Web-based Collaborative Workflow Design

Dissertation

der Fakultät für Informations- und Kognitionswissenschaften

der Eberhard-Karls-Universität Tübingen

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

(Dr. rer. nat.)

vorgelegt von

Dipl.-Inform. Markus Held

aus Balve im Sauerland

Tübingen

2010

Tag der mündlichen Qualifikation:	07.07.2010
Dekan:	Prof. Dr.-Ing. Oliver Kohlbacher
1. Berichterstatter:	Prof. Dr. Wolfgang Küchlin
2. Berichterstatter:	Prof. Dr. Herbert Klaeren

Dedicated to my family.

Acknowledgements

As much as this thesis is about collaboration, it could not have come to existence without the support from many people.

First, I would like to express my deep gratitude towards my advisor Prof. Dr. Wolfgang Küchlin, and towards Dr. habil. Wolfgang Blochinger for the invaluable guidance they have given to me in the course of my dissertation project.

Dr. Dierk Wanke, Dr. Kenneth Berendzen, and Prof. Dr. Klaus Harter of the *Zentrum für Molekularbiologie der Pflanzen* have given me helpful advice on contemporary biology and cooperated in the supervision of Moritz Werning's diploma thesis. Moritz Werning has been of great influence in the course of the project. Daniel Lehle, Stefan Raue, and Max Günther helped a lot with their undergraduate work. Philip Effinger, Benjamin Albrecht, Stefan Kottler and Prof. Dr. Michael Kaufmann cooperated on BPEL visualization.

Many colleagues at the Symbolic Computation Group and other groups of the Wilhelm Schickard Institute provided me with input, new ideas, and other forms of support. Sven Schulz supervised my own diploma thesis, leading to my first co-authorship of a journal article. I enjoyed many philosophical debates with Eray Gençay and Julia Trieflinger, to which Matthias Sauter provided a healthy down-to-earth attititude.

Many of my personal friends encouraged me to take up the path leading to this dissertation. Some even encouraged me to pursue other projects in the future.

During my undergradute and graduate studies, my fraternity, the *Katholische Studentenverbindung Rechberg*, gave me the room to explore my own creativity, and practice social skills and leadership. Some of my frat brothers I count among my closest friends.

My internship at Titan Corporation was a very special influence in my career, where Barbara and Dave Rogers provided me with outstanding hospitality and friendship.

My parents Helga and Hans-Bernd Held strongly supported my dissertation project, as did my sister Monika Tegethoff and her husband Thomas. I close these acknowledgements in loving memory of my grandparents Maria and Ferdinand Held, and Elisabeth and Helmut Neuhaus.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

One man draws out the wire, another straights it, a third cuts it, a fourth points it, a fifth grinds it at the top for receiving the head: to make the head requires two or three distinct operations: to put it on is a particular business, to whiten the pins is another ... and the important business of making a pin is, in this manner, divided into about eighteen distinct operations, which in some manufactories are all performed by distinct hands, though in others the same man will sometime perform two or three of them.

(Adam Smith, The Wealth of Nations, Pt. I, Ch. 1.)

1.1 Introducing Workflows

Modern societies are based upon the division of labor between differently specialized personnel and machines to achieve common goals. Productive and administrative *services* are implemented as *business processes* consisting of interdependent *tasks*. Such processes may interact with each other and an entity which is viewed as an atomic task on one level of abstraction may be seen as a complex process on a lower level. As some business processes are repeated frequently, expose a complexity which is beyond human administrative skills, or are too valuable to be compromised, the idea of computerized execution arises.

Traditionally, processes are automated by monolithic, hard-coded programs, e.g. a banking application may be encoded as a COBOL program running on a mainframe computer. The underlying business processes of an application may change over time, though. In the example of the banking application, new laws may force a bank to adapt their business process, leading to changes in the program. As well, the business processes of enterprises may differ significantly, although they might share some characteristics.

The adoption of object-oriented programming techniques and distributed object systems in the 1990s has enabled more flexible system architectures, where functionality is provided by remote modules (e.g. Enterprise Java Beans or CORBA remote objects), which are described via *interface definition languages*. A yet more flexible approach is to use scripting languages to couple remote components. Distributed object systems suffer from a lack of compatibility between different vendors, though.

The advent of *Web Services* and *Service Oriented Architectures* (SOA) in the early 2000s marks a turning point in distributed computing, because now a set of commonly accepted standard protocols and interface definition languages has been established. In SOA, *workflows* serve as the glue between business processes and the IT infrastructure. A workflow incarnates a business process as *executable* logic and in a *flexible* way, as it is usually implemented in a specialized scripting language. Such *workflow languages* often can partially be visualized, while some business process *notations* can be compiled to — usually incomplete — workflows. Workflow interpreters (dubbed *workflow engines*) execute workflows, and collect customizable log data, which may serve to analyze, fix and optimize business processes.

1.2 Workflows as a Means and Product of Collaboration

In recent years Scientific Workflows have enormously gained importance [TDGS06], while Workflow Management has been a major aspect of enterprise systems since the 1990s. Based on Web Services, the Business Process Execution Language (BPEL) [AAA$^+$07] has gained the status of a standard language for enterprise workflows. Scientific workflow management, on the other hand, still lacks common standards [TDGS06, YB05, TS07, BvH08], although BPEL has been applied successfully [EBC$^+$05, ESCR07].

Contemporary software development in general is a task for teams, whose members are often scattered among different sites. A trend toward global software development makes it desirable to access resources anywhere and to coordinate development at distant sites [Her07]. Software development teams can be inherently distributed, as scientific research is often conducted in multi-institutional projects. Hence, collaborative software development tools become a necessity [Whi07]. As discussed subsequently, this is especially true for workflow development.

The scientific workflow design process can involve people from very different backgrounds. While some scientists will be eager to work with graphical workflow editors, it cannot be expected that they will fully comprehend the details of a workflow language. As Friese et al. [FSF+06] point out for engineering applications, we have to distinguish between domain and grid/SOA experts involved in workflow design. Domain experts bring in their knowledge about the high-level scientific processes to be modeled, while grid experts utilize their knowledge about grid infrastructure and implementation details.

Workflows often integrate capabilities and resources of distributed sites. Different research groups pool their services, thus forming *Virtual Organizations* [FKT01]. Gil et al. [GDE+07] point out that workflows are used by scientists "to collaborate with each other across organizations and physical locations." The activities of a workflow can thus include services provided by different vendors, business units, or research groups. Hence, the correct integration of these services may require special knowledge only available at a remote site. Moreover, workflow design is the most dynamic part in building and executing advanced grid applications, further driving the demand for sophisticated tool support with collaborative features. Gil et al. [GDE+07] conclude: "Scientific applications are driving workflow systems to examine issues such as [...] collaborative support for workflow design [...]."

In 1987 Osterweill [Ost87] pointed out that "Software Processes are software too," which greatly inspired Software Process Improvement during the 1990s. We argue that workflows are at least as much a product of collaboration as they are a means of organizing collaboration. Summing up, we see a need for tool support for collaborative workflow design in a way that enables easy integration into existing Problem Solving Environments.

1.3 The Case of Business Workflows

The same arguments for collaborative workflow design, we have presented here, also apply to enterprise workflows. In business, distributed software development teams are an effect of globalization and outsourcing. Moreover, enterprise workflows frequently involve several departments, which may be scattered across countries or even continents. Finally, business or engineering processes are designed by domain experts as well.

While contemporary enterprises rely on fast changing business processes, the domain modeling and technical modeling of workflows are not sufficiently coupled, which leads to problems in the design and operation of information systems [TLD+07]. For instance, Thomas et al. [TLD+07] have identified a gap between modeling business processes by domain experts, and their actual implementation by IT experts in executable workflow models:

> Durch die bestehende Lücke zwischen fachlichem und technischem Modell ist keine konsistente Überführung fachlicher Anforderungen in unterstützende IT-Systeme gewährleistet.[1]

1.4 The Case of E-Biology Workflows

Biological *Problem Solving Environments* (PSEs) are frequently provided as web applications. In the trivial case, such a PSE provides a single service or a limited set of services as a web application. Biologists "connect" these services manually by cutting and pasting data between web sites. Since this process is error-prone and hard to retrace, a plethora of different bioinformatics workflow systems have emerged [TDGS06, TS07]. Increasingly web portals are used to access and to share scientific workflows (e.g. [RGS07, CM07]). Some portals provide workflow construction facilities via Java Web Start (e.g. [CM07, SK05]), while true web-based workflow construction tools lack sophisticated user experience (e.g. [CG06, BCMS07]). Often, scientific workflow management systems are provided as desktop applications with rather complex user interfaces (e.g. [OAF$^+$04, SHS$^+$04]).

Similar to programming languages, the complexity of the user interface and the workflow notation remains an area of conflict. The expressiveness of workflow languages is often investigated by their set of supported *Workflow Patterns* [WABA03]. Supporting a large set of different workflows and different types of services increases the complexity of a workflow language, thus reducing usability by domain experts. In contrast, high user-friendliness limits the possible set of workflow patterns and access to arbitrary services. Domain experts should be able to compose and execute workflows in a domain specific modeling system, and still fall back to a collaboration with software engineers if a more complex workflow model is needed. Similar to high-level programming languages that are compiled to a common machine language, we envision the Business Process Execution Language (BPEL) as the "assembly language" of an arbitrary set of domain-specific workflow languages [AAA$^+$07].

[1]engl. "The existing gap between domain model and technical model prevents guaranteeing a consistent implementation of domain requirements in supporting IT systems."

1.5 Goals

To address the issues presented so far, this dissertation aims at three different goals:

1. The dissertation provides a basis for collaborative workflow development by investigating requirements, as well as suitable software processes and software metrics.
2. It also investigates the relationship between process modeling and scientific workflows, leading to a new perspective on scientific workflow development.
3. Finally, we show the applicability of modern Web technology for collaborative and scientific workflow development, thus proving the applicability of the Web 2.0 to software development in general.

1.6 Overview

The rest of this dissertation is organized as follows. In Chapter 2, we introduce the technological background of this thesis and present related work. We outline the development of workflow management systems both in business and in science. In particular, we discuss scientific workflow systems in the life sciences and formulate open issues. Chapter 3 presents our approach of workflow development processes. Here we investigate requirements of collaborative workflow development processes, and apply our findings to the development and use of scientific workflows in biology. Afterwards, we discuss synchronization algorithms in Chapter 4 and workflow metrics in Chapter 5. We present the implementation of the collaborative BPEL development system HOBBES in Chapter 6 and extensions to HOBBES in Chapter 7. Then we present the e-Biology workflow system CALVIN, which is based on the same technology stack as HOBBES, in Chapter 8. The integration of HOBBES and CALVIN in mashups is discussed in Chapter 9. We conclude in Chapter 10 by summing up the findings of this dissertation.

Chapter 2

Workflow Management for Business and Science

This chapter gives an introduction to the technical background and the context of this dissertation. Section 2.1 first introduces workflow management systems as used in industry. This section especially presents the Business Process Execution Language (BPEL), which is a technical basis for our work. Section 2.2 describes the concept of *Scientific Workflows*, which have emerged in the context of *E-Science*. We then proceed by describing Grid workflow systems in Section 2.3. Afterwards, we discuss life science workflows in more detail in Section 2.4, and the application of BPEL to scientific workflows in Section 2.5. We conclude the chapter with a summary of the presented scientific workflow systems in Section 2.6, where we also outline some open research questions.

2.1 Workflow Management Systems

2.1.1 Business Processes and Workflow Management

Georgakopoulos et al. [GHS95] define workflows as “a collection of tasks to accomplish some business process,” where “a task can be performed by one or more software systems, one or a team of humans, or a combination of these.” They define workflow management as “a technology supporting the reengineering of business and information processes,” where a business process is a description of “an organization’s activities implemented as information processes or material processes” [GHS95]. In this context, an information process consists of automated and partially automated computerized tasks, while material processes address the actual manipulation of physical objects.

Van der Aalst et al. [vdAtHW03] identify workflow management as a part of *business process management* (BPM), which they define as “Supporting business processes using methods, techniques, and software to design, enact, control, and analyze operational processes involving humans, organizations, applications, documents and other sources of information." They describe BPM as a lifecycle consisting of the four subsequent phases "diagnosis," "process design," "system configuration," and "process enactment." The latter three phases are addressed by workflow management, while the "diagnosis" phase, where "the operational processes are analyzed to identify problems and to find things that can be improved," is left without tool support.

According to Schmidt [Sch99b] the first workflow management systems were applied to document-oriented business processes, where documents were passed between and processed by different people: "the workflow coordinates the flow of documents between people, enforcing business rules for routing and making deliveries to electronic in-baskets." According to Krafzig et al. [KBS04] the notion of *business processes* became popular in the early 1990s, in the context of *business process reengineering*, i.e. the attempt to improve efficiency in enterprises "by completely reinventing business processes."

An important aspect of business process management is business process modeling, i.e. the analysis, formalization and evolution of existing business processes. Havey [Hav05] claims five benefits of business process modeling:

1. If existing processes in an enterprise are formalized, it is likely that possible improvements are discovered.

2. A formalization enables the automation of process flow. This leads to a more efficient execution of processes, since time for passing data between activities is reduced, and parallelism is exploited.

3. BPM increases productivity and enables the execution of business processes with fewer workers.

4. As BPM enables the automation of tasks, it enables human beings to focus on hard problems, by relieving them from those tasks which can be facilitated by a machine.

5. BPM simplifies the compliance to regulations.

Leymann and Roller [LR00] distinguish between four types of workflows, according to business value and repetition:

- *Collaborative workflows* are of high business value, involve a single large project and many individuals.

- *Ad-hoc workflows* occur to process unforeseen events. Thus, ad-hoc workflows are less formalized than other workflows. For example, policy changes often cannot be predicted, but have to be implemented nonetheless.
- *Administrative workflows* refer to those workflows which are necessary for the operation of an enterprise, but do not directly affect its core business activities. Examples include accounting and controlling.
- *Production workflows*, on the other hand, steer the execution of core business activities.

As workflow systems became popular in the 1990s, a multitude of incompatible workflow management systems emerged. In the appendix of their survey article from 1995, Georgakopoulos et al. [GHS95] list over 50 different systems.

2.1.2 Workflow Modeling

The Workflow Reference Model

The *Workflow Reference Model* (WFRM) was published in 1995 by the *Workflow Management Coalition* industry consortium [Hol95]. The focus of the WFRM is to clearly define a terminology and a reference architecture for workflow management systems.

In the WFRM, a *workflow* is defined as

> The computerised facilitation of automation of a business process, in whole or part.

while a *workflow management system* is defined as

> A system that completely defines, manages and executes “workflows” through the execution of software whose order of execution is driven by a computer representation of the workflow logic.

Figure 2.1 depicts the basic functional characteristics of a workflow management system as defined by the WFRM. According to the WFRM, a workflow management system is comprised of build time tools, which enable the analysis and modeling of business processes and their implementation as workflows, and a workflow enactment service.

The core of a workflow management system is comprised by a *workflow enactment service*, consisting of one or more workflow engines, for the creation, management

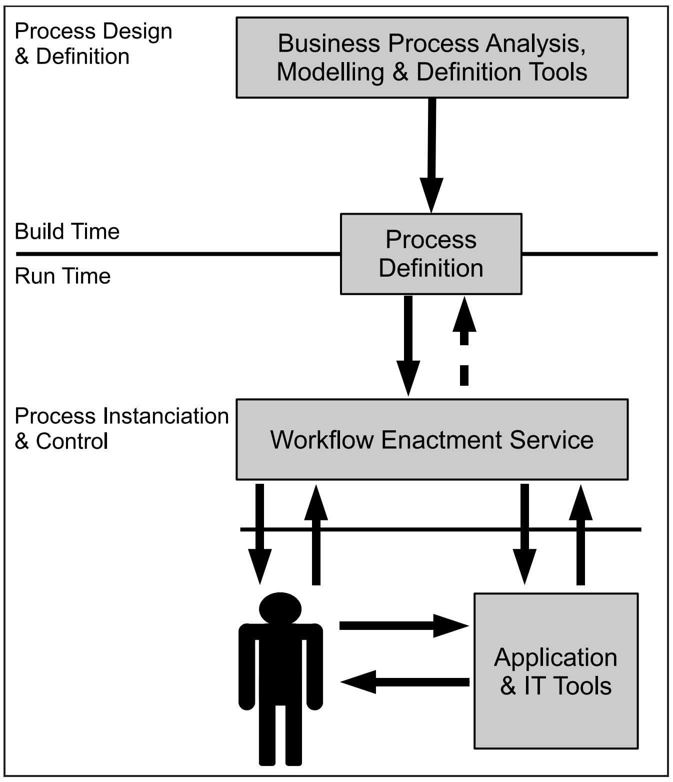


Figure 2.1: The Workflow Reference Model: Workflow System Characteristics

and execution of workflow instances. The workflow enactment service has to provide an application programming interface, which enables access to its functionality. The workflow enactment service is logically separated from those entities actually executing a task. The actual tasks can be processed either by machines or by human end users. Clients can access the workflow enactment service with application programming interface, whose functionality is also defined in the WFRM.

Workflow Patterns

Van der Aalst et al. [WABA03] have introduced the notion of *Workflow Patterns*. A workflow pattern describes a common and basic functional capability which a workflow language or notation may or may not be able to express. Workflow patterns thus can be used to express requirements for workflow management systems and to compare workflow languages and notations. Subsequently, in several papers van der Aalst et al. have compiled a set of workflow patterns from existing workflow management systems and languages, which are presented at the web site `http://www.workflowpatterns.com`.[1]

They distinguish between control-flow patterns, data patterns, resource patterns, and exception handling patterns. An simple example is the *sequence* control flow pattern, i.e. the sequential execution of ordered tasks. *Parallel split*, i.e. the fork of parallel execution threads of activities, is another control flow pattern example.

Workflow Notations

We will now present two commonly used and typical examples of workflow modeling notations, i.e. Event-Driven Process Chains and the Business Process Modeling Notation. In both examples activities are represented as rounded rectangles and the notations are generally control-flow oriented. Control structures are modeled as special nodes, which enable parallel and conditional execution paths by splitting and joining control flow edges. Loops are modeled as circles in the control flow graph.

Event Driven Process Chains Scheer [Sch99a] has presented the ARIS framework (“Architecture of Integrated Information Systems”) as an approach to modeling the structure, processes, information systems and resources of enterprises or other organizations. One major aspect of ARIS is the *Event Driven Process Chains* (EPC) process modeling notation.

[1] Last accessed 27/10/2009.

EPC models describe the control flow of a business process as alternating between *events*, i.e. passive elements describing process states, and *functions*, i.e. active elements describing the execution of an activity. Events are represented as hexagons and functions as rounded rectangles. Control flow is depicted as arrows between the elements mentioned so far. Control structures consist of nodes representing *logical connectors*, which enable parallel and conditional execution paths by splitting and joining control flow edges.

To describe data flow or the shipping of material objects between physical activities, *information*, *material* or *resource* elements, depicted as rectangles, can be added to the diagram and connected to functions as inputs or outputs.

A function can be connected to *organization units* and *supporting systems*, involved in the activity modeled by the function. An organization unit can be a role (e.g. “sales manager”) or a department in the enterprise. Organization units are represented as ellipses containing a vertical line, while supporting systems are depicted as rectangles with two lines parallel to the vertical edges.

Since EPCs are well suited for high-level modeling of cooperative processes, EPCs will at some points be used to illustrate high-level workflow development processes in the further course of this dissertation.

The Business Process Modeling Notation The *Business Process Modeling Notation* is a standard by the Object Management Group, which had originally been developed by the Business Process Management Initiative[2] [bpmb, bpma].

In contrast to EPC, workflow activities are directly connected by control flow edges, dubbed *sequence flows*. Event nodes may be added to a process diagram to describe incidences arising from external causes or caused by the process but affecting the outside world. Every BPMN diagram contains at least one start event and one end event node. Other event nodes include symbols for timeouts or incoming/outgoing messages. BPMN distinguishes between

Every participant in a process is represented by a *pool*, i.e. a rectangle containing all the activities performed by the participant. The pools may be further divided into *lanes* to categorize activities. Activities in different pools are not connected by sequence flows, but by *message flows*.

[2]`http://www.bpmi.org`, last accessed 27/10/2009.

2.1.3 Web Services and the Service Oriented Architecture

We will now outline the structure of the *Service Oriented Architecture* and introduce its main elements. Afterwards, we will discuss the workflow language BPEL in more detail in Section 2.1.5

In a reference model, OASIS defines SOA as “a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains” [OAS06]. The reference model points out that “although SOA is commonly implemented using Web services, services can be made visible, support interaction, and generate effects through other implementation strategies.”

In practice, however, the term “Service Oriented Architecture” is used to describe loosely coupled distributed systems based on Web Services, as defined by the W3C. The W3C defines a Web Service as a service accessible by the SOAP protocol, whose interface is described in a machine-readable format, which is usually but not necessarily the Web Services Description Language.

The appearance of a multitude[3] of standards based on WSDL and SOAP, has led to the usage of the acronym *WS-** to describe WSDL and SOAP related standards. The complexity of the WS-* specifications has led to the inception of the *WS-I Basic Profile*[4] standard by the *Web Services Interoperability Organization*.[5] The goal of the WS-I Basic Profile is to provide guidance on the usage of WS-* standards in order to ensure the interoperability of Web Service implementations. The WS-I Basic Profile includes test scenarios for Web Service software.

SOAP The SOAP protocol is the W3C standard protocol for accessing Web Services [W3C07a, W3C07b, W3C07c]. SOAP specifies the transmission of messages containing XML data over transport layer protocols like HTTP or SMTP. SOAP messages consist of a *body* element, containing the content data, and an optional *header* element, containing Meta-Information. The actual content of the header depends upon the WS-* standards used by the message.

[3]For example, the “Web Services Specifications Index Page” of the Microsoft Developer Network lists 39 different standards:
`http://msdn.microsoft.com/en-us/library/ms951274.aspx`, last accessed 31/08/2009. An overview poster from innoQ.com lists 70 different standards documents concerning Web Services:
`http://www.innoq.com/soa/ws-standards/poster/innoQ%20WS-Standards%20Poster%202007-02.pdf`, last accessed 31/08/2009.

[4]`http://www.ws-i.org/deliverables/workinggroup.aspx?wg=basicprofile`, last accessed 31/08/2009.

[5]`http://www.ws-i.org/`, last accessed 31/08/2009.

The Web Services Description Language The *Web Services Description Language* (WSDL) is a W3C standard for the description of Web Service interfaces. WSDL is an XML-based Interface Definition Language, which can be used to describe services independent of the programming language they are implemented in or the network protocol bindings that they provide. Although WSDL 2.0 is the current recommendation by the W3C, WSDL 1.1 still presents the de-facto standard for describing Web Services.

WSDL documents consist of a root element called *description*, which contains XML namespace declarations. In WSDL 2.0 XML schema types are used as message types to describe input and output formats in the *operations* a service provides [W3C04a, W3C04b, W3C04c]. WSDL operations roughly correspond to the *methods* provided by classes in object-oriented programming. The operation elements are collected in *interface* elements, which define logical access points of a service. The actual *service* types are defined as sets of *endpoints*, each of which provides a specific interface. The services are bound to network addresses and protocols via *binding* elements. Via the *endpoint* and *binding* elements, the different *interfaces* are bound to concrete implementations.

In WSDL 1.1 the types of the input and output messages are defined as *message* elements, containing *part* elements. Each *part* element consists of a name and a reference to an XML Schema type. Also, in WSDL 1.1 interface elements are called *port types*, while endpoint elements are called *ports*. Table 2.1 presents an informal mapping of object oriented concepts to WSDL elements.

OOP concept	WSDL 2.0	WSDL 1.1
interface	interface	port type
method	operation	operation
method parameters, return types	XML schema	message, message part
data types	XML schema	XML schema
class instance	service	service
object reference (network address)	endpoint	port
n/a	binding	binding

Table 2.1: *OOP and WSDL.* Mapping of object-oriented concepts to WSDL elements.

Universal Description Discovery and Integration *Universal Description Discovery and Integration*[6] (UDDI) is an OASIS standard for Web Service registries, which are provided as Web Services themselves. UDDI has been designed for interoperability with SOAP and WSDL. Entries consist of contact information of Web Service providers, a categorization of Web Services based on standard taxonomies, and technical information about the services published by a provider. From 2000 until its shutdown[7] in 2005, Microsoft, IBM, and SAP operated a public *UDDI Business Registry*.

2.1.4 Alternatives to SOA: The REST architecure style

In recent years, Web Services based on the *Representational State Transfer* (REST) architecture style have been promoted as an alternative to SOA and Web Services [FT02, zNS05, PZL08]. In his dissertation, Roy Fielding [Fie00] has introduced the REST architecture style as a description of the architecture of the World Wide Web. The basic idea of REST is to provide services with uniform interfaces as addressable resources, in the same way the World Wide Web provides documents to browsers. Zur Muehlen et al. [zNS05] describe REST as a “retrospective abstracting of the principles that make the World Wide Web scaleable.”

Fielding et al. [FT02] describe REST as a hybrid architectural style that yields layered, cacheable, reliable, code-on-demand, stateless client/server architectures with a uniform interface, which are highly scalable and can be used across organizational boundaries. Statelessness in this context does not preclude the server from state changes, but rather demands that all necessary information for understanding a request has to be included in the message.

A resource is defined as a temporarily changing membership function that maps an identifier to a set of entities, or rather as “the semantics of what the author intends to identify, rather than the value corresponding to those semantics at the time the reference is created” [FT02]. Every resource has a resource identifier that remains static over time, e.g. a web site is identified by its URL, but its content may change. The set of operations for manipulating resources is fixed and limited to creation, update or delete requests, which are encoded in HTTP methods. REST interfaces thus provide functionality similar to the traditional *Create-Read-Update-Delete* (CRUD) database paradigm.

[6] `http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm`, last accessed: 30/08/2009; `http://uddi.xml.org/`, last accessed: 30/08/2009.
[7] `http://uddi.microsoft.com/about/FAQshutdown.htm`, last accessed 30/08/2009.

In a purist REST style, create-operations would be mapped to HTTP-Put, read-operations to HTTP-Get, update-operations to HTTP-Post and delete-operations to HTTP-Delete. Pautasso et al. [PZL08] point out that HTTP-Put and HTTP-Delete are not allowed by all proxies and firewalls and are not supported by XHTML-forms. Some REST web services avoid these problems by using HTTP-Post instead, which compromises the idea of providing a uniform CRUD interface, though.

HTTP services implementing the REST principles are often called *RESTful Web Services*. While REST in principle can be implemented based on a distributed object system or SOA, in practice REST is usually implemented based on the HTTP protocol. Often, data is transmitted as XML or JSON.[8]

In contrast to SOA, no standard REST interface definition language exists, although WSDL 2.0 supports the description of RESTful Web Services.

Pautasso et al. [Pau08] recommend RESTful Web Services for ad-hoc integration, while they describe WS-* Web Services as useful for “advanced quality of service requirements commonly occurring in enterprise computing.” Pautasso has proposed extensions to the BPEL language for supporting RESTful Web Services.

Tim O’Reilly [O’R] identifies RESTful Web Services as a major building block of the Web 2.0, as they enable easily composable (“hackable”), social Web Applications with rich user interfaces. RESTful Web Services can easily be integrated in Web Applications both for client-side GUIs and server-side business logic. Web Applications which mainly consist of the aggregation of RESTful Web Services are often called *Mashups*. Mashups can be interpreted as a Web 2.0 equivalent to workflows. Some mashup systems, e.g. Yahoo Pipes [Yah], even enable the graphical composition of components similar to workflow modeling.

2.1.5 The Business Process Execution Language

Overview

The *Business Process Execution Language* (BPEL) [ACD$^+$03, AAA$^+$07] is an XML-based Web Service orchestration language, which has been standardized by the Object Management Group (OMG). BPEL was derived from two earlier workflow execution languages; i.e. the block-oriented XLANG language by Microsoft, and the graph-oriented Web Services Flow Language by IBM. The current BPEL standard supports Web Services with interfaces defined in WSDL 1.1, although extensions for RESTful Web Services have been proposed.

[8]The acronym JSON stands for *Javascript Object Notation*.

Wohed et al. [WWDA02] have analyzed BPEL based on workflow patterns. They point out that BPEL supports a large set of workflow patterns and that its communication constructs distinguish BPEL from traditional workflow modeling languages. They conclude that BPEL is a "complex language because it offers (too) many overlapping constructs" and that "the simple fact that many of the patterns can be realized using 'XLANG style' and 'WSFL style' illustrates its complexity" [WWDA02].

We will now describe the BPEL language, as defined in the WS-BPEL 2.0 OASIS standard from 2007 [AAA$^+$07]. The first BPEL standard BPEL4WS 1.1 dates from 2003 [ACD$^+$03].

BPEL is an imperative, structured programming language, which contains a mixture of block-oriented and graph-oriented elements. In contrast to most scientific workflow languages, BPEL is a control-flow-oriented language.In contrast to other high level imperative languages, BPEL does neither encompass any concepts of modules, libraries or classes, nor does it include any concept of functions, procedures or closures. Since BPEL processes are deployed as Web Services, every BPEL process can be interpreted as a function, available to other BPEL processes.

Elements

Declarations Connections to communication partners are declared as *partner links*, whose according *partner link types* have been defined in WSDL files. Partner links allow BPEL processes to exchange a series of messages with peer processes.

If a BPEL process is to be called by a communication partner, the execution engine needs to identify the correct process instance addressed by the message. This can be achieved by declaring *correlation sets* for message exchange sequences. A correlation set is a common set of properties for a set of message types, which can comprise a sequence of related message exchanges between two communication partners. The value of a correlation set is determined at the first communication event in such a sequence, and remains constant for subsequent message exchanges.

BPEL *variables* are typed either as *XML Schema* types or as WSDL *messages*. Other BPEL declarations include *event handlers*, *fault handlers*, as well as *compensation handlers* to model transactions.

Activities The BPEL language distinguishes between *basic activities* and *structured activities*. Atomic tasks are modeled as basic activities, which are treated by the workflow as "black boxes." Variable values can be changed using the Assign activity. An Assign activity may contain an arbitrary number of assignment operations, expressed as `copy` elements. The activities Receive and Reply are used to model communication with a client of the BPEL process, while Invoke calls an operation on a Web Service. All communication partners assume roles defined in *partner links* (see above). Input and output data are passed via variable references. Exceptions can be generated with the basic activity Throw and passed on with Rethrow. Other basic activities are used for waiting, for terminating the process, or for validating the adherence of variables to an XML schema.

Most control structures in BPEL are expressed as *structured activities*. In contrast to basic activities, structured activities contain child activities, so BPEL workflow models can be nested.

Sequences and Scopes The structured activity Sequence is used to express the subsequent execution of an ordered set of activities. The functionality of Sequence thus resembles the concept of bracket-enclosed blocks in C-like programming languages. However, Sequence activities cannot be used for local variable declarations.

Instead, the Scope structured activity is used to define subworkflows with local variables, exception handers, and compensation handlers.

Conditionals and Loops BPEL employs XPath as a standard expression language [xpa99]. Expressions are used for conditionals, for triggering links between activities and for assignments. The If activity contains a set of child elements, which are bound to conditions. Either the first child activity, whose condition evaluates to $true$, or a default activity, or no child activity is executed.

The structured activity Pick resembles the If conditional, but does not depend on the evaluation of a boolean expression. Instead, Pick blocks until the occurrence of an event, which can be the reception of a message or a scheduled date.

Loops are declared with the activities While, RepeatUntil, and ForEach.

Parallelism Concurrency can be modeled using the structured activities Flow and ForEach. Flow allows the definition of Directed Acyclic Graphs of activities, while ForEach loops may be marked as "parallel."

Links between activities are always declared inside a Flow activity, and may never form a circle. Links are tagged with *target conditions*. If activity α has been executed and the target condition of a link (α, β) evaluates to $true$, then (α, β) is triggered. If no explicit target condition is set, then it is implicitly set to $true$. Every activity will be executed, if its corresponding *join condition* evaluates to $true$. The join condition is a boolean function of the incoming links, and — depending on the implementation — is by default set either to a disjunction or to a conjunction of the incoming links.

Links declared inside a Flow may cross the boundaries of structured activities within the Flow, with the exception of loops.

BPEL4People and WS-Human Tasks

The BPEL language does not define the integration of human tasks in workflows, i.e. Web Services providing functionality implemented by people. In 2005, IBM and SAP published a joint white paper proposing extensions to BPEL and WSDL for the facilitation of human tasks [KKL$^+$]. Active Endpoints, Adobe Systems, BEA Systems, IBM, Oracle, and SAP subsequently published two draft documents, one describing the implementation of human tasks as Web Services [AAD$^+$a], and the second describing extensions to WS-BPEL to invoke such Web Services [AAD$^+$b]. In 2008, an OASIS technical committee[9] was formed to standardize "WS-HumanTasks" and "BPEL4People," which as of December 2009 has not produced any documents.

2.2 Scientific Workflows

2.2.1 Defining Scientific Workflows

Although the term "Scientific Workflow" has become popular after the year 2000, to the best of our knowledge, its earliest occurrence has been in the title of a position paper at the NSF Workshop on Workflow and Process Automation in Information Systems in 1996 [SV96]. In this paper, Singh and Vouk [SV96] define Scientific Workflows as "coarse-granularity, long-lived, complex, heterogeneous, scientific computations."

Today it is hard to find a definition of the term "Scientific Workflow," that encompasses all existing scientific workflow systems. Ludäscher et al. [LWMB09] point out that "there seems to be no single set of characteristic features that would uniquely define what a scientific workflow is and isn't."

We can identify two strains of development in the history of scientific workflows. On the one hand, parallel programming models from High-Performance Computing have evolved into Grid-based workflow systems for large-scale applications. On the other hand, the needs of laboratory scientists to analyze and organize experimental data and electronic experiments — dubbed *in silico* experiments in biology — has led

[9]OASIS WS-BPEL Extension for People (BPEL4People) TC, `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=bpel4people`, last accessed 28/10/2009.

to the rise of end user oriented workflow management systems, often with graphical user interfaces. Only after the year 2000 has the scientific community slowly begun to pick up influences from business-oriented workflow management, especially with the adoption of SOA standards.

2.2.2 Surveys on Scientific Workflows

This chapter draws from survey works by several authors. In 2005, Yu and Buyya [YB05] presented a taxonomy of grid workflow systems. In 2006, Taylor et al. [TDGS06] published a book on E-Science workflows, presenting several systems and defining research questions. Tiwari and Sekhar [TS07] later have given an survey of life science workflow systems, while Shah et al. [SBL$^+$08] investigated the general use of Web and Grid technologies in life sciences. Deelman et al. [DGST09] gave another survey of contemporary scientific workflow systems based on the end user perspective. Gil et al. [GDE$^+$07] report of research questions formulated at the National Science Foundation Workshop on Scientific Workflows of 2006. Barker and van Hemert [BvH08] targeted the specification of research directions in their survey, as well. Recently, Ludäscher et al. [LWMB09] gave a survey, aiming at a comparison of Scientific Workflows and Business Processes.

2.2.3 Motivations for Scientific Workflows

Relief of Repetitive Manual Tasks In many cases, scientific analyses are comprised of passing data between individual processing systems. If these scientific analyses have to be repeated, scientific workflows give the chance of relieving human beings from the coordination tasks, which can be automated.

Explication and Traceability of Research Processes If a researcher manually concatenates computational tools or web applications, all input, output, and intermediate data, all output-input relationships between tools and all execution parameters have to be recorded, to make the process traceable. As this is a cumbersome and error-prone task, a major motivation for Scientific Workflows is to explicate scientific research processes, and to automatically persist results. Gil et al. [GDE$^+$07] identify the “reproducibility of scientific analyses and processes” as a major application requirement for scientific workflow systems. This goal is often dubbed *provenance*, since it enables retracing the origin of scientific results [DF08]. Other aspects of provenance include the question, *who* developed or executed a workflow, and *how* workflows are related to each other.

Large-Scale Data Processing Some scientific workflow systems target the automation of large-scale data processing, where a large number of tasks are generated in order to parallelize data-intensive problems. In this case, workflows are used as one specific kind of parallel programming model for data-parallel applications. Section 2.3 gives examples of Grid workflow systems targeting large scale data processing.

Data Aggregation from Distributed Sources Another application of scientific workflows is the aggregation of data from locally available or remote data sources. For instance, many databases worldwide contain experimental or derived data from genomics, proteomics and other sub-disciplines. A typical application of bioinformatics workflows is to coordinate queries of several such databases to investigate relationships between genes from different species. The application of workflows in bioinformatics is discussed in more detail in Section 2.4.

2.2.4 Differences to Business Workflows

Development by Domain Experts In Business Process Management, domain experts and software engineers play different roles in workflow development. Since domain experts have deep knowledge about the processes to be modeled, while software engineers are needed to implement workflow in a low-level language like BPEL, a major concern is to facilitate and coordinate the cooperation between these groups. Scientists, on the other hand, often want to explore the possibilities of a workflow by themselves, before actually laying down a precise definition. In contrast to BPM, Scientific Workflow Management lacks a clear distinction between business processes to be modeled and workflows, that implement them. Ludäscher et al. [LWMB09] point out that the main goal of BPM is to model business processes in order to enable later implementations, while scientific "workflow designs can be viewed as executable specifications." Hence, "workflow modeling and design has not yet received the attention it deserves in scientific workflows" [LWMB09], as contemporary scientific workflow models are not sufficiently easy to reuse and evolve.

Predominance of Machine Tasks Business Workflows often contain human tasks, which are processed by one or several persons. Scientific workflows, however, usually consist of automated tasks like database queries or computations only.

Human intervention, if possible at all, is usually limited to “runtime decisions [that] occasionally require user input” like branching decisions, authentication information, or providing missing parameter values [LWMB09].

Data-Flow Orientation The Business Process Execution Language is a control-flow oriented workflow language. Scientific workflow systems, however, often use data-flow oriented languages. In many cases, the distinction between data-flow and control-flow oriented workflow languages is not clear cut, as some data-flow oriented languages contain some control-flow oriented concepts like loops or conditionals.

No Need for Transactions Scientific Workflows are used to generate new scientific data from mining existing data sources. Thus, in contrast to business workflows, Scientific Workflows usually do not alter database entries, so failure of a Scientific Workflow does not imply the necessity to “roll back” the effects of any activity.

2.3 Scientific Workflows on the Grid

2.3.1 Grid Computing

Grid Computing aims at harnessing idle compute cycles or other resources within *Virtual Organizations*, i.e. a dynamic set of individuals or institutions who are sharing resources [FKT01]. The main motivation of Grid Computing lies in enabling large scientific applications, which otherwise would be impossible to tackle with the resources of individual institutions. While many Grid Computing applications are based on embarrassingly parallel problems (e.g. all applications in the BOINC[10] Desktop Grid), and others rely on complex programming models like Message Passing (based on MPICH-G[11]) or strict multithreading (e.g. applications of the SATIN[12] system), other applications can be decomposed into control-flow and data-flow graphs *a priori*, thus lending themselves towards the workflow metaphor. We will now present a selection of Grid workflow systems.

[10] `http://boinc.berkeley.edu/`, last accessed 22/11/2009.
[11] `http://www3.niu.edu/mpi/`, last accessed 22/11/2009.
[12] `http://www.cs.vu.nl/ibis/`, last accessed 22/11/2009.

2.3.2 Predecessors of Grid Workflow Systems

Coarse-grained parallel programming models can be interpreted as predecessors of grid workflow systems. In the 1990s, graphical development tools for some parallel programming models have been presented, which resemble contemporary graphical workflow editors.

In 1992, Newton and Browne presented the “CODE 2.0” programming environment for the definition of directed data-flow graphs in a GUI [NB92]. Arcs represent data-flow dependencies between tasks, while nodes represent either sequential computations or other “CODE 2.0” graphs. Every node is associated with a “firing rule,” that governs if the node is ready for execution whenever a new data item arrives. The graphs may contain cycles to model loops, where the loop condition is controlled by a designated node. CODE 2.0 data-flow graphs are compiled to Ada source code for shared memory MIMD machines.

In 1993, Beguelin et al. [BDG+93, BDG+94b] introduced the “Heterogeneous Network Computing Environment” (HeNCE), which enabled the GUI-based, explicit parallelization of Fortran or C programs as control-flow DAGs of independent processes. HeNCE DAGs are compiled to parallel PVM programs, which can be deployed on clusters.

Individual nodes represent either subroutines or control structures [BDG+94a]. Edges between subroutines represents control flow dependencies. The HeNCE control structure nodes are conditionals, loops, *fans* and *pipes*. Data-flow between subroutines is based on parameter variables. Subworkflows enclosed by a *Fan-In* and a *Fan-Out* node are replicated for parallel execution. The number of replicates depends on the output of the subroutine represented by the *Fan-In* node. Similarly, subworkflows enclosed by a *Pipe-In* and a *Pipe-Out* node are expanded for parallelism, but with additional dependencies between instances of the same subroutine.

In 1997, Kacsuk et al. [KDF97] presented the Graphical Environment for Parallel Programming (GRADE) system, which is an ancestor of the P-GRADE workflow portal. GRADE contains a graphical editor for the graphical language GRAPNEL [KDF96]. GRAPNEL programs are compiled to parallel C programs based on MPI and PVM. GRADE “generates all message-passing library calls automatically on the basis of the visual code.” Parallel programs are dubbed “process groups” and can be nested. GRAPNEL defines two different, graphical views on processes. In the “Process Communication View,” individual processes and nested “process groups” are depicted as boxes, with attached “ports.” The message passing “channels” between processes are visualized as connections between “ports.” The “Process Internal View” is a diagram visualizing the control-flow code of a process and "communication actions," i.e. abstractions of message passing operations.

