modelling support to the user, freeing her from dealing with the technicalities of workflow development. In Section 4.1 we describe how the PROPHETS plugin addresses this need.

2. A need for more explicit data handling:
The graphical workflow models in jABC3 define the flow of control, while the exchange of data between the individual SIBs is done via the ExecutionContext, a shared memory where the SIBs can put and access data objects. The control flow representation has proven quite easy to grasp, but the ExecutionContext in its current form is apparently less intuitive. With its different scopes and the ContextKeys and ContextExpressions needed to control the flow of data, it is among the jABC concepts that have turned out to be more difficult to understand for new users. For example, a student who developed a “Protein classification workflow” [100] 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. [...]”. In fact, while control flow is nearly self-evident, the ExecutionContext and related concepts have to be explained and practised more elaborately before jABC novices are able to use them proficiently. Furthermore, the untyped ContextKeys also make a type-correct definition of the data flow difficult.

Hence, there is clearly a need for more explicit and more flexible data and data-flow modelling. Especially for scientists for whom the data in the applications is typically central, an increased visibility of the processed data and the data flow in the workflows would be beneficial. Similarly, an easy way of defining custom data structures and their direct availability in a database would be helpful, as are easy means for accessing them from the process models. In Section 4.2 we describe how we address this need with the currently developed jABC4 and DyWA (Dynamic Web Application) frameworks.

4 Further Developments and Future Perspectives
The jABC framework has proven itself to be easily applicable and has been successfully used in the field of scientific workflows, as well as in several other application domains. Nevertheless, we have observed several obstacles in the workflow design process still hampering users without classical programming skills, and also scientists who have specialised in their respective non-IT disciplines. These observations have led to further developments within the framework:

- Addressing the need for further semantic simplification of the workflow design process, the PROPHETS plugin [88] adds means for constraint-driven workflow synthesis according to the loose programming paradigm [63]. With PROPHETS, the user does not
need to do the entire workflow design manually anymore, but can just sketch the most salient elements and let PROPHETS complete it into a compatible and executable service composition.

- In response to the need for more explicit and more flexible data and data-flow modelling, the DyWA [94] and jABC4 [92] frameworks facilitate the definition of domain-specific data models and the graphical definition of data flows in the jABC, respectively. This facilitates the co-design and co-evolution of data structures and workflows that operate on them. The evolutionary aspect is particularly important for workflow variations and customisation, as well as for maintainability and reuse.

All these pieces are going to be combined into one comprehensive framework with the emerging Cinco Metatooling Suite [1,90], the next generation of the XMDD framework, which enables the generation of tailored domain-specific frameworks and tools from metamodels. We believe that with Cinco the jABC as a workflow development environment will achieve a level of domain specificity that drastically simplifies the tasks of the end users, while at the same time maintaining or even enhancing the flexibility needed when dealing with variant-rich application domains. This capability to customise while maintaining compatibility and integrability of the workflows (the design outcomes) would pave the way towards true user-level workflow design for scientific applications, as envisaged and developed in [54] specifically for the bioinformatics domain. The following Sections 4.1, 4.2 and 4.3 introduce the PROPHETS plugin, and the jABC4/DyWA and Cinco frameworks, respectively.

4.1 PROPHETS: Synthesis-Based Loose Programming

Ideally, a scientific workflow framework should “speak the language of the user”, be able to understand the user’s analysis problems and wishes, and provide solutions automatically [54]. Semantics-based service handling approaches 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 this spirit, a number of approaches to semantics-based service discovery and composition in the scientific domain have been seen in the last years, such as the BioMoby web service registry [28,126,131], its successor SADI/SHARE [127,128,131], jOrca [48,84], the Wings semantic workflow system [35] that extends the Pegasus workflow system, ASKALON and also the Declarative Service Flow Language (DecSerFlow) [117].

With the loose programming paradigm [63], we have developed a particularly user-oriented approach towards this aim. Loose programming promotes a form of semantically assisted and model-based graphical software development specifically tailored to enabling workflow developers to design their application-specific workflows in an intuitive fashion. In particular, it aims at making highly heterogeneous services accessible in a uniform way to application experts that need to design and manage complex workflows. After an adequate domain modelling that structures the domain concepts and artefacts in a vocabulary familiar to the users, technically supported by ontologies and taxonomies, the users should ultimately be able to proficiently and efficiently work with a world-wide distributed collection of services and data that populates the design space and refers to those ontologies and taxonomies. They would thus operate as workflow designers and developers while using their own domain language. This infrastructural aspect is well aligned with the semantic web movement (for domain modelling), as well as with the user-driven design paradigms that foresee users at the centre of the design activities. Moreover, loose programming enables users to specify their intentions about a workflow in a very sparse way, by just giving intuitive high-level specifications that refer to concepts and activities from the domain-specific vocabulary, because it offers a mechanism that automatically translates such requests into syntactically correct and executable running workflows.