BioOpera From 1999 until 2003, the BioOpera[13] project at ETH Zürich investigated the development of bioinformatics workflows for Computer Clusters [Alo, BPSA02]. After 2004 it has been continued in the JOpera[14] project, with a refined focus on general Web Services workflow development.

BioOpera workflows consist of sequences of tasks with explicitly defined dataflow dependencies. The workflows can be edited, executed and monitored in a GUI.

Available activities are registered in an SQL database containing descriptions of their input and output interfaces. BioOpera can invoke Linux, Solaris, Windows or MacOS programs or services based on RPC, CORBA or SOAP.

2.3.3 Grid Workflow Management Systems

Condor DAGMan Condor is a batch system for computer clusters and computational Grids, which has been developed since 1988, containing a task-based workflow manager called *DAGMan* since the late 1990s [TDGS06, ch. 22]. The acronym "DAGMan"[15] stands for *Directed Acyclic Graph Manager*. DAGMan workflows consist of task specifications which are connected via file-based communication. The tasks are executed as jobs by the Condor system, which maps them to resources. Optionally, preprocessing and postprocessing scripts can be specified to run on the central node before or after the execution of a job. Only acyclic dependencies are allowed, making DAGman resemble a parallelized version of the classical *make* tool.

A task can be specified to be retried a fixed number of times in the case of a failed execution. In the case of a task failure, DAGMan executes the rest of the tasks until no more progress is possible. Afterwards, based on the original workflow, DAGMan constructs a "Rescue DAG" file, where those tasks which have been successfully completed are marked as "done."

Pegasus The Pegasus system is a planning tool for generating executable, task-based workflows from abstract workflows, runnable on Grids, Condor clusters, or

[13]`http://www.iks.inf.ethz.ch/projects/projects/bioopera/`, last accessed 31/08/2009.

[14]`http://www.iks.inf.ethz.ch/jopera`, last accessed 31/08/2009.

[15]`http://www.cs.wisc.edu/condor/manual/v7.0/2_10DAGMan_Applications.html`, last accessed: 21/08/2009.

local machines [TDGS06, ch. 23][DSS$^+$]. Deelman et al. [DSS$^+$] distinguish between *workflow templates*, providing the general control and data flow structure of a workflow, *workflow instances* or *abstract workflows*, additionally containing input data, and *executable workflows*/*concrete workflows* including a mapping of tasks to resources as well as management tasks (e.g. for staging in data). Pegasus facilitates the mapping of workflow instances to executable workflows, with file-based communication between tasks.

Pegasus targets large-scale workflows. It has been applied in the Grid Physics Network (GriPhyN) in conjunction with an AI-based planner for the creation of abstract workflows from a Virtual Data Language. In bioinformatics, Pegasus has been applied to perform two large scale runs of the BLAST[16] algorithm, producing "on the order of 10,000 jobs and approximately 70 GB of data" [TDGS06]. Other application areas include astronomy, high-energy and gravitational wave physics, as well as earthquake science.

ASKALON The ASKALON system enables the composition and execution of task-based workflows for Computational Grids [TDGS06, ch. 27][FPT].

Workflows can be edited as UML 1 activity diagrams using the graphical editor Teuta, a Java application [PFT$^+$04, PQF]. The UML diagrams are compiled to the control-flow oriented XML-language AGWL, which supports conditionals, sequential and parallel loops, DAG parallelism, and nested subworkflows. Data-flow in AGWL is defined by connections of input and output ports of activities. AGWL activities can be tagged with resource constraints, e.g. minimum amount of RAM, which have to be fulfilled by the execution system.

The ASKALON workflow execution engine uses a resource manager and a scheduler based on the Globus toolkit. For performance reasons, the scheduler initially transforms the workflow to a DAG representation by making assumptions about the decisions in conditionals, i.e. branch predictions, and the behavior of loops [TDGS06, ch. 27]. The DAG tasks are tagged with *execution contracts* representing the initial assumptions and then scheduled for execution. During execution, incorrect assumptions, i.e. violations of the execution contracts, generate events for the scheduler. The scheduler processes these events by sending a notification of the current state of the workflow execution to the execution engine. The execution engine then generates a new workflow description, which is again sent to the scheduler.

[16]The acronym BLAST stands for *Basic Local Alignment Search Tool*.

Triana The Triana[17] Workflow System originally was developed for the GEO 600 project, which investigates gravitational waves using a laser interferometer [GEO] [TDGS06, ch. 20]. It is based upon the GridLab Architecture for Grid Computing [ADD$^+$03], which provides more abstract access to Globus Grid resources via the *Grid Application Toolkit* API (GAT).

The Triana Workflow System enables the construction of workflows as directed graphs of tasks, where tasks and task types are called “Tools,” and connections “cables.” Every input or output port has one input and one output data type, which may be defined as a disjunction of several basic data types. For example, a port can consume a string input only, or consume either a string or a number input. Tools can be executed once input data for all cables is available.

Triana supports conditional splits and joins as special tools. Triana workflows can be nested to enable clear arrangements of tasks on the screen. Loops can be constructed using a special “loop tool” task [Wan03]. The looped sub-workflow is either nested in the loop task, or connected to special in/out ports of the loop task. In the standard case, the exit condition of a loop can be given as a boolean expression of plain arithmetic equations or inequations. The expressions may contain constants and variables representing the ports of nested tasks, the data passed by the current loop state, or the current number of iterations. Custom exit conditions can be implemented as Java classes.

Triana workflow models are stored in XML files. Triana components are implemented as Java classes, containing public variables and a *process* method. The variables are mapped to input and output ports, as specified in an XML document, similar to the definition of Web Service interfaces in WSDL files.

Figure 2.2 gives an overview of the architecture of Triana. The entire system has been implemented as a Java application. Triana workflows are composed using a GUI, but can also be executed from the command line.

Triana supports workflows consisting of local Java components, grid jobs, and service invocations. Grids are accessed using the *Grid Application Toolkit* API (GAT), while services are embedded via a *Grid Application Prototype* Interface (GAP). GAP provides bindings to several service protocols, i.e. Web Services and Grid Services respectively, JXTA, and P2PS. Web Services can be integrated into the Triana toolbox from UDDI registries. Using UDDI as well as the respective discovery mechanisms from JXTA and P2PS, GAP populates the Triana toolbox.

Triana uses GAP to allow two different modes of parallel task execution. In the first case, the Triana coordinator receives the results of all tasks and farms out new

[17] `http://www.trianacode.org/`, last accessed 22/12/2009.

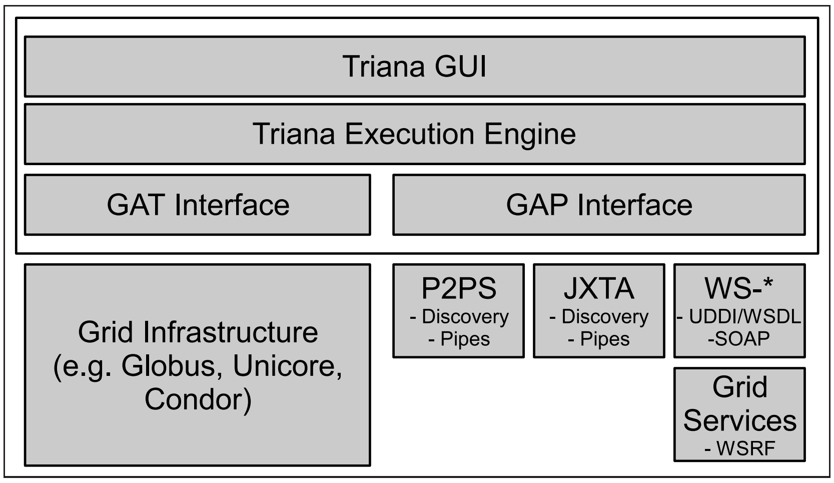


Figure 2.2: The Architecture of the Triana Scientific Workflow System.

tasks as soon as their input data is available. In the second case, the Triana tasks know about their individual data dependencies and communicate directly. While the former case allows a more dynamic aggregation of resources, the latter approach avoids that the coordinator becomes a “bottleneck.”

Kepler The Kepler workflow system is an extension of the Ptolemy II modeling system, which originally was developed for designing embedded systems [EJL+03][TDGS06, ch. 7]. Kepler and Ptolemy II have been implemented as Java applications. It has been applied to domains like chemistry, ecology, geology, molecular biology, phylogeny, and oceanography.

In the Kepler system, tasks are represented by *actors*, which are implemented in the Java language, and may be used to “wrap” code from other languages or from Web Services. Every actor has a set of typed input and output *ports*, which can be connected via dataflow edges transporting *data tokens*. Besides ports, actors expose *parameters*, which can be set at build time. Nested subworkflows are represented by *composite actors*. Loops can be implemented by connecting one output port of an actor, with an input port of the same actor. The “BooleanSwitch” actor is used to implement an “if-then-else” conditional. Workflows can be prototyped using a non-functional “Design” actor which can be annotated with design information and later be replaced with other actors.

Directors are a unique feature of the Kepler language. A director is an execution strategy for a workflow, governing when and how actors are activated for execution.

Changing the director of a workflow thus changes the semantics of the execution. For example, the “SDF Director,” will only permit the sequential execution of actors, while the “PN Director” will try to exploit parallelism. The question, whether a loop will deadlock depends on the director, as well as the behavior of a conditional.

The Kepler distribution contains 350 actors, which among other things enable user interaction, graphical output, file i/o, web service invocations, database access, using the statistical programming language “R.”

P-GRADE The *Parallel Grid Run-Time and Development Environment* Grid portal provides editing and execution facilities for task-based workflows [SK05, SK][TDGS06, ch. 18].

Workflows are represented as DAGs with data flow edges representing file-based communication between individual jobs. Input files of a job can be either copied from the workflow developer’s desktop PC, from a storage resource in a Grid, or from the output of another job. Output files can be stored on the portal server or a storage resource. Jobs can be sequential or parallelized via MPI or PVM, and will only be executed once all necessary input files are available. The workflow manager then transfers all input files and the job executable to the computational resource allocated for the execution of the job.

In the P-GRADE workflow editor, the workflow graph is constructed via drag and drop, while nodes representing jobs are configured in form windows. Input and output files are represented as “ports,” which can be configured in form windows as well.

P-GRADE provides a version of the workflow editor which supports the collaborative editing of workflows by locking elements of the workflow. If a user wants to edit a workflow component, he/she has to acquire a lock on the component from the P-GRADE portal server first. New nodes can be added every time to the workflow and are automatically locked for the creator. A link between two nodes can only be created, if both nodes are locked by the same user.

Modifications of locked components are performed locally first. At any time, users can upload their version of the workflow model to the portal server, which integrates their changes into a global workflow model. Similarly, they can download the current state of the global workflow model at any moment.

The P-GRADE portal has been implemented on top of the Globus toolkit, and can execute jobs in single or multiple Globus virtual organizations. The workflow editor has been implemented as a Java Web Start application.

2.4 Life Science Workflows

In this section, we review the development and current state of workflow technology for the life sciences. We will first give an overview of the development of life science workflows, by presenting a selection of bioinformatics workflow systems. We will then discuss three systems in detail, the BioMoby Web Service registry, the Taverna workflow management system, and the ^{my}Experiment workflow sharing platform.

2.4.1 Bioinformatics Workflow Management Systems

Data processing pipelines implemented in scripting languages like Perl can be seen as the predecessors of modern day bioinformatics workflow systems.

LabFlow In 1995 Rozen et al. [RSG95, GRSS98] presented the LabBase system, a specialized database management system for biological laboratory genome data, with a query language based on Prolog. LabBase was later extended with a workflow component called LabFlow [SRG94, BSR96, GRS98]. Stein et al. [SRG94] identify the application area of LabFlow as "large semi-automated laboratory projects," where "experimental protocols are complex and involve many steps typically with multiple branch points and cycles." LabFlow has been implemented as a Perl library, which interprets textual workflow definitions.

The workflows are expressed as tables describing data-flow DAGS, where nodes correspond to "steps," expressed in the Perl language. Steps process and generate tokens, which represent biological data items from LabBase. LabView supports the parallel evaluation of DAG paths as well as subworkflows.

The functionality of a step is encapsulated in the "execution" method of a "Worker" object, which is called once the step is ready for execution, takes one or a list of tokens, and returns a token [GRS98]. For every step a Unix process is instantiated at the beginning of a workflow.

Discovery Net The service-based Discovery Net was one of the first workflow systems targeting bioinformatics. Its goal is to "allow scientists to create meaningful data analysis processes and then execute them using an underlying Grid infrastructure without being aware of the protocol used by individual services" [RKO+03].

Based on the Discovery Net API, graphical clients have been provided for workflow construction and execution. Discovery Net workflows are directed acyclic graphs, given in an XML-language. Workflows consist of components, defined by a *service* *descriptor*, providing a description of the corresponding input and output data types, and user parameters. The components are plain HTTP-based services, Web services using the SOAP protocol, or OGSA Grid services.

The Discovery Net system primarily consists of a component service and an execution service. The component service manages access to components/services, while the execution service interprets workflow models, eventually invoking services via the component service. Discovery Net includes standard Data Access, Computational and InfoGrid services, to simplify development of specialized components.

The system has been applied to Nucleotide Annotation, Protein Annotation, and Process Annotation workflows.

Sight The Sight system enables the construction of tree-like workflows from web applications with HTML form interfaces [MLHJR04]. For a given HTML form, Sight generates an agent representing the implemented service with input and output fields. Users can compose workflows by connecting the output fields of one agent with the input fields of another agent. Sight can be interpreted as an early example of a scientific mashup system, although it does not support RESTful Web Services.

Pegasys Pegasys enables the creation and execution of task-based workflows, which can be run as sequential jobs on a Cluster [SHS$^+$04]. Workflows are represented as DAGs, where edges represent control-flow dependencies. Vertices represent either programs or data input sources or output sinks, which are available as predefined components. Task parameters are configured in property windows. Components are Unix programs, which are integrated in the Pegasys system via proxy classes.

While the server has been implemented in Java, the Pegasys GUI is based on the Qt C++ library. Pegasys workflows are submitted as jobs to the OpenPBS batch queuing system and executed sequentially.

Wildfire Wildfire is a task-based bioinformatics workflow system, where atomic tasks based on the EMBOSS application suite are executed on computer clusters or Grids [TCH$^+$05]. Wildfire uses a control-flow oriented graphical notation, where nodes represent atomic tasks, or composite constructs like parallel or sequential loops, conditionals, or sub-workflows. The parameters of atomic tasks are edited in form windows similar to the JEMBOSS GUI. Data dependencies between atomic tasks represented as textual parameters. Instead of a boolean expression language, Wildfire conditionals and loops executed external programs, whose output is interpreted as “true” or “false.” Control-flow links may only form a directed acyclic graph, and may not cross the boundaries of composite constructs. Besides programs from the EMBOSS suite, users can add their own command line programs.

KDE Bioscience Lu et al. [LHC$^+$06] presented the Java-based KDE Bioscience platform.

Workflows are dataflow DAGs consisting of different types of data processing nodes representing “more than 60 commonly used bioinformatics programs covering the alignment of nucleotide and protein sequences” [LHC$^+$06], special data input, output and visualization nodes, and a selection of web site hosted applications and Web Services respectively.

Lu et al. [LHC$^+$06] point out that “it is not feasible for a genome project to be handled by only one person,” so that “typically, a team will be formed involving several experts who focus on different parts of the project” [LHC$^+$06]. They conclude that “it is necessary for an integrative informatics platform to provide a mechanism for biologists to work together and share data and designs or even construct the same workflow.” To enable cooperation in a team of biologists, in KDE Bioscience “users belonging to the same group can share a public user space,” i.e. a common directory of workflows and data items, so that “the same workflow can be edited and executed by different users in the same group” [LHC$^+$06].

KDE Bioscience provides a client Java application for editing workflows, which communicates with an EJB-based server deployed in JBOSS. The server contains an execution engine, a processing nodes manager, and access to storage facilities.

BioWMS The BioWMS system enables the web-based construction and execution of service-oriented workflows in a control-flow-oriented workflow language [BCMS07]. Workflows are stored as XPDL, while the used notation resembles BPMN.

The user interface is based on HTML pages, which are dynamically generated on the BioWMS server. Workflow elements are depicted by images embedded in HTML. Any changes on the workflow lead to the new generation of the HTML page on the server.

BioWMS supports the sequential and parallel execution of activities, conditionals and loops. Nested subworkflows are called “complex activites.” Parameters and data flow connections of activities are configured in HTML forms.

Biowep Romano et al. [RBB$^+$07] presented the Biowep portal for the sharing and execution of workflows from BioWMS and Taverna. The workflows available on the Biowep server can be searched based on annotations, or listed by popularity or application domain. The central Biowep portal server[18] currently does not seem to be available.

Bio-jETI The Bio-jETI system is an Eclipse-based workflow design system, targeted at enabling collaboration between software engineers and biologists, by providing a graphical workflow editor, which compiles source code for lower level workflow execution languages or programming languages [MKS08, LMS09].

Workflows are visually defined as directed control-flow graphs with fork/join parallelism, where nodes represent service invocations or sub-workflows, and data-flow between nodes is configured in forms. The workflows can be executed in the Bio-jETI editor, deployed as Web Services in an AXIS Web Service container, or compiled to Java, C++, or BPEL.

VisTrails VisTrails is a desktop-based Workflow Management System implemented in Python, which has originally been intended as a visualization system (“visualization trails”) [SKS$^+$08]. Workflows are represented as dataflow graphs which may contain loops and conditionals.

VisTrails automatically saves all changes that a user makes to a workflow in a workflow version tree, which can be stored in a database. Optionally the version tree can be accessed by more than one user to enable collaboration [EKA$^+$08]. In contrast to a version control system, changes are not automatically merged, but are represented as new branches of exploration, where users may switch between the different versions.

While VisTrails is not a life science workflow system, it is a unique approach to scientific workflows in its emphasis on workflow versioning.

2.4.2 BioMoby

BioMoby is a repository which enables semantic searches for bioinformatics Web Services [WL02, The08]. BioMoby Web Services are published in the BioMoby Central repository and have to adhere to certain conventions to enable their seamless communication.

The BioMoby system contains three ontologies, which are used to describe biological namespaces (i.e. bioinformatics database identifiers), biological data objects (e.g.

[18] http://bioinformatics.istge.it:8080/biowep/, last accessed 26/08/2009.

genome sequences), and BioMoby service types. The BioMoby namespace "ontology" is a list of strings, which are accepted as namespaces by the BioMoby system. The BioMoby service ontology is a hierarchy of "is-a" relationships. In the BioMoby object ontology, the relationship between object types is described using the relations "HAS-A" and "IS-A." For example, a gene sequence *is a* sequence, which *has a* length field, which *is an* integer. The ontologies can be extended by the end users, i.e. the service providers, to accommodate new service categories, database domains, or data types.

A *BioMoby object* is a representation of a biological entity with a database namespace and an identifier. It can be identified by a *Moby triple* of the form `<Object namespace="" id="" />`. In particular, BioMoby services communicate via String parameters containing XML data which describes *Moby objects*.

For example, the Moby triple

```
<Object namespace="Genbank/AC" ID="AY070397.1"/>
```

identifies an Arabidopsis APETALA3 mRNA sequence, which can be given as a BioMoby object of the form:

```
<Sequence namespace="Genbank/AC"
                 ID="AY070397.1">
  <Length>960</Length>
  <SequenceString>
   aacaaaaagattaaacaaagagag
   aagaat atggcgagag ggaagatccagat
   caaga. . .
  </SequenceString>
</Sequence>
```

Optionally, BioMoby objects can contain a *cross references* block, containing a list of Moby triples which identify the same biological entity referenced by the Moby object in the context of other databases.

For every available Moby service, the Moby Central server holds a description including the service name, the service type, input and output objects, a URI identifying the service provider, a URL pointing to the service, and a textual, human-readable description. The Moby Central server can be queried using an API and returns a dynamically generated WSDL document in case a service is to be invoked.

Wilkinson has presented the GBrowse Moby browser for the BioMoby Central registry, which enables users to invoke services without having to use another client

[Wil06]. GBrowse Moby has been implemented as a CGI-program in Perl and provides an HTML-based interface. After a service invocation, GBrowse Moby presents the result data and links to possible further processing steps. The sequence of steps in a GBrowse Moby session can be saved as a workflow in the Scufl language of the Taverna workflow system.

The Seahawk BioMoby browser presented by Gordon and Sensen [GS07b] strongly resembles GBrowse Moby in terms of functionality. Seahawk has been implemented in Java, and can be embedded in other Java applications.

Carrere and Gouzy [CG06] have presented the workflow system Remora which enables users to generate dataflow DAGs of BioMoby Web Services. Similar to BioWMS, the Remora user interface is based on traditional Web pages.

Users can define input data types and add compatible BioMoby Web Service tasks consuming the inputs. In the same way, users can add compatible BioMoby tasksm, which consume the output of an existing BioMoby task. If a task consumes more than one input, additional dataflow edges can be added.

Workflows and results can be stored on the Remora server, and searched for based on keywords, or in a "Frequently Asked Workflows" list.

2.4.3 Taverna

Overview

The *Taverna* workbench is a popular bioinformatics workflow management system from the ^{my}Grid[19] project [OAF^{+}04][TDGS06, ch. 19] for composing and executing service-oriented workflows, given in the XML-language *Scufl* [XSc]. Taverna runs as a desktop application, but an extension for executing workflows on remote hosts exists. Stevens et al. [SRG03] describe the goal of ^{my}Grid as the development of middleware layers appropriate for bioinformatics information grids. They identify the "target users of myGrid [as] tool and service providers who build applications for a community of biologists" [SRG03]. In particular, according to Stevens et al. [STW^{+}04] "^{my}Grid can be considered a toolbox of components with which a bioinformatician developer can construct targeted applications through a generic mechanism."

[19] `www.mygrid.org.uk/`, last accessed: 22/09/2008.

The Scufl Language

No standardized SCUFL specification exists and the developers warn that "the language itself should be considered volatile."[20] Glatard and Montagnat have shown the Turing-completeness of SCUFL with a set of Java extensions [GM08].

SCUFL uses special nodes in the workflow graph to depict input and output parameters. The SCUFL terminology does not distinguish between activity types and activities. Activities are represented by *Processors*. Processors are an abstraction for arbitrary services, e.g. Web Services or Java methods. Workflow nesting is provided by a special processor type. Every processor type is identified by a $name$, and tuples of type input and output ports, respectively:

$$P_{type} = (name, (i_1, \ldots, i_m), (o_1, \ldots, o_n)).$$

An individual processor in a workflow is given by its processor type, and an explicitly or implicitly defined *iteration Strategy* I:

$$P = (P_{type}, I)$$

.

Data links connecting an output port of one processor and an input port of another represent a data dependency between the two processors. SCUFL also includes *coordination constraints*, which state that a processor may only run after the successful execution of another one.

If a processor with one input port receives a list of n input items, the server underlying the processor will be invoked n times, and the results repackaged as a list. This approach does not suffice for processors with more than one input port. Hence, SCUFL contains the concept of *iteration strategies*, where every processor P with more than one input port is associated with an iteration strategy $I = (op_1, \ldots, op_n)$, where $op_i \in \{\otimes, \odot\}$. $\otimes$ and $\odot$ represent mappings $D^\infty \times D^\infty \to (D^2)^\infty$, where D is the set of possible data items:

$$(a_1, \ldots, a_n) \odot (b_1, \ldots, b_n) \mapsto ((a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n))$$

$$(a_1, \ldots, a_n) \otimes (b_1, \ldots, b_n) \mapsto ((a_1, b_1), (a_1, b_2), \ldots, (a_n, b_{n-1}), (a_n, b_n))$$

The default iteration strategy for a processor with n input ports is $I_{default,n} = \{\otimes\}^{n-1}$, the sole application of the cross product.

[20]`http://www.ebi.ac.uk/~tmo/mygrid/XScuflSpecification.html`, last accessed 17/11/2009.

Oinn et al. [OGA+06] state that Taverna uses SCUFL because the development of Taverna had been started before the standardization of BPEL, and because an evaluation of the BPEL predecessor WSFL showed that it did not abstract away from the intricacies of the Web Service stack. They conclude:

> "For Taverna users a migration path, so that workflows prototyped in SCUFL could be transformed to BPEL for more 'heavyweight' use, would be attractive."

Architecture

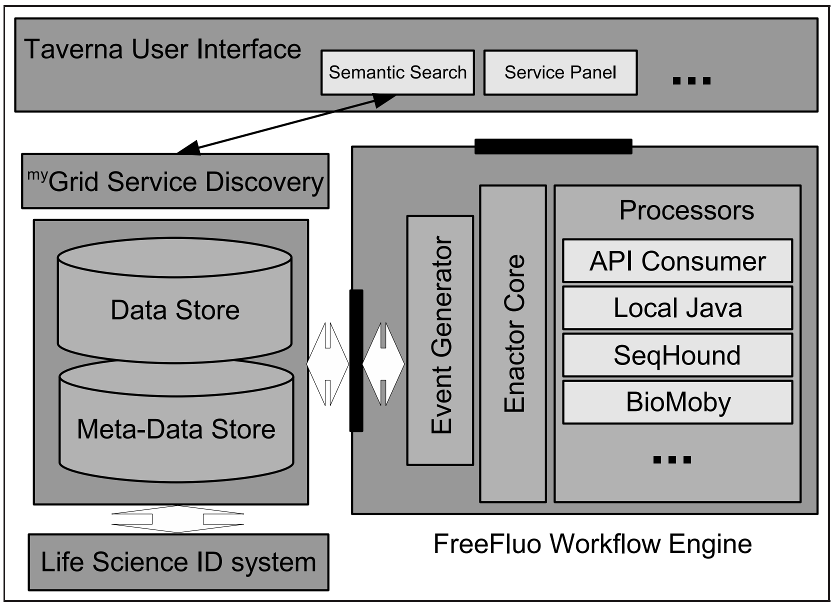


Figure 2.3: A simplified view of the Taverna architecture.

Figure 2.3 gives an overview of the Taverna architecture. The Taverna GUI is implemented as a Java application, which can be extended with additional components via a plugin-framework. ^{my}Grid/Taverna employs the Java-based open source workflow engine *FreeFluo*, which can interpret the WSFL workflow language, a predecessor of BPEL, as well as SCUFL [Fre]. Processors are implemented as Java classes, which can wrap arbitrary services. The Taverna distribution includes processors for WSDL-based Web Services, local Java functions, Soaplab Web Services, BioMoby Web Services, the RESTful SeqHound service, a JDBC-based interface to the BioMart database, and Styx Grid Services. A special processor enables the invocation of nested SCUFL workflows. The BioMoby Central registry can be queried to find semantically matching successors for a given BioMoby processor [KSW06]. The FreeFluo engine provides an interface to enable external observers to subscribe to events generated by workflow executions. Taverna uses the Life Science Identifier System (LSID) [CML04] to identify biological data items in provenance records.

Taverna can be deployed via Java Web Start for integration in web portals, which is e.g. used in the ^{my}Experiment workflow sharing platform.

User Interaction Lanzen and Oinn [LLO08] point out that "in the standard version of the Taverna Workbench, a user cannot control the behaviour of a workflow once it is running" and have introduced an interaction extension. Users are notified of interactive steps by email and can visit a web site to perform the actual interactions. Two standard interaction activities are available, where users have the choice to either accept or reject "a piece of textual data," or to edit a given genome sequence using a Java Web Start initiated editor.

The Taverna Data Playground Gibson et al. [GGW+07, GGW+09] have presented the Data Playground plugin, which enables an "investigative, data-oriented mode" of workflow construction. Starting with pieces of input data, a bioinformatician can explore possibilities of data manipulation by repeatedly applying processor invocations. This mode of workflow construction strongly resembles the mode of operation already enabled by BioMoby browsers (see Section 2.4.2), which can also be used to construct SCUFL pipes in a "programming by example" fashion. Consequently, the Data Playground has been implemented based on the Taverna BioMoby plugin, and sessions can be recorded and saved as SCUFL workflows.

2.4.4 ^{my}Experiment

^{my}Experiment is a Web-based platform for sharing workflows, which has been developed in the ^{my}Grid project. De Roure et al. [RGB+08, dRGS08] describe ^{my}Experiment as a *Social Virtual Research Environment* which they define by four key capabilities. First, a social virtual research environment should enable the sharing of *Research Objects*, which are the digital entities used by scientists, including data and method definitions, as well as publications. Second, it should facilitate a *social model*, which should enable the attribution of credit, as well as "fine control over the visibility and sharing of research objects," since "producers of research objects should have incentives to make them available” [RGB+08]. The *social model* should also comprise capabilities for the curation, i.e. the categorization and evolution of research objects, by the community. Third, it should be an *open and extensible environment*, providing APIs for the integration with other systems. Fourth, it should allow *actioning research*, by allowing users to directly deliver research objects to external services.

^{my}Experiment enables the up- and download of workflows and arbitrary other files as *research objects*, which users can tag, comment on and review. Users can create and join groups, which can be used to exchange messages. A special class of research objects are *Packs*, i.e. collections of existing research objects.

Although ^{my}Experiment allows sharing arbitrary *research objects*, it is tightly integrated with the Taverna system, as on the one hand, ^{my}Experiment provides Taverna deployment via Java Web Start, and on the other hand Taverna includes a ^{my}Experiment plug-in to access the ^{my}Experiment workflow base. While general workflows are only presented with textual descriptions and graphical representations, Taverna workflows can be inspected online on the element level, i.e. users can view the individual steps in a workflow.

^{my}Experiment has been developed in the Ruby on Rails[21] web framework, following the Model View Controller Design Pattern [GHJV95]. It has been deployed on a Mongrel[22] application server cluster, with an Apache HTTP Server frontend, a mySQL database server, and an Apache Solr[23] search server. It exposes a RESTful API, which enables the development of further plugins or mashups.

2.5 The Adoption of BPEL in Scientific Workflow Systems

2.5.1 Scientific Workflows in BPEL

Sedna and OMII BPEL To the best of our knowledge, Wassermann et al. [TDGS06, ch. 26] were the first to apply BPEL to scientific workflows. They presented the visual BPEL editor Sedna in the context of the OMII BPEL distribution from the UK e-Science project. While Sedna has been implemented as an Eclipse plugin, the OMII BPEL execution engine is based on Active BPEL.

[21] `http://rubyonrails.org/`, last accessed 28/12/2009.
[22] `http://mongrel.rubyforge.org/`, last accessed 28/12/2009.
[23] `http://lucene.apache.org/solr/`, last accessed 28/12/2009.

In the version presented by Wassermann et al. [TDGS06, ch. 26], Sedna supports several extensions to the BPEL4WS 1.1 language, to improve its capabilities for the development of scientific workflows. The extensions have been implemented by compiling the original source to standard BPEL4WS 1.1. First, a parallel for loop has been implemented, similar to, but less powerful than the ForEach structured activity in WS-BPEL 2.0. It supports counting loops, which are compiled to parallel activities inside a Flow.

BPEL in bioinformatics In 2004 de Knikker et al. [dKGlL$^+$04] have compared three different implementations of a genome analysis workflow based on Java, BPEL and Taverna. They conclude that "due to the immaturity of currently available web services engines, it is still most practical to implement a simple, ad-hoc XML-based workflow by hard coding the workflow in Java."

In 2007 Thiele and Habich [TH07] have implemented a workflow for clustering gene expression data based on the Oracle BPEL engine. They use a relational database to store analysis results, which is accessed via stored procedures that are exposed as Web Services.

Tan et al. [TMMF09] have compared Taverna and BPEL in a use case from the caGrid project. They conclude that BPEL is more complex and well suited for implementing any kind of process, while "Taverna provides a more compact set of primitives and a functional programming model that eases data flow modeling" [TMMF09]. They suggest a "multi-stage modeling approach in adapting BPEL to scientific workflow, leveraging its capability and retaining the simplicity", where scientists can define workflows on a higher level of abstraction, which are then compiled to BPEL [TMMF09].

2.5.2 BPEL and the Grid

The Web Services Resource Framework Since 2004, the original Grid Service specification of the Open Grid Services Infrastructure has been succeeded by the Web Services Resource Framework (WSRF).[24] In contrast to OGSI, WSRF relies on existing WS-* specifications and is compatible with the Web Services Architecture. WSRF has been standardized by OASIS in five different specifications (WS-Resource WS-ResourceProperties, WS-ResourceLifetime, WS-ServiceGroup, WS-

[24] `http://www.globus.org/wsrf`, last accessed 12/10/2009

BaseFaults). WSRF has been implemented in the Globus[25] Toolkit 4 and Apache Muse.[26]

The WSRF specification defines a *resource* as a logical entity that has to be identifiable, has a set of zero or more properties which are expressible in XML, and may have a lifecycle [WSR06a]. It further defines a *WS-Resource* as the composition of a resource and a Web service through which the resource can be accessed. WS-Resources can be addressed using a *WS-Addressing endpoint reference* [WSA06]. The WS-Adressing standard defines a way of passing routing information in SOAP envelopes using two different XML constructs, i.e. *endpoint references* and *message information headers*. Endpoint references consist of an *address* element containing a URI and other optional elements. The properties of a WS-Resource are declared in a *Resource Properties Document* and can be accessed by standard getter/setter interface [WSR06c]. WSRF also defines a subscription mechanism for property changes, based on WS-BaseNotification and WS-Topics [WSB06, WST06]. The WS-ResourceLifetime standard defines “message exchanges to standardize the means by which a WS-Resource may be destroyed, and resource properties that may be used to inspect and monitor the lifetime of a WS-Resource” [WSR06b]. The WS-ServiceGroup specification defines a set of resource properties for representing groups of WSRF web services [WSS06].

WSRF and BPEL According to OASIS,[27] the WS-BPEL 2.0 standard does not support WS-Addressing, which is a prerequisite for WSRF [AAA$^+$07, sec. 3]. On the other hand, the WS-BPEL 2.0 standard introduces a mechanism to assign implementation dependent endpoint references to partner link roles: “the actual partner service may be dynamically determined within the process. WS-BPEL uses a notion of endpoint reference, manifested as a service reference container <sref:service-ref>, to represent the data required to describe a partner service endpoint” [AAA$^+$07, sec. 6]. At least some BPEL engines support WS-Addressing endpoint references to imple-

[25]`http://www.globus.org`, last accessed 02/09/2009.
[26]`http://ws.apache.org/muse/`, last accessed 02/09/2009.
[27]`http://www.oasis-open.org/committees/download.php/23858/WS-BPEL-2.0-FAQ.html` last accessed 02/09/2009.

ment dynamic partner links, e.g. IBM Web Sphere,[28] Apache ODE,[29] and Glassfish BPEL SE.[30]

Leymann [Ley06] and Slominski [Slo06][TDGS06, ch. 14] have been among the first authors to theoretically discuss the integration of BPEL engines into Grids.

Ezenwoye et al. [ESCR07] have presented their experiences with orchestrating WS-Resources with BPEL in an application which attempts to match protein sequences. They have identified the "centralized nature of data movement" as "a problem for high-performance computing," and point out that "the BPEL specification does not make provisions for how dynamically obtained information such as resource identifiers, usernames and passwords can be specified within SOAP message headers" [ESCR07]. Thus, embedding WS-Resources in BPEL workflows depends on implementation specific features of the BPEL engine.

Janciak et al. [JKB08] have presented a BPEL enactment engine, which is deployed as a stateful WSRF Web Service, where each process is represented as a WS-Resource. Brinkman et al. [BGH$^+$08] have implemented a BPEL execution engine for UNICORE Grid Services, based on the ActiveBPEL engine and extensions of the UNICORE system. While the ActiveBPEL engine has not been modified, the UNICORE container is extended with a Workflow Management Web Service, which is used as a proxy to represent Unicore Grid Services. Gunarathne et al. [GHCM09] have discussed extensions to the Apache ODE BPEL engine in the context of the LEAD Grid Portal [CM07]. Similar to other authors they especially point out the intricacies of invoking stateful Web Services.

2.5.3 Collaborative BPEL Development Systems

Friese et al. [FSF$^+$06] have presented an Eclipse-based collaborative BPEL Editor, which supports a synchronous editing process based on the Eclipse Communication Framework, where the collaboration initiator acts as a coordinator. They motivate collaborative workflow design by an example, where a casting engineer, a numerical simulation expert and a grid expert cooperate on the design of an engineering workflow in the domain of metal casting. They argue, that especially small companies will gain from collaborative workflow development, since such companies will often have to hire external experts for implementing aspects of a workflow. Users can join

[28] `http://www.ibm.com/developerworks/library/ws-addressing/index.html`, last accessed 02/09/2009.

[29] `http://ode.apache.org/endpoint-references.html` last accessed 02/09/2009.

[30] `http://wiki.open-esb.java.net/Wiki.jsp?page=UsingDynamicPartnerLinksAndDynamicAddressing`, last accessed 02/09/2009.

a session and create, delete, connect and move basic activities concurrently. They neither address the issue of collaboratively editing complex workflows consisting of several structured activities nor how participants should negotiate the configuration of a specific activity.

Phung et al. [PHL09b, PHL09a] have presented a collaborative workflow editor to enable the collaborative organization of health care processes in a Grid infrastructure for monitoring and nursing elderly people. The workflows are collaboratively designed in BPMN and afterwards compiled to BPEL, with bindings to Grid services representing health care professionals.

Phung et al. [PHL09a] point out that "the potential support ratio (the number of people aged 15-64 years per one older person) has fallen from 12 to 9 between 1950 and 2000 and will continue to fall." They outline the scenario of a treatment workflow for a patient suffering from a heart complication, with the purpose to "automatically [prompt] nurses for appropriate treatment and monitors the patient's progress" [PHL09a]. In this scenario the attending doctor uses a collaborative workflow editor to generate a treatment plan in cooperation with a radiologist and a cardiologist. During the treatment, the monitoring activities notify the doctor of changes in the patient's condition. The doctor thus decides to stop the current workflow and modify the workflow model in cooperation with a physiotherapist.

While all users receive updates in real time, only one user at a time is allowed to make changes on the workflow model. A user normally holds a global lock on the workflow until he/she voluntarily releases it or a configurable timeout event occurs. However, the system also enables users to "urgently" demand the lock, in which case the lock of the current owner will be revoked and granted to the demander.

Similar to the approach of Friese et al. the workflow editor is based on Eclipse, and has been implemented based on the Eclipse BPMN Modeler,[31] and the Eclipse Communication Framework.

In a blog entry on the *SAP Community Network*, Dreiling [Dre09] has presented a video of a collaborative BPMN editor based on Google Wave.[32] Introduced in May 2009,[33] Google Wave is a collaboration platform, which can be embedded in Web applications to support concurrent modifications of hosted documents (dubbed *waves*) in a Wiki-like fashion with near real time updates. It is based on an extension of the XMPP instant messaging protocol, which is defined in RFC 3920, RFC 3921, RFC 3922, and RFC 3923. In current implementations, the XMPP messages are tunneled

[31] `http://www.eclipse.org/bpmn` last accessed 01/09/2009.

[32] `wave.google.com`, last accessed 17/12/2009.

[33] `http://www.techcrunch.com/2009/05/28/google-wave-drips-with-ambition-can-it-fulfill-googles-grand-web-vision/`, last accessed 28/12/2009.

through AJAX-based HTTP posting and polling, though. Document modifications are handled with an operational transformation algorithm.

2.6 Summary

The presented *Grid Workflow Systems* have evolved from parallel programming models and address the issue of coarse-grained parallel applications. While Pegasus addresses the issue of exceptionally large workflows, Systems like ASKALON, Triana, P-GRADE, and Kepler enable the graphical composition of workflows, but do not achieve a higher level of abstraction than textual programming languages. P-GRADE is the only system which addresses the question of collaborative development.

Similar to Grid Workflow Systems, *Life Science Workflow Systems* originally were implemented with textual interfaces only. They differ in two ways, though. First, Life Science Workflow Systems mainly serve the purpose of composing data sources, while parallel execution is a minor issue. Second, Life Science Workflow Systems attempt to support workflow composition by end users, i.e. bioinformaticians or biologists. Systems like Taverna have indeed succeeded in providing workflow programming capabilities on an appropriate level of abstraction for bioinformaticians.

The adoption of the BPEL standard for scientific workflows appears to be possible on a technical level as the examples presented in Section 2.5 show. The BPEL language is too complex for use by non-experts, though.

Summing up, three related issues remain unsolved concerning scientific workflow modeling, which we will now discuss.

Comparatively Low Level Languages While all presented systems provide graphical abstractions of control flow constructs, they still resemble traditional programming languages. Systems like Triana and Kepler greatly simplify access to Grid resources and are not directed at users from a single domain. However, since they do not target any specific audience, they provide general notations. The Taverna workflow system successfully addresses the needs of bioinformaticians, but remains too complex for biologists, whose education does not usually encompass programming courses.

In 2007, Gordon and Sensen [GS07a] conducted a study on the usability of Taverna for end users (i.e. Biologists) and programmers. The sample included four biologists, three experienced programmers, who were given a task, and had to fill out a questionnaire afterwards. Additionally, "two additional developers intimately familiar with Taverna participated in the questionnaire alone.” In the study, Gordon an Sensen identify four major obstacles of the Taverna user interface:

1. They point out that Taverna has many “incompatible and incomprehensible data formats/types.”
2. They see a “difficulty linking workflow elements.” While Taverna is generally dataflow oriented, control constructs remain unintuitive.
3. While a plethora of processors are available, the system suffers from “poor visibility, searchability and typing of processors.”
4. Taverna enables “no direct manipulation of [the] workflow diagram by default.”

Thus, even though Taverna enables the graphical construction of workflows, its user interface and language features amplify the inherent complexity of workflow development.

No Separation between Modeling and Implementation Business Process Management endorses a separation of roles between domain experts and software engineers. While domain experts have the best knowledge about core business processes, they cooperate with workflow experts and software engineers in defining the actual business process models which are later used by software engineers in the actual workflow implementation. Several layers of abstraction — ranging from EPC via BPMN to BPEL — enable the exact specification of the purposes, requirements and functionality of Business Processes, thus further enabling future repurposing, reimplementation or reuse.

Scientific workflow systems, however, are often intended to be used in every aspect by domain scientists. While the wish of scientists to define executable workflows by themselves is served by current systems, there is no sufficient support for sketching processes, which can be implemented by software engineers. Thus, scientific workflows can be unnecessarily hard to understand and hard to adapt to changing technical requirements, e.g. to novel service infrastructures.

Lack of Standards The lack of common standards for scientific workflow systems remains a problem. In 2005, Yu and Buyya [YB05] discussed 13 different Grid workflow systems, while in 2007, Tiwari and Sekhar [TS07] presented 23 different workflow systems in the life science domain only. Yu and Buyya [YB05] conclude that “more effort is thus needed towards workflow modeling standardization.” Tiwari and Sekhar [TS07] even see the “lack of standards in workflow based development and implementation in life science domain [as] a major issue.” While new scientific workflow systems are evolving, Barker and van Hemert [BvH08] believe that “Yet another workflow language is the last thing researchers should be doing” and give researchers the advice to “stick to standards.” They see a problem in the uncertain future of workflow languages that are developed by the scientific community: “If software development and tool support terminates on one of the proprietary frameworks, workflows will need to be reimplemented from scratch” [BvH08]. This appears to be a realistic scenario even for successful scientific workflow systems. Once research funding terminates, it is unclear whether a system will be developed further, or even if documentation of a workflow language will be of any use.

Barker and van Hemert [BvH08] see a solution in the adoption of industry standards as execution languages in combination with novel modeling techniques: “Instead, standards need to be viewed in a different way, research needs to be targeted at providing powerful abstraction mechanisms for languages as BPEL and providing integrated tool support.”

In the conclusion to their review of Grid systems for bioinformatics, Shah et al. [SBL+08] point out that some projects “seem to be discontinued in terms of information updating,” the cause of which they assume to be “funding problems or difficulties associated with their implementation.” The lack of standards for scientific workflow languages thus constitutes an operational risk for scientists’ ability to understand and execute archived workflows in the future.

Chapter 3

Workflow Design Processes

This chapter describes workflow development processes which lead our design efforts in the further course of this dissertation. First, we investigate the influence of different levels of abstraction on collaboration in workflow development. Then, we lay out a general process of collaborative workflow development. A second aspect of this chapter is the investigation of E-Biology processes, to understand requirements for e-Biology workflow systems. Our findings yield a set of application requirements, which serve as a guide to the design of corresponding Rich Internet Applications. The chapter closes with a discussion of the relationships between the presented processes and the development systems HOBBES and CALVIN.

3.1 Levels of Abstraction and Collaboration in Workflow Design

Thomas et al. [TLD$^+$07] describe the application of the Service Oriented Architecture for Business Process Management as an interplay of three levels of abstraction. In this model, Event-Driven Process Chains are used for high-level business process modeling by domain experts, while BPMN is used to communicate between domain experts and software engineers, and BPEL models are used for the actual workflow implementation. Blake [Bla07] identifies three corresponding phases in service-centric software systems management, where he distinguishes between business process management, design time software engineering, and runtime software engineering (i.e. on-demand composition and rebinding of Web Services).

Similar to Thomas et al. [TLD$^+$07], we can identify four levels of abstraction in workflow design systems, which are visualized in Figure 3.1. The three lower layers of abstraction are found in general purpose workflow management systems today. As we have seen in Chapter 2, contemporary scientific workflow systems often do not distinguish between workflow notations and languages.

We will now describe the levels of abstraction by proceeding from the bottom level to the top. The lowest layer contains *workflow execution languages*, e.g. BPEL in the standard case. Since BPEL is a full-fledged, Turing-complete programming language, it is intended for the use by software engineers. However, as BPEL is a verbose, XML-based language, developing workflows manually via text editors is a cumbersome task, which can only be performed by experienced software engineers.

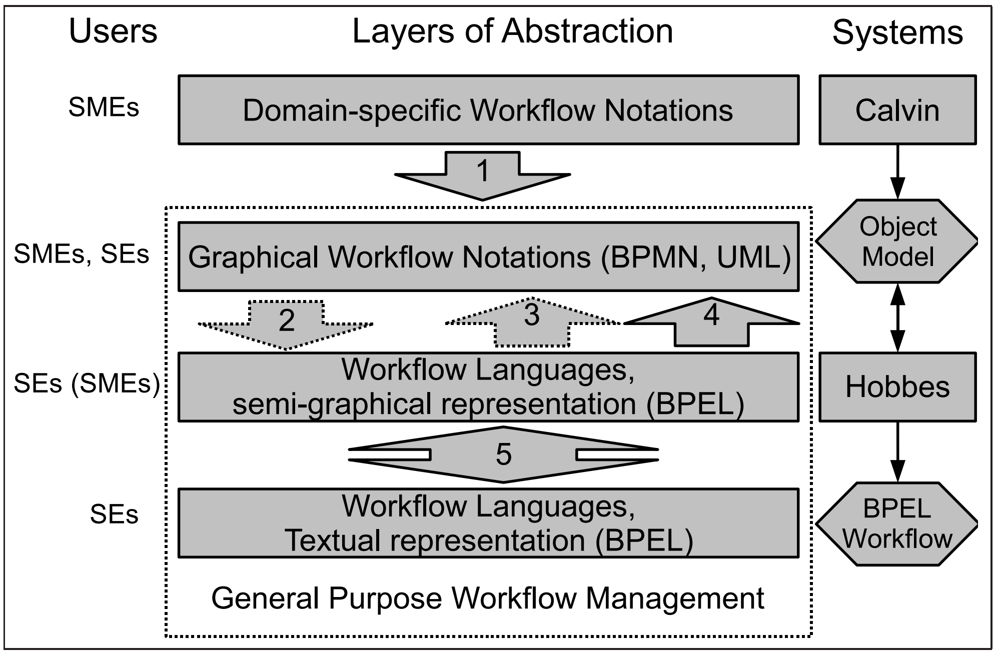

Users
Layers of Abstraction
Systems
SMEs
Domain-specific Workflow Notations
Calvin
1
Object Model
SMEs, SEs
Graphical Workflow Notations (BPMN, UML)
2
3
4
SEs (SMEs)
Workflow Languages, semi-graphical representation (BPEL)
Hobbes
5
SEs
Workflow Languages, Textual representation (BPEL)
BPEL Workflow
General Purpose Workflow Management

Figure 3.1: Layers of Abstraction in Workflow Design.

Hence, editing systems have been developed, which visualize at least parts of the workflow model, and present other parts as widgets. Especially the control flow or data flow described by a workflow model often can be visualized, yielding a graphic depiction of the model. However, a workflow model can also contain elements which cannot easily be visualized, e.g. declarations of variables or events. An editor can combine visualizations of the BPEL control flow with form-based editing of non-visual properties. Examples for such editors include the *ActiveBPEL Designer* and *Oracle JDeveloper*. While the workflow object model used on this level includes more information than a BPEL program, i.e. information about the placement of graphical objects, a bijective mapping between the semantically relevant parts of the object model and the textual representation exists (see Figure 3.1, arrow 1). While data about the placement of visual artifacts is lost in the compilation, after parsing a

BPEL file a new layout can be calculated without compromising the correctness of the workflow model. At this level of *semi-graphical representation* of workflow models, a cooperation between software engineers and *subject matter experts* (also dubbed *domain experts*) is possible, although close guidance by the software engineers is necessary.

Graphical Workflow Modeling Notations enable subject matter experts and software engineers to reason about workflows on par with each other. Examples for such notations include BPMN, Event-Driven Process Chains, and UML2 activity diagrams. Models in these notations can be compiled to BPEL, and BPEL models can be parsed to these notations (see Figure 3.1, arrows 2, 3). Since BPEL is a block-oriented language, where the same process can be modeled in redundant ways, to the best of our knowledge there is no bijective mapping between any of these languages and BPEL. The notations can always be used for visualization purposes, though (see Figure 3.1, arrow 4).

The levels described so far are used for workflow management in the context of business process management. Scientific workflow systems are intended to enable individual scientists to define and execute workflows, though. The area of scientific workflows thus leads to the assumption of a level of *domain-specific workflow notations*, above the stack of general purpose workflow management.

This model of layers of abstraction reveals two distinct directions of collaboration. On the one hand, software engineers and SMEs can cooperate asynchronously, by refining workflow models from higher levels of abstraction to lower levels. We can identify this as the *vertical* axis of collaboration. On the other hand, software engineers and SMEs can cooperate synchronously or asynchronously at the same level, which we can identify as the *horizontal* axis of collaboration.

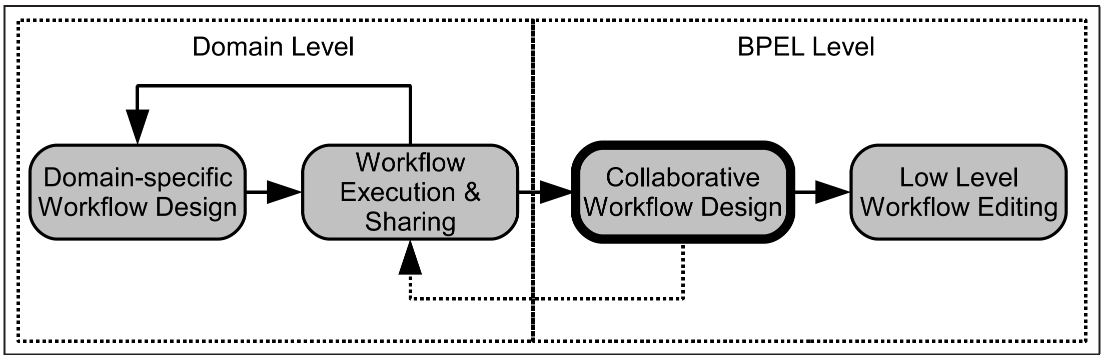

Domain Level
BPEL Level
Domain-specific Workflow Design
Workflow Execution & Sharing
Collaborative Workflow Design
Low Level Workflow Editing

Figure 3.2: An overview of tasks in the collaborative development of workflows.

Figure 3.2 visualizes a set of possible tasks on the identified levels of abstraction, and their dependencies. Each dependency corresponds to a mode of asynchronous collaboration, i.e. the arrows in Figure 3.2 show that results of one task can be used in another task. In scientific domains, *domain-specific workflow design* tasks result in workflows which can be *executed or shared* with colleagues. These workflows can be refined beyond the limits of a domain-specific notation on the BPEL level. The synchronously collaborative task *collaborative workflow design* is marked by a thick border. Note that editing or executing refined workflow models on the domain level is not always possible, as workflow structures and interfaces can be altered arbitrarily on the lower levels of abstraction. This is signified by the dotted arrow in Figure 3.2. Low level workflow editing, e.g. in a text editor, can be needed to fine-tune a workflow model.

3.2 Collaborative Workflow Development

In this Section, we describe a process of collaborative workflow development on the abstraction level of a semi-graphical representation of the BPEL language.

3.2.1 An Example of Collaborative Workflow Design

As a use case, we assume a life science project that spans several research groups in different countries. In the project, a biologist from Germany and a physician from France are designing an in-silico experiment composed of several Web Services. New, non-trivial Web Services needed in the experiment are provided by research institutions in several European countries. To achieve a correct integration of these services, partners from the provider institutions have to be contacted. The results of the workflow are to be stored in a database located in Belgium. Figure 3.3 illustrates the situation of the project partners.

The workflow project is a collaboration of specialists that are too far away from each other to meet face-to-face. We can further observe that the domains of expertise of the participants differ from each other, e.g. the biologist and the physician share knowledge about the biological process to be modeled, while the service providers know best how to configure their services. As none of them is very experienced in workflow design, they consult a workflow expert.

Bridging the gaps between distinct domains and bridging the geographic distances between team members requires support by a collaborative software system. Collaborative manipulation of a workflow model implies that no single person has a com-

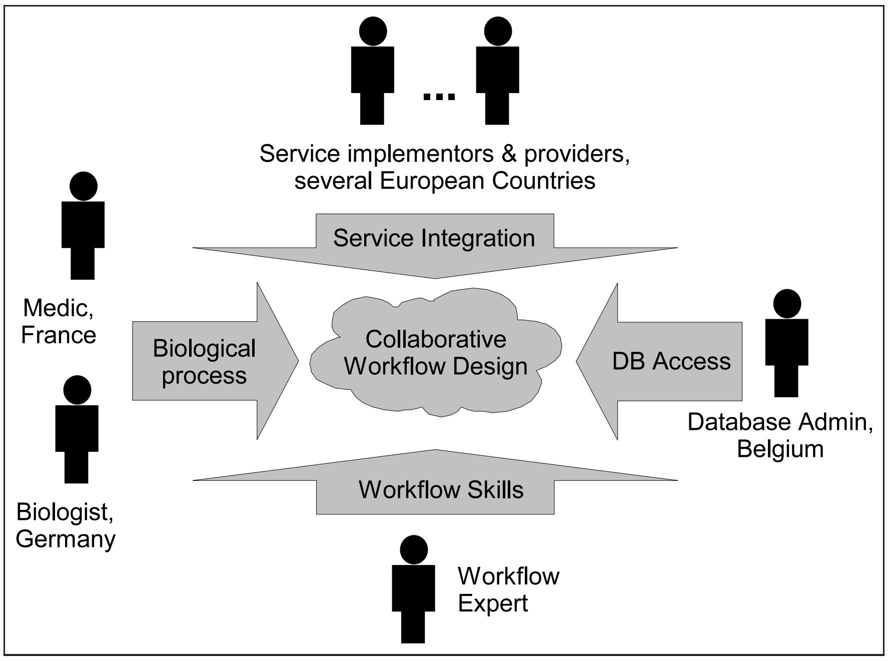


Figure 3.3: A workflow design collaboration.

plete view of the model. However, a team leader needs to make decisions about how to proceed in the collaboration, e.g. by assigning tasks to team members. A collaborative workflow design system should thus help the team leader in assessing workflow models. Such workflow analysis support has to consider that the workflow model is "under construction" and that it is the result of the cooperation of distinct partners. Verification techniques can prove the syntactic correctness of a workflow model, but will not help in identifying syntactically correct but troublesome parts of the workflow, i.e. design flaws. Although collaborative development improves the integration of domain expertise and technological expertise in a workflow, it also leaves the risk that design flaws are left unnoticed, since no single member of the design team has a complete view of the model all of the time. Finding design flaws in source code can be simplified by using software metrics. As the team leader needs to identify design flaws in the workflow model, we propose to use workflow metrics in the collaborative workflow design system, i.e. workflow-specific software metrics. The team leader can use these metrics to gather information, which can help identifying design flaws in the workflow model.

Collaborative development of Scientific Workflows is a natural fit for Problem Solving Environments, which are often implemented as web portals. Moreover, department policies can prevent the installation of yet another software package on client machines. Thus, the collaborative workflow design system should be provided as a web application.

However, classical web applications cannot provide real-time interactivity and lack the look and feel of desktop GUIs. Many users, on the other hand, will only accept the system if they feel comfortable with its interface. Hence, we will implement the system using Web 2.0 technology, which enables desktop-like user experience and synchronous interaction of team members.

3.2.2 The Collaborative Workflow Design Process

An Event-Driven Process Chain of Collaborative Workflow Development

Weiseth et al. [WMTL06] identified three sub-processes of collaboration, which they call *coordination*, *production* and *decision-making*. Coordination denotes the planning and assignment of tasks to collaborators, while production means collaborative development. Decision-making is the base of coordination and requires analysis of the manipulated artifacts. Our Collaborative Workflow Design approach accounts for each of these sub-processes in distinct phases (see Figure 3.4). After initializing the collaborative process, each design phase will be composed of a cycle of synchronous *collaboration* of the team members, and *analysis* as well as *annotation* of the workflow model by the team leader.

First, the team leader instantiates a new session in the collaborative workflow design system. She can make initial decisions by laying out and configuring elements. Finally, the team leader will decide to share the workflow and invite team members.

Team members can join or leave at any time after sharing the workflow design session. The team synchronously works on the design until the team leader decides that it is time to analyze the quality of the workflow model. During the analysis and annotation phase the workflow is globally locked, so that only the team leader has full access. The team leader uses analysis tools, e.g. workflow metrics, to asses the workflow and find sub-workflows that should be refactored. If the team leader decides that the workflow needs further refinement, she will annotate the workflow model to delegate responsibility for specific tasks in sub-workflows to individual team members. Afterward she will switch the session to a new collaborative phase.

Figure 3.5 provides a more detailed description of the collaborative workflow design process, modeled as an Event-Driven Process Chain [Sch99a].

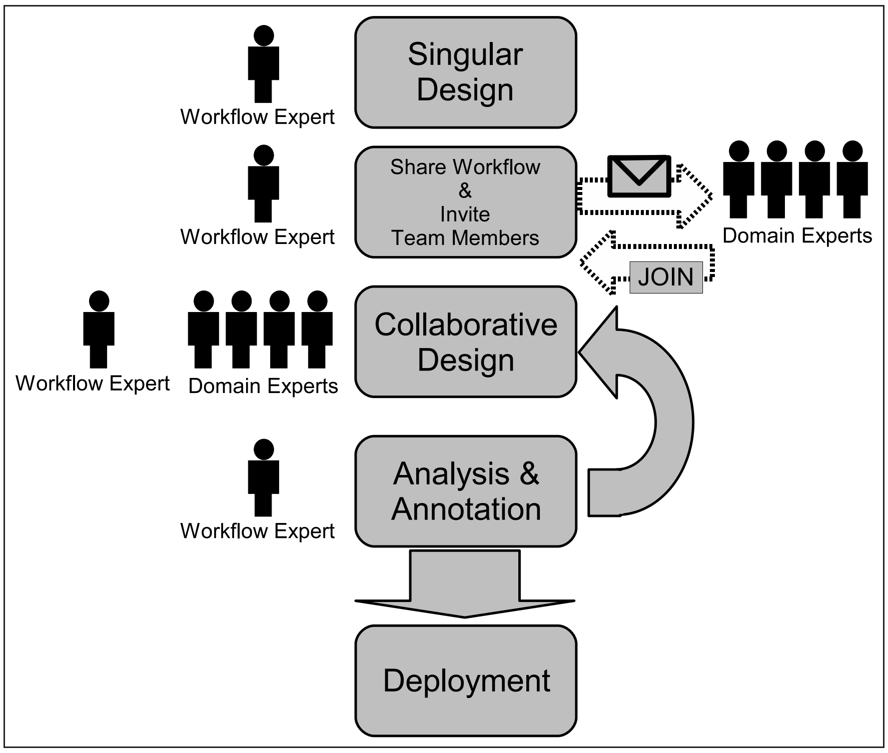


Figure 3.4: Overview of the collaborative workflow design process.

It shows that workflow analysis (decision-making) and annotation (coordination) are closely related, as coordinative actions depend upon decisions. The activity following the top left XOR represents metrics aided analysis of the workflow, which alternates with the annotation activity on the right. The cycle of analysis and annotation is entered after a design activity. Both activities involve only the team leader and require a workflow model that is left unchanged by other users actions. Thus, in a collaborative workflow design system both can be handled within the same phase.

The HOBBES system, which is presented in this dissertation, supports this Event-Driven Process Chain, with the exception of the hatched area. The Process Chain shows that the same tools used for enabling collaborative workflow design can be used for collaboration between a workflow expert and an administrator in case a workflow deployment has led to problems.

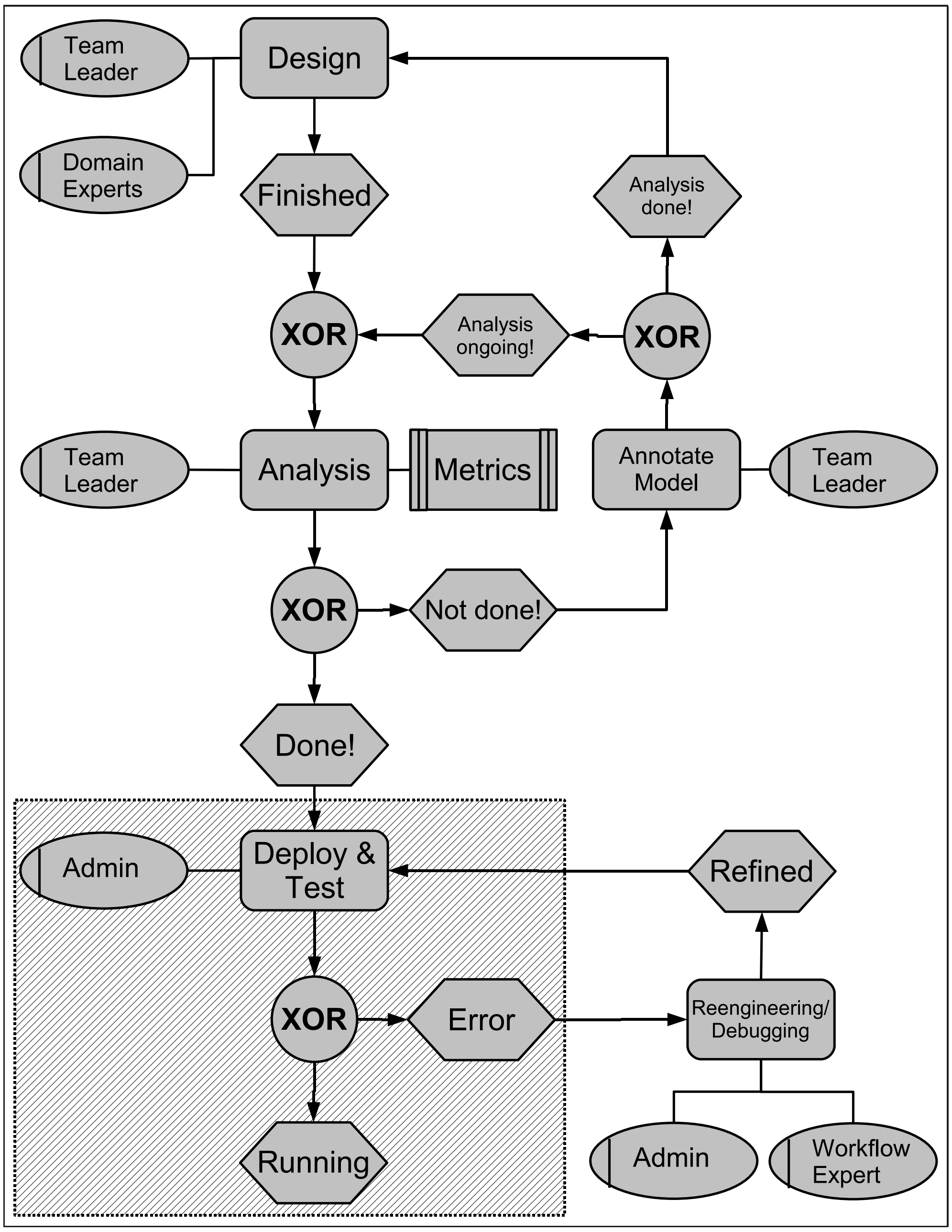


Figure 3.5: The complete collaborative workflow design process as an Event-Driven Process Chain.

3.2.3 Requirements of Collaborative BPEL Development Systems

We will now discuss requirements for collaborative workflow design systems, which support the process described in Section 3.2.2, by proceeding from general to more specific issues.

General Requirements To enable the seamless integration of workflow technology in Problem Solving Environments, workflow design tools will have to be accessible on the web, and should provide a nearly desktop-like user experience. Thus, we need a client that can be accessed from the web browser, but provides a user experience similar to desktop applications. Users need to be able to create team sessions, whose members work on the same workflow project and can synchronously add, remove, and connect activities.

Relaxed WYSIWIS and Late Joining A fundamental design decision for collaborative software is, which information should be forwarded to other participants. A naive approach would be to give all users exactly the same view on the shared document (*strict* “What You See Is What I See” (WYSIWIS)) [SBL$^+$86]. In the case of complex documents (like workflows) this would diminish productivity because it is unlikely that all parts of the document can be shown in one view at the same time. A special concern arises if users attempt to make conflicting changes to the view, e.g. scroll into different directions or open pop-up windows. Such conflicts are sometimes dubbed “scroll-war” or “window-war.” Hence, a user’s actions should only affect another user’s view if it is semantically necessary.

Begole et al. [BRS99] discuss several usability requirements for the collaborative adaptation of legacy applications. They especially point out that a user should be able to begin sharing a document at any time after its creation, that users should be able to join a shared session at any time (“late-joining”) and that users need to be allowed to have independent views of the shared document (“location-relaxed WYSIWIS”).

Workflow Locking Mechanisms Synchronous collaboration encourages users to work on the same objects at the same time. This stands in contrast to each individual user’s demand for privacy while working on a specific hard problem in a sub-workflow. Sub-workflows, however, may contain complex domain-specific logic. One way to enhance group awareness — knowledge about other participants’ actions — is the display of representations of foreign mouse cursors, called telepointers.

Telepointers make much less sense in a web-based infrastructure due to the relatively high latency. Thus as an alternative means of raising group awareness, we have to allow synchronous work but also provide mechanisms to temporarily protect a user from being disturbed. To prevent the possibility of one user blocking the progress of the workflow design process, the team leader may grant or release such locks.

Collaborative Document Navigation Additionally, there should be a way to lead another user's view to a specific part of the workflow. We call such functionality "collaborative document navigation," since it signifies the possibility to change another team member's view on the workflow. This resembles face-to-face collaboration, where participants hint at a part of an object when communicating about it. Its implementation can be simplified, if the graphical representation reflects the tree structure of BPEL. Collaborative navigation is not only a better alternative to telepointers when collaborating in large, structured documents, but also improves awareness of dependencies between team members, since it enforces direct communication.

The configuration of an individual activity may depend on knowledge from a domain expert and from an infrastructure expert. Consequently, it makes sense to provide means that encourage a direct negotiation of these parameters, i.e. two team members should be able to debate the non-visual parameters of an activity. We thus have to provide means to enable two partners to collaborate on configuring a workflow activity.

Design Project Phases As the design process will often be split up into distinct phases, which can be subject to a number of iterations, the team leader needs tool support for terminating a collaborative phase. Phase switching has to be a very lightweight operation to anticipate a quick alternation of phases. During the analysis phase team members should not be forced to leave the session. The analysis and annotation phase is non-collaborative, since any changes of the workflow by users other than the team leader would make analysis impossible. Effectively, switching to the analysis and annotation phase is equivalent to locking the workflow for other users than the team leader. Hence it can be seen as a special case of the issues presented in the paragraph **Workflow Locking Mechanisms**.

Workflow Metrics The team leader will need support for workflow model analysis, which has two main purposes. First, the team leader needs to assess whether the workflow model is finished, i.e. it is runnable and it mirrors the requirements. While the first can easily be verified from the source code, the second has to be checked by a human being.

The second purpose is the detection of design flaws in sub-workflows. Design flaws are not identical with errors in the workflow model. In contrast to errors, design flaws do not affect syntactical correctness or prevent the execution of a workflow, and thus can only be detected by a human user.

We are particularly interested in comparing manipulated artifacts according to their size and complexity, to enable searching for "code smells" or "bad smells" which indicate a need of refactoring [Fow99]. In this context, complexity informally can be understood as the intricacy of understanding a specific artifact. For instance, Cardoso [Car05] defines workflow complexity as "the degree to which a process is difficult to analyze, understand or explain." It is not to be confused with algorithmic complexity of mathematical problems.

Scientific and business workflows are supposed to be reusable, so they should be maintainable, adaptable, and comprehensible. Scientific workflows often are larger than business workflows, making it harder to fulfill these demands. For scientific workflows, performance is an additional factor, so parallelism should be exploited and data-manipulation be avoided where possible.

Hence, a collaborative workflow design system should provide a view on several workflow metrics, which help the team leader in decision-making. In particular, recursively defined metrics are needed, that enable analyzing sub-trees of the BPEL model.

Delegating Responsibility After analyzing the workflow model, the team leader will prepare the next collaborative phase. This preparation especially includes decisions about who is supposed to work on which workflow parts. As sub-workflows in BPEL are induced by structured activities, this can be done by annotating activities. Hence, the team leader should have a means of annotating an activity to express the delegation of a specific task to a set of team members. In addition to the delegation itself, in some cases the affected sub-workflow should be locked for other team members. A special case arises, if a sub-workflow is to be locked for all team members to prevent further modification. The team leader also needs the right to remove these annotations at any time in the future. The different delegations should be individually visualized according to their effects for each team member.

Notification Web applications are limited by the available network bandwidth and latency compared to desktop software. Collaboration constitutes a logical peer-to-peer relationship between the clients. However, web-clients may only communicate with the server. Thus, we need to use a server-based notification scheme to enable server-initiated communication.

Operation Granularity One important difference between collaborative text editing and collaborative workflow design is the granularity of the manipulations. A workflow model consists of entities like activities, links and control structures. These are relatively complex objects, represented by parameter maps and object relationships. Inserting, deleting and connecting activities is handled by mouse input. Changing the configuration of an activity is a very coarse grained operation, when compared to inserting or deleting a letter. Hence operation messages are more coarse-grained when compared to collaborative text editing systems.

In comparison to collaborative vector graphics editing, the set of visual operations, i.e. adding, removing, connecting and dragging activities, is very small. According to Sun and Chen [SC02], *create* and *delete* cannot conflict with other operations, if operations are sorted in a partial ordering relation, yet *move* operations can conflict. The x and y coordinates and z-index of an activity do not affect the syntax of the resulting BPEL document, though, so inconsistencies between the clients concerning the position of activities could be permitted. Also, collaborative workflow design does not need any grouping or merging operations, which is difficult to implement in collaborative graphics editing systems according to Ignat and Norrie [IN]. In contrast to collaborative graphics editing the visualization of the control flow graph of a workflow may differ on the clients. Thus, collaborative workflow design is not a special case of collaborative graphics editing.

Scalability The whole software system defined by a workflow and its services will probably be the product of a large group of developers. However, the design team for the workflow itself will be significantly smaller. We expect workflow design teams to consist of fewer than ten persons, according to Brooks [Bro78] the upper limit of a “*small* sharp team.”

Brooks uses the analogy of a surgical team, which consists of team members that assume different roles. He stresses the role of a chief surgeon, similar to the team leader in our approach. We believe that this estimation is realistic as e.g. the case study on a collaborative workflow design process presented by Friese et al. [FSF$^+$06] has three participants who bring in their different kinds of expertise.

3.3 Workflow Development Processes in e-Biology

In this section, we investigate service usage in Biology and derive requirements for scientific workflow systems for biologists. To distinguish between software systems for bioinformaticians and biologists, we use the term *e-Biology* in cases where a system addresses biologists. We will not outline a domain-specific workflow development process for e-Biology, but describe the general usage of Web Services in biology departments, which yields a description of e-Biology as a class of processes, which we dub a “meta-process.” Cooperation on this level will take the form of *workflow sharing* for the re-use of existing workflows, a form of asynchronous collaboration. In contrast to business analysts, scientists generally expect to be able to sketch runnable workflows by themselves. The refinement of workflows at a lower level of abstraction is the second direction of asynchronous collaboration, enabled by compiling high-level domain-specific workflow notations to BPEL.

3.3.1 Computer Use and Infrastructure in Biology Departments

We now analyze contemporary e-Science usage in biology departments, at the example of the Center for Plant Molecular Biology (ZMBP[1]) of the University of Tübingen. The ZMBP consists of fifteen research groups which are grouped into four branches. We have cooperated with researchers from the plant physiology branch, which consists of seven research groups.

We have held several meetings and interviews with ZMBP research staff to evaluate their requirements on a bioinformatics workflow system. In an agile fashion, we have given demonstrations of our evolving workflow system to staff members with differing levels of expertise in e-Science.

In the ZMBP, *in silico experiment* web applications are frequently used to prepare “wet” experiments in the laboratory. Existing workflow solutions have been evaluated but failed to satisfy the users, especially due to intricate user interfaces and lack of user interaction.

In the course of our investigations, ZMBP researcher Dierk Wanke sketched an informal drawing of what he identified as common practices in e-Biology at the ZMBP (see Figure 3.6). We have used this sketch to simplify communication about

[1] `http://www.zmbp.uni-tuebingen.de`, last accessed: 22/09/2008.

e-Biology processes. The diamond shapes symbolize the three different kinds of input data mainly used in e-Biology, i.e. database ids (*accessions*), amino acid sequences representing proteins, and nucleotide sequences representing genes. These three kinds of data items can be consumed or produced by Web Services in varying data formats. Data items representing an entity in a specific type in general can be converted to corresponding data items of another type. Ids are used to represent gene sequences or protein structures in a specific database. Via search services, gene or protein data can be mapped to ids representing the same or related genes or proteins. Genes, on the other hand, serve as a blueprint of protein structures. For example, a database identifier can be mapped to a gene sequence, which can be mapped to a specific protein structure encoded by the gene sequence. The different data formats used by e-biology Web Services usually are encoded as string parameters. The rectangles symbolize different classes of tasks, which are usually performed via Web Services, provided as Web Forms. The ZMBP research staff emphasizes the need to "manually" inspect data produced by Web Service invocations.

3.3.2 Roles in e-Biology

Letondal and Mackay [LM] have investigated the cooperation of biologists, bioinformaticians and software engineers in software development for biology applications. In a survey at the Institut Pasteur in Paris, they identified five different groups [LM]:

- *occasional users*, with low computer usage (36%)
- *non-Unix users* using PCs, mainly with statistical software (15%)
- *young scientists* with interest in bioinformatics, who where able to build web sites (15%)
- *learners*, who had taken a computing course recently (15%)
- *gurus* with profound computing and programming skills (6%).

As the survey took place in 1996, we expect that the number of computer-using biologists will have increased significantly. On the other hand, biology curricula usually do not encompass programming courses. Thus, we expect computing skills of biologists to vary strongly.

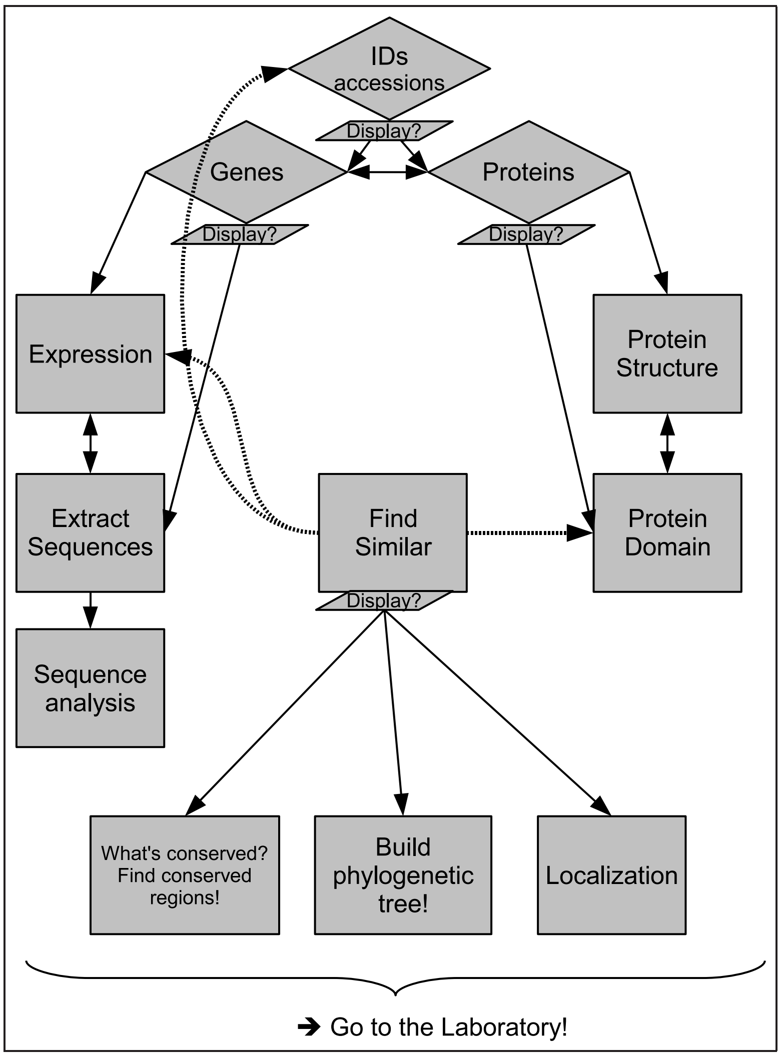


Figure 3.6: A meta-process of e-Biology, as sketched by Dierk Wanke.

3.3.3 The Meta-Process of Workflow Usage in Plant Science

Figure 3.7 and Figure 3.8 show the representation of a *Meta*-Process of computational plant science, given in the Event-Driven Process Chain notation. We have derived the Meta-Process from our interviews at the ZMBP and from informal drawings from ZMBP staff members as described in the previous Section.

The figures do not represent a specific bioinformatics workflow or a general process of research, but outline the daily usage of web applications by biologists. Every path between the events “Data entered” and “Data” represents a pipeline of data processing that could be automated. The “view/edit” activity triggers either the execution of a new pipeline or the end of the in silico experimenting phase. In many cycles, the “view/edit” activity can effectively be left out, and is only enforced by the “copy & paste” mode of operation. In some cycles, however, it is necessary to enable a biologist to edit the data or decide, whether to go on with the workflow. Additionally all input and output data has to be recorded for tracing the experiment. Figure 3.8 points out, that once genomic sequence data is available several options for further analysis exits, e.g. sequence alignment or comparison (sequence analysis), or building phylogenetic trees depicting evolutionary relationships.

Effectively, the set of cycles performed by a biologist before going on to a “wet” experiment, forms a tree, where the result of one in silico experiment is the input to a set of other in silico experiments.

3.3.4 e-Biology Use Cases

From our observations, we have derived a UML 2 use case diagram of e-Biology workflow systems (see Figure 3.9). We have identified six use cases and five types of actors, i.e. traditional biologists, “E”-Biologists, and experienced “E”-Biologists, as well as software engineers and workflow experts.

The central use case is *workflow execution*. We have to distinguish between *workflow development* and *workflow sketching*. The use case workflow development has been investigated in related work, as presented in Chapter 2. The prime example in e-Biology is the Taverna system, while BPEL is the standard case in the industry. As we have seen in Chapter 2, in Business Process Management, *workflow sketching* is done via a number of different notations, some of which support varying levels of abstraction (e.g. UML), while in e-Science, making the distinction between low-level and high-level workflow development is yet uncommon. *Workflow refinement* is a special

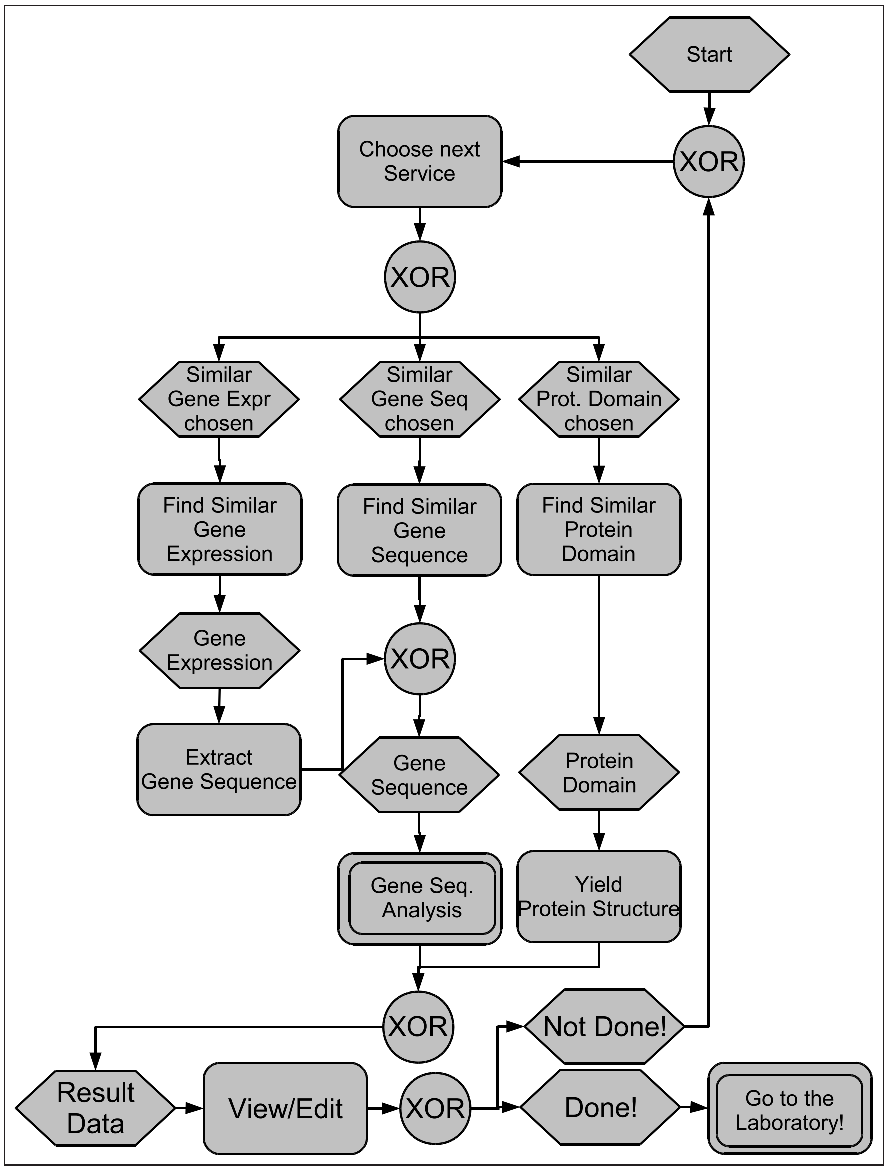


Figure 3.7: The Process of in silico Microbiology in Plant Science (Part 1).