The PROPHETS plugin [88] to jABC3 is the current reference implementation of the loose programming paradigm. Based on existing model synthesis techniques [77, 112, 114], it provides means for constraint-driven, synthesis-supported workflow design according to its ideas. With PROPHETS, the workflow designer neither needs to model fully executable workflows (as usually necessary) nor to formally specify a synthesis or planning problem in terms of some first-order or temporal logic (as usually required in state-of-the-art planning environments). While behind the scenes the synthesis algorithm still requires formal specifications of the synthesis problem, PROPHETS hides this formal complexity from the user and replaces it by intuitive (graphical) modelling concepts of workflow sketches which build on the (formalised) domain knowledge and artefacts. In other words, it integrates an automatic, application-level service composition methodology into the jABC framework.

To illustrate how workflow design with PROPHETS looks from a user’s perspective, and the role of constraints in this, Figure 8 shows a small example from a case study on Microarray Data Analysis pipelines [54, Chapter 6]. The domain model describes the available services in terms of the service and data type terminology defined by the respective taxonomies. It also comprises a set of so-called domain constraints, which capture general properties of the targeted workflows. Synthesis can then be applied to derive possible concretisations from loosely specified workflows. A very simple, yet common, example of a loose workflow is shown at the lower left of the figure: It starts with the loading of an available data set, and ends with a display of re-
Together with a constraint that enforces the use of an (also recommended) workflows automatically. Accordingly, the exemplary workflow composition problem step, the synthesis solutions will comprise at least two additional steps, as shown in that enforces the use of a possible solutions, consisting merely of a start analysis pipeline. As Figure 6.10 (top) shows, such a minimal loosely specified model Microarray data analysis workflows are typically pipelines (i.e., linear sequences of the expression values are visualized before and after the preprocessing step, and and accordingly the

**Figure 6.9.: Type taxonomy of the microarray data analysis domain model.**

.. .. ..

Synthesis

domain constraints

Loosely specified workflow:

Possible synthesis results:

with no further constraints (only constraints from the domain model):

with additional constraints:

1. Enforce the use of Filtering.
3. If Preprocess or Filter is used, Annotation has to be used next.
4. If StatisticalAnalysis is used, Annotation has to be used subsequently.
5. If Annotation is used, WriteHtmlFile has to be used next.

Figure 8. Constraint-Driven Workflow Design with PROPHETS.

results, this way utilising the synthesis for a comprehensive exploration of the many possible analysis processes. Two possible synthesis outcomes are shown in the figure. If no further constraints are applied, the workflow at the upper right would be a default shortest possible solution. If the five textually described constraints are applied in addition, one of the shortest solutions in this case is the sequence shown below. The rounded rectangles around the SIBs in the workflow show which of the constraints were involved for their inclusion in the solution. Note that the constraints can indeed be formulated in terms of natural-language as indicated in the figure. PROPHETS’ constraint editor provides natural-language templates for the constraints, and services and types instances and classes can be selected from dropdown lists that are derived from the domain model to guarantee that the users find and use the correct terminology.

Like jABC, PROPHETS has been conceived as a domain-independent framework, and the domain modelling is a crucial phase of working with it. Also here, working with users and applications from different domains provided us with rich and valuable experience regarding the usability of the constraint-driven workflow development methodology, which drives our further developments (see next sections). Most of its applications so far have been in the area of scientific workflows (see, e.g. [54,64,65]), predominantly in bioinformatics. Here, the domain modelling hugely benefited from the data type and operation vocabularies provided by the EM- BRACE Data and Methods ontology (EDAM) [42].

4.2 jABC4 and DyWA: More Attention to Ease of Data Management

In our quest for a truly uniform easy-to-use design and development environment, we sought to incorporate easier means to define and manage the data flow in the framework. The recently introduced jABC4 [92,93,96] and DyWA [31,94] frameworks address these needs. The jABC4 is an extension of the third generation of the jABC that, among other things, adds type-awareness to the ExecutionContext (so that type-safety can already be validated during workflow design) and allows for graphically supported data-flow definition to the editor (so that the data flow also becomes visible in the graphical workflow model). Another central idea of the jABC4 is the introduction of higher-order semantics: by treating services and processes as first-class citizens, they may be moved around just like data and plugged and played into activities at runtime, thus enabling higher-order process engineering (HOPE) [92].