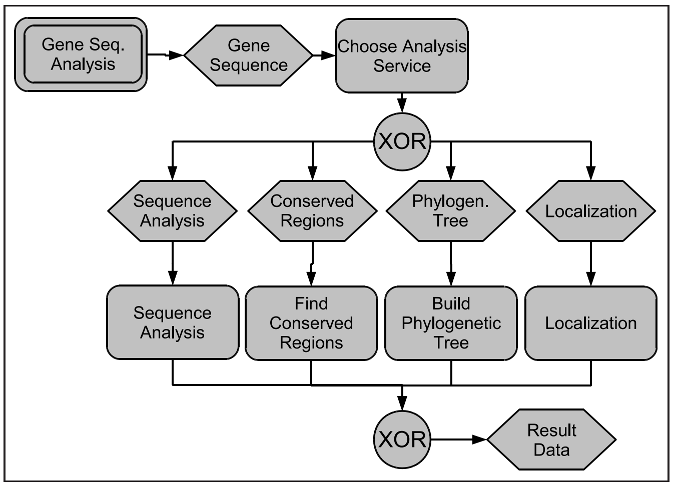


Figure 3.8: The Process of in silico Microbiology in Plant Science (Part 2): Possible Analysis steps for Gene Sequences

case of *workflow development*, which is a special case of *service development*, if the resulting workflows are deployed as Web Services.

General Constraints Two biologists B_1 and B_2 are employed at a research department. They share their offices with several colleagues, with whom they may or may not collaborate on a set of projects. Their second work place is the laboratory, where several work benches are available, some with Internet access. While B_1 lacks any interest in or education about programming, the biologist B_2 feels comfortable using computer technology, has at least some programming skills, and is willing to invest time for learning new technologies. It cannot be expected that B_1 or B_2 have administrative rights on all workstations they might use for e-Biology.

Use Case: Workflow Execution B_1 or B_2 want to trigger the execution of an existing workflow. The results of the workflow execution are to be recorded together with any intermediate results. B_1 or B_2 may use workflow execution for the preparation of laboratory experiments as well as during an experiment to check results. In contrast to the execution of Web Services in general, users expect the workflow not to act as a plain “black box”, but to reveal information about the actual workflow run. In particular, users are interested in the *provenance* of the produced results, i.e. the question which chain of operations yielded the result data.

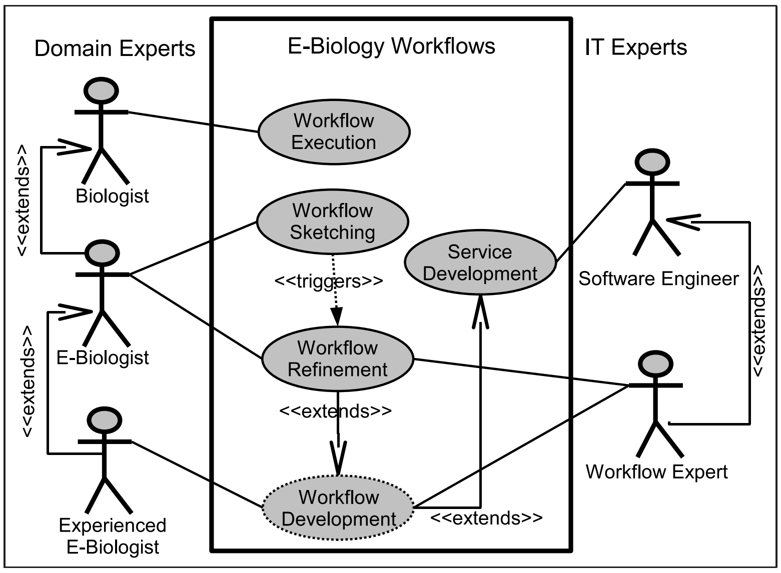


Figure 3.9: A UML 2 Use Case Diagram of e-Biology.

Use Case: Workflow Sketching Similar to business Subject Matter Experts, B_1 wants to formalize the steps she expects taken in an e-Biology process. We thus see workflow sketching as the equivalent of business process modeling in e-Science. B_1 expects to be able to sketch pipes or exploratory steps, which can be extended to arbitrary processes if necessary. In contrast to business SMEs, B_1 often wants to try out runnable versions of the sketched processes immediately.

The modeling notations have to be less complex than actual workflow languages, thus limiting the set of possible workflows. The set of workflows W_N which can be sketched in a notation N is a subset of all workflows W. For a notation N which is easily comprehensible by workflow non-experts, we can assume W_N to be a strict subset of W. The set of runnable sketchable workflows $W_{N,r}$ is a subset of W_N. Thus $W_{N_r} \subseteq W_N \subset W$.

Use Case: Workflow Development Workflow development is the use case which is addressed by existing bioinformatics workflow systems. In contrast to B_1, B_2 is willing to invest time for actual workflow programming. Workflow development can be a special case of service development, if the workflow system supports the deployment of a workflow as a Web Service, which is not necessarily the case.

Use Case: Workflow Refinement Workflow refinement can be interpreted as a special case of workflow development, where existing workflows are extended or refactored. Since workflow sketching yields workflows, which are limited in their expressiveness, augmenting workflow sketching with workflow refinement extends the set of possible workflows in a scientific workflow management system to W.

Use Case: Service Development Service development is a use case, which is not specific for a life science workflow management system. It remains an instance of software engineering, mainly performed by bioinformaticians and computer scientists.

3.3.5 Requirements of e-Biology Workflow Systems

Application Requirements for Scientific Workflows

Gil et al. [GDE+07] have identified three different application requirements for next-generation scientific workflow management systems. First, such systems have to support collaborations between scientists and other domain experts. The second requirement is the "reproducibility of scientific analyses and processes." Finally, "flexible environments" should both support every-day scientific analysis and enable unforeseen new characteristics of workflows. We will now analyze how these general requirements apply to bioinformatics.

Supporting Collaborations Scientific progress is inherently a collaborative process, in which experts from different domains participate. Hence, Gil et al. [GDE+07] see a need to "orchestrate the steps of scientific discovery and bridge the differing expertise of collaboration members." In life sciences, collaborations consist of biologists, bioinformaticians, and software engineers. Biologists and bioinformaticians will prefer to work with "stripped down" workflow notations that enable them to easily sketch *runnable* workflows, while software engineers will prefer a complete workflow language that effectively gives them the power of a programming language.

Reproducibility of Scientific Analyses and Processes In the "copy & paste" processes explained in Section 3.3.3, biologists see all results of all service invocations. However, there is no guarantee that a service will always give the correct answers, as databases can be corrupted or services be subject to software errors. Hence, each workflow result will have to document both the actual control-flow and data-flow of the workflow instance, i.e. the results of every activity in the workflow are a part of its result. Thus, we need to provide an interface for biologists to easily navigate through results, and pick up information about intermediate results as needed.

Flexible Environments Gil et al. [GDE$^+$07] point out that “systems must be flexible, in terms of supporting both common analyses that many scientists perform, as well as unique analyses,” that may incorporate “workflows with previously unseen combinations and configurations of models.” In terms of programming languages, this means providing a high-level domain specific language that is compiled to a representation in a general programming language. Hence, we need to enable two different views on the workflow models. Limited biology workflow models can enable every day life science research for non-expert users. If biological workflow models are compiled to a general-purpose workflow language, software engineers can provide arbitrarily sophisticated programming constructs if they are needed.

Biology-specific Requirements

Intuitive and Simple Composition In contrast to other sciences (e.g. physics, bioinformatics), biology education does not encompass software programming. Thus, biology-specific workflow notations have to be kept very simple and may only provide necessary features, thus leaving out any control structures where possible. Barker and van Hemert [BvH08] point out, that “wet laboratory biologists are uncomfortable using even the most abstract workflow tools currently available.”

Immediate Service Invocations at Build Time For two reasons, users may want to invoke services at build time. First, this enables an exploratory way of workflow construction. As Gibson et al. [GGW$^+$07] point out, users sometimes will start composing a workflow with a given piece of information without knowing the goal. On the other hand, life science web services are publicly provided by research institutions, that cannot guarantee the correctness or the availability of a service. Thus, trying out a service before incorporating it into a workflow enhances the chance of yielding a correct workflow for a given task.

Semantic Service Discovery When constructing a workflow, a typical task is finding an applicable service, that can serve as a successor for a given activity (see Section 3.3.3). However, the opposite scenario can also be the case, where a predecessor for an activity is needed. This situation arises, if a user knows, which result type she wants and is searching for a way to get to this result type.

User Interaction A major issue for the usability of workflows is the possibility to interact with a workflow at predefined events in the workflow. This matches the “view/edit” activity from Section 3.3.3. In data-flow oriented workflow models, these events can be identified as the invocation or return of activities. In aspect-oriented terminology user interaction happens at a *pointcut* before or after the invocation of an activity [KLM$^+$97]. After viewing the input or the output of an activity, the user may decide either to resume the workflow, to alter the data and resume or to quit the workflow execution.

Group Level Workflow Sharing Roure et al. [RGS07] have identified the necessity to share workflows with other scientists, similar to publishing experiments. However, as Goble has pointed out, life scientists would “rather share their toothbrushes than their data” [The08], i.e. they share preliminary results only within their research group. Hence, before publishing their workflows, biologists will want to share their workflows on the group level.

Technical Requirements

BioMoby Support A new workflow system will only be accepted, if a sufficient number of services is available. Hence, it should re-use an existing and well-established web service database. The BioMoby framework provides such a database and also enables semantic service discovery [WL02, The08].

Web Integration The web browser is the standard e-Science tool for biologists (see Section 3.3.1). Users are most likely to accept a new system if it is integrated into the environment they are used to. On the other hand, department policies often prevent the installation of new software. Even if department policies allow installing new software, many biologists prefer using pre-configured and centrally managed systems. These arguments make the web integration of a bioinformatics workflow system highly desirable.

Standards Conformance Current bioinformatics workflow systems lack a common standard. To guarantee that a workflow can be understood and executed by future users, we need to ensure its compatibility to a standard workflow language. In the business world, the *Business Process Execution Language* (BPEL) has gained general acceptance as the standard workflow execution language [AAA+07]. Thus, Barker and van Hemert [BvH08] recommend to use novel workflow notations in conjunction with BPEL.

3.4 The development systems HOBBES and CALVIN

We have described a collaborative workflow development process, as well as the general structure of e-Biology processes. Figure 3.10 visualizes the mapping of the processes, which have been investigated in this chapter to actual tools, which will be presented in the further course of this dissertation.

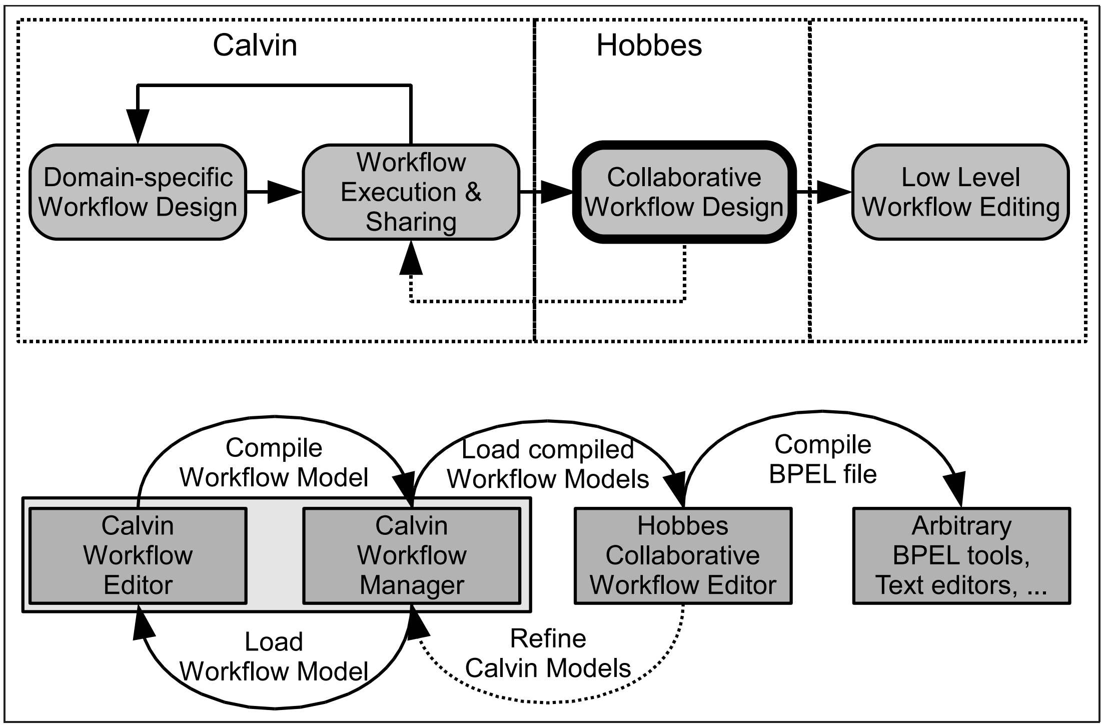

Calvin
Hobbes
Domain-specific Workflow Design
Workflow Execution & Sharing
Collaborative Workflow Design
Low Level Workflow Editing
Compile Workflow Model
Load compiled Workflow Models
Compile BPEL file
Calvin Workflow Editor
Calvin Workflow Manager
Hobbes Collaborative Workflow Editor
Arbitrary BPEL tools, Text editors, ...
Load Workflow Model
Refine Calvin Models

Figure 3.10: Mapping tasks in collaborative workflow development to software systems.

While HOBBES enables synchronous collaboration on the same workflow model for a team of users, CALVIN enables the development of a workflow by a single domain expert. CALVIN also allows users to share their workflows on the server. CALVIN workflow models are compiled to BPEL and can be exported as HOBBES object models. On the HOBBES level, domain experts and software engineers are able to cooperate in refining a workflow model. We have also developed the prototype of a collaborative WSDL editor, called LEVIATHAN. We will discuss HOBBES in Chapter 6 and CALVIN in Chapter 8. Prior to our presentation of the HOBBES implementation, we will discuss the approach of HOBBES to consistency in Chapter 4, and workflow metrics used in HOBBES in Chapter 5. Additionally we will present the LEVIATHAN prototype in Chapter 7.2. The integration of HOBBES and CALVIN using Web 2.0 techniques is discussed in Chapter 9.

Chapter 4

Collaboration and Consistency

Consistency is contrary to nature, contrary to life. The only completely consistent people are the dead.

(Aldous Huxley, "Do What You Will," 1929.)

A major concern when implementing a synchronous collaborative workflow design tool is to keep the workflow models on all participating clients consistent. This can be achieved either by preventing inconsistencies between the clients, or by repairing them after a limited amount of time.

First, we characterize variants of inconsistency, and afterwards describe corresponding consistency algorithms. Then we describe the approach taken in the HOBBES system. A special case of consistency preservation arises, if clients join a collaborative session at an arbitrary time after its beginning (*late joining*). We conclude this chapter by describing the approach to *late joining* used in HOBBES.

4.1 Manifestations of Inconsistency

According to Sun et al. [SJZ$^+$98], inconsistency can manifest itself as *divergence*, *causality violation*, or *intention violation*. As we discuss these types of inconsistency, we will use examples, which resemble the situation of concurrent operations on the object model of HOBBES.

Divergence Divergence means, that the resulting model of a collaborative design process differs on the participants' editors. Then it is impossible to decide which model is the "correct" one, or even if a correct model exists at all. Temporal divergence can be accepted in many cases. For example if a user's actions are immediately executed on his/her client, the effects of these actions could be altered when the collaboration system detects inconsistency. Thus, how much temporal divergence a system can accept is a design decision. At least at the end of the design session, however, the state of the manipulated model has to be the same on all clients.

Causality Violation Causality violation can happen if two operations, one of which depends on the other, arrive at a client in the wrong order. For example, the deletion of an element can only succeed if the element has already been created. Depending on the implementation of a collaborative system, two different negative effects are possible in this example. On the one hand, a delete operation could be lost. On the other hand, the wrong element could be deleted. Both cases of causality violation would eventually lead to divergence.

Figure 4.1 shows an example of causality violation, which can happen when collaboratively editing tree-like models, similar to BPEL models. The Figure shows the screens of two clients which concurrently manipulate a tree-like object model. Client 1 has a view on object *B* and initiates adding a child object *C* to *B*. Concurrently client 2 has a view on object *A*, the parent of *B*, and initiates the deletion of *B*.

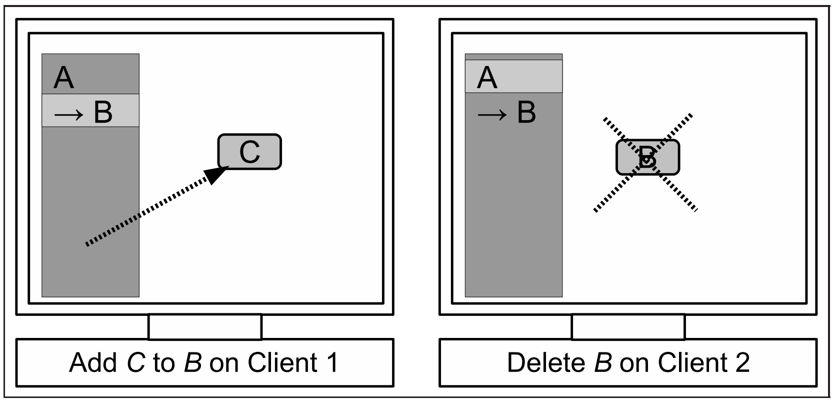


Figure 4.1: Concurrent Add and Delete. The user of client 1 wants to add an element *C* as a child to element *B*, while the user of client 2 intends to delete *B*. See Figure 4.2 for possible results.

Figure 4.2 visualizes the possible results of the example, if the system processes the commands in the order *delete* **B**, *add* **C** *to* **B**. The system can decide to ignore the second operation, yielding the first possible result. This behavior is appropriate, if adding a new element to the object model is a simple operation. However, if adding an element is relatively complex, e.g. by answering several questions to a Wizard, the other possible results may be appropriate. If *C* does not depend on the existence of *B*, it can be added to *A* instead (possible result 2). Otherwise, a viable solution may be to undo the deletion of *B* (possibility 3).

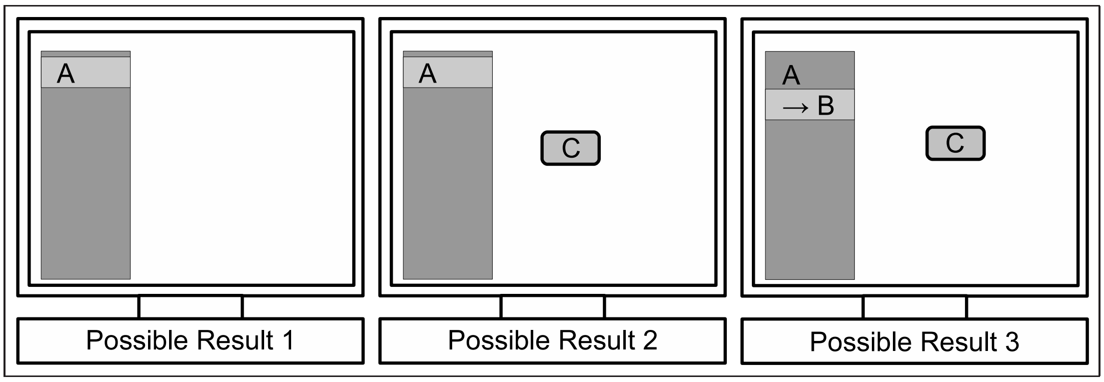


Figure 4.2: Concurrent Add and Delete: Possible Results.

Intention Violation Intention violation means that the *intended* effect of an operation is compromised by the order in which operations are processed, e.g. if two clients demand to move an element of a sequence up one step, the result could be that the element moves two steps up. Obviously the real intention behind an operation cannot be determined in every case.

For example, two users could simultaneously decide to add a service invocation activity to the same sub-workflow. Then it is unclear whether both user wanted to add the same service invocation, or whether they wanted to add different invocations. Hence, the system cannot decide that only one activity has to be added, even if both users wanted to invoke the same service and only one invocation makes sense.

Another example for intention violation can arise if two users concurrently decide to move an element in a sequence (see Figure 4.3). Both user u_1 and user u_2 intend to switch the activities 1 and 2, by moving 1 one step down in the sequence $s = (1, 2, 3)$. Both clients thus issue the command “down activity 1 by 1 step” to the collaboration system. With the information included in both commands, the system will yield the sequence $s = (2, 3, 1)$, while $s = (2, 1, 3)$ has been intended by the users. In contrast to the previous example, this case of intention violation can be prevented by including a precondition in the command, e.g. by demanding that $pos_s(2) = pos_s(1)+1$, where $pos_s(x)$ yields the position of x in the sequence s.

4.2 Algorithms for Enforcing Consistency

As in all distributed systems, one can distinguish between consistency algorithms for centralized client/server and for decentralized peer-to-peer architectures.

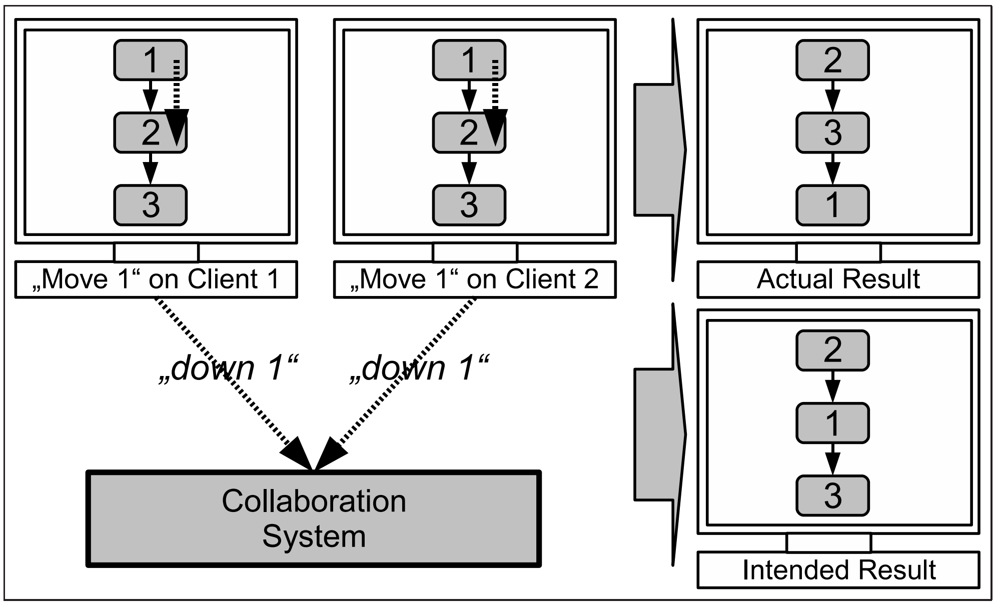


Figure 4.3: Concurrent Moves can lead to intention violation.

In his dissertation, Vogel [Vog, ch. 4] distinguishes between *soft state* and *hard state* approaches to sharing a model. In a soft state approach, one or more participants periodically announce the current state of the model. This method is simple and does not need a reliable transport protocol. Its disadvantages are possibly high notification times, frequent temporary inconsistencies, and very high data rates, which makes it unsuitable for use on the web.

In a hard state approach, all operations are explicitly and immediately sent to all clients. It needs a reliable transport protocol, extra support for late joining clients, and consistency has to be enforced by a special consistency algorithm. We further have to distinguish between *optimistic* and *pessimistic* consistency algorithms. In her dissertation, Ignat [Ign06, ch. 2] gives an overview of optimistic and pessimistic consistency algorithms.

Optimistic Consistency Algorithms *Optimistic* consistency algorithms execute operations locally at once and try to recognize and fix inconsistencies if they appear. Since they often rely on every editor instance to enforce consistency, they are well suited for peer-to-peer systems. Optimistic algorithms enable a high responsiveness, but are hard to implement and verify.

A simple optimistic approach is to leave out any consistency algorithms at all, and solely rely on communication between human operators to prevent or deal with inconsistency. Ignat [Ign06] describes such attempts as *social protocols for mediation*. A different, yet related approach is to detect concurrent changes via time stamps and let conflicts be resolved by *human intervention*.

In *optimistic locking* protocols, a client requests a lock on those objects which are to be manipulated. The client does not wait until the lock request has been answered, but immediately allows the user to perform the corresponding operations on the local object model. If the request is denied, the effects of the operations are reversed to the previous state.

Optimistic serialization algorithms assume one globally correct order of incoming operations. On every client, all locally known operations are stored in a linearly ordered history buffer, where every operation is assigned a priority. If conflicts appear, the conflicting operation with the lower priority and those operations depending on it are removed from the history buffer, which is re-executed afterward. The collaborative graph drawing system *Draw-Together* is an example of the application of optimistic serialization [IN06].

In *multi-versioning* protocols, in the case of a conflict, multiple versions of objects are created. Sun and Chen [SC02] have presented the collaborative graphics editor GRACE, which uses a combination of *operational transformation* with multi-versioning of objects. As some types of conflicts cannot be resolved by operational transformation, it keeps multiple versions of the affected objects. Ignat and Norrie [IN06] point out that keeping multiple versions yields the problem of how users can navigate through the alternative versions. In GRACE multiple object versions are displayed at the same time and marked by highlighting.

An example of an optimistic approach often used in peer-to-peer architectures is *Operational Transformation* [EG89, SE98]. Local operations are immediately executed and then broadcast to the other editors. All operation messages are tagged with vector clocks, thus implying a partial ordering of the operations. Incoming operations from the peers are transformed to fit into the local model, thus preventing inconsistencies.

In optimistic *validation techniques* each operation performed on an editor is transmitted to a central node, checked for validity and afterwards forwarded to all clients or revoked if invalid. The collaborative text editor TeNDaX [HWD05] is an example of this approach, using an object-oriented database holding a central model of the text. For each character, the TeNDaX database contains an entry with a unique identifier and links to the entries of the previous and next character in the text. In the

TeNDaX editor client, the text is presented as an array of consecutive characters with a mapping to the corresponding database IDs.

Pessimistic Consistency Algorithms *Pessimistic* consistency algorithms first check the validity of an operation and only then propagate it. This means, that either at least those parts of a document which can be affected by an operation have to be locked temporarily, or that the operations have to be processed in a sequential order by one designated host.

Turn-taking protocols explicitly grant exclusive access to the whole document to one user at a time, while *pessimistic locking* approaches grant privileged access to document parts. Both approaches can further be distinguished by whether privileged access is managed by the system or by a human coordinator. In *access control* protocols, the success of an operation depends upon the question, which user is allowed to actuate which kinds of operations on a set of objects. In *pessimistic serialization* protocols, all operations are executed in the same order at every client.

4.3 Consistency in the HOBBES System

In Chapter 6, we will present our collaborative Rich Internet Application HOBBES, which implements the approach to consistency preservation that we will now discuss. Since we assume small teams (see Chapter 3.2.3), a centralized approach can be applied, thus enabling an implementation within a client/server architecture, such as the Web architecture.

On the HOBBES server, incoming operations are processed sequentially by a *controller* routine to prevent both divergence and causality violation. Thus, in the terminology used by Ignat [Ign06], collaboration in HOBBES is based on a *pessimistic serialization* protocol.

Moreover, the controller routine checks the validity of incoming operations to avoid a corruption of the central workflow model. An operation is considered to be valid, if all potentially affected objects are available, and if the user has the right to manipulate the affected sub-workflow. Thus, before actually executing an operation, the server gathers all needed elements. If a resource is unavailable, i.e. if it is not a part of the workflow model or if it has been locked, the operation is aborted. Checking the preconditions of an operation is simplified by the tree structure of BPEL, the small set of possible operations and their precise semantics.

Operations are impossible, if they depend on a workflow element which does not exist. In the following cases, the deletion of an element by client *B* makes an operation intended by *A* impossible:

1. *A* wants to add an activity α to a structured activity σ. *B* has deleted σ, which has not yet been propagated to *A*. The controller fails to look up σ and then decides to deny the operation. *A*'s client gets informed about the deletion of σ. The client then automatically navigates the view to the workflow's top element and notifies *A* about the deletion.

2. *A* wants to connect α_1 and α_2. α_2 has been deleted by user *B*. The controller denies the operation and propagates the deletion of α_2 to *A*. α_2 is removed from *A*'s screen.

3. *A* issues a drag operation of α, which has been deleted by *B*. This case is handled as in 1 and 2.

4. *A* wants to delete activity α, which has already been deleted by *B*. The controller ignores *A*'s operation and just propagates *B* deletion operation.

5. *A* has opened a configuration window for the properties of α. While *A* is filling values into the form, *B* deletes α. When *A* commits the form, the server informs *A* about the loss of α and ignores the form data.

A special case consists of adding an activity to a structured activity that can only have one child (e.g. ForEach). If *A* and *B* concurrently try to add a child to such an activity, only one operation will succeed. If user *A*'s add-operation fails, she will see the addition of *B*'s activity and receive additional feedback by a pop-up notification.

Another special case is changing the sequence number of an activity that is a child of a Sequence or another structured activity that depends on the order of its children. Sequence numbers can only be changed by stepwise increments or decrements, i.e. by switching the positions of two activities α_1 and α_2. Thus, the controller has to check the existence of σ, α_1 and α_2 and that $pos(\alpha_1) - pos(\alpha_2) = 1$ to prevent intention violation.

Operations can either affect the BPEL structure defined by the model or only its visual structure. Changing the position of an element within a Flow affects the visualization of the workflow, but not the BPEL model itself. Thus, concurrent drag operations within a Flow can lead to overlapping activities, which could be considered as an intention violation (see Figure 4.4). However, attempts by the system to prevent such a situation would probably lead to irritation of the users who can easily resolve it by themselves by dragging the activity on top. If two concurrent drag operations of the same element occur, it is impossible to decide, which one should succeed from both user's points of view, because drag operations are solely characterized by their outcome.

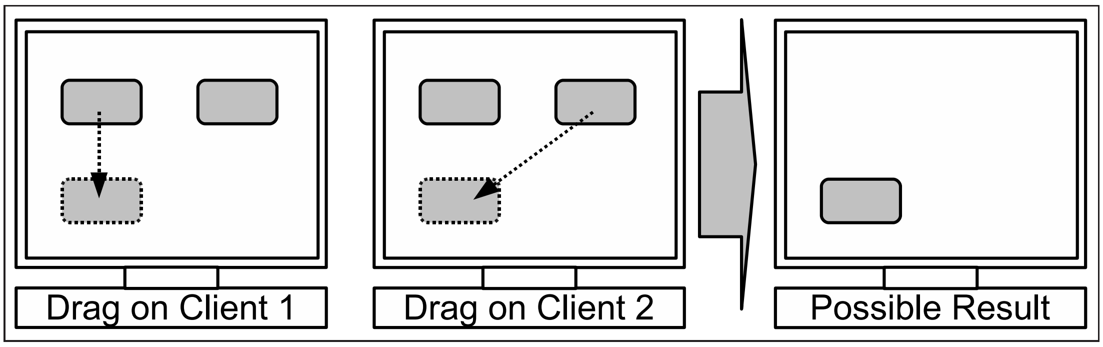


Figure 4.4: Concurrent Drag Operations can lead to overlapping elements.

In HOBBES, non-visual properties are edited with form widgets and loaded on demand. Hence, changes which affect non-visual properties only do not need to be propagated to the clients. Examples for non-visual properties include the configuration of an invocation or the condition in a loop activity. If two clients concurrently update a set of non-visual properties, only one of these modification operations will succeed. This does not compromise causal consistency, since any set of such modifications is applied to the workflow model in a sequential order. The view on a non-visual aspect of the workflow model can be temporarily divergent, though, until a all clients have reloaded the according set of properties.

Table 4.1 gives an overview of collaborative editing operations in HOBBES, as well as their resources and preconditions.

4.4 Late Joining

As has been noted in Chapter 3.2.3, we cannot assume all clients to be present at the beginning of a design session. A client can join a session after any number of operations have been performed on the model. All announcements of changes by the server are tagged with a counter. If a client joins a session, it first subscribes to operation notifications and begins buffering all incoming operation messages. The client then sends a join-request to the server, whose reply includes the current BPEL

Operation	Resources	Additional Conditions
Add new Activity to Parent P.	P	$typeof(P) \in \{\text{“}Flow\text{”}, \text{“}Sequence\text{”}\}$ $\vee$ $(typeof(P) \in \{\text{“}Scope\text{”}, \text{“}While\text{”}, \text{“}RepeatUntil\text{”}, \text{“}ForEach\text{”}\}$ $\wedge\ children(P) = \emptyset)$
Link from A_1 to A_2	A_1, A_2	$\exists F \in Activities(X)$, $typeof(F) = \text{“}Flow\text{”}$ $\wedge \{A_1, A_2\} \subset descendants(F)$ $\wedge A_1 \notin descendants(A_2)$ $\wedge A_2 \notin descendants(A_1)$ $\wedge A_1 \notin successors(A_2)$ $\wedge A_2 \notin successors(A_1)$
Drop X to position $(left, top)$ in F.	X, F	$typeof(F) = \text{“}Flow\text{”}$ $\wedge F = parent(X)$
Move A_1 up behind A_2.	A_1, A_2	$pos(A_1) - (A_2) = 1$ $\wedge parent(A_1) = parent(A_2)$ $\wedge typeof(parent(A_1)) = \text{“}Sequence\text{”}$
Add new branch to X.	X	$typeof(X) \in \{\text{“}If\text{”}, \text{“}Pick\text{”}\}$
Alter properties of X.	X	
Delete X.	X	

Table 4.1: Conditions for the validity of operations in HOBBES. The system first checks whether the necessary resources are available. Afterwards, the additional conditions are evaluated. The function $type(X)$ returns the activity type of activity X, while $children(X)$ yields the set of its children. Starting from activity X, $descendants(X)$ is recursively defined as $descendants(X) := children(X) \cup \bigcup_{C \in children(X)} descendants(C)$. For any X, Y, $Y \in successors(X)$ iff. there exists a control flow path from X to Y.

model and the value of the operation counter at the time the server received the join-request. Joining and leaving are handled atomically on the server. When the client receives the reply, it constructs the BPEL model and then processes any messages that have been lost between the join-request and reply. This ensures that after the join process is complete, the client has a consistent document model.

Figure 4.5 shows a sequence diagram, which describes late joining. Client $n + 1$ attempts to join the group of clients $1 \ldots n$. First, it subscribes to the notification system of the server, afterwards receiving any broadcast messages, including those messages, which modify the workflow model. Then it sends a join request to the server, which sends the current state of the reference workflow model to the client. Note, that the client uses a different channel for sending messages to the server and receiving answers, than for receiving notification messages. Hence, any changes, other clients actuate on the workflow model may be received earlier by client $n + 1$, than the workflow model state. Thus, these change requests are stored in a buffer, until the workflow model state has arrived at client $n + 1$.

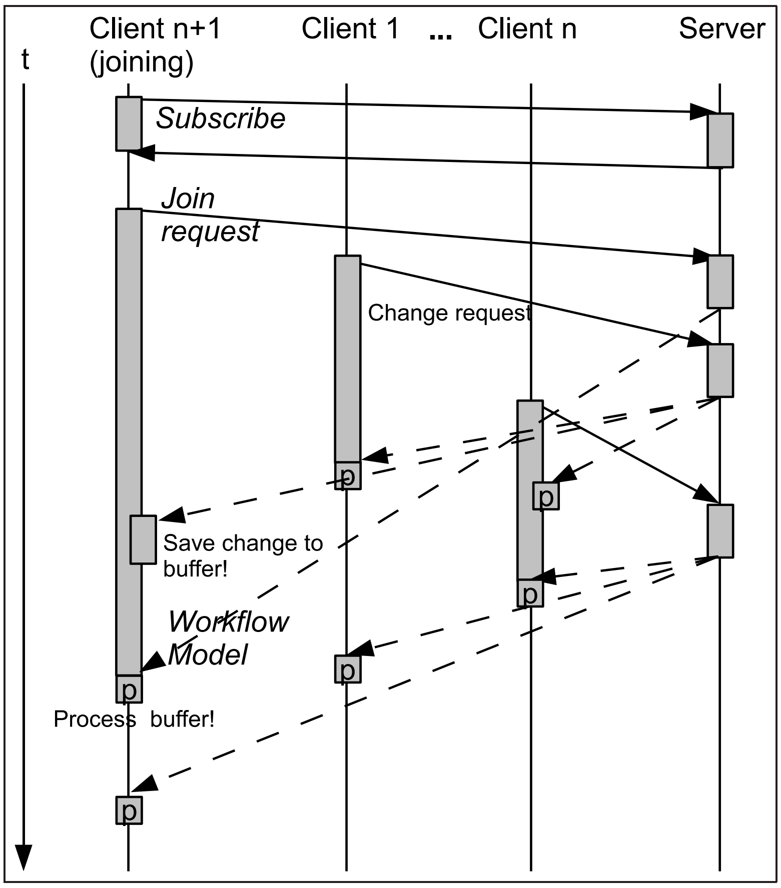


Figure 4.5: A sequence diagram depicting message flow in a late joining scenario. The boxes labeled *p* represent the processing of change requests on the clients.

Chapter 5

Workflow Metrics

There is nothing new to be discovered in physics now. All that remains is more and more precise measurement.

(William Thomson, 1st Baron Kelvin, quoted after "Superstring: A theory of everything?," P. Davies, J. Brown, 1988.)

This Chapter discusses a set of workflow metrics for steering collaborative BPEL development processes. First, we motivate the necessity of workflow metrics in the context of collaborative development in Chapter 5.1. Afterward, we analyze which properties of BPEL affect possible workflow metrics. We then introduce metrics for the size, the complexity of control and data flow, and explicit parallelism of BPEL workflows, as well as some additional metrics. In Chapter 5.8, we evaluate the metrics according to a set of desirable properties. We then use the metrics to define *code smells*, which help team leaders in identifying design flaws in workflow models, and afterwards conclude the Chapter with an example of workflow analysis and redesign using the metrics.

5.1 Evening out Intricacies of Collaboration

Although collaborative workflow design fosters the integration of domain-specific and IT-specific knowledge within a workflow, it may increase the probability of design flaws. While every collaborator brings in additional expertise, he also is a source of unpredictable behavior. On the other hand, no single person has a view of all changes taking place in the model. Thus, the structural quality of a workflow model can deteriorate in the course of collaborative workflow design. To even out these effects, the team leader will repeatedly analyze the workflow model to identify necessary improvements (see Chapters 3.2.2 and 3.2.3).

In this scenario, we aim at improving a workflow model in a way, which does not affect its functionality, but improves non-functional qualities, such as sustainability, understandability, or performance. In object-oriented programming, such measures are commonly dubbed *refactoring* [Fow99]. Refactoring is initiated after the detection of *code smells*, i.e. symptoms in source code which indicate that the code could be improved [Fow99].

A traditional approach to support decisions about code quality is the use of *Software Metrics*. According to Fenton and Neil [FN00] *Software Metrics* is used as an umbrella term describing "the very wide range of activities concerned with measurement in software engineering," including "classic software 'metrics'," which "produce numbers that characterise properties of software code." A special form of these traditional software metrics are *Workflow Metrics*, i.e. numerical measures of the properties of workflow models.

In the course of this Chapter, we present a set of workflow metrics for BPEL and use these metrics to define code smells for BPEL models.

5.2 Workflow Metrics for BPEL

As we intend to use workflow metrics to support the decision-making sub-process of collaborative workflow design, we need to consider which metrics can be applied to the Business Process Execution Language in a meaningful way. The following aspects of BPEL affect what can be measured in BPEL workflow models:

1. Web Services are treated as "black boxes," giving no insight into their functionality or internal structure.
2. BPEL lacks any concept of modules or objects.
3. It does not include any notion of functions or procedures.
4. It is a structured programming language, since control structures are block-oriented and links between activities are limited to form directed acyclic graphs.
5. In contrast to other workflow languages, control-flow and data-flow are defined separately.
6. BPEL enables parallel task execution.
7. BPEL workflow models are inherently structured as a tree.

Item 7 indicates that we can develop recursively defined metrics that indicate the status of individual sub-workflows. Our choice of metrics is also affected by the necessity to use only such information which is available at design time.

We will now present a set of metrics for measuring *size*, *control-flow*, *data-flow* and *parallelism* of (sub-)workflows, some of which have already been presented by other authors. Afterwards we will evaluate the set and present its application to identify code smells.

5.3 Size Metrics

The size of programs is often measured using the Lines-Of-Code metric (LOC), which counts the number of statements within a program source code. Since workflows are usually designed with graphical editors, the LOC metric looses impact on design considerations. An LOC implementation could count the lines of code of an XML document generated by a graphical editor. The number of lines would depend on the used code formating algorithm, though.

Cardoso et al. [CMNR06] have proposed to count the number of basic activities (*NOA*) in a workflow as an alternative to the Lines-Of-Code metric. In contrast to the LOC metric this leaves out any information about control structures. Thus, they also propose to count the number of activities and control structures (*NOAC*) as well.

In BPEL, the Assign activity may contain any number of copy statements, each of which represents one assignment. As an additional size metric, we propose counting the number of activities, control structures, and copy statements (*NOACC*).

5.4 Measuring Control Flow

5.4.1 McCabe's Cyclomatic Complexity

Thomas McCabe [McC76] has proposed to measure the cyclomatic complexity of programs, i.e. the number of linearly independent paths in the control flow graph. McCabe's Cyclomatic Complexity (MCC) is often used to evaluate the complexity of individual modules or functions. According to [WM96], it "measures the amount of decision logic in a single software module." In a study for the National Security Agency, Drake [Dra96] found MCC to be a "credible quality indicator." The cyclomatic number of a graph g with n vertices, e edges and p connected components is defined as:

$$V(g) = e - n + p$$

For software modules, p can be identified as the number of entry points, e.g. C++ method declarations. In BPEL, the entry points of a workflow are *start activities*, i.e. Receive or Pick activities with the attribute `createInstance = "yes"`. Each BPEL workflow contains at least one start activity [AAA+07, Chapter 5.5]. For every binary decision in a control structure (i.e. every binary predicate in a conditional), the e is increased by 2 and n is increased by 1. In structured programming languages, MCC can be computed by counting the number of binary decision points d, and setting $MCC = d + 1$.