The DyWA framework has been designed to “support application experts without programming knowledge to model (according to their professional knowledge and understanding) both the domain-specific data models and the business process models that act on the data via automatically generated elementary data operations” [94]. In essence, it provides a database together with easy means for the definition of domain-specific data structures and to access the database from within the workflows, which can of course also be used in scientific application domains.

A first case study where we use the jABC4 and DyWA frameworks in the field of scientific workflows is currently ongoing in a large interdisciplinary project led by the Cancer Metabolism Research Group at the Institute of Biomedical Sciences at University of São Paulo in Brazil.
There, our frameworks are used to define the data structures and workflows for the collection and analysis of data for cancer and cachexia research [71]. This research project involves extensive collections of detailed sample data within a complex environment that includes (1) clinical and research methods, (2) medical, assessment, and measurement equipment and (3) the regulatory requirements for data privacy, quality assurance, and security. Thus it provides a challenging application and a good evaluation basis for the combined capabilities of the further developing jABC4 and DyWA frameworks.

4.3 Cinco: Generating Tailored Frameworks

The broader use context envisaged for our tools is Service Centred Continuous Engineering [5], which extends the continuous design and continuous deployment of [105] to a continuous lifecycle support paradigm where users co-define and co-manage the design and the evolution of deployed systems. This approach requires the coherent support of different levels of expertise and different concerns among the cooperating participants, and thus requires adaptability also of the technological means, including the design environment (like the jABC) itself.

With the Cinco SCCE Meta Tooling Suite [1,90], a generator-driven development environment for domain-specific graphical modelling tools, the next generation of the XMDD frameworks is ahead. As has already been shown in [91] for an example from concurrent systems modelling, Cinco can be used for the automatic generation of a wide range of graphical modelling tools from an abstract high-level specification.

In fact, we have observed that the more a software framework is tailored to a specific domain or even application, the easier is its uptake especially by non-CS user communities. Thus, we are convinced that Cinco’s capability will open the technologies to a much broader audience by tailoring the development tools beyond the provision of suitable domain-specific service libraries by directly hooking onto domain-specific practices, conventions or individual preferences for workflow modelling.

5 Conclusion

jABC is a framework for process modelling and execution according to the XMDD (eXtreme Model-Driven Design) paradigm, which advocates the rigorous use of user-level models in the software development process and throughout the software life cycle. We have used the jABC in the field of scientific workflows for almost a decade now. The exchange of experiences with other researchers from the scientific workflow community and the work on several case studies has taught us how to best proceed to service-enable different subdomains of scientific processes and workflows and how to use our technology to orchestrate complex analyses of experimental data. In this paper, we have taken stock of our experiences in the field. We analysed scientific workflow applications that have been implemented with the jABC, discussed our experiences with service integration and workflow modelling, and described the developments that are now being made in response to frequently observed obstacles in the workflow design process, especially regarding the adequacy for users without classical programming skills, who are common among scientists that have specialised in their respective fields.

The PROPHETS synthesis framework provides functionality for constraint-driven (semi-) automatic workflow design, and jABC4, among other features, simplifies the definition of the data flow in the workflow models. The Cinco and DyWA technologies allow us to tailor future frameworks to the respective application domains or even concrete project scenarios. In fact, the importance of thorough domain modelling cannot be emphasised strongly enough. It often does take huge efforts, which however usually pay off. Even more, a lack of commitment to proper domain modelling can be sufficient to cause projects to fail. We believe that with the next generation of the XMDD framework that is ahead, we can simplify the provisioning of tools with a level of domain specificity that is adequate for end users, while at the same time maintaining the flexibility needed for dealing with a variant-rich application domain, thus paving the way towards true user-level workflow design.

Acknowledgments

Many thanks go to our students and cooperation partners for all the scientific workflow applications that would not have been implemented without them. We also thank Alexander Wickert for his assistance in the process modelling courses and for his contributions to the service usage analysis functionality of jABCStats.

This work was supported, in part, by Science Foundation Ireland grant 13/RC/2094 and co-funded under the European Regional Development Fund through the Southern & Eastern Regional Operational Programme to Lero - the Irish Software Research Centre (www.lero.ie).

References