We calculate the MCC metric by counting the appearances of

- Receive, where the attribute `createInstance` equals `"yes"`, as such activities signify the entry points in the control flow
- conditional branches (i.e. If branches or Pick branches), loops and events
- the *join*/*transition conditions* of Flow links
- logical conjunctions and disjunctions in XPath Expressions.

In a Flow, if *join conditions* and *transition conditions* of links are not explicitly specified, they are implicitly assumed by the BPEL execution engine [AAA+07]. The implicit *join condition* of an activity is a disjunction of the status of all incoming links, i.e. if an activity has n links, it consists of $n - 1$ disjunctions. The implicit *transition condition* of an activity consists of a predicate, which yields true if the activity has completed successfully. We do not count Pick activities with the attribute `createInstance = "yes"`, as their branches are already counted.

5.4.2 Control-Flow-Complexity

Cardoso [Car, Car05, Car07] has proposed a control flow complexity workflow metric (CFC), which he has also adopted to BPEL. The CFC metric is defined recursively for every possible activity of a BPEL process (see Table 5.1).

The CFC definition does not mention the Scope activity. We propose to set the CFC value of a Scope to the CFC value of its child activity, since scopes do not affect the control flow.

Activity X	$CFC(X) =$
process	$DOP(a), \quad a \in X$
basic activities	1
sequence	$\sum_{a \in X} CFC(a)$
flow	$(n-l)! \cdot \sum_{a \in X} CFC(a)$ $n = \|X\|, l = \|links(X)\|$
if	$\|X\| \cdot \sum_{a \in X} CFC(a)$
loops (while, for, repeat)	$log_2(CFC(a)+2) \cdot CFC(a),$ $a \in X$
pick	$(2^n - 1) \cdot \sum_{a \in X} CFC(a)$

Table 5.1: Control Flow Complexity.
The BPEL Control Flow Complexity metric.

The CFC value of the Flow activity is defined as

$$CFC(F) = (n-l)! \cdot \sum_{a \in F} CFC(a),$$

where $n = |F|$, $l = |links(F)|$. Thus, it is implicitly assumed that $n > l$, which is not necessarily the case. Hence, we propose to use the term $CFC(F) = \sum_{a \in F} CFC(a)$, iff. $l >= n$.

Since the CFC metric differs from MCC in its treatment of sequences and basic activities and its consideration of flow parallelism, it makes sense to use both metrics together when measuring control flow in BPEL.

5.5 Measuring Parallelism

We have not found any hints on measuring workflow parallelism in the literature. In BPEL, tasks can be created via the structured activities Flow and ForEach. Both differ significantly from each other in their level of abstraction.

The Flow activity defines static parallel execution of tasks in a directed acyclic graph. The ForEach activity can be used to create a number of tasks based on a numeric XPath expression. Hence, it is possible that the actual number of tasks generated by a ForEach activity cannot be known at compile time. The maximum number of concurrently running basic activities thus may depend on knowledge which is not available at build time. In the terminology of [ST98], both constructs yield explicit parallelism. However, tasks are *implicitly* decomposed with ForEach and *explicitly* decomposed using Flow.

Thus, ForEach parallelism cannot be measured, but is defined on a higher level of abstraction than Flow parallelism. Hence, we propose a parallelism metric *DOP* (degree of parallelism), which gives an upper limit to the maximum number of concurrently executed activities which have been created by explicit decomposition.

For any Flow activity F, parallel child tasks form a clique in the non-directed graph induced by the relationship τ_1 is parallel to τ_2 (See Figure 5.1).

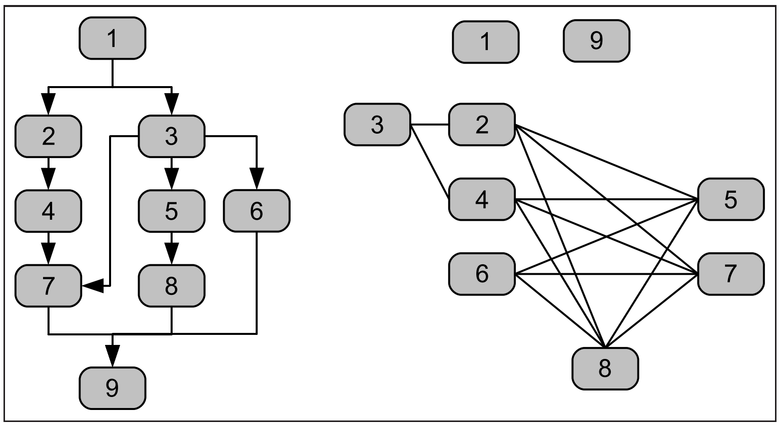


Figure 5.1: Flow graph and parallelism graph.

Hence, we construct a parallelism graph G_p and then set

$$\omega_F := max_{c \in C(G(F))} \omega(c)$$

$$DOP(F) := max_{(A \subseteq F) \wedge (|A| = \omega_F)} \sum_{a \in A} DOP(a)$$

where $C(G)$ yields the connected subgraphs of G, $\omega(c)$ is the size of the biggest clique, and $G(F)$ is the parallelism graph induced by F. The Flow graph shown in Figure 5.1 has a DOP value of 3, if all activities are basic activities.

Since Clique is NP-complete, it only makes sense to compute $\omega(c)$, where $G(F)$ is small. In our implementation we use the Bron-Kerbosch algorithm when $|F| < 40$ and an upper bound[1] for larger graphs [BK73].

As the child activities of a Sequence are processed in a sequential order, the DOP of a Sequence is the maximum DOP of its children. The same applies the treatment

[1] $\omega'(G) = \frac{3+\sqrt{9-8(n-m)}}{2}$., see [BBPP99].

Activity X	$DOP(X) =$
process	$DOP(a), \quad a \in X$
basic activities	1
flow	$max_{(A \subseteq X) \wedge (\lvert A \rvert = \omega_X)} \sum_{a \in A} DOP(a)$
sequence, if, pick	$max_{a \in X} DOP(a)$
loops, scope	$DOP(a), \quad a \in X$

Table 5.2: Degree of parallelism.

of conditionals (If) and external events (Pick), where one activity of a set of possible activities is selected.

The algorithm for the DOP metric is presented in Table 5.2. In addition to DOP, we propose counting the number of parallel ForEach activities.

The DOP metric is not to be confused with the *degree of concurrency* of a parallel algorithm [GGKK03], which is defined as the maximum number of tasks which can be executed simultaneously at any time. DOP differs from the degree of concurrency in that it measures only explicit parallelism and in its use of an upper bound in the computation of the maximum clique in the concurrency graph.

5.6 Measuring Data Flow Intensity

There has been little work on measuring data flow in BPEL, although data flow is a major aspect of workflows. We use a simple data flow intensity metric (DFI), inspired by Liggesmeyer [Lig95]:

$$DFI = \frac{|definitions| + |references|}{|decisions| + 1}$$

where $references$ is the set of variable references both in predicates and computations, while $definitions$ is the set of assignments and $decisions$ is the set of decision points. In contrast to Liggesmeyer, we set the divisor to $|decisions| + 1$ instead of $|decisions|$. Thus, the metric is always defined, even if a workflow does not contain any decisions.

Liggesmeyer [Lig95] originally defined this metric for C-like languages where loops (*while*, *do*) and conditionals (*if*, *switch*) form the set of decision points. In BPEL we can identify $|definitions|$ as the number of copy statements and message receiving activities and $|references|$ as the number of variable references in different elements, while $|decisions|$ is the number of appearances of the activities If, Pick, For, RepeatUntil and While. Alternatively, $|decisions|$ could be defined using the number of branches defined by If and Pick elements. This would not affect any of the mathematical properties of the metric, since any If or Pick with n branches can be simulated by n Ifs/Picks with adapted expressions/events. $|decisions|$ can be interpreted as an additional control flow metric. Thus, instead of $|decisions|$, we can also use MCC or CFC to define variants of DFI.

To gain insight into the structure of the data flow, we need to augment DFI with other measures. For instance, the ratio of assignments per invocation and the ratio of *copy* operations per Assign, can indicate the complexity of integrating the given Web Services. The ratio of data-manipulating activities (receive, invoke, assign) to NOA can be used as a rough measure to assess the impact of data flow compared to control flow.

5.7 Additional Structure Metrics

We can augment the set of metrics by adding several derived metrics and other statistical values. These values are not necessarily meaningful by themselves, but can help interpreting the given metrics. The metrics proposed in this section are inspired by [Lig95]. They should only be used in conjunction with complexity and size metrics:

- In addition to the NOA and NOAC metrics, it makes sense to count the appearance of any distinct element type within a workflow, so its impact on workflow complexity can be assessed.

- Similar to the decision density metric $\frac{MCC}{LOC}$, we propose using the ratio of decisions per activity/basic activity/ per invocation:

  $$\frac{MCC}{NOAC} \quad \frac{MCC}{NOA} \quad \frac{MCC}{|invoke(X)|}$$

  Here $invoke(X)$ denotes the set of invocation activities in the BPEL process X.

- The number of simple predicates per decision and the number of simple and compound predicates per decision can indicate how complex the decisions in a workflow are. In this case a decision would correspond to a branch condition of an If, to a loop condition or to a *join* or transition condition respectively.

- The ratio of arithmetic operators to the number of copy statements can yield the average complexity of calculations within the workflow:

  $$\frac{|arithmetic\ operators(X)|}{|copy(X)|}$$

  Here, $arithmetic\ operators(X)$ denotes the set of XPath arithmetic operators and $copy(X)$ is the set of *copy* elements of a BPEL process X).

5.8 Evaluation of the Metrics

5.8.1 Weyuker's Properties

For our purposes, the set of metrics that we propose both has to cover meaningful aspects of BPEL workflows and to provide sufficient granularity. Weyuker [Wey88] has presented a set of nine desirable properties of software metrics. The Weyuker properties assess the granularity of and the effects of program composition on software metrics. While we have not found any meaningful metric in the literature that satisfies all of the properties, if a set of metrics fulfills many or all of the properties, this can be interpreted as an indication of its usefulness. For a metric μ, with P, Q, R denoting programs, the Weyuker properties are defined as:

1. $\exists P \exists Q : \mu(P) \neq \mu(Q)$.
   This property enforces that not all programs are assigned the same metric value.

2. $\forall c > 0 : |\{P, \mu(P) = c\}| \neq \infty$
   Weyuker intended this property to prevent that a metric is too "coarse-grained," by dividing the set of possible programs into only "a few" classes.

3. $\exists P \exists Q : P \neq Q \wedge \mu(P) = \mu(Q)$
   This property is supposed to prevent that a metric is too "fine-grained," by assigning an individual value to each possible program.

4. $\exists P \exists Q : P \equiv Q \wedge \mu(P) \neq \mu(Q)$, where $\equiv$ means that P and Q produce the same output data for the same input.
   This property expresses that a metric is to measure the *structure* of programs, independently from their functionality.

5. $\forall P \forall Q : \mu(P) \leq \mu(P|Q) \wedge \mu(Q) \leq \mu(P|Q)$, where | means concatenation. The components of a program should not be more complex than the program itself.

6. $\exists P \exists Q \exists R : \mu(P) = \mu(Q) \wedge \mu(P|R) \neq \mu(Q|R)$
   $\exists P \exists Q \exists R : \mu(P) = \mu(Q) \wedge \mu(R|P) \neq \mu(R|Q)$
   When programs are concatenated, they can interact with each other. Property 6 mirrors the intuition, that if two programs P and Q interact with another program R in different ways, this should influence the metric value.

7. There exist programs P and Q such that Q is formed by a permutation of the statements of P, and $\mu(P) \neq \mu(Q)$.
   A metric should be responsive to the order of statements in a program.

8. If P is a renaming of Q, then $\mu(P) = \mu(Q)$.
   While the meaningfulness of variable or function names can affect readability of a program, it is impossible to measure the sensibleness of names.

9. $\exists P \exists Q : \mu(P) + \mu(Q) < \mu(P|Q)$
   If two programs are concatenated, they will probably interact. This should be reflected by a higher metric value.

5.8.2 Weyuker Properties of the DOP Metric

Lemma 1. *Weyuker property 1 holds for DOP.*

Proof. See Table 5.3 for an example, where $DFI(P) \neq DFI(Q)$.

`<sequence>`
`        <invoke />`
`        <invoke />`
`</sequence>`
	`<flow>`
	`        <invoke />`
	`        <invoke />`
	`</flow>`

DOP = 1	DOP = 2

Table 5.3: Proof of lemma 1. $\exists P \exists Q : \mu(P) \neq \mu(Q)$.

$\square$

Lemma 2. *Weyuker property 2 does not hold for DOP.*

Proof. Since sequential workflows of arbitrary length yield the same DOP value, we can form infinitely many workflows with the DOP value 1: $|\{w|DOP(w) = 1\}| = \infty$. See Table 5.4 for an example.

`<sequence>`	`<sequence>`
`  <invoke />`	`  <invoke />`
`  <invoke />`	`  <invoke />`
`</sequence>`	`  <invoke />`
	`  <invoke />`
	`</sequence>`
DOP = 1	DOP = 1

Table 5.4: Proof of lemma 2. Sequential workflows of arbitrary length yield the same DOP value.

□

Lemma 3. *Weyuker property 3 holds for DOP.*

Proof. The lemma is a corollary to Lemma 2, which has been proven by showing that an arbitrary number of workflows with the same degree of parallelism but differing sequential parts can be constructed (see Table 5.4).

□

Lemma 4. *Weyuker property 4 holds for DOP.*

Proof. Sequential and parallel invocations of stateless web services with the same arguments yield the same output. □

Lemma 5. *Weyuker property 5 holds for DOP.*

Proof. BPEL workflows can be concatenated using either sequences or flows. If P and Q are workflows, then

$$DOP(P|Q) = max\,\{DOP(P), DOP(Q)\} \geq DOP(P), DOP(Q),$$

holds for sequential concatenation, while for parallel concatenation

$$DOP(P|Q) = DOP(P) + DOP(Q) \geq DOP(P), DOP(Q).$$

□

Lemma 6. *Weyuker property 6 does not hold for DOP.*

Proof. If P, Q, and R are workflows and $DOP(P) = DOP(Q)$, then

$$DOP(P|R) = max\{DOP(P), DOP(R)\} = max\{DOP(Q), DOP(R)\} = DOP(Q|R)$$

for sequential concatenation and

$$DOP(P|R) = DOP(P) + DOP(R) = DOP(Q) + DOP(R) = DOP(Q|R)$$

for parallel concatenation. ☐

Lemma 7. *Weyuker property 7 holds for DOP.*

Proof. Exchanging activities between sequential sub-workflows can lead to different DOP values. For an example see Table 5.5.

`<flow>`	`<flow>`
`  <sequence>`	`  <sequence>`
`    <flow>`	`    <flow>`
`      <invoke />`	`      <invoke />`
`      <invoke />`	`      <invoke />`
`    </flow>`	`    </flow>`
`    <invoke />`	`    <flow>`
`  </sequence>`	`      <invoke />`
`  <sequence>`	`      <invoke />`
`    <flow>`	`    </flow>`
`      <invoke />`	`    <invoke />`
`      <invoke />`	`  </sequence>`
`    </flow>`	`  <sequence>`
`    <invoke />`	`    <invoke />`
`    <invoke />`	`    <invoke />`
`  </sequence>`	`  </sequence>`
`</flow>`	`</flow>`
DOP = 4	DOP = 3

Table 5.5: Proof of lemma 7. Permuting activites can lead to different DOP values.

☐

Lemma 8. *Weyuker property 8 holds for DOP.*

Proof. Element names do not affect workflow structure or execution. Thus, renaming the elements of a workflow does not have an influence on its explicit parallelism. □

Lemma 9. *For DOP, Weyuker property 9 does neither hold for sequential concatenation nor for parallel concatenation.*

Proof. $DOP(P|Q) = max\{DOP(P), DOP(Q)\} \leq DOP(P) + DOP(Q)$ for sequential concatenation. $DOP(P|Q) = DOP(P) + DOP(Q)$ for parallel concatenation.

□

5.8.3 Weyuker Properties of the DFI Metric

In the following, we will use the abbreviations $def(X)$ for $|definitions(X)|$, $ref(X)$ for $|references(X)|$ and $dec(X)$ for $|decisions(X)|$. Since the DFI metric only depends on counting appearances of definitions, references and appearances of conditionals, we do not need to distinguish between sequential and parallel concatenation of BPEL programs.

Lemma 10. *Weyuker property 1 holds for DFI.*

Proof. Sequential processes contain no decisions but an arbitrary number of definitions and references. Thus, for any $n \in \mathbb{N}, n \geq 2$, we can construct a sequential workflow X_n with $n = def(X_n) + ref(X_n)$ and $dec(X_n) = 0$, such that $DFI(X_n) = \frac{n}{1} = n$. Table 5.6 lines out the structure of corresponding workflows.

□

Lemma 11. *Weyuker property 2 does not hold for DFI.*

Proof. The number of decisions and the number of variable references can be increased at the same time by using XPath custom functions. The XPath term

$$h_1(x, h_2(x, (...h_k(x)))$$

contains k references to the variable x. For any BPEL workflow X, and any $n \in \mathbb{N}, n > 1$ we can construct a workflow X', with $X \neq X'$ and $DFI(X) = DFI(X')$. We add an XPath expression E with

$$ref(E) = (def(X) + ref(X)) \cdot (n - 1).$$

Accordingly $k = (dec(X) + 1) \cdot (n - 1)$ decisions will be added using conditionals with a condition of the form $f()$, where f is an XPath custom function. The

X_2	X_n

`<sequence>`
`<!-- define Message`
`     Variable -->`
`  <receive />`
`<!-- reference Message`
`     Variable -->`
`  <reply />`
`</sequence>`
	`<sequence>`
	`<!-- define Message`
	`     Variable -->`
	`  <receive />`
	`<!-- define or reference`
	`     Message Variable`
	`     (n - 2) times -->`
	`  <assign />`
	`  ...`
	`  <assign />`
	`<!-- reference Message`
	`     Variable -->`
	`  <reply />`
	`</sequence>`

$DFI(X_2) = 2$	$DFI(X_n) = n$

Table 5.6: Proof of lemma 10. For any $n \in \mathbb{N}, n \geq 2$, a sequential workflow X_n with $DFI(X_n)$ exists.

added statements could be of the form `<if><exit /></if>` or `<if><empty /></if>`. Then

$$\begin{aligned}
DFI(X') &= \frac{def(X) + ref(X) + ref(E)}{dec(X) + k + 1} \\
&= \frac{def(X) + ref(X) + (def(X) + ref(X)) \cdot (n-1)}{dec(X) + 1 + (dec(X) + 1) \cdot (n-1)} \\
&= \frac{(def(X) + ref(X)) \cdot n}{(dec(X) + 1) \cdot n} \\
&= \frac{def(X) + ref(X)}{dec(X) + 1} \\
&= DFI(X)
\end{aligned}$$

Hence, we can construct an infinite number of workflows with the same DFI value. □

Lemma 12. *Weyuker property 3 holds for DFI.*

Proof. The property holds, since the XPath expressions $\$x * \$y$ and $\$x + \$y$ contain the same number of references but a different operator. ☐

Lemma 13. *Weyuker property 4 holds for DFI.*

Proof. Property 4 holds since any number of unused[2] assignments can be made by assigning constant values. For an example see Table 5.7.

```
<sequence>
  <receive />
  <!--
  The following "assign"
  manipulates variable V:
  V = f(V)
  -->
  <assign />
  <!--
  The following "invoke"
  only reads V
  -->
  <invoke />
  <reply />
</sequence>
```

```
<sequence>
  <receive />
  <!--
  The following assign
  sets variable V to
  a constant value:
  -->
  <assign />
  <!--
  The following "assign"
  manipulates variable V
  -->
  <assign />
  <!--
  The following "invoke"
  reads V
  -->
  <invoke />
<reply />
</sequence>
```

$DFI = 5$	$DFI = 6$

Table 5.7: Proof of lemma 13. Two workflows with the same output and a different DFI value.

☐

Lemma 14. *Weyuker property 5 does not not hold for DFI.*

Proof. See Table 5.8 for an example, where $DFI(P) > DFI(P|Q)$.

[2]By the term “unused assignments,” we signify those assignments, which define the value of a variable at a point in the control flow which prevents any impact of the assignment on the further control or data flow of the program.

P	Q	$P\|Q$
`<sequence>`	`<sequence>`	`<sequence>`
`  <receive />`	`  <receive />`	`  <receive />`
`  <invoke />`	`  <if>`	`  <invoke />`
`  <invoke />`	`    <assign />`	`  <invoke />`
`  <reply />`	`  </if>`	`  <reply />`
`</sequence>`	`  <reply />`	`  <receive />`
	`</sequence>`	`  <if>`
		`    <assign />`
		`  </if>`
		`  <reply />`
		`</sequence>`
$DFI(P) = 6$	$DFI(Q) = 2$	$DFI(P\|Q) = 5$

Table 5.8: Proof of lemma 14. Two workflows P and Q and their concatenation $P|Q$.

□

Lemma 15. *Weyuker property 6 holds for DFI.*

Proof. Assume

$$\begin{aligned} P &: \ def(P) + ref(P) = 16, dec(P) = 3, DFI(P) = 4 \\ Q &: \ def(Q) + ref(Q) = 8, dec(Q) = 1, DFI(Q) = 4 \\ R &: \ def(R) + ref(R) = 2, dec(R) = 2. \end{aligned}$$

Then $DFI(P) = DFI(Q)$, but $DFI(P|R) = \frac{18}{6} = 3$ and $DFI(Q|R) = \frac{10}{4} = 2\frac{1}{2}$.

□

Lemma 16. *Weyuker property 7 does not hold for DFI, while Weyuker property 8 holds.*

Proof. Since DFI only depends on counting the appearance of elements, neither permuting nor renaming operands affects the metric value. □

Lemma 17. *Weyuker property 9 does not hold for DFI.*

Proof. The concatenation of two BPEL programs P and Q yields the term

$$DFI(P|Q) = \frac{def(P) + def(Q) + ref(P) + ref(Q)}{dec(P) + dec(Q) + 1}.$$

Assume $DFI(P) + DFI(Q) < DFI(P|Q)$. Then

$$DFI(P) + DFI(Q) < DFI(P|Q)$$
$$\Leftrightarrow$$
$$\frac{def(P) + ref(P)}{dec(P) + 1} + \frac{def(Q) + ref(Q)}{dec(Q) + 1} < \frac{def(P) + ref(P) + def(Q) + ref(Q)}{dec(P) + dec(Q) + 1}$$

Since $dec(P) \geq 0 \wedge dec(Q) \geq 0$, we may multiply with $(dec(P) + 1) \cdot (dec(Q) + 1) \cdot (dec(P) + dec(Q) + 1)$:

$$\Leftrightarrow$$
$$(def(P) + ref(P)) \cdot (dec(Q) + 1) \cdot (dec(P) + dec(Q) + 1)$$
$$+ (def(Q) + ref(Q)) \cdot (dec(P) + 1) \cdot (dec(P) + dec(Q) + 1)$$
$$< (def(P) + ref(P) + def(Q) + ref(Q)) \cdot (dec(P) + 1) \cdot (dec(Q) + 1)$$
$$\Leftrightarrow$$
$$(def(P) \cdot dec(Q) + ref(P) \cdot dec(Q) + def(P) + ref(P)) \cdot (dec(P) + dec(Q) + 1)$$
$$+ (dec(P) \cdot def(Q) + dec(P) \cdot ref(Q) + def(Q) + ref(Q)) \cdot (dec(P) + dec(Q) + 1)$$
$$< (def(P) + ref(P) + def(Q) + ref(Q)) \cdot (dec(P) \cdot dec(Q) + dec(P) + dec(Q) + 1)$$

$$\Leftrightarrow$$
$$def(P) \cdot dec(P) \cdot dec(Q) + ref(P) \cdot dec(P) \cdot dec(Q) + def(P) \cdot dec(P)$$
$$+ ref(P) \cdot dec(P) + def(P) \cdot dec(Q)^2 + ref(P) \cdot dec(Q)^2$$
$$+ 2 \cdot def(P) \cdot dec(Q) + 2 \cdot ref(P) \cdot dec(Q) + def(P) + ref(P)$$
$$+ dec(P)^2 \cdot def(Q) + dec(P)^2 \cdot ref(Q) + 2 \cdot dec(P) \cdot def(Q) + 2 \cdot dec(P) \cdot ref(Q)$$
$$+ dec(P) \cdot def(Q) \cdot dec(Q) + dec(P) \cdot ref(Q) \cdot dec(Q) + def(Q) \cdot dec(Q)$$
$$+ ref(Q) \cdot dec(Q) + def(Q) + ref(Q)$$
$$< def(P) \cdot dec(P) \cdot dec(Q) + ref(P) \cdot dec(P) \cdot dec(Q) + dec(P) \cdot def(Q) \cdot dec(Q)$$
$$+ dec(P) \cdot ref(Q) \cdot dec(Q) + def(P) \cdot dec(P) + ref(P) \cdot dec(P) + dec(P) \cdot def(Q)$$

$$+ dec(P) \cdot ref(Q) + def(P) \cdot dec(Q) + ref(P) \cdot dec(Q) + def(Q) \cdot dec(Q)$$
$$+ ref(Q) \cdot dec(Q) + def(P) + ref(P) + def(Q) + ref(Q)$$
$$\Leftrightarrow$$
$$def(P) \cdot dec(Q)^2 + ref(P) \cdot dec(Q)^2 + def(P) \cdot dec(Q) + ref(P) \cdot dec(Q)$$
$$+ dec(P)^2 \cdot def(Q) + dec(P)^2 \cdot ref(Q) + dec(P) \cdot def(Q) + dec(P) \cdot ref(Q)$$
$$< 0$$

$\lightning$, since $def(X) \geq 0, ref(X) \geq 0, dec(X) \geq 0, \quad \forall X.$

$\square$

5.8.4 Summary

Table 5.9 gives an overview of the adherence of the presented metrics to the Weyuker properties. It shows that all Weyuker properties are satisfied by at least one of the metrics. The same arguments Weyuker has used for investigating the LOC metric, hold for the size metrics NOA, NOAC and NOACC [Wey88]. Weyuker [Wey88] also has investigated the adherence of MCC to the Weyuker properties , while Cardoso [Car] has investigated the adherence of CFC to the Weyuker properties. The adherence of DOP and DFI to the Weyuker properties have been discussed in Chapters 5.8.2 and 5.8.3. MCC and DFI are defined non-recursively. Both can be computed for any sub-workflow, though, since they only depend on counts.

Metric	Category	recursive	1	2	3	4	5	6	7	8	9
NOA	size	yes	+	+	+	+	+	-	-	+	-
NOAC	size	yes	+	+	+	+	+	-	-	+	-
NOACC	size	yes	+	+	+	+	+	-	-	+	-
MCC	control flow	no	+	-	+	+	+	-	-	+	-
CFC	control flow	yes	+	-	+	+	+	-	+	+	+
DFI	data flow	no	+	-	+	+	-	+	-	+	-
DOP	parallelism	yes	+	-	+	+	+	-	+	+	-

Table 5.9: The set of workflow metrics and the Weyuker properties.

The other metrics presented in this paper are either derivations from the metrics listed in Table 5.9 or are only meant to be used in conjunction with them. Thus we do not evaluate their adherence to the Weyuker properties.

5.9 Applying the Metrics: BPEL Code Smells

The set of metrics proposed here enables investigating BPEL documents for code smells that indicate a need of refactoring, which has been required in Chapter 3.2.3. In the following, X and Y denote sub-workflows.

5.9.1 Code Smell 1: Similar Functionality, Distinct Complexity

If the sub-workflows X and Y are expected to perform a similar functionality, the code smell

$$\begin{aligned} & MCC(X) >> MCC(Y) \\ \vee\, & NOACC(X) >> NOACC(Y) \\ \vee\, & CFC(X) >> CFC(Y) \\ \vee\, & DFI(X) >> DFI(Y) \\ \vee\, & |invoke(X)| >> |invoke(Y)| \end{aligned} \tag{5.1}$$

indicates that either one implementation is incorrect or X is overly complex. This can happen, if collaborators either differ in their idea about the functionality of the workflow or in their level of experience.

5.9.2 Code Smell 2: Flow Links Sequence

In BPEL, sequences of activities can be modeled either with the structured activity *sequence* or via links in a *flow*. The second case has the disadvantage of bloating the resulting BPEL code, since a link element and source and target elements have to be generated for every involved activity. The following smell indicates, that sub-workflow X is linear, but contains at least one Flow element with *join* and *target* conditions and probably should be refactored to a Sequence:

$$\begin{aligned} & CFC(X) = NOA(X) \\ \wedge\, & MCC(X) > 1 \end{aligned} \tag{5.2}$$

5.9.3 Code Smell 3: Too Many Assigns

When collaborators independently add assignments to a workflow, this can lead to redundancy. Every *assign* activity can hold an arbitrary number of *copy* elements.

Thus, often different assigns can be merged. The following smell indicates that sub-workflow X may contain Assign activities that can be merged:

$$\frac{|assign(X)|}{|invoke(X)|} > 1 \wedge \frac{|copy(X)|}{|assign(X)|} \approx 1 \tag{5.3}$$

For an example, see Table 5.10.

`<sequence>`	`<sequence>`
`  <assign>`	`  <assign>`
`    <copy />`	`    <copy />`
`  </assign>`	`    <copy />`
`  <assign>`	`    <copy />`
`    <copy />`	`    <copy />`
`  </assign>`	`  </assign>`
`  <assign>`	`  <invoke />`
`    <copy />`	`</sequence>`
`  </assign>`	
`  <assign>`	
`    <copy />`	
`  </assign>`	
`  <invoke />`	
`</sequence>`	
$\frac{\lvert assign(X)\rvert}{\lvert invoke(X)\rvert} = 4$ $\frac{\lvert copy(X)\rvert}{\lvert assign(X)\rvert} = 1$	$\frac{\lvert assign(X)\rvert}{\lvert invoke(X)\rvert} = 1$ $\frac{\lvert copy(X)\rvert}{\lvert assign(X)\rvert} = 4$

Table 5.10: *Too Many Assigns*. The workflow on the left contains more Assign activities than needed.

5.9.4 Code Smell 4: Unnecessary Nesting

Unnecessary Nesting signifies the multiple nesting of structured activities without any influence on control flow. For example, nesting several Sequence activities is legal, but only reduces code readability without having any effect on the control flow. Here, sub-workflow X probably contains unnecessary nesting:

$$\begin{aligned} &NOAC(X) > NOA(X) + 1 \\ \wedge\ &CFC(X) \approx NOA(X) \end{aligned} \tag{5.4}$$

Unnecessary nesting can appear if designers work at different levels of a model tree and activities on the upper layer are removed. Another source of unnecessary nesting are inexperienced designers, who attempt to use the structured activity "of their choice." An example is shown in Table 5.11.

`<sequence>`	`<sequence>`
`  <flow>`	`  <invoke />`
`    <sequence>`	`  <invoke />`
`      <invoke />`	`</sequence>`
`      <invoke />`	
`    </sequence>`	
`  </flow>`	
`</sequence>`	
$NOAC(X) = 5 \wedge NOA(X) = 2$ $\wedge CFC(X) = 1$	$NOAC(X) = 3 \wedge NOA(X) = 2$ $\wedge CFC(X) = 1$

Table 5.11: An example of unnecessary nesting.

5.9.5 Code Smell 5: Unexploited Potential Parallelism

Scientific workflows usually exploit parallelism for efficiency. If X contains compute-intensive tasks, the following smell should lead to questioning a domain-expert about potentially parallel execution of affected invocations:

$$\begin{aligned} & DOP(X) = 1 \\ \wedge\ & |parallel\ forEach(X)| = 0 \\ \wedge\ & |invoke(X)| > 1 \end{aligned} \tag{5.5}$$

5.9.6 Code Smell 6: Similar Functionality, Distinct Parallelism

It can be expected, that in general similar functionality of sub-workflows implies a similar degree of potential parallelism. If two sub-workflows X and Y are known to perform a similar functionality,

$$DOP(X) >> DOP(Y) \tag{5.6}$$

is another code smell for unused parallelism.

5.9.7 Code Smell 7: Too Much Computation

The sole purpose of a workflow language is to coordinate the activities of other services and particularly not to perform complex computations. Because BPEL is not intended as a general programming language, a high density of arithmetic operators or a very high data flow indicates a need of new custom functions or web services:

$$\frac{|arithmetic\ operators(X)|}{|copy(X)|} > c, \quad c > 1\ const. \tag{5.7}$$

The actual value of c will depend upon experience and on the characteristic details of the workflow project. Similarly, if the size or complexity of a process are large compared to the number of invocations, the developers should consider whether an implementation of the service in a general purpose programming would be preferable.

5.10 An Example of Workflow Design and Analysis

5.10.1 Purpose

Our example is a pipe process, which converts gene sequence identifiers of the plant Arabidopsis thaliana to gene sequences. Arabidopsis thaliana is widely used as a model organism for plant science. We have chosen this particular example, since it is a small and simple workflow, yet it represents a common case in bioinformatics. We have developed this example with the aid of the BioMoby Web Service database and an early version of CALVIN [WL02].

The example workflow consists of the sequential execution of three tasks and takes a list of AGI (Arabidopsis Genome Initiative) Locus Identifiers as input. The first task invokes a service, which converts the AGI Locus identifiers to NASC (Nottingham Arabidopsis Stock Centre) code identifiers. The second task invokes another service, that converts the NASC identifiers to EMBL (European Molecular Biology Laboratory) accession numbers, while the third task uses yet another service to yield actual DNA, RNA or amino acid sequences from the EMBL accession numbers. The corresponding BPEL code for the presented workflows is listed in Appendix A.

5.10.2 Original Design

The designers have attempted to implement the process in a semi-modular fashion, where every service invocation is encapsulated within a Sequence activity, thus representing every one of the tasks as a sub-workflow. Each of these sequences consists of two phases. In the following, these sequences are dubbed *sub-workflow* 1, 2, and 3. First assignments are made to configure the invocation, second the actual invocation is actuated. The sequences are embedded in a Flow activity and connected by links, because originally it had not been obvious, whether the workflow would contain concurrent invocations. In the first phase, three designers have worked in parallel to implement the workflow model, with each designer being responsible for one sub-workflow. However, one of the designers is inexperienced in workflow design and has committed several design flaws when implementing sub-workflow 1. The workflow is visualized in Figure 5.2.

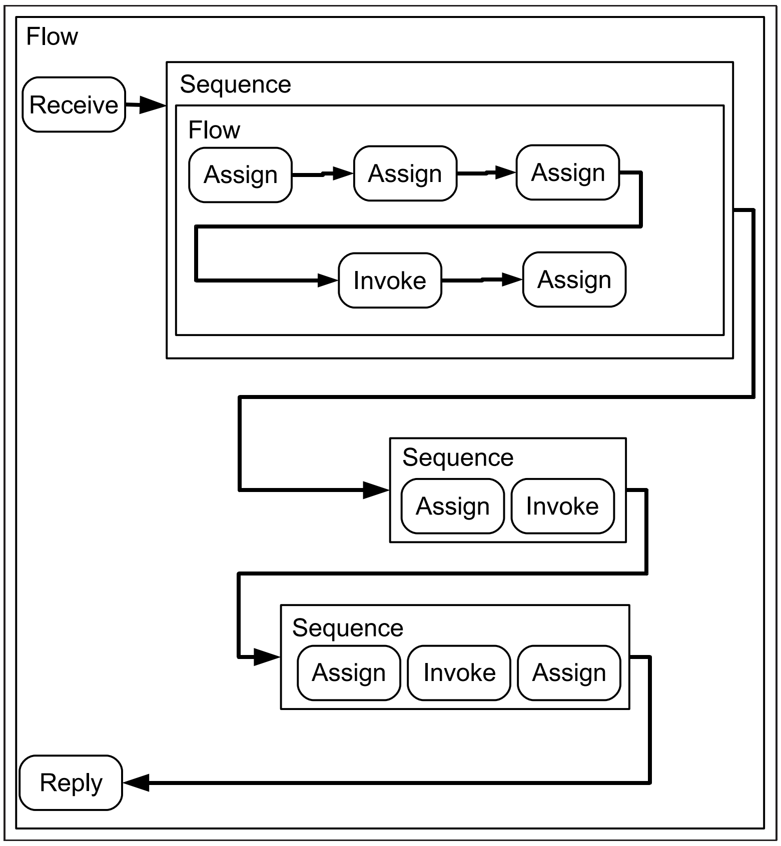


Figure 5.2: A pipeline workflow.

5.10.3 Analysis and Redesign

While the rationale behind the workflow at hand is a relatively clear cut design, a look at some of the metrics reveals that it can be improved (see Table 5.12).

Workflow Metric	original design	X_1	X_2	X_3	refined design	final design
$NOA(X)$	12	5	2	3	9	9
$NOAC(X)$	16	6	2	3	13	10
$NOACC(X)$	29				25	22
$\|StructuredActivities(X)\|$	4				4	1
$\|decisions(X)\|$	0	0	0	0	0	0
$\frac{\|assign(X)\|}{\|invoke(X)\|}$	4				1.33	1.33
$\frac{\|copy(X)\|}{\|assign(X)\|}$	1				3	3
$MCC(X)$	17	8	1	1	6	1
$CFC(X)$	12	5	2	3	9	9
$DOP(X)$	1	1	1	1	1	1
$DFI(X)$	36	10	10	12	34	34
$DFI_{MCC}(X)$	2.1	1.25	10	12	5.66	34

Table 5.12: Workflow metrics of the example. The terms X_1, X_2, and X_3 denote the sequences in the first version of the workflow.

Since the number of basic activities is equal to its control-flow-complexity and its degree of parallelism is zero, the workflow is a pipe.

Moreover, the ratio of copy-elements per assign is one, while there are more than one assign-activities per invocation, as described in Chapter 5.9.3 (code smell 3, “Too Many Assigns”). The metrics also imply unnecessary nesting, as described in Chapter 5.9.4 (code smell 4, “Unnecessary Nesting”). Since the workflow does not include any conditionals, some of the assign activities can be merged.

While it is known that all three sub-workflows are responsible for a similar purpose, i.e. the invocation of a service, there is an imbalance between the metrics of the sub-workflows. As stated in Chapter 5.9.1 (code smell 1, “Similar Functionality, Distinct Complexity”), this indicates that the first sub-workflow is overly complex. Closer inspection also reveals that it contains a redundant Assign activity. Refactoring leads to the new pipe shown in Figure 5.3.

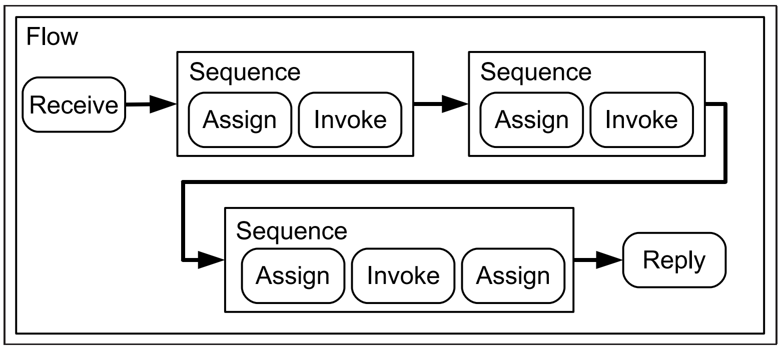


Figure 5.3: The refined pipeline workflow.

5.10.4 Further Analysis and Redesign

Still, the MCC value is too high for a simple pipe, as it is the result of join and target conditions as described in Chapter 5.9.2 (code smell 2, "Flow Links Sequence"). Hence, the workflow can still be improved by refactoring all structured activities to a single sequence element. Size and complexity of the new workflow have diminished substantially (see Figure 5.4).

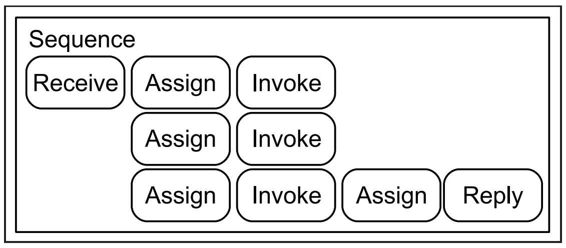


Figure 5.4: The final pipeline workflow

Compiling the initial workflow yields about 180 LOC, while the refactored Workflow contains about 100 LOC.[3] Since the workflow is comparatively small (containing only three invocations), we can see that larger workflows absolutely require metrics support for effective steering of collaborative design.

[3]N.B.: By the way, the code is formatted in the Appendix, these values correspond to 350 and 270 lines of printed code respectively.

Chapter 6

The HOBBES System for Collaborative Workflow Design

This Chapter shows how the principles of collaborative workflow design presented in the previous Chapters 3, 4, and 5 are implemented in the workflow design system HOBBES. Section 6.1 gives an overview of the general BPEL editing features of HOBBES, while Section 6.2 presents its collaborative features. Section 6.3 explains the architecture and implementation of HOBBES. We discuss extensions to the HOBBES system in the following Chapter 7.

6.1 BPEL-Editing

HOBBES is a Web-based workflow editing system supporting collaborative workflow development processes. It can be used either as an ordinary BPEL designer for a single user or in a team mode, which allows synchronous and asynchronous work of several users on the same workflow. We will briefly describe the general features of HOBBES first and then elaborate on the collaborative features. The activities of the workflow are edited and arranged in a graphical view (see Figure 6.1). User dialogs can be opened to edit the references of the BPEL document to WSDL files, its Partner Links to Web Services, and other properties, like global variables, fault and event handlers.

6.1.1 Activities

Figure 6.1 shows the main screen of HOBBES. It consists of an accordion panel on the left (see Figure 6.1, ❶), which provides diverse editing options, a canvas ❷ showing the children of a structured activity, and a navigation tree ❸ on the right. The editing canvas may differ according to the kind of structured activity currently shown. For instance, while Figure 6.1 shows the children of a Flow activity, Figure 6.2 shows the

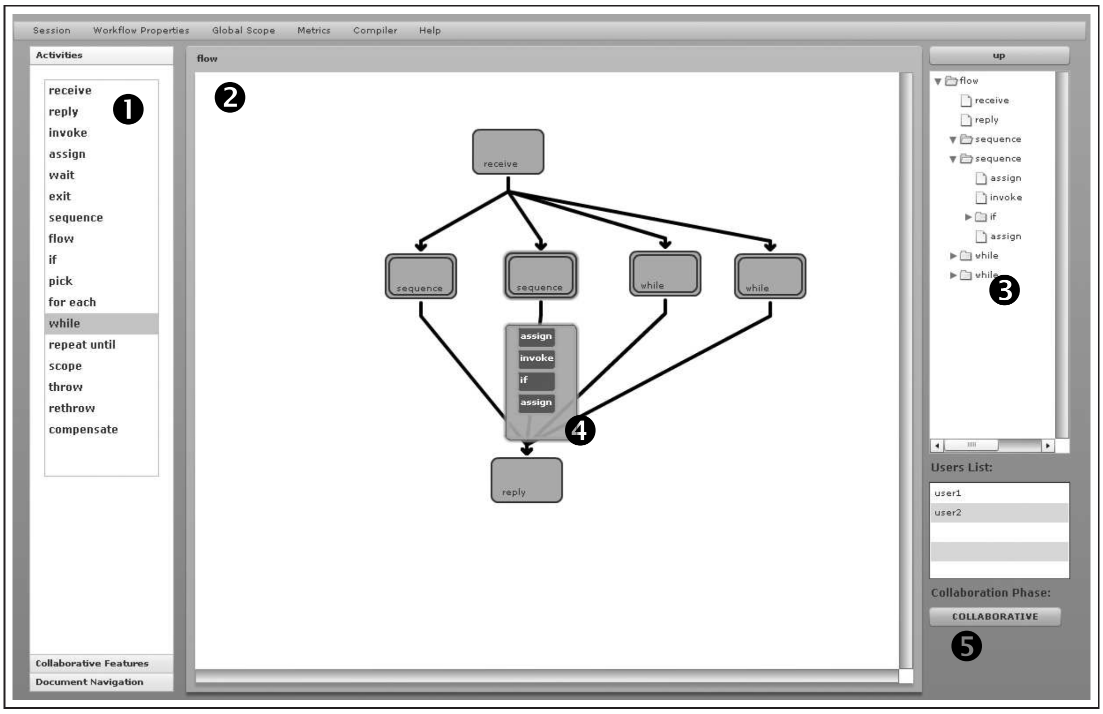


Figure 6.1: The HOBBES user interface: The screenshot shows the children of the top-level Flow activity. The contents of structured activities are hidden. Holding the mouse button over a structured activity reveals a preview of its contents.

editing canvas for a Sequence activity, where element positions can be switched by the user. HOBBES supports basic and structured activities, links, handlers, message exchanges and scopes but does not support correlation sets.

Similar to other workflow notations, basic activities are represented as rounded rectangles with a single line border, while structured activities have a double line border, similar to sub-processes in the Event-Driven Process Chain notation. Moving the mouse cursor on certain hotspots of an activity representation reveals buttons, which enable actions like configuring parameters of an activity, connecting it to to following activities or removing it from the workflow. This concept allows visualizing activity representations without using pop-up-menus. Activities within a Flow can be connected by Links forming a DAG. The Link between two activities is automatically added to the closest Flow surrounding both of them.

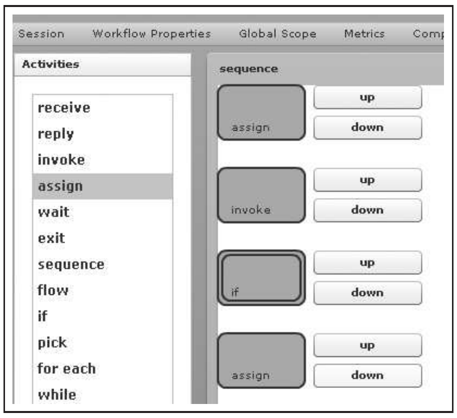
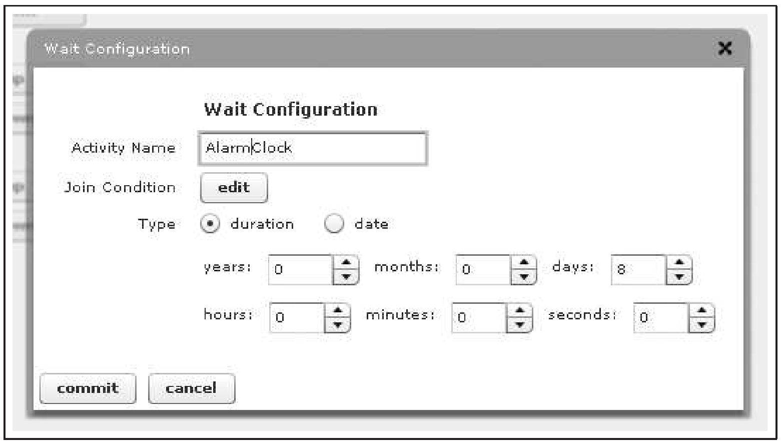


Figure 6.2: Editing the contents of a Sequence activity. A similar editing mode is also used for If and Pick activities.



Activity Name	AlarmClock
Join Condition	edit
Type	[✓ duration] date
years: 0	months: 0	days: 8
hours: 0	minutes: 0	seconds: 0

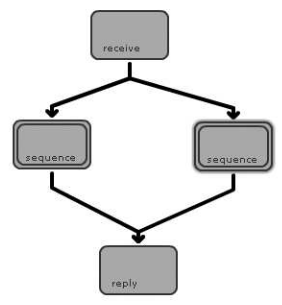
Figure 6.3: Editing the properties of a Wait activity.

Conditional and arithmetic XPath-expressions are edited visually. The expressions are represented as trees, where nodes correspond to operators, functions, and constant values. Other non-visual properties are edited in forms, e.g. Figure 6.3 shows the form for a Wait activity, where a date or a duration can be entered.

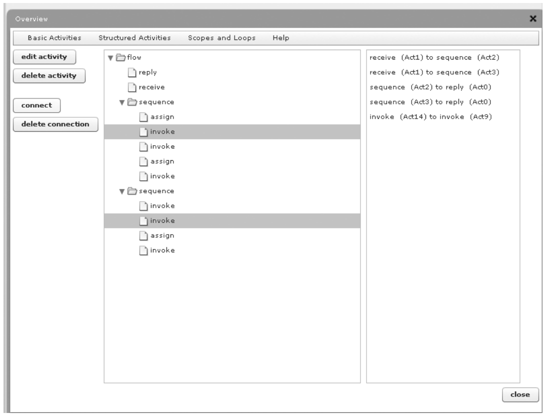
6.1.2 Document Navigation

HOBBES directly reflects the tree structure of the BPEL document by showing only the immediate child elements of a structured activity and their connections. Clicking on a flow or sequence activity reveals a preview of its child elements (e.g. see Figure 6.1, ❹). Navigation through the document is simplified by a navigation tree widget, which is shown right of the editor view. The structured activity currently being edited is highlighted. HOBBES's document navigation adheres to relaxed WYSIWIS ("What you see is what I see," see Chapter 3.2.3), so team members can have different views on the document.

For an overview of the workflow model, users can open a larger tree view (see Figure 6.4, b), which can be used for some editing operations. This tree view also enables editing Flow links between children of different structured activities.



(a) Two Sequence structured activities.



```
▼ flow
    reply
    receive
  ▼ sequence
      assign
      invoke
      invoke
      assign
      invoke
  ▼ sequence
      invoke
      invoke
      assign
      invoke
```

```
receive (Act1) to sequence (Act2)
receive (Act1) to sequence (Act3)
sequence (Act2) to reply (Act0)
sequence (Act3) to reply (Act0)
invoke (Act14) to invoke (Act9)
```

(b) Adding a link between children of the two Sequence activities.

Figure 6.4: Connecting children of different structured activities.

6.2 Collaborative Features

6.2.1 Workflow Sessions and Phase Management

At the beginning of a team session, the team leader has sole access to the workflow until she decides to share the session, after which other users may join. Clients are notified of users joining and leaving the session and display a list of present users. A chat feature enables communication without depending on external programs.

The current collaboration mode is shown as a screen icon (see Figure 6.1, ❺), which the team leader may click to change to a different collaboration mode. The mode can be *non-shared*, *collaborative*, and *private*. Above the collaboration mode icon, HOBBES shows a list of the team members. The team leader may at any time switch to a different mode, which is propagated to all clients. When the system changes from the collaborative design mode to the analysis and annotation mode, the team leader gains private access to workflow metrics as well as the ability to annotate workflow elements, as described later. During the analysis mode, other team members can view the changes the team leader makes on the workflow. They are prevented from modifying the workflow themselves, though. The team leader may compile and persist the current state of the workflow model at any time.

6.2.2 Collaborative Editing

We need to support synchronous collaboration and also provide means for temporarily locking workflow parts (see Chapter 3.2.3). Team members can concurrently add, remove, drag, and connect activities. Such changes, which affect visual elements of the workflow model, are instantly propagated to all clients.

If a user actuates a modification of the workflow a request is sent to the server. The server then decides upon the validity of the operation and includes the necessary instructions in its response message and informs the other clients via a broadcast. The validity of an operation can be compromised if users concurrently actuate operations (see Chapter 4).

For example, if user *A* actuates the creation of an Invoke activity as a child of a Sequence activity, the server will only allow the modification if the Sequence still exists within the model. If this is not the case because user *B* has deleted it, *A*'s client will receive notification about the delete operation. It will then automatically navigate to the root of the workflow and perform the delete operation.

However, users may lock parts of the workflow, thus preventing others from modifying it. If user *A* wants to lock a structured activity, she will click its "lock" button.

The lock operation is rejected if another user still holds a lock on the activity or any of its descendants. Otherwise, the team leader is asked for allowing *A* to lock the activity and its children. If the team leader currently is logged out, the lock operation succeeds anyway. If a user has sole access to a specific part of the workflow, he cannot be disturbed by other user's actions. This can be of great importance for implementing details. Because all changes to the sub-workflow are still propagated to other clients, all users stay aware of the progress of the design process. Locks can be released by the lock owner herself or by the team leader.

6.2.3 Collaborative Activity Configuration

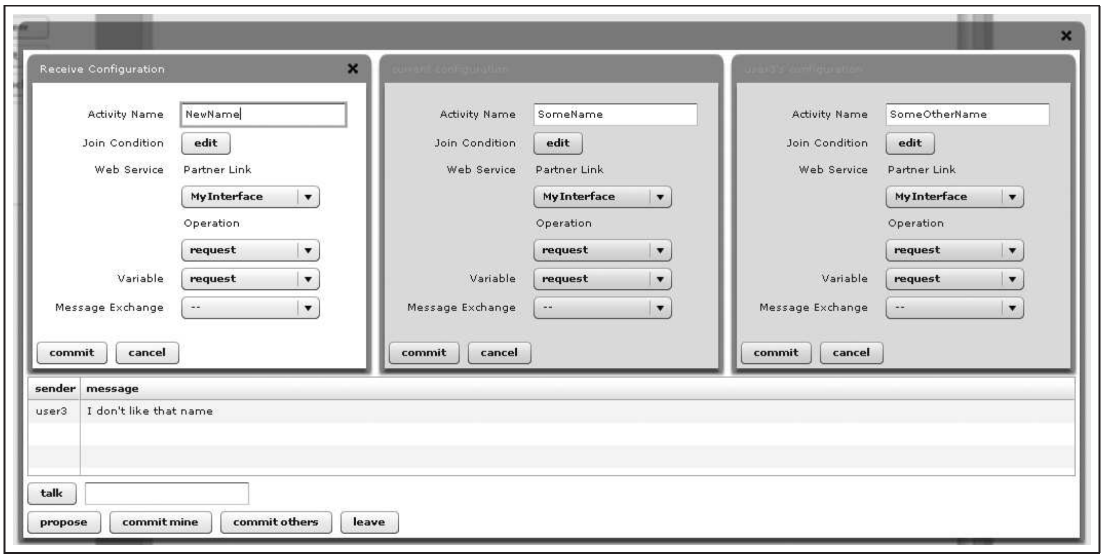
As discussed in Chapter 3.2.3, sometimes a user will need another user's advice, when configuring an activity. For this case, HOBBES includes a special configuration "co-edit" window (see Figure 6.5). User *A* may select an activity and then choose the "co-edit" feature. A window dialog will pop up for inviting another user *B* to collaboratively edit the activity. *B* is informed of the invitation and may accept or reject it. In case *B* accepts, a window will appear on both users' screens, which shows three activity configuration forms, two of which are disabled. In one form the user can input a configuration and send it to his partner by pushing the "propose" button. The two disabled forms contain the opposite user's last proposal and the current configuration of the activity. Both users can exchange and discuss each other's proposals until *A* either commits one of the two proposals or cancels the operation.

6.2.4 Collaborative Document Navigation

With relaxed WYSIWIS, one user's actions do not directly influence any other user's view on the model. In a face-to-face collaboration, however, the first step in communicating about a specific object is to show it to a partner. We have thus included a feature which enables such a communication, as discussed in Chapter 3.2.3. If user *A* clicks on the "co-navigate" button, she can invite another user *B* to navigate to her view of the model. *B* is notified by a pop-up window and may accept or reject. If *B* accepts, his client will navigate to the same structured activity.

6.2.5 Workflow Metrics

We consider workflow metrics as a collaborative feature of HOBBES, since it is used in the collaborative workflow development process described in Chapter 3.2.


Receive Configuration

	current configuration	user3's configuration
Activity Name	NewName	SomeName	SomeOtherName

	Receive Configuration	current configuration	user3's configuration
Activity Name	NewName	SomeName	SomeOtherName
Join Condition	edit	edit	edit
Web Service – Partner Link	MyInterface	MyInterface	MyInterface
Operation	request	request	request
Variable	request	request	request
Message Exchange	--	--	--

sender	message
user3	I don't like that name

Figure 6.5: Collaboratively editing the properties of an activity.

In the analysis phase of the workflow design process, the team leader may view the current metrics of the workflow model (see Figure 6.1, ❺). The team leader can also access a graph view that presents the development of the global metrics of a workflow over time (see Figure 6.6).

The graphs of individual metrics are drawn with different colors. Additionally, each metric may be viewed separately. The metrics window also includes a tree view for those metrics which are defined recursively.

6.2.6 Activity Annotations

During the analysis and annotation phase, the team leader may annotate activities in the workflow to delegate tasks to team members (see Chapter 3.2.3). Annotations automatically are also applied to the sub-tree induced by structured activity.

The team leader may annotate an activity α in the following ways:

1. α is delegated to a set of team members, and locked for others.

2. α is delegated to a set of team members, but not locked for others.

3. α is locked for everyone.

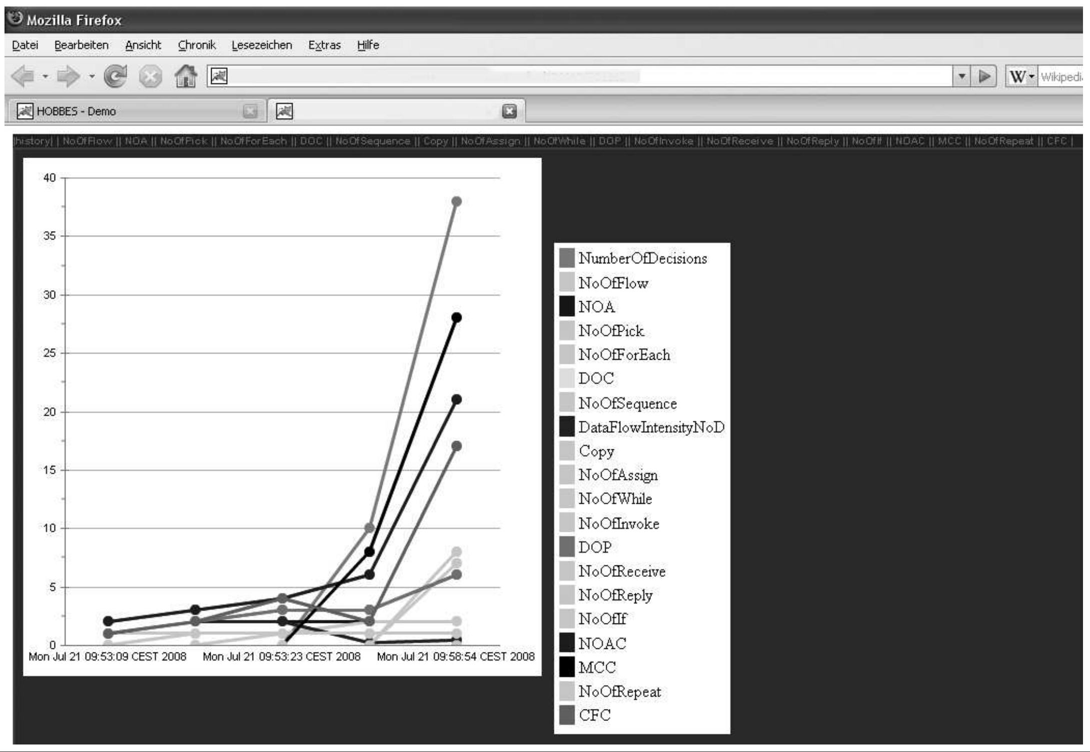
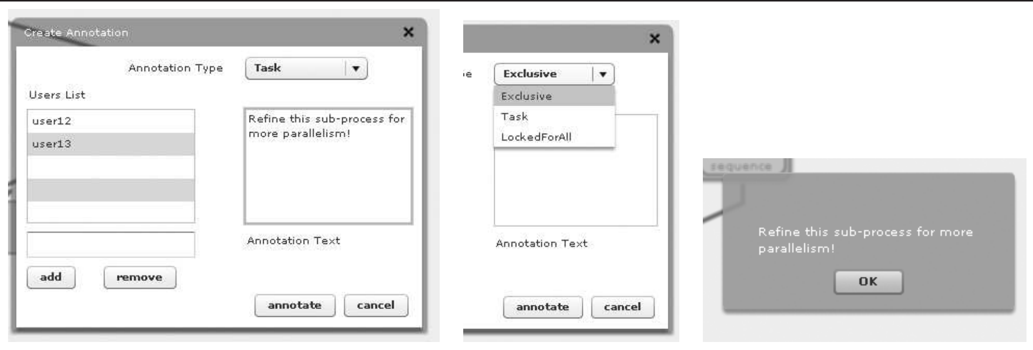

Mozilla Firefox
Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe
HOBBES - Demo
history| || NoOfFlow || NOA || NoOfPick || NoOfForEach || DOC || NoOfSequence || Copy || NoOfAssign || NoOfWhile || DOP || NoOfInvoke || NoOfReceive || NoOfReply || NoOfIf || NOAC || MCC || NoOfRepeat || CFC |
40
35
30
25
20
15
10
5
0
Mon Jul 21 09:53:09 CEST 2008
Mon Jul 21 09:53:23 CEST 2008
Mon Jul 21 09:58:54 CEST 2008
NumberOfDecisions
NoOfFlow
NOA
NoOfPick
NoOfForEach
DOC
NoOfSequence
DataFlowIntensityNoD
Copy
NoOfAssign
NoOfWhile
NoOfInvoke
DOP
NoOfReceive
NoOfReply
NoOfIf
NOAC
MCC
NoOfRepeat
CFC

Figure 6.6: Metrics History Graph.


Create Annotation

Annotation Type: Task

Users List: user12, user13

Refine this sub-process for more parallelism!

Refine this sub-process for more parallelism!

(a) The team leader can create annotations. (b) The available annotation types. (c) Viewing a task description.

Figure 6.7: Annotations.

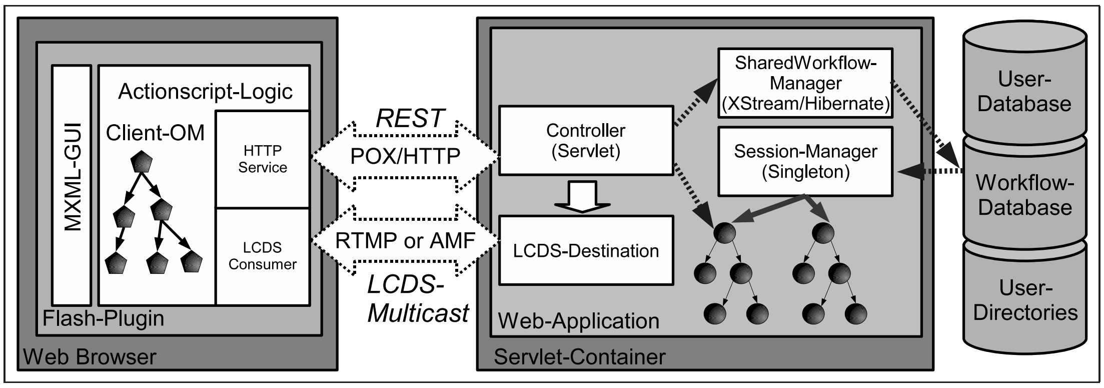


Figure 6.8: The architecture of HOBBES.

Annotation 1 can be used to both grant privileged access to a workflow part to an expert and to instruct her about her responsibilities in the next collaborative phase. Annotation 2 delegates responsibility to a team member but does not prevent collaboration, while annotation 3 can especially be useful to prevent those parts of a workflow model from further modification, which are considered complete. According to their meaning to a specific user, the annotated activities are marked with colors. Additionally the annotations may include a text which describes the expected task.

Figure 6.7(a) shows the annotation editor window used by the team leader, while Figure 6.7(b) shows the available annotation types. Figure 6.7(c) depicts the window, which presents the text description of an annotation to a team member.

6.3 Architecture and Implementation

6.3.1 The HOBBES Architecture

Figure 6.8 shows an overview of the architecture of HOBBES. Basically, it implements the Model-View-Controller Pattern [GHJV95] using the notification facilities of the Adobe Flex framework. Concurrent input is processed on the server, which decides upon the validity of operations and subsequently notifies the clients of changes of the BPEL model. Clients post HTTP requests to the server, whose responses are handled via callback methods. The full functionality of a workflow design session is exposed via a RESTful interface using POX/HTTP messages.[1] Hence alternative non-collaborative clients could be developed as well, e.g. for mashups.

[1]The acronym *POX* stands for *Plain Old XML*.

On Rich Internet Applications and Adobe Flex Rich Internet Applications provide both web access and desktop like usability. They often are identified with *AJAX* technology [Gar05], though alternatives exist. AJAX stands for "Asynchronous Javascript And XML." It denotes the combination of Javascript and HTML Document Object Models with an *XMLHttpRequest*-API which enables asynchronous server queries. The Web architecture does not provide any means or concepts for server-to-client notification. If such capabilities are needed, they either have to be simulated by HTTP-Polling or implemented using different protocols, which may lead to problems with firewalls.

The Adobe Flex framework, which is based on the Flash plugin, is a stable and complete RIA solution [fle]. It supports declarative GUI design in an XML-based language (MXML) with direct mapping to the object-oriented scripting language Actionscript. Actionscript strongly resembles Javascript, but includes (optional) build-time type checking and a class system. Flex supports HTTP requests and its communication facilities can be enhanced with optional server side components called *Lifecycle Data Services* (LCDS), which among other things enable messaging according to the Publisher-Subscriber pattern, i.e. server-to-client notification.

6.3.2 The BPEL Object Model

The structure of the resulting BPEL document is reflected by a BPEL Object Model (BOM). It mainly consist of an object tree, where every node represents a BPEL activity and additional information concerning its visualization.

The full BOM is held on the server. The client side object model contains less information and thus is more light-weight than the server side object model. This reduces the amount of data that has to be sent when joining a session and also simplifies the implementation of the client. Every relevant object (activities, links, etc.) has an identifier which is identical in the client model as well as in the server model. The identifiers are created on the server, which guarantees for their uniqueness within a session.

On the server a BOM also maintains a collection of references to WSDL file representations, which include information about partner link types and message types.

Activity objects can be marked as unlocked or as locked by a specific user. The server uses this information to decide about the validity of an operation, while the client needs it for visualization purposes.

An activity object holds information relevant for compilation to BPEL (its configuration, its incoming and outgoing links, etc.), for internal management (e.g. its parent activity) and for visualization purposes (e.g. its x- and y-position within a flow). Structured activities also contain references to their children.

Client side activity representations contain some, but not all of the information of the server side objects, e.g. the configuration information is only held on the server. Server side activity representations contain a map of the properties of the affected activity. For manipulating its configuration, the client fetches the current configuration from the server and presents it in a configuration form. Structured activity representations also include the container widget, which is used for editing its children.

XPath Expressions are represented by their own object trees and later compiled to XPath strings. If a user wants to edit a condition, its current status is loaded from the server.

6.3.3 The HOBBES Server

The server's main components are a session manager singleton which manages access to the single user and shared sessions and a controller servlet, which processes input from the clients. Users authentication is handled by a login manager singleton, which either queries a user database or assigns temporal access for demo purposes. Compiled workflows optionally are stored in a database using the Hibernate[2] Object/Relational-mapper. The workflows are stored as Base-64 encoded, zip-compressed archives, whose format is discussed in Chapter 9.

Every workflow design session consists of a session information object and a BPEL object model, representing the current state of the workflow. A session information object includes the session's creator who acts as a team leader and the set of users logged into the session.

Input messages are tagged with a user-id, a session-id and a number identifying the requested operation. Further parameters identify the involved elements and specify the kind of modification. The controller decides upon the validity of a user request, manipulates the concerned BOM and sends an answer message via HTTP reply, if necessary. If a manipulation can affect the views of other clients (e.g. creating an activity), it is propagated via an LCDS destination.

Some types of operations are only available to the team leader, e.g. granting locks. In these cases, the controller checks if the user id of the message is the team leader's user id to decide upon the validity of the operation. Operations which affect the BOM are processed in short critical regions within the controller servlet. Every critical region first checks whether the resources necessary to fulfill a request exist and whether

[2] `http://www.hibernate.org`, last accessed 18/11/2009.

the affected activities are available to the user. This may not be the case if an activity has been locked. If this check and further consistency checks as described in Chapter 4 succeed, the operations are performed and then announced to the requester and all other clients. If a check does not succeed, the reaction of the system depends on the nature of the request. If e.g. two users tried to delete the same activity at the same time, only one deletion request will succeed but no special information about the failed deletion request needs to be published. If on the other hand, a client requests to import a WSDL file which is not available on the server any longer, the reply will contain an error message.

The workflow metrics are implemented as *strategy* classes which define a recursive method that visits every activity of a workflow [GHJV95]. Every time, the metrics are computed, their global values are saved with a time stamp, thus defining a metrics history of the workflow project. The history data is later visualized using the Yahoo Charting API, a RESTful Web Service providing sophisticated charting facilities as a part of the Yahoo UI library.[3] Every metric has a unique identifier used both on the client and server. The metrics are registered with a mediator which is used to apply the metrics to a workflow.

[3]`http://developer.yahoo.com/yui/`, last accessed: 17/11/2009.

Chapter 7

Extensions to HOBBES

This Chapter presents three different extensions to the *Hobbes* system. In Section 7.1 we describe, how HOBBES has been extended with a Video Conferencing component that is based on a RESTful API called *VCR*. Video-conferencing is not supported in the public demo version of HOBBES, due to department policies concerning media servers. The *VCR* API implementation can be downloaded from the HOBBES web site, though. Section 7.2 describes the collaborative WSDL editor LEVIATHAN, which can be used independently from HOBBES. The Chapter closes in Section 7.3 with a brief description of a BPEL model visualization component, which has been developed in cooperation with the Parallel Computation Group.

7.1 Video Conferencing

7.1.1 Video Conferences via REST

To simplify the integration of video conferencing in HOBBES, we have developed a RESTful[1] video conferencing API. The purpose of the Video Conferencing with REST API (VCR) is to provide and govern access to a video conferencing service. A video conferencing service differs from a pure video streaming service, as it provides full video conferencing capabilities without further client development. The code-on-demand feature of the REST architecture style enables decoupling administration and representation of resources [Fie00]. Exploiting the code-on-demand property of the REST architecture style thus enables a late binding of media services to conference sessions.

[1]For a general overview of the REST architecture style see Chapter 2.1.4.

7.1.2 Resources and States of Video conferencing

On a conceptual level, video conferencing clients deal with two distinct kinds of services. An administrative service A serves as the front-end for developers who want to integrate video conferencing into their products. A administers access to a set of media services and a temporal function S_A, where $S_A(t)$ yields the set of available sessions at time t. A video conference session s is a temporarily available resource, consisting of a set of users $U_s = \{u_1, \ldots, u_n\}$, a media service M_s and corresponding service parameters $P_s = (p_1, \ldots, p_m)$:

$$s = (id_s, U_s, M_s, P_s)\,.$$

We assume U_s, M_s, and P_s to be fixed over time. P_s designates the used video and audio codecs as well as quality information (e.g. bit rate, screen size). In object-oriented terminology, sessions are created using a *Factory* method provided by A , which yields an identifier [GHJV95]:

$$A.createSession() \rightarrow id_s.$$

A user u joining a session is redirected to the corresponding media service if it exists and u is a participant of the session:

$$A.getSession(id_s, u, t) \rightarrow \left\{ \begin{array}{cc} M_s(u), & u \in s \wedge s \in S_A(t) \\ -1, & u \notin s \vee s \notin S_A(t) \end{array} \right\}.$$

A can be queried about the currently available sessions: $A.getSessions(t) \rightarrow S_A(t)$. Sessions also can be deleted.

7.1.3 A Relaxed REST Paradigm

For modeling the resources described in Section 7.1.2, we have applied a relaxed version of the REST paradigm to overcome the problems mentioned in Chapter 2.1.4, thus leading to a REST-RPC-hybrid architecture [RR07]. While pure REST directly maps CRUD operations to HTTP-methods, we provide an alternative mapping. *Read* operations are mapped solely to HTTP-Get methods. *Create* operations, however, are mapped to HTTP-Put, HTTP-Get, and HTTP-Post, *Delete* operations to HTTP-Delete, HTTP-Get, and HTTP-Post and *Update* operations are mapped to HTTP-Get and HTTP-Post.

Providing an HTTP-Get method for every command enables using the API directly from any web browser, while providing HTTP-Post as an alternative to HTTP-Put respectively HTTP-Delete helps avoiding firewall problems. On top of this REST-RPC-hybrid architecture, we have implemented a RESTful interface, which adopts and redirects REST-operations to their hybrid counterparts (see Figure 7.1). In contrast to traditional REST services, the Put operation does not take the resource Id as a parameter, but returns a generated resource Id.

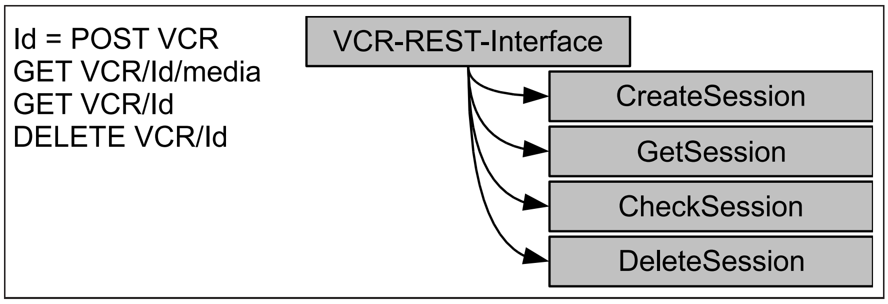


Figure 7.1: A pure REST interface on top of a REST-RPC-hybrid architecture.

7.1.4 Architecture and Implementation

Architecture

Four different servers are involved in managing video conference sessions (see Figure 8.8). A web server provides the actual web application which is to be augmented with video conferencing. When initiating a video conference, the web application issues HTTP-requests to the conference management server, which sets management information, especially access rights, in a database server. A media server provides its streaming capabilities. When a client wants to access an existing session, the conference management server redirects the client to an HTML-page with an embedded Flash-application which connects to the media server. The media server grants or denies access according to the information in the database. It does not query the database directly, but requests information from the management server.

An advantage of this architecture is the decoupling of session management and the actual video streaming service. Since the client software is provided on demand, both the video client implementation and the media server can be switched at any time. This can be used for load balancing reasons or for using alternative video

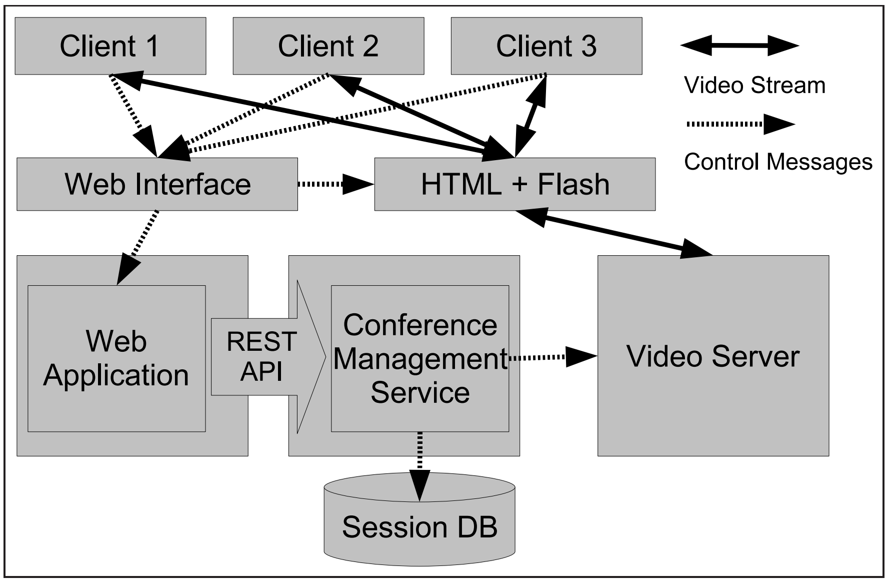


Figure 7.2: The architecture of VCR.

service implementations. Currently the VCR system supports the Adobe Flash Media Server[2] and the the open-source media server Red5.[3]

The VCR API is implemented by servlets and grants access to arbitrary video conferencing solutions using four commands dealing with the creation and deletion of session resources, one command to retrieve a list of the existing sessions, and another to access a session. An additional servlet provides a translation to pure REST, as described in Section 7.1.3. Figure 7.3 shows the video conference client provided by VCR.

RESTful Integration in HOBBES Video conferencing is intended to augment the collaborative features of HOBBES, but must not disturb developers during the workflow design process. The permanent reception of audio-visual information on all clients involved in a development session would be a rather harsh disturbance. The initiation of a video conference should thus normally be limited to the design team leader. The team leader thus is granted the privilege to invoke a video conference at any time with a set of participants of his choice. The participants' clients will immediately respond by entering the video conference.

[2]`www.adobe.com/products/flashmediaserver/`, last accessed: 12/09/2008.
[3]`http://www.osflash.org/red5`, last accessed: 09/09/2008.

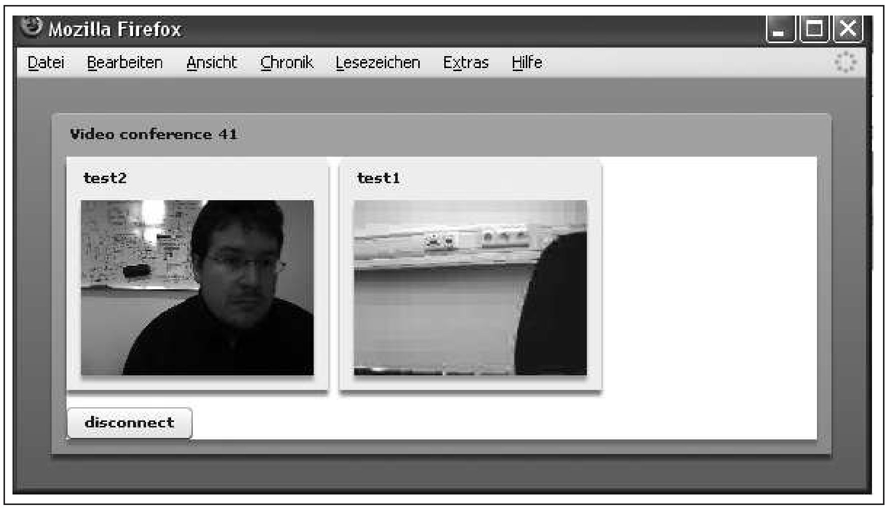


Figure 7.3: The Video Conference Client.

Figure 7.4 shows the control message flow used for integrating VCR in HOBBES. At the beginning of a video conference, the team leader selects the participants from the team members. The list of participants is sent to the HOBBES control server (see ❶), which creates a conference session using `CreateSession` (see ❷). Afterward, the HOBBES server broadcasts the creation of the new session on its notification channel to the clients (see ❸). The clients of participating users react by opening a video conference session in a new window via Javascript, which call `GetSession` on the VCR server (see ❹). This yields a video conference client, which immediately connects to the media server (see ❺).

7.2 Collaborative WSDL Editing

7.2.1 Interfaces in the Context of Collaborative Workflow Design

As mentioned in Chapter 2.1.5, the interfaces of BPEL workflows are described in WSDL 1.1. Thus, while workflow development as a process is decoupled from WSDL design, it depends upon the existence of a WSDL file describing the workflow interface. We have thus devised the prototype of a collaborative WSDL development system, which uses the same technology and principles as HOBBES. The collaborative WSDL design system LEVIATHAN can be used independently of HOBBES, but

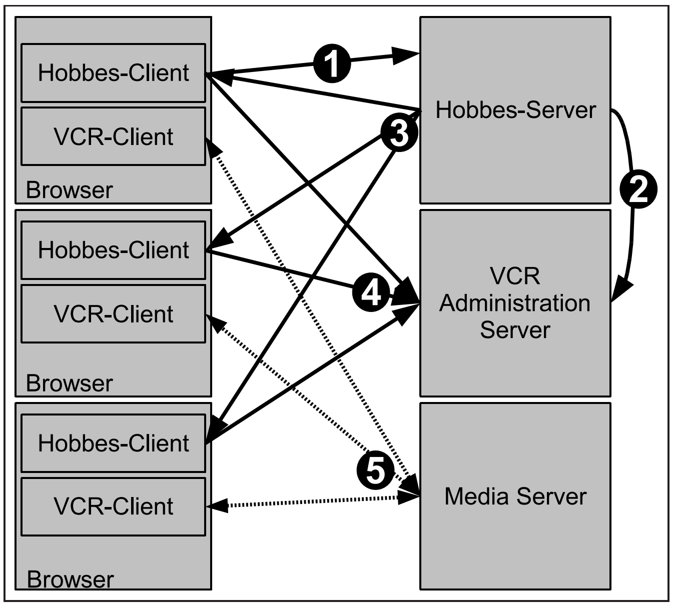


Figure 7.4: Message Flow in HOBBES with Video conferencing.

resulting WSDL documents may be used in HOBBES or other BPEL development systems.

Figure 7.5 gives an overview of the process of collaborative workflow design, extended by an activity of service interface design. We have discussed the activities in block ❷ in detail in the previous chapters, and will now derive a description of collaborative interface design (❶).

7.2.2 Requirements of Collaborative Web Service Interface Design

Some, but not all concepts from collaborative workflow development can be applied to collaborative interface design. We will now identify differences between workflow models and interface definitions, which influence the structure of a collaborative interface design process.

Fixed set of layers While workflows can be arbitrarily nested, interface definitions consist of a fixed set of logical layers. For example, BPEL programs consist of nested *structured activities* and BPMN workflows can contain any number of sub-

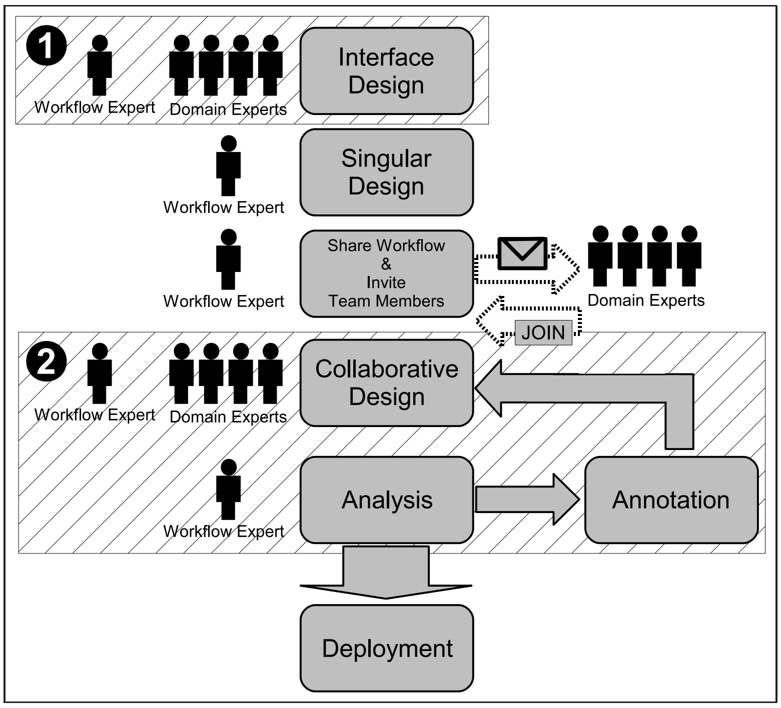


Figure 7.5: The context of Collaborative Interface Design.

workflows. WSDL definitions, on the other hand consist of a tuple of XML Schema declarations, message types, port types, services, and bindings.

Non-intuitive Visualization Workflow models can be visualized as their control flow graph. WSDL documents could be visualized as a layered graph showing the relation of services to port types and port type operations to messages respectively. In contrast to workflow visualizations, this does not convey an intuitive understanding of the Web Service functionality.

Size and Complexity With the exception of trivial workflows, we can expect workflow models to be more complex than their corresponding interface definitions.

No Metrics In collaborative workflow development, workflow-specific software metrics can be applied to simplify process steering for the team leader. In contrast to BPEL or other workflow languages, WSDL is not a programing language. Thus, traditional software metrics cannot be applied.

7.2.3 The Process of Collaborative Interface Design

Figure 7.6 illustrates the process of collaborative interface design as an Event Driven Process Chain. The team leader initiates a session, which can afterwards be joined and left by other team members at any time. The team leader will temporarily grant privileged editing access to one team member, who will assume the role of an *editor*, while other team members will receive updates on their clients, thus assuming the role of *observers*. During the design phase, all team members can communicate, especially to comment on the editor's actions.

7.2.4 User Interface

Using options from the top menu, the team leader can grant privileged access to the document to individual team members or reset the WSDL document to a different state from a tree of older document versions.

Figure 7.7 shows a screenshot of the Leviathan user interface. The middle panel consists of a set of editors for the different parts of the WSDL document. On the left side, a tree view enables an easy navigation through the document, while a text view is available on the right side. The left side also contains a chat widget.

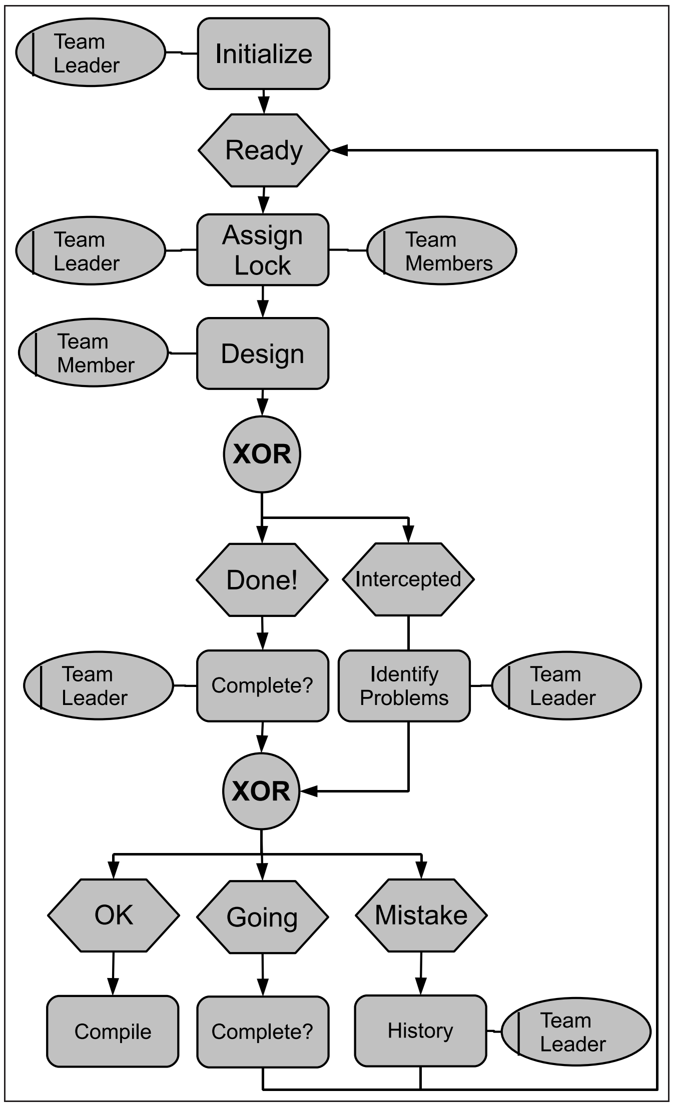
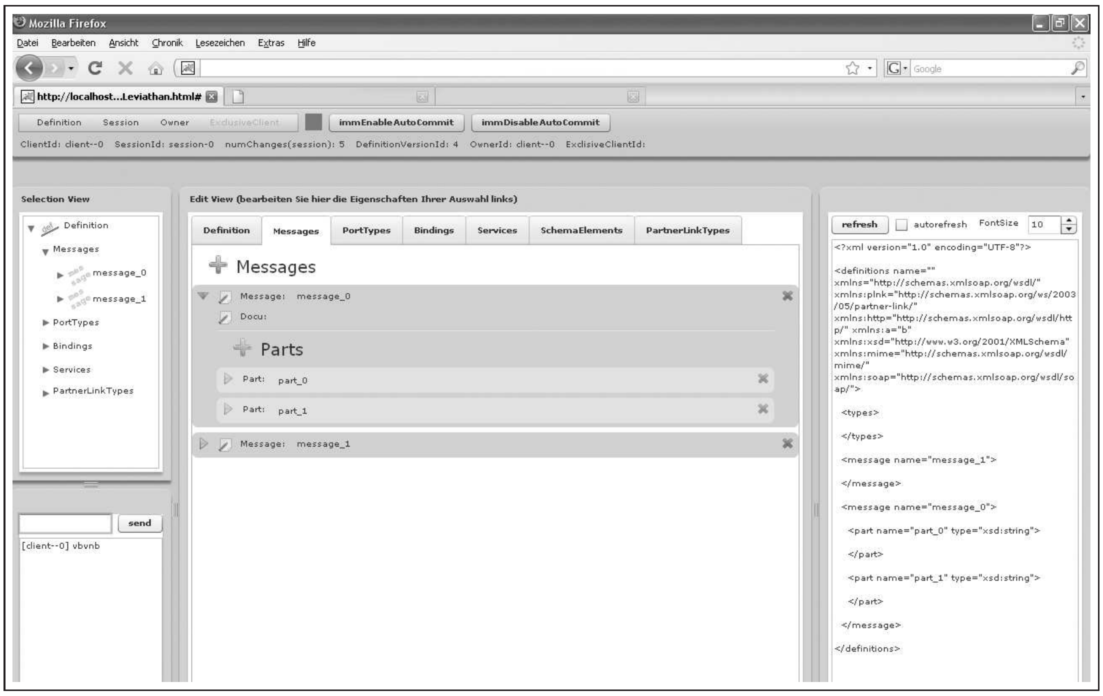


Figure 7.6: The Process of Collaborative WSDL design.



Mozilla Firefox
Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe
http://localhost...Leviathan.html#
Definition Session Owner ExclusiveClient
immEnableAutoCommit
immDisableAutoCommit
Selection View
Edit View (bearbeiten Sie hier die Eigenschaften Ihrer Auswahl links)
Definition
Messages
message_0
message_1
PortTypes
Bindings
Services
PartnerLinkTypes
Definition Messages PortTypes Bindings Services SchemaElements PartnerLinkTypes
Messages
Message: message_0
Docu:
Parts
Part: part_0
Part: part_1
Message: message_1
send
refresh
autorefresh
FontSize 10

ClientId: client--0 SessionId: session-0 numChanges(session): 5 DefinitionVersionId: 4 OwnerId: client--0 ExclisiveClientId:

[client--0] vbvnb

```
<?xml version="1.0" encoding="UTF-8"?>

<definitions name=""
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:plnk="http://schemas.xmlsoap.org/ws/2003
/05/partner-link/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/htt
p/" xmlns:a="b"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/
mime/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/so
ap/">

 <types>

 </types>

 <message name="message_1">

 </message>

 <message name="message_0">

  <part name="part_0" type="xsd:string">

  </part>

  <part name="part_1" type="xsd:string">

  </part>

 </message>

</definitions>
```

Figure 7.7: The User Interface of LEVIATHAN.

7.2.5 Architecture and Implementation

Similar to HOBBES, LEVIATHAN has been implemented using the Adobe Flex framework [fle] and server-side Java. Like HOBBES, LEVIATHAN uses a central server model to validate change requests and broadcast operations. In contrast to HOBBES, the joining protocol of LEVIATHAN is based on invalidating the client model in the case of inconsistencies. Such inconsistencies are detected via timestamps and lead to reloading the central model as a whole. The LEVIATHAN client can be used in a disconnected mode, where model changes are performed on the local model, and sent to the server when reconnecting. If possible, the operations are then performed on the central model, which is reloaded by the joining client.

7.3 Workflow Visualization

7.3.1 Overview

In cooperation with the Parallel Computation Group, we have developed a workflow visualization module for HOBBES. It uses a workflow notation inspired by UML activity diagrams and BPMN, but adjusted to the BPEL language, with an emphasis on explicitly depicting BPEL elements and enabling visualizations of strongly nested workflows. The layout algorithm is based upon an extension of the Sugiyama algorithm [STT81] devised by Philip Effinger and Benjamin Albrecht, and is well-suited for visualizing strongly nested workflows. It has been implemented using the *yFiles* library.[4]

7.3.2 The Notation

Figure 7.8 gives an overview of elements used in the BPEL visualization notation. The workflow models are represented as control flow graphs, which are drawn from the top of a page to the bottom. Activities are distinguished by color, shape and labels. With the exception of Sequence activities, those structured activities which contain an arbitrary number of children are visualized by boxes of adjustable size, where the upper and lower margin represent control flow nodes. For example Figure 7.8 ❶ depicts a Flow activity. Similar to other workflow notations, basic activities are depicted as rounded rectangles of a fixed size (see Figure 7.8 ❷). Loops and sequences are represented by special start and end nodes (see Figure 7.8 ❹, ❺). Pick

[4]`http://www.yworks.com`, last accessed 03/12/2009.

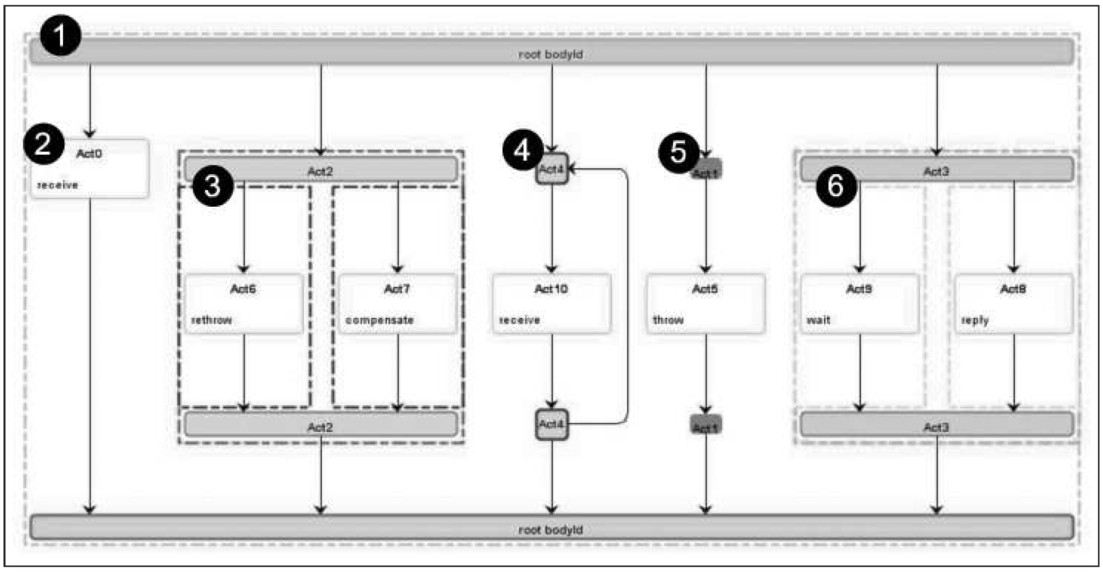


Figure 7.8: The BPEL notation used by the HOBBES visualization extension.

and If activities are drawn similar to Flow activities, with *swimlane*-like borders limiting the alternative execution paths (see Figure 7.8 ❸, ❻). The algorithm attempts to minimize crossings, which result from Flow links.

Chapter 8

The CALVIN Life Science Workflow System

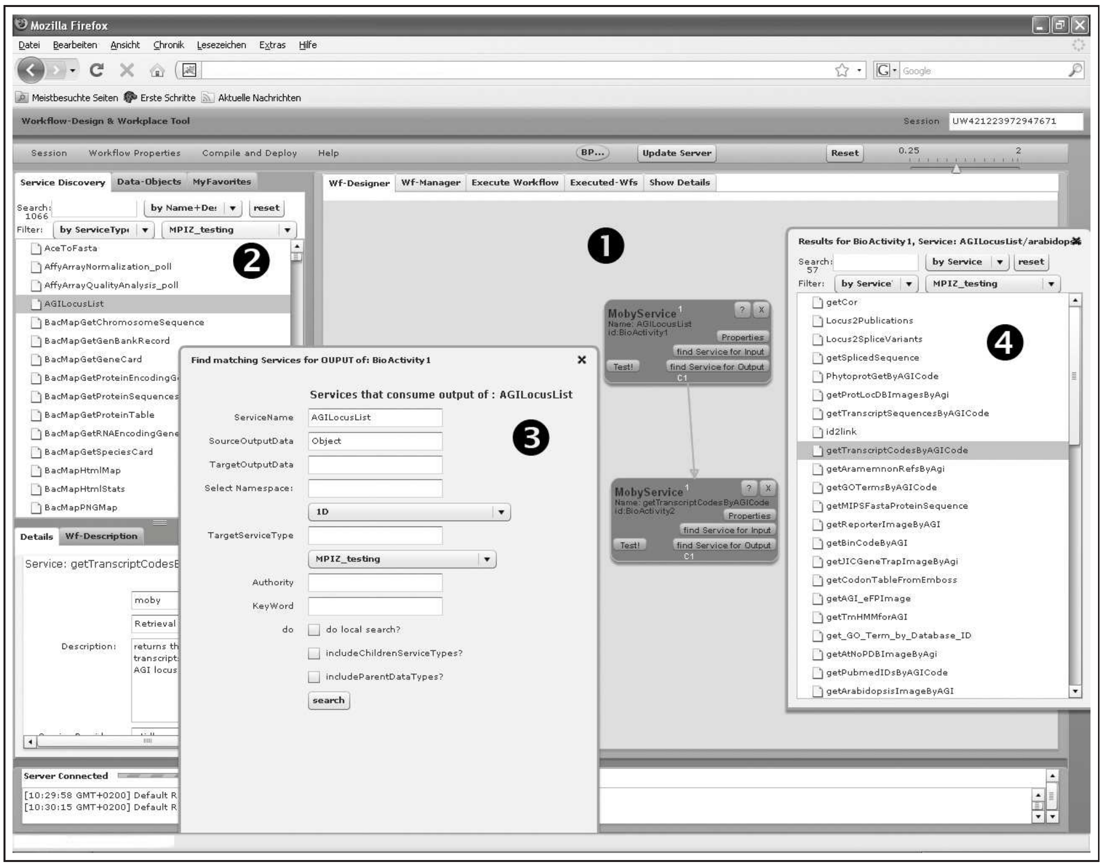
In this Chapter we introduce the CALVIN system, which fulfills the requirements for a biology-specific workflow system described in Chapter 3.3.5. CALVIN builds on the same technology stack as HOBBES. In contrast to HOBBES, CALVIN is a domain-specific workflow management system, which provides a simple and intuitive notation, as well as workflow execution facilities. Section 8.1 introduces the CALVIN user interface, followed by an example of a CALVIN workflow in Section 8.2. The internal representation of CALVIN workflows and their compilation to BPEL is described in Section 8.3. The Chapter closes with a discussion of the architecture and implementation of CALVIN in Section 8.4.

8.1 The CALVIN User Interface

8.1.1 Workflow Editing

Figure 8.1 shows the main screen of the CALVIN workflow design system. It consists of an editor canvas, showing the activities of the workflow as rounded boxes (❶) and data-flow connections between activities as arrows. Activities can be dragged from a palette of available services (❷). The arrows are green if types are compatible, red if not, and orange if this cannot be decided. The CALVIN workflow notation is kept very simple, to ease its use for biologists. Effectively, it represents a tree of exploration steps, similar to the e-science processes described in Chapter 3.3.3. Hence, users do not have to deal with complex control structures.

Services that process the output type or produce the input type of an activity can be found using a semantic search facility. By clicking on an activity, a query window is opened (❸) which can be used to search the BioMOBY database for a list of adequate services (❹). Users can open a description of the service represented by an activity, or invoke the service directly from the activity properties menu. This enables testing the


Mozilla Firefox
Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe
Meistbesuchte Seiten Erste Schritte Aktuelle Nachrichten
Workflow-Design & Workplace Tool
Session Workflow Properties Compile and Deploy Help
BP...
Update Server
Reset
Service Discovery
Data-Objects
MyFavorites
Wf-Designer
Wf-Manager
Execute Workflow
Executed-Wfs
Show Details
Details
Wf-Description
1
2
3
4

Session UW421223972947671

0.25 2

Search: 1066 by Name+De: reset
Filter: by ServiceType MPIZ_testing

AceToFasta
AffyArrayNormalization_poll
AffyArrayQualityAnalysis_poll
AGILocusList
BacMapGetChromosomeSequence
BacMapGetGenBankRecord
BacMapGetGeneCard
BacMapGetProteinEncodingG
BacMapGetProteinSequences
BacMapGetProteinTable
BacMapGetRNAEncodingGene
BacMapGetSpeciesCard
BacMapHtmlMap
BacMapHtmlStats
BacMapPNGMap

Service: getTranscriptCodesB
moby
Retrieval
Description: returns th transcript AGI locus

Server Connected
[10:29:58 GMT+0200] Default R
[10:30:15 GMT+0200] Default R

Find matching Services for OUPUT of: BioActivity1

Services that consume output of : AGILocusList

ServiceName: AGILocusList
SourceOutputData: Object
TargetOutputData:
Select Namespace:
1D
TargetServiceType:
MPIZ_testing
Authority:
KeyWord:
do: do local search?
includeChildrenServiceTypes?
includeParentDataTypes?
search

MobyService 1
Name: AGILocusList
id:BioActivity1
Properties
find Service for Input
Test! find Service for Output
C1

MobyService 1
Name: getTranscriptCodesByAGICode
id:BioActivity2
Properties
find Service for Input
Test! find Service for Output
C1

Results for BioActivity1, Service: AGILocusList/arabidops

Search: 57 by Service reset
Filter: by Service MPIZ_testing

getCor
Locus2Publications
Locus2SpliceVariants
getSplicedSequence
PhytoprotGetByAGICode
getProtLocDBImagesByAgi
getTranscriptSequencesByAGICode
id2link
getTranscriptCodesByAGICode
getAramemnonRefsByAgi
getGOTermsByAGICode
getMIPSFastaProteinSequence
getReporterImageByAGI
getBinCodeByAGI
getJICGeneTrapImageByAgi
getCodonTableFromEmboss
getAGI_eFPImage
getTmHMMforAGI
get_GO_Term_by_Database_ID
getAtNoPDBImageByAgi
getPubmedIDsByAGICode
getArabidopsisImageByAGI

Figure 8.1: The CALVIN Workflow Editor. CALVIN enables the drag and drop composition of Web Services from the BioMoby registry. Successors and predecessors for a given activity can be searched for based on the input and output data types accepted by a service.

behavior of a service prior to running the workflow, thus enabling an explorative and data-oriented way of workflow composition. Activities can be marked as requiring user interaction, before or after their execution. Finished workflows are compiled to BPEL and deployed to a workflow execution engine.

Deployed CALVIN workflows can be re-used as new services. For every output of a CALVIN workflow, its corresponding activity has a virtual output port that can serve as an input of another activity. The representations of CALVIN workflow activities are distinguished from BioMoby activities by their lighter blue color.

8.1.2 Workflow Managment and Execution

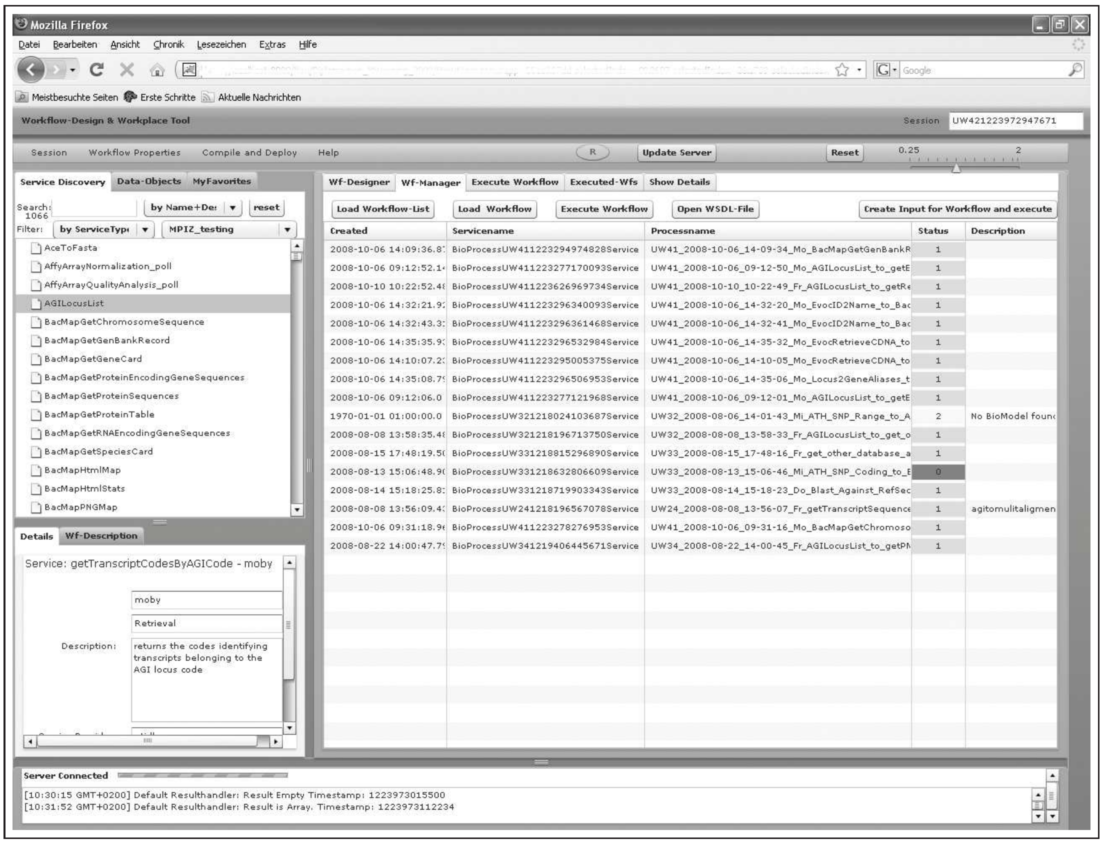
The Workflow Managment Console Figure 8.2 shows the Workflow Management Console, which presents a list of the workflows that are available on the server and can be loaded into the editor. The workflows are already deployed in the BPEL execution engine and can be executed from the console. From the workflow management console, deployed workflows can be added to the editor as special activities, as described in Section 8.1.1.

Editing Input Data Before beginning the actual execution, a window will show a graphical editor for workflow inputs (see Figure 8.3). Two different kinds of input parameters are available, primary inputs, which consist of biological data, and secondary parameters that influence the behavior of a service (e.g. its output format). Standard parameters are provided which can be overwritten by the user. Every input variable is presented with a description provided by the BioMOBY database.

Monitoring and Interaction After a workflows has been invoked from the workflow management console, its execution can be monitored on an HTML page provided by the Active BPEL engine. If the workflow execution reaches an activity that is marked as an interaction point, the user can be prompted to verify if the workflow should go on, or to view and edit result data (see Figure 8.4).

External clients can be developed using the WSDL file provided by a workflow. Those WSDL-files are annotated with documentation data from the BioMoby database which yield a complete description of the semantics of each service invocation.

The Provenance Browser Every workflow execution yields a result file containing the time of invocation, the inputs and service parameters, and all final and



Session: UW421223972947671

Search: 1066
Filter: by ServiceTyp | MPIZ_testing

AceToFasta
AffyArrayNormalization_poll
AffyArrayQualityAnalysis_poll
AGILocusList
BacMapGetChromosomeSequence
BacMapGetGenBankRecord
BacMapGetGeneCard
BacMapGetProteinEncodingGeneSequences
BacMapGetProteinSequences
BacMapGetProteinTable
BacMapGetRNAEncodingGeneSequences
BacMapGetSpeciesCard
BacMapHtmlMap
BacMapHtmlStats
BacMapPNGMap

Created	Servicename	Processname	Status	Description
2008-10-06 14:09:36.87	BioProcessUW411223294974828Service	UW41_2008-10-06_14-09-34_Mo_BacMapGetGenBankR	1	
2008-10-06 09:12:52.14	BioProcessUW411223277170093Service	UW41_2008-10-06_09-12-50_Mo_AGILocusList_to_getE	1	
2008-10-10 10:22:52.48	BioProcessUW411223626969734Service	UW41_2008-10-10_10-22-49_Fr_AGILocusList_to_getRe	1	
2008-10-06 14:32:21.92	BioProcessUW411223296340093Service	UW41_2008-10-06_14-32-20_Mo_EvocID2Name_to_Bac	1	
2008-10-06 14:32:43.31	BioProcessUW411223296361468Service	UW41_2008-10-06_14-32-41_Mo_EvocID2Name_to_Bac	1	
2008-10-06 14:35:35.93	BioProcessUW411223296532984Service	UW41_2008-10-06_14-35-32_Mo_EvocRetrieveCDNA_to	1	
2008-10-06 14:10:07.23	BioProcessUW411223295005375Service	UW41_2008-10-06_14-10-05_Mo_EvocRetrieveCDNA_to	1	
2008-10-06 14:35:08.79	BioProcessUW411223296506953Service	UW41_2008-10-06_14-35-06_Mo_Locus2GeneAliases_t	1	
2008-10-06 09:12:06.0	BioProcessUW411223277121968Service	UW41_2008-10-06_09-12-01_Mo_AGILocusList_to_getE	1	
1970-01-01 01:00:00.0	BioProcessUW321218024103687Service	UW32_2008-08-06_14-01-43_Mi_ATH_SNP_Range_to_A	2	No BioModel found
2008-08-08 13:58:35.48	BioProcessUW321218196713750Service	UW32_2008-08-08_13-58-33_Fr_AGILocusList_to_get_o	1	
2008-08-15 17:48:19.50	BioProcessUW331218815296890Service	UW33_2008-08-15_17-48-16_Fr_get_other_database_a	1	
2008-08-13 15:06:48.90	BioProcessUW331218632806609Service	UW33_2008-08-13_15-06-46_Mi_ATH_SNP_Coding_to_E	0	
2008-08-14 15:18:25.81	BioProcessUW331218719903343Service	UW33_2008-08-14_15-18-23_Do_Blast_Against_RefSec	1	
2008-08-08 13:56:09.43	BioProcessUW241218196567078Service	UW24_2008-08-08_13-56-07_Fr_getTranscriptSequence	1	agitomulitaligmen
2008-10-06 09:31:18.96	BioProcessUW411223278276953Service	UW41_2008-10-06_09-31-16_Mo_BacMapGetChromoso	1	
2008-08-22 14:00:47.79	BioProcessUW341219406445671Service	UW34_2008-08-22_14-00-45_Fr_AGILocusList_to_getPM	1	

Service: getTranscriptCodesByAGICode - moby

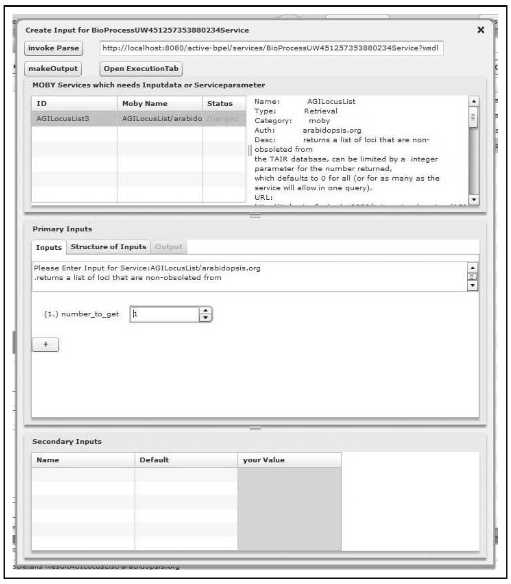
moby

Retrieval

Description: returns the codes identifying transcripts belonging to the AGI locus code

```
[10:30:15 GMT+0200] Default Resulthandler: Result Empty Timestamp: 1223973015500
[10:31:52 GMT+0200] Default Resulthandler: Result is Array. Timestamp: 1223973112234
```

Figure 8.2: The CALVIN Workflow Management Console. The console presents a list of workflows available on the server. Workflows which are presumed to be runnable and editable are marked green, non-editable workflows for which no Calvin object-model is available are marked yellow, and non-runnable workflows are marked red. A workflow can become non-runnable if it contains services, which the BioMoby central server deems to be dead. Each workflow has a service name, which is used in its WSDL definition, and a process name from its BPEL file.



Create Input for BioProcessUW451257353880234Service

http://localhost:8080/active-bpel/services/BioProcessUW451257353880234Service?wsdl

MOBY Services which needs Inputdata or Serviceparameter

ID	Moby Name	Status
AGILocusList3	AGILocusList/arabido	changed

```
Name:      AGILocusList
Type:      Retrieval
Category:  moby
Auth:      arabidopsis.org
Desc:      returns a list of loci that are non-
obsoleted from
the TAIR database, can be limited by a  integer
parameter for the number returned,
which defaults to 0 for all (or for as many as the
service will allow in one query).
URL:
```

Primary Inputs

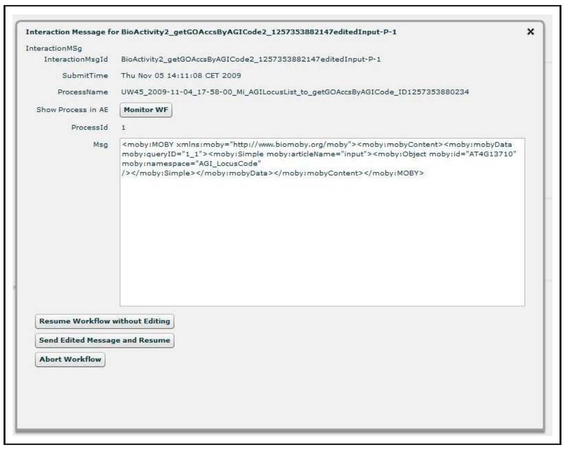
Please Enter Input for Service:AGILocusList/arabidopsis.org
.returns a list of loci that are non-obsoleted from

(1.) number_to_get 1

Secondary Inputs

Name	Default	your Value

Figure 8.3: The CALVIN execution input mask. In this example, a service expects an integer number as input. A description of the expected input data is given in the top right panel, while secondary parameters of any service could be edited in the table on the bottom.


×
Monitor WF
Resume Workflow without Editing
Send Edited Message and Resume
Abort Workflow

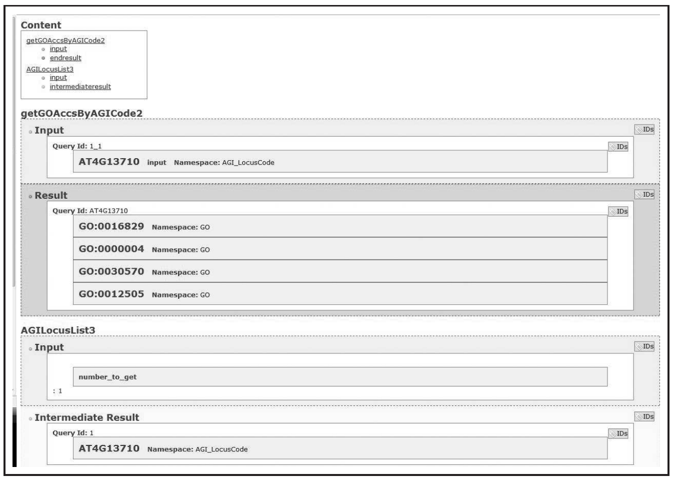
Interaction Message for BioActivity2_getGOAccsByAGICode2_1257353882147editedInput-P-1

InteractionMSg

InteractionMsgId	BioActivity2_getGOAccsByAGICode2_1257353882147editedInput-P-1
SubmitTime	Thu Nov 05 14:11:08 CET 2009
ProcessName	UW45_2009-11-04_17-58-00_Mi_AGILocusList_to_getGOAccsByAGICode_ID1257353880234
Show Process in AE	Monitor WF
ProcessId	1
Msg	`<moby:MOBY xmlns:moby="http://www.biomoby.org/moby"><moby:mobyContent><moby:mobyData moby:queryID="1_1"><moby:Simple moby:articleName="input"><moby:Object moby:id="AT4G13710" moby:namespace="AGI_LocusCode" /></moby:Simple></moby:mobyData></moby:mobyContent></moby:MOBY>`

Figure 8.4: User interaction with a workflow. The user can inspect the data which is passed from one service to another, alter the data if needed, and resume or exit the workflow.

intermediate results. Together with the serialized workflow model, this enables retracing every event in the workflow and the origin of its results. The result files can either be downloaded from the server or be explored using an HTML based provenance browser, which is shown in Figure 8.5. The provenance browser is presented to the user as a dynamically generated HTML page. Since the result files can be large, this page in general only presents a preview of the result file contents. If the user wishes to view a specific part of a large result file, it is loaded from the server.


Content

getGOAccsByAGICode2
- input
- endresult

AGILocusList3
- input
- intermediateresult

getGOAccsByAGICode2

Input IDs

Query Id: 1_1 IDs

AT4G13710 input Namespace: AGI_LocusCode

Result IDs

Query Id: AT4G13710 IDs

GO:0016829 Namespace: GO

GO:0000004 Namespace: GO

GO:0030570 Namespace: GO

GO:0012505 Namespace: GO

AGILocusList3

Input IDs

number_to_get

: 1

Intermediate Result IDs

Query Id: 1 IDs

AT4G13710 Namespace: AGI_LocusCode

Figure 8.5: The CALVIN provenance browser.

8.2 Example: Searching for Orthologous Gene Sequences

We now present a workflow which has been implemented with CALVIN. More examples are available at our web site. Figure 8.6 shows a workflow for searching for orthologous gene sequences, a common case in genomics. Genes from two species are called orthologous[1], if they are divergent copies of a single gene in a common ancestor species.

[1]"If a species diverges into two separate species, the divergent copies of a single gene in the resulting species are said to be orthologous.", `http://en.wikipedia.org/wiki/Orthologue#Orthology`, last accessed: 09/10/2008.

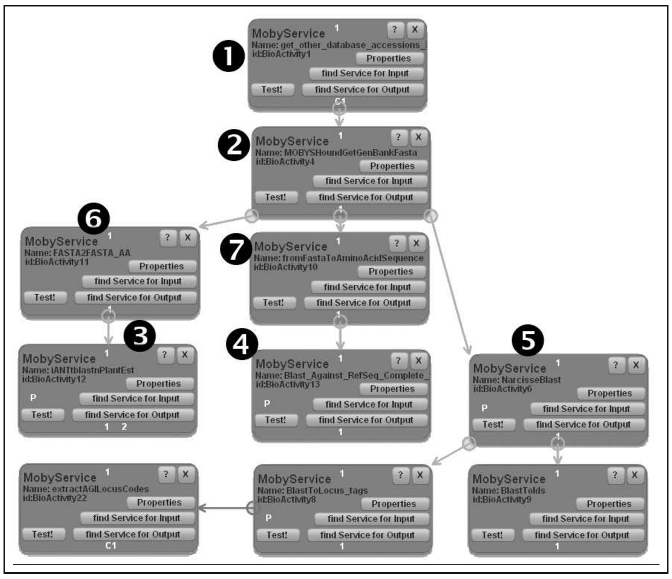


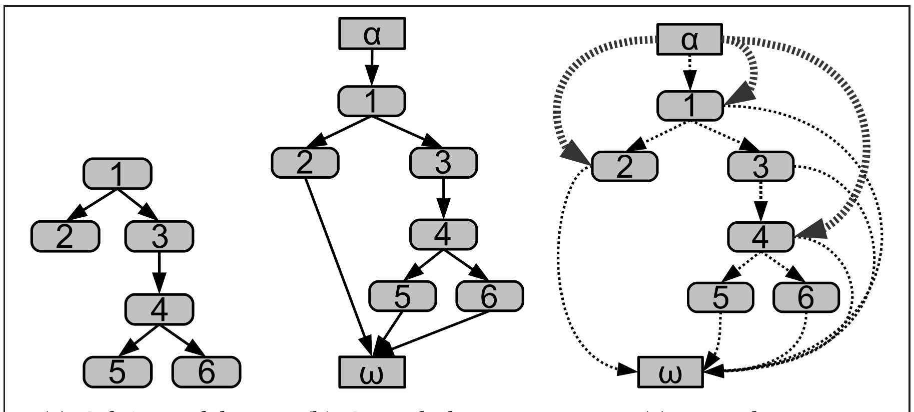
Figure 8.6: Search for Orthologous Gene Sequences.

The input is a list of sequence identifiers, which first are converted to the needed ID format in step ❶ and then yield the according sequences in step ❷. The sequences are subject to three different BLAST searches (❸, ❹, ❺) to query different databases with different parameters at the same time. Before that, two sequence conversions are necessary (❻, ❼). Afterward, sequence identifiers are extracted from the results, which can be used for further processing.

8.3 Representation of Biology Specific Workflow Models

The CALVIN notation represents workflow as a tree of exploration steps (see Chapters 8.1.1 and 8.2), which has to be transformed to a complete workflow. This implies on the one hand that not all workflows can be expressed via CALVIN, and that on the other hand, CALVIN workflows have to be automatically augmented by the system.

During the transformation process, the data-flow links are compiled to control-flow links and variable assignments. Additional data-flows augment the tree to a directed acyclic graph which collects all produced data elements. Figure 8.7(a) shows a CALVIN workflow as viewed by a biologist. It is represented by a tree $T = (N, E), E \subset N \times N$ with root n_1, where N is a set of BioMoby activities.



(a) Calvin Model. (b) Control Flow. (c) Data Flow.

Figure 8.7: Transformation of a CALVIN workflow.

CALVIN compiles the tree model T to a complete workflow model W, by adding a start activity α and a final activity ω, a set of control flow edges C, a set of data flow edges D which connect activity inputs and outputs, and a set of parameter edges P:

$$\begin{aligned}
W &= (N \cup \{\alpha, \omega\}, C, D, P), \quad \alpha \notin N \wedge \omega \notin N \\
C &= E \cup \{(n, \omega) | \forall k \in N : (n, k) \notin E\} \cup \{(\alpha, n_1)\} \\
D &= E \cup \{(\alpha, n_1)\} \cup N \times \{\omega\} \\
P &= \{(\alpha, n) \in \{\alpha\} \times N, \; where \; n \; has \; parameters\}
\end{aligned}$$

The complete workflow model W thus consists of three graphs defined on the same node set $N \cup \{\alpha, \omega\}$. α is responsible for taking the client's input and serving parameters to those activities, which have parameters. ω takes the output of all activities and sends the answer message. Figure 8.7(b) shows the control flow graph of W, while Figure 8.7(c) presents the data flow edges from D as dotted lines, and the edges from P as thick dotted lines respectively.

The interface of a CALVIN workflow is given as a Tuple $\tau = \big(\iota, o(n_1), \ldots, o(n_{|N|})\big)$, where ι is the BioMoby type of its input message, and the $o(n_i)$ denote those message parts of its output message, which accord to the output of a BioMoby activity. An $o(n_i)$ is a tuple of a message part name and a description of its BioMoby type.

If n_j represents an embedded CALVIN workflow, it yields a tuple $(o(n_{j,1}), \ldots o(n_{j,k}))$, which is added to τ instead of a single $o(n_j)$ Within the CALVIN object model, every output of a CALVIN service is treated as a virtual activity with its own identifier and can thus be used as an input source for other activities.

8.4 Architecture and Implementation

8.4.1 The CALVIN Architecture

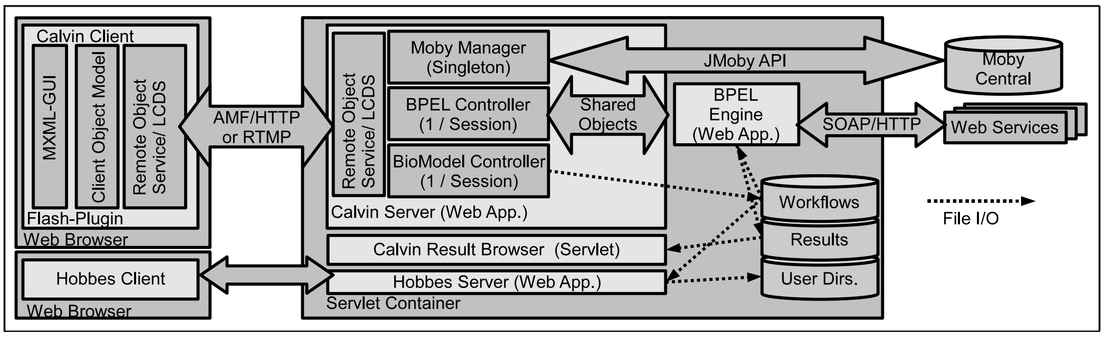


Figure 8.8: The CALVIN architecture. As shown in the picture CALVIN and HOBBES can be deployed on the same server. Note that the architecture of HOBBES is described in detail in Chapter 6. The integration of CALVIN and HOBBES via Web 2.0 techniques is discussed in Chapter 9.

Figure 8.8 shows an overview of the CALVIN architecture. Like HOBBES, CALVIN clients have been implemented with the Adobe Flex[2] framework (see Chapter 6.3). Client/server Communication in CALVIN is facilitated via the LCDS *Remote Object Services*, which transparently map client-side Action Script objects to server side Java objects.

CALVIN exposes three different kinds of remote objects. For every session, one *BioController* object is instantiated which holds a server-side workflow model. To enable communication with the BPEL execution engine, one *BPEL Engine Controller* object is instantiated per session. The *MOBY Manager* singleton is exposed as a remote object, to enable service queries. Using the JMoby API[3] it provides access to the BioMOBY registry [WL02].

[2]`http://www.adobe.com/devnet/flex/`, last accessed: 24/10/2007.
[3]`http://biomoby.open-bio.org/CVS_CONTENT/moby-live/Java/docs/`, last accessed 20/10/2008.

We use the *ActiveBPEL* execution engine as a workflow container. ActiveBPEL[4] is available as an open source project and deployed as a Java Web Application. Thus it can easily be integrated with other web applications like HOBBES and CALVIN.

The provenance browser has been implemented as a servlet, which processes three sources of input data: the serialized workflow model, the WSDL definition with BioMoby annotations, and the result file. An initial request to the servlet generates an HTML page, which displays a shortened version of the file content. If the user navigates to a part of the result file which is not present on the page, it gets loaded from the server in a separate window.

8.4.2 Transformation of CALVIN Workflows to BPEL Object Models

We now describe the compilation of CALVIN workflows to BPEL based on the considerations of Chapter 8.3. For each session, CALVIN holds a server-side and a more light-weight client-side object model, which communicate via transfer objects. When compiling and deploying a CALVIN workflow, it is transformed to a HOBBES BPEL Object Model (BOM), which is afterward compiled to a BPEL document.

Before compiling the BPEL model, the BioMoby WSDL definitions have to be adapted by adding Partner Link Type definitions. CALVIN generates a WSDL definition for the process, including input and output messages whose message parts are annotated with BioMoby activity descriptions. The BOM is then generated by first importing all WSDL definitions and adding a global Flow activity with a Receive and a Reply activity. For every CALVIN activity, a Sequence activity is added, with an input and an output assignment, and a service invocation. Control flow edges are mapped to BPEL Flow Links, and data flow edges to Assign activities. If user interaction is required before or after the execution of an activity, CALVIN proceeds as described in Chapter 8.4.3. Transforming CALVIN workflows to BOMs thus consists of the following steps:

1. Load the BioMoby WSDL definitions for all involved services.
2. For every WSDL definition:
    - Assign a new (generated) namespace.
    - Add Partner Link Type definitions.
3. Generate a WSDL-definition for the BPEL process.

[4] `www.activevos.com`, last accessed 10/12/2009.

4. Generate the BOM:
    - Import all needed WSDL-definitions.
    - Generate a global Flow activity.
    - Add data flow variables as needed. We assume one output variable for each BioMoby service. For a CALVIN service, an output variables is generated for every output of an encapsulated service invocation.
    - For every CALVIN activity, add a sequence activity, with an input and an output assignment, and a service invocation for either a remote BioMoby Web Service or a local CALVIN Web Service. If user interaction is required, proceed as described in Chapter 8.4.3.
    - Map CALVIN data flow links to BPEL Flow Links.
    - Generate a Receive and a Reply activity.
    - Connect the Receive activity to all initial activities.
    - Connect all final activities to the Reply activity.

After the transformation, the BOM is compiled to a BPEL file, which is saved to a temporary directory with its imported WSDL files and additional deployment description files. A system-dependent deployment script compresses the temporary directory to an ActiveBPEL archive, which is then copied to the workflow deployment directory.

After a CALVIN workflow has been deployed, it can be reused as an activity within other CALVIN workflows. Workflows which have been edited via HOBBES can be re-used as long as their WSDL definition is compatible with the CALVIN system. In contrast to BioMoby services CALVIN workflows are provided as Web Services by the same server as the CALVIN system itself.

8.4.3 User Interaction

User interaction (see Chapters 8.1.1 and 8.1.2) has been implemented using XPath custom functions. An XPath custom function is a user defined function in the XPath language which usually is implemented as a Java object. Figure 8.9 outlines the algorithm as pseudo-code. The first three assignments invoke XPath custom functions to communicate with the user. Messages for the client are passed through a singleton via the LCDS notification service, while messages from the user are stored in a blocking queue. The communication loop includes a timeout mechanism to prevent waiting infinitely for user feedback. After finishing the polling loop, the conditional tests, whether the workflow just has to resume or whether changed data has to be assigned. If the user decided to quit the workflow, a *throw* activity is executed.

```
<sequence>
  <assign /> <!-- initiate user interaction -->
    <repeatuntil>
      <sequence>
        <wait />
        <assign /> <!-- poll -->
      </sequence>
    </repeatuntil>
    <assign /> <!-- read user interaction result -->
    <if>
      <case>
        <empty/> <!-- resume workflow w/o changes -->
      </case>
      <case>
        <assign /> <!-- read changed data -->
      </case>
      <else>
        <throw /> <!-- quit workflow -->
      </else>
    </if>
</sequence>
```

Figure 8.9: BPEL-Pseudo-Code for User Interaction.

We have decided against using BPEL4People [KKL$^+$] to implement the user interaction facilities because it has not been standardized yet. As of November 2009 proposals for standardizing BPEL4People and WS-HumanTask exist, but appear to be overly complex for our purposes. For a more detailed discussion of BPEL4People see Chapter 2.1.5.

Chapter 9

Integrating CALVIN and HOBBES

This Chapter discusses the integration of CALVIN and HOBBES as RESTful Web Services. Section 9.1 presents a high-level motivation for loosely integrating CALVIN and HOBBES. Afterwards, Section 9.2 discusses the prerequisites for file-based integration, while Section 9.3.1 deduces requirements for corresponding REST interfaces from the use in mashups. Section 9.3.2 introduces the REST interfaces of HOBBES and CALVIN, while Section 9.3.3 discusses their implementation.

9.1 The Interplay of CALVIN and HOBBES

CALVIN enables biologists to easily define workflows for common tasks in a domain specific workflow notation. As we have pointed out in Chapter 3.1, in a domain-specific environment editing or executing workflow models refined on a lower level of abstraction is not always possible. A general mapping from BPEL to the CALVIN notation is impossible, since BPEL is Turing-complete. Leaving the WSDL interface of a compiled CALVIN workflow unchanged is a sufficient condition for the invocation of the workflow from CALVIN, though. Thus, in case more sophisticated workflows are needed which cannot be expressed in the CALVIN notation, biologists can request the help of software engineers or bioinformaticians, who can refine the workflow using our collaborative BPEL development system HOBBES. Figure 9.1 visualizes the possible interplay of CALVIN and HOBBES.

The entire circle can in principle be realized by copying files from CALVIN servers to HOBBES servers, and vice versa. This depends upon a coordinated interplay of the development team with system administrators or other users with read/write permissions in the corresponding directories, as illustrated in Figure 9.2(a). In this scenario, HOBBES and CALVIN are completely decoupled applications with — fully or partially — compatible file interfaces. Since HOBBES and CALVIN are based on the same technology stack, they could also be integrated as one web application (see Figure 9.2(b)), which is a rather inflexible solution. A third option is to provide

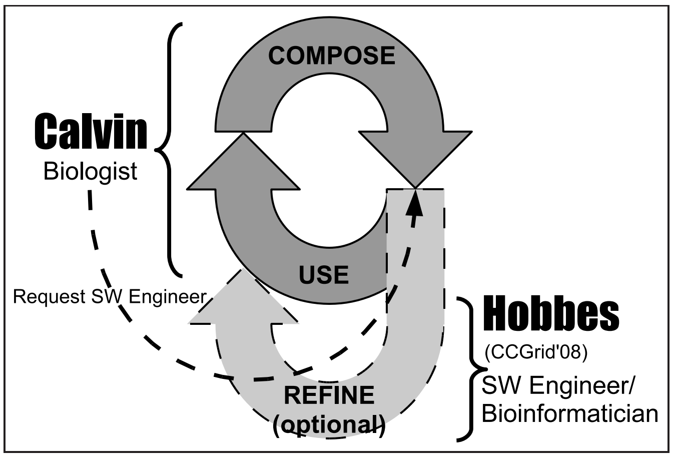
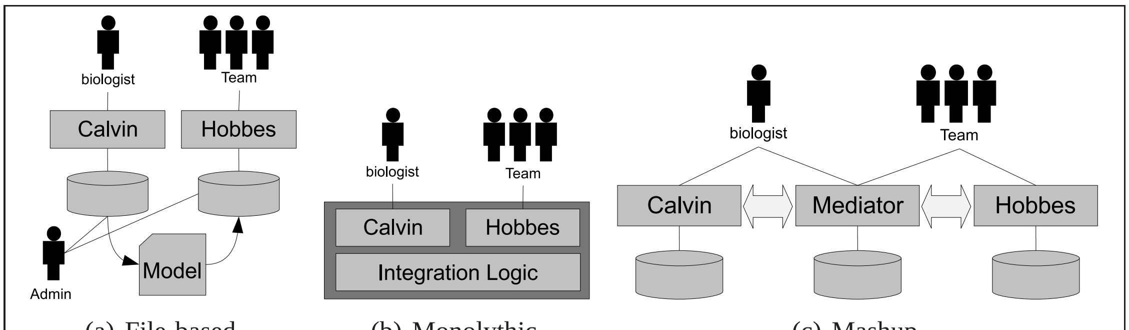


Figure 9.1: The interplay of CALVIN and HOBBES.

Web Service interfaces to control HOBBES sessions and to load and store CALVIN workflows (see Figure 9.2(c)). As Pautasso et al. [PZL08] point out, RESTful Web Services are well suited for the ad-hoc integration of applications on the Web. In this third scenario, Web Portals and Mashups respectively resemble service aggregators in the Service Oriented Architecture [WCL$^+$05], which combine web services to new functionality. The approach is also similar to the *mediator* pattern [GHJV95], which fosters loose coupling.



(a) File-based. (b) Monolythic. (c) Mashup.

Figure 9.2: Integration scenarios for CALVIN and HOBBES.

9.2 Workflow Archives in HOBBES, CALVIN, and ActiveBPEL

In order to integrate workflow applications, the workflow editors and execution engines have to use compatible file formats. The ActiveBPEL workflow execution engine uses zip archives with the extension `.bpr` for workflow deployment. The archives contain one BPEL workflow file, a WSDL file describing the interface of the workflow, and possibly several WSDL files describing the SOAP interfaces of the process partners.

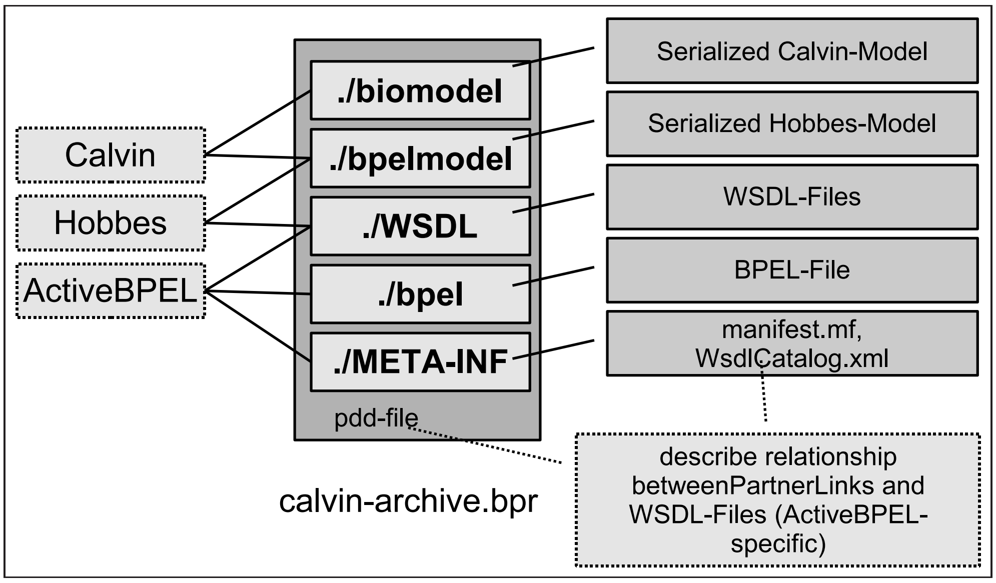


Figure 9.3: The CALVIN archive format is based on ZIP Files and held compatible with HOBBES and ActiveBPEL.

CALVIN and HOBBES workflow archives are implemented as sub-types of ActiveBPEL archives. HOBBES workflow archives are zip files, which besides the BPEL file and the referenced WSDL files contain a serialized HOBBES BPEL Object model. CALVIN workflow archives additionally contain a serialized CALVIN workflow model, effectively making CALVIN workflow archives a sub-type of HOBBES workflow archives.

Hence, CALVIN workflow archives can be processed by the HOBBES session interface. CALVIN workflow archives are the single resource type to which the CALVIN REST provides access.

9.3 Mashing up HOBBES and CALVIN

9.3.1 Implications of Mashup Architectures

We can distinguish two different types of mashup architectures. In the first case, a mashup consists only of client-side integration code, based on Javascript.This has the disadvantage of limiting the set of potential services to those of a single web site, as contemporary Web Browsers prevent cross-site scripting due to security reasons. The solution is to base mashups on server-side code generating Web pages. In contrast to the first solution, this enables the integration of arbitrary services, but has the disadvantage of additional complexity.

We can expect that HOBBES and CALVIN could be used in both settings. This yields the question, of how to pass workflow archive files to arbitrary clients.

Encoding of archives The Javascript language, which is the de-facto standard for client-side scripting on the Web, is defined in the ECMA-262 standard [ECM99]. While ECMA-262 does not define any data types for handling binary data, it provides sophisticated String manipulation facilities and does not pose any size limit on the String data type. We have thus decided to pass Workflow archives as Base64-encoded data [bas06]. This enables the integration of the interfaces in purely Javascript-based mashups, depending on the browser implementation of the Javascript String data type.

Publication of archives and sessions While no common notification standards exist in the Web Architecture, two families of standards for *Web feeds* are commonly accepted, i.e. *RSS* and *ATOM* [Say05]. A Web feed is a dynamically generated XML document indexing a set of Web resources, which can be accessed on a Web server.

9.3.2 The HOBBES and CALVIN REST Interfaces

To facilitate their integration on the Web 2.0, we have extended both HOBBES and CALVIN with RESTful interfaces. We have introduced the REST architecture style in Chapter 2.1.4 and discussed an example of a RESTful API in Chapter 7.1.

HOBBES provides a RESTful session management interface, which enables creating, deleting, and accessing sessions, but also gives access to the compiled workflow archives. Session creation parameters include the identifiers of the team leader and the team members. Optionally, HOBBES session can be created with initial workflow models passed as Base64-encoded parameters. Finally, a URL can optionally be passed that serves as a callback interface. In this case, every time the team leader decides to compile a session, a notification is passed to the callback URL.

If a HOBBES session is created with a HOBBES or CALVIN workflow archive as a parameter, the HOBBES server extracts and loads the BPEL Object Model and the corresponding WSDL files from the archive, and copies the files to a temporary work directory. Figure 9.4(b) shows a HOBBES session, which has been created from a CALVIN workflow model.

The workflow models are transfered as Base64-encoded compressed data via HTTP-POST for uploads and HTTP Get for downloads. Session access has been implemented via HTTP forwarding. The stored HOBBES workflow archives and active development sessions are published as RSS and ATOM feeds. The feeds consist of *entry* elements, which contain meta-information about the workflow archives and sessions respectively. In the case of the workflow archives an entry includes a database identifier, a URL for downloading the workflow archive, and its creation date. The session feed entries consists of the identifier of the corresponding session, a URL leading to the session client, a description containing the identifiers of the team leader and the team members, and the session creation date.

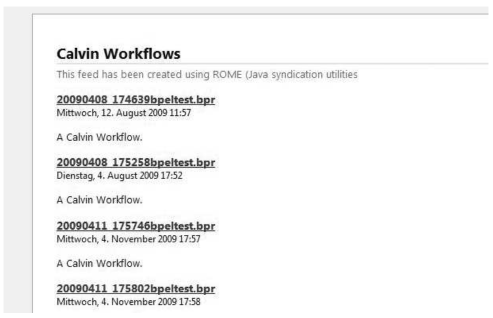
To access the CALVIN workflow base online, CALVIN provides a RESTful interface, which enables listing and downloading the available workflows, as well as posting new CALVIN workflow archives. HOBBES and CALVIN can thus be combined via Web Portals or *Mashups* which administer access to editing sessions and workflow models, thus enabling the dynamic steering of workflow development processes.

The list of deployed CALVIN workflow archives is published via RSS and ATOM feeds (see Figure 9.4(a)). Similar to the HOBBES archive feeds, the feed entries contain information the file name of the CALVIN archive, a URL for downloads, and the file creation date.

9.3.3 Implementation

The Figures 9.5 and 9.6 give an overview of the architectures of the HOBBES and CALVIN REST services respectively.

The HOBBES REST interface is integrated in the HOBBES web application to directly access the HOBBES session manager singleton. In Figure 9.5, the HOBBES REST interface is accessed by the URL path `hobbes/sessions`. Compiled work-



Calvin Workflows

This feed has been created using ROME (Java syndication utilities

20090408_174639bpeltest.bpr
Mittwoch, 12. August 2009 11:57

A Calvin Workflow.

20090408_175258bpeltest.bpr
Dienstag, 4. August 2009 17:52

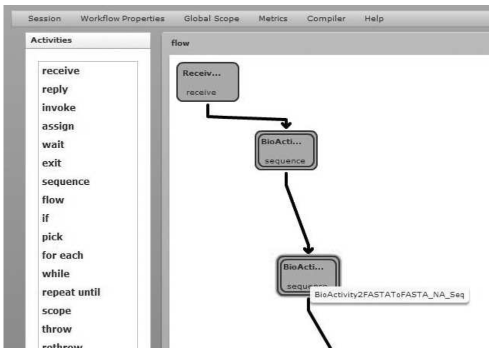
A Calvin Workflow.

20090411_175746bpeltest.bpr
Mittwoch, 4. November 2009 17:57

A Calvin Workflow.

20090411_175802bpeltest.bpr
Mittwoch, 4. November 2009 17:58

(a) A CALVIN Atom feed in Firefox.



(b) A CALVIN model in HOBBES.

Figure 9.4: A CALVIN feed and a CALVIN model in HOBBES. The left Figure depicts an Atom feed of CALVIN workflows archives as shown in Mozilla Firefox, while the right Figure shows a CALVIN generated BPEL model that has been loaded in HOBBES. Note that the activity names have been generated by CALVIN without involving the user and that the layout of the top level activities has been generated from the layout of the original CALVIN model.

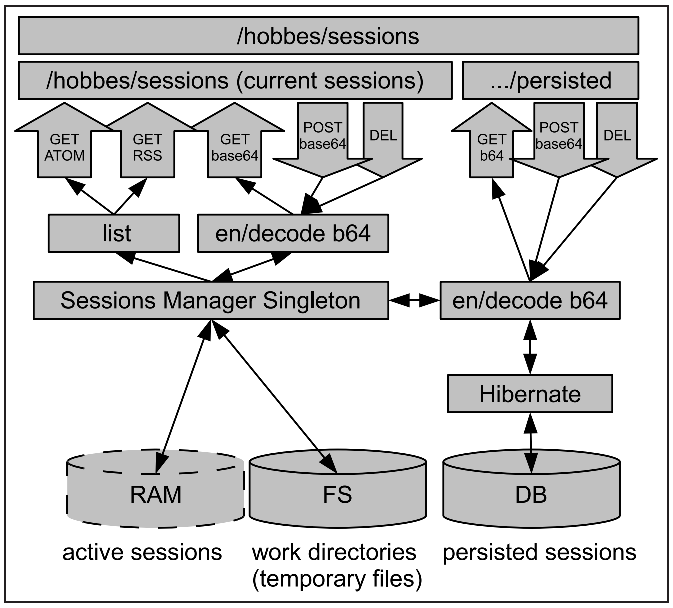


Figure 9.5: The HOBBES REST interface.

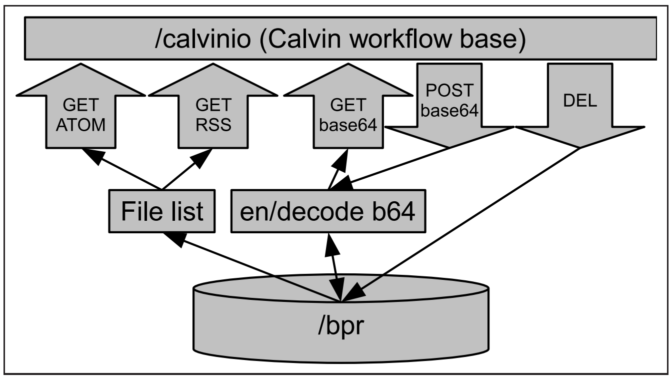


Figure 9.6: The CALVIN REST interface.

flow archives are stored in a database using the Hibernate[1] object-relational mapper. The Session Manager singleton controls the set of active sessions in memory, and assigns work directories for temporary files to the sessions.

The CALVIN REST interface is implemented as a Web Application (on the URL path `/calvinio` in Figure 9.6) which can be deployed independently from CALVIN. It consists of two servlets, one providing the CRUD functionality to access the workflow model resources, and the other generating the RSS and ATOM feeds respectively. The calvinio system accesses the ActiveBPEL deployment directory via the standard Java file IO library to generate the list of deployed CALVIN workflows.

In both cases, the Base64 en/decoding is implemented using the Apache Commons Codec library.[2] The feeds are generated by the ROME library[3] which enables changing the feed format from the default RSS 2.0[4] to arbitrary RSS or ATOM versions via a request parameter.

[1]`http://www.hibernate.org`, last accessed 17/11/2009.
[2]`http://commons.apache.org/codec/`, last accessed 17/11/2009.
[3]`https://rome.dev.java.net/`, last accessed 17/11/2009.
[4]`http://www.rssboard.org/rss-specification` format, last accessed 19/11/2009.

Chapter 10

Conclusion

... Science is the knowledge of consequences, and dependence of one fact upon another...

(Thomas Hobbes, Leviathan, Pt. I, Ch. 5)

10.1 Results

Due to the transition of enterprise IT systems to Cloud Computing, it is likely that more and more software development systems will be Web-based and Cloud-based respectively. Moreover, permanent availability of Internet access makes Web-based collaborative development environments desirable. This dissertation has investigated the intricacies of providing workflow development systems as collaborative Rich Internet Applications, thereby addressing the opportunities and challenges of Web-based collaborative development environments in general. The investigation has yielded results in terms of concepts and their implementation in prototype systems, which we will now summarize.

In Chapter 1.5 we have stated three goals:

1. To provide a theoretical basis for collaborative workflow development by investigating requirements, as well as suitable software processes and software metrics.
2. To investigate the relationship between process modeling and scientific workflows.
3. To show the applicability of modern Web technology for collaborative and scientific workflow development, thus proving the applicability of the Web 2.0 to software development in general.

The first goal is addressed in the Chapters 3, 4, and 5. A suitable software process for collaborative workflow development is presented in Chapter 3 and a set of corresponding workflow metrics in Chapter 5. The relationship between process modeling and the life cycle of scientific workflows is also addressed in Chapter 3, and Chapter 8 presents a corresponding scientific workflow system as a proof of concept. In the Chapters 6, 7, 8, and 9 we have given a detailed account of the implementation of collaborative and scientific workflow management systems on the Web 2.0.

- In contrast to single user IDEs, collaborative development systems demand a closer look at application specific requirements, at development processes, and at team structures. Based on the processes presented in Chapter 3, we have devised requirements for corresponding development tools and discussed approaches to addressing these requirements. In particular, we have shown the applicability of a set of workflow metrics for guiding the development process. We have analyzed the set of workflow metrics using a collection of desirable properties and devised guidelines for the application of the metrics in the form of *code smells*.

- As we can see at the example of HOBBES, providing sophisticated development environments as Rich Internet Applications is possible. Moreover, we have shown the applicability of modern Web technology for implementing collaborative development environments in general. HOBBES goes beyond the limits of traditional IDEs in its support of close realtime cooperation between team members. In contrast to traditional collaborative text, graphics or workflow editors (e.g. [sub, SK05, FSF$^+$06, PHL09b, PHL09a]), HOBBES more strongly supports structured development processes. While earlier collaborative workflow editors are based on traditional desktop technology [SK05, FSF$^+$06, PHL09b, PHL09a], HOBBES may be the first example of a new generation of Web-based collaborative workflow development systems.

- As workflows directly reflect business processes or virtual scientific experiments, the question of closely integrating domain experts in development processes becomes more urgent. The life science workflow system CALVIN is an example of a domain-specific workflow system targeting end users, which applies the same technology stack as HOBBES, but differs in purpose and capabilities. At the example of HOBBES and CALVIN we have discussed the integration of a domain-specific workflow system with a general collaborative workflow development system.

Nearly two years after the submission of our first paper on HOBBES to IEEE CCGrid'08 we find evidence of research on Web-based collaborative workflow development in the industry [HB08]. In September 2009 SAP presented the promotion video of a new research prototype of a collaborative, Web-based BPMN editor [Dre09], while earlier this year, Software AG introduced a project on collaborative business process management with social software [AG, WD09]. In a blog post entitled "The time is right for collaborative BPM," Ward-Dutton [WD09] claims that "it's time for BPM tools and working environments that foster and support collaboration between disparate groups of stakeholders." In this dissertation, we have discussed principles and processes of collaborative workflow development and have presented corresponding implementations, thus proving that the time is right, indeed.

10.2 Publications

In the context of this dissertation project, we have published a number of scientific articles, which are now briefly presented.

10.2.1 Journal Publications

- M. Held and W. Blochinger. Structured Collaborative Workflow Design. In *Future Generation Computer Systems - The International Journal of Grid Computing: Theory, Methods and Applications*, 25(6):628-653, http://dx.doi.org/10.1016/j.future.2008.12.005.
  Abstract: *Workflow design is often an effort of distributed and heterogeneous teams, thus making tool support for collaboration a necessity. We present a novel concept of collaborative workflow design which combines cooperation and workflow model analysis. Workflow analysis is simplified using workflow metrics, which help identifying problematic aspects of the workflow model. Our findings are implemented in a collaborative workflow design system, which is easily accessible on the Web, but provides a desktop-like user experience.*

- S. Schulz, W. Blochinger, M. Held, and C. Dangelmayr. COHESION - A Microkernel Based Desktop Grid Platform for Irregular Task-Parallel Applications. In *Future Generation Computer Systems - The International Journal of Grid Computing: Theory, Methods and Applications*, 24(5):354-370, 2008.
  Abstract: *We present COHESION, a novel approach to Desktop Grid Computing. A major design goal of COHESION is to enable advanced parallel programming models and application specific frameworks. We focus on methods for irregularly structured task-parallel problems, which require fully dynamic*

problem decomposition. COHESION overcomes limitations of classical Desktop Grid platforms by employing peer-to-peer principles and a flexible system architecture based on a microkernel approach. Arbitrary modules can be dynamically loaded to replace default functionality, resulting in a platform that can easily adapt to application specific requirements.We discuss two representative example applications and report on the results of performance experiments that especially consider the high volatility of resources prevailing in a Desktop Grid.

10.2.2 Conference Publications

- M. Held and W. Blochinger and M. Werning. E-Biology Workflows with Calvin. In *Proceedings of the 10th International Conference on Web Information Systems Engineering (WISE'09)*, October 5-7 2009, Poznan, Poland, Springer Lecture Notes in Computer Science.
  Abstract: *Web portals enable sharing, execution and monitoring of scientific workflows, but usually depend on external development systems, with notations, which strive to support general workflows, but are still too complex for every-day use by biologists. The distinction between web-based and non-web based tools is likely to further irritate users. We extend our work on collaborative workflow design, by introducing a web-based scientific workflow system which enables easy-to-use semantic service composition with a domain specific workflow notation.*

- M. Held and M. Günter. Collaborative Web Service Interface Design on the Web 2.0. In *Proceedings of the First International Conference on Social Informatics*, June 22-24 2009, Warsaw, Poland, IEEE Computer Society.
  Abstract: *In the Service Oriented Architecture, Web Services are the standard means of providing functionality to a remote partner. The interfaces of these Web Services are defined in the Web Services Description Language (WSDL). If two or more partners are involved in the development of a Web Service, a need to collaboratively edit service interface definitions can arise in order to increase productivity and to prevent ambiguities. In this paper, we investigate the process of collaborative interface design and present a web-based, collaborative WSDL development system, which can be integrated with a collaborative workflow development system.*

- M. Held and D. Lehle. Augmenting Collaborative Web Applications with RESTful Video Conferencing. In *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Networks*, February 16 – 18, 2009, Innsbruck, Austria.
  Abstract: *Although Video Conferencing is a useful addition to many collaborative applications, it is not supported by most contemporary collaborative web applications. In this paper we describe the integration of video conferencing into the collaborative workflow design system HOBBES and present an easy-to-use RESTful API for integrating video conferencing in web applications.*

- M. Held and W. Blochinger. Collaborative BPEL Design with a Rich Internet Application. In *Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid'08)*, Lyon, France, 19-22 May 2008.
  Abstract: *Workflow design is often an effort of distributed and inhomogeneous teams. We present a collaborative workflow design tool, which is implemented as a Rich Internet Application. The tool provides a desktop like user experience, but can easily be embedded into web sites. It enables synchronous as well as asynchronous cooperation of design team members and encourages a well coordinated design process.*

10.2.3 Other

- B. Albrecht, P. Effinger, M. Held, M. Kaufmann and S. Kottler. Visualization of Complex BPEL Models. In Proceedings of the 17th International Conference on Graph Drawing (GD'09), Springer LNCS, poster, in press.

Appendix A

Listings

A.1 Listings of Examples from Chapter 5

A.1.1 Example Process 1

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
BPEL-Example no 1
-->
<bpel:process xmlns:BioActivity1="http://biomoby.⤸
                                 org/AGILocusList-⤸
                                 arabidopsis.org" xmlns:⤸
                                 BioActivity2="http:⤸
                                 //biomoby.⤸
                                 org/getNASC_codebyAGI_locus-⤸
                                 arabidopsis.info" xmlns:⤸
                                 BioActivity3="http:⤸
                                 //biomoby.⤸
                                 org/getEMBLaccessionByNASCst⤸
                                 ockCode-arabidopsis.info"⤸
                                 xmlns:BioActivity4="http:⤸
                                 //biomoby.⤸
                                 org/MOBYSHoundGetGenBankWhat⤸
                                 everSequence-bioinfo.⤸
                                 icapture.ubc.ca" xmlns:aecf-⤸
                                 xmlstring="http://docs.⤸
                                 active-endpoints.⤸
                                 com/activebpel/sample/custom⤸
                                 Function/2006/09" xmlns:⤸
                                 bpel="http://docs.oasis-⤸
                                 open.org/wsbpel/2.⤸
                                 0/process/executable" xmlns: ⤸
                                 bpelx="http://schemas. ⤸
                                 oracle.com/bpel/extension" ⤸
                                 xmlns:bpws="http://schemas. ⤸
                                 xmlsoap. ⤸
                                 org/ws/2003/03/business- ⤸
                                 process/" xmlns:sns="http: ⤸
                                 //org.apache.axis2/xsd" ⤸
                                 xmlns:tueb="http: ⤸
                                 //tuebigrid.sample.org" ⤸
                                 xmlns:xsd="http://www.w3. ⤸
                                 org/2001/XMLSchema" ⤸
                                 name="AGILocusList_to_MOBYSH ⤸
                                 oundGetGenBankWhateverSequen ⤸
                                 ce" targetNamespace="http: ⤸
                                 //tuebigrid.sample.org">
<bpel:import importType="http://schemas.xmlsoap.org/wsdl/" ⤸
                        location=".. ⤸
                        /wsdl/AAgetNASC_codebyAGI_locus- ⤸
                        arabidopsis.infoAA.wsdl" ⤸
                        namespace="http://biomoby. ⤸
                        org/getNASC_codebyAGI_locus- ⤸
                        arabidopsis.info"/>
<bpel:import importType="http://schemas.xmlsoap.org/wsdl/" ⤸
                        location=".. ⤸
                        /wsdl/AAMOBYSHoundGetGenBankWhateverS ⤸
                        equence-bioinfo.icapture.ubc.caAA. ⤸
                        wsdl" namespace="http://biomoby. ⤸
                        org/MOBYSHoundGetGenBankWhateverSeque ⤸
                        nce-bioinfo.icapture.ubc.ca"/>
<bpel:import importType="http://schemas.xmlsoap.org/wsdl/" ⤸
                        location=".. ⤸
                        /wsdl/AAgetEMBLaccessionByNASCstockCo ⤸
                        de-arabidopsis.infoAA.wsdl" ⤸
                        namespace="http://biomoby. ⤸
                        org/getEMBLaccessionByNASCstockCode- ⤸
                        arabidopsis.info"/>
<bpel:import importType="http://schemas.xmlsoap.org/wsdl/" ⤸
                        location="../wsdl/AAAGILocusList- ⤸
                        arabidopsis.orgAA.wsdl" ⤸
```

```
                              namespace="http://biomoby. ⤸
                              org/AGILocusList-arabidopsis.org"/>
<bpel:import importType="http://schemas.xmlsoap.org/wsdl/" ⤸
                              location="../wsdl/BioProcess.wsdl" ⤸
                              namespace="http://tuebigrid.sample. ⤸
                              org"/>
<bpel:partnerLinks>
<bpel:partnerLink name="AGILocusListPL" ⤸
                             partnerLinkType="BioActivity1: ⤸
                             AGILocusListServicePLT" ⤸
                             partnerRole="AGILocusListService"/>
<bpel:partnerLink myRole="BioProcess" name="BPELMyServicePL" ⤸
                               partnerLinkType="tueb: ⤸
                               BioProcessPLT"/>
<bpel:partnerLink name="MOBYSHoundGetGenBankWhateverSequenceP ⤸
                             L" partnerLinkType="BioActivity4: ⤸
                             MOBYSHoundGetGenBankWhateverSequenceSe ⤸
                             rvicePLT" ⤸
                             partnerRole="MOBYSHoundGetGenBankWhate ⤸
                             verSequenceService"/>
<bpel:partnerLink name="getNASC_codebyAGI_locusPL" ⤸
                             partnerLinkType="BioActivity2: ⤸
                             getNASC_codebyAGI_locusServicePLT" ⤸
                             partnerRole="getNASC_codebyAGI_locusSe ⤸
                             rvice"/>
<bpel:partnerLink name="getEMBLaccessionByNASCstockCodePL" ⤸
                             partnerLinkType="BioActivity3: ⤸
                             getEMBLaccessionByNASCstockCodeService ⤸
                             PLT" ⤸
                             partnerRole="getEMBLaccessionByNASCsto ⤸
                             ckCodeService"/>
</bpel:partnerLinks>
<bpel:variables>
<bpel:variable messageType="BioActivity3: ⤸
                                 getEMBLaccessionByNASCstockCodeOut ⤸
                                 put" ⤸
                                 name="getEMBLaccessionByNASCstockC ⤸
                                 odeOutput"/>
<bpel:variable messageType="BioActivity1:AGILocusListOutput" ⤸
                                 name="AGILocusListOutput"/>
```

```
<bpel:variable name="tmp"  type="xsd:string"/>
<bpel:variable messageType="BioActivity1:AGILocusListInput" ⤸
                               name="AGILocusListInput"/>
<bpel:variable messageType="BioActivity3: ⤸
                               getEMBLaccessionByNASCstockCodeInp ⤸
                               ut" ⤸
                               name="getEMBLaccessionByNASCstockC ⤸
                               odeInput"/>
<bpel:variable messageType="tueb:callBioProcessInputMessage" ⤸
                               name="inputMessage"/>
<bpel:variable messageType="BioActivity2: ⤸
                               getNASC_codebyAGI_locusOutput" ⤸
                               name="getNASC_codebyAGI_locusOutpu ⤸
                               t"/>
<bpel:variable messageType="BioActivity4: ⤸
                               MOBYSHoundGetGenBankWhateverSequen ⤸
                               ceInput" ⤸
                               name="MOBYSHoundGetGenBankWhatever ⤸
                               SequenceInput"/>
<bpel:variable messageType="BioActivity2: ⤸
                               getNASC_codebyAGI_locusInput" ⤸
                               name="getNASC_codebyAGI_locusInput ⤸
                               "/>
<bpel:variable messageType="tueb: ⤸
                               callBioProcessOutputMessage" ⤸
                               name="outputMessage"/>
<bpel:variable messageType="BioActivity4: ⤸
                               MOBYSHoundGetGenBankWhateverSequen ⤸
                               ceOutput" ⤸
                               name="MOBYSHoundGetGenBankWhatever ⤸
                               SequenceOutput"/>
</bpel:variables>

<bpel:flow>

<bpel:links>
<bpel:link name="Link1"/>
<bpel:link name="Link2"/>
<bpel:link name="Link3"/>
<bpel:link name="StartingLink"/>
```

```
<bpel:link name="EndingLink"/>
</bpel:links>

<bpel:receive createInstance="yes" ⤸
                                        name="ReceiveBioWFUserRequest" ⤸
                                        operation="callBioProcess" ⤸
                                        partnerLink="BPELMyServicePL" ⤸
                                        portType="tueb: ⤸
                                        BioProcessPortType" ⤸
                                        variable="inputMessage">
<bpel:sources>
<bpel:source linkName="Link1"/>
</bpel:sources>
</bpel:receive>

<bpel:sequence>

<bpel:targets>
<bpel:target linkName="Link1"/>
</bpel:targets>

<bpel:sources>
<bpel:source linkName="Link2"/>
</bpel:sources>

<bpel:assign name="AS_Beginning">
<bpel:copy>
<bpel:from part="BioActivity1" variable="inputMessage"/>
<bpel:to variable="tmp" />
</bpel:copy>
</bpel:assign>

<bpel:assign name="AS_CUSTFKT">
<bpel:copy>
<bpel:from>aecf-xmlstring:xmlStringToElement($tmp, ⤸
                                                          "AGI_LocusCode") ⤸
                                                          </bpel:from>
<bpel:to part="data" ⤸
                    variable="getNASC_codebyAGI_locusInput"/>
</bpel:copy>
```

```
</bpel:assign>

<bpel:assign name="AS_in2End">
<bpel:copy>
<bpel:from part="data" ⤸
                    variable="getNASC_codebyAGI_locusInput"/>
<bpel:to part="BioActivity2I" variable="outputMessage"/>
</bpel:copy>
</bpel:assign>

<bpel:invoke inputVariable="getNASC_codebyAGI_locusInput" ⤸
                                        name="getNASC_codebyAGI_locus" ⤸
                                        operation="getNASC_codebyAGI_locus ⤸
                                        " ⤸
                                        outputVariable="getNASC_codebyAGI_ ⤸
                                        locusOutput" ⤸
                                        partnerLink="getNASC_codebyAGI_loc ⤸
                                        usPL" portType="BioActivity2: ⤸
                                        getNASC_codebyAGI_locusPortType"/>

<bpel:assign>
<bpel:copy>
<bpel:from part="body" ⤸
                    variable="getNASC_codebyAGI_locusOutput"/>
<bpel:to variable="tmp"/>
</bpel:copy>
</bpel:assign>

</bpel:sequence>

<bpel:sequence>

<bpel:targets>
<bpel:target linkName="Link2"/>
</bpel:targets>

<bpel:sources>
<bpel:source linkName="Link3"/>
</bpel:sources>

```

```
<bpel:assign name="AS_Int2tmp">
<bpel:copy>
<bpel:from part="body" ⤸
                variable="getNASC_codebyAGI_locusOutput"/>
<bpel:to variable="tmp"/>
</bpel:copy>
</bpel:assign>

<bpel:assign name="AS_CUSTFKT">
<bpel:copy>
<bpel:from>aecf-xmlstring:xmlStringToElement($tmp, ⤸
                                             "NASC_code")< ⤸
                                             /bpel:from>
<bpel:to part="data" ⤸
              variable="getEMBLaccessionByNASCstockCodeInput" ⤸
              />
</bpel:copy>
</bpel:assign>

<bpel:assign name="AS_in2End">
<bpel:copy>
<bpel:from part="data" ⤸
                variable="getEMBLaccessionByNASCstockCodeInpu ⤸
                t"/>
<bpel:to part="BioActivity3I" variable="outputMessage"/>
</bpel:copy>
</bpel:assign>

<bpel:invoke inputVariable="getEMBLaccessionByNASCstockCodeIn ⤸
                           put" ⤸
                           name="getEMBLaccessionByNASCstockC ⤸
                           ode" ⤸
                           operation="getEMBLaccessionByNASCs ⤸
                           tockCode" ⤸
                           outputVariable="getEMBLaccessionBy ⤸
                           NASCstockCodeOutput" ⤸
                           partnerLink="getEMBLaccessionByNAS ⤸
                           CstockCodePL" ⤸
                           portType="BioActivity3: ⤸
                           getEMBLaccessionByNASCstockCodePor ⤸
```

```
                                tType"/>

</bpel:sequence>

<bpel:sequence>

<bpel:targets>
<bpel:target linkName="Link3"/>
</bpel:targets>

<bpel:sources>
<bpel:source linkName="EndingLink"/>
</bpel:sources>

<bpel:assign name="AS_Int2tmp">
<bpel:copy>
<bpel:from part="body" ⤸
                    variable="getEMBLaccessionByNASCstockCodeOutp ⤸
                    ut"/>
<bpel:to variable="tmp"/>
</bpel:copy>
</bpel:assign>

<bpel:assign name="AS_CUSTFKT">
<bpel:copy>
<bpel:from>aecf-xmlstring:xmlStringToElement($tmp, ⤸
                                                      "identifier")< ⤸
                                                      /bpel:from>
<bpel:to part="data" ⤸
                  variable="MOBYSHoundGetGenBankWhateverSequenceI ⤸
                  nput"/>
</bpel:copy>
</bpel:assign>

<bpel:assign name="AS_in2End">
<bpel:copy>
<bpel:from part="data" ⤸
                    variable="MOBYSHoundGetGenBankWhateverSequenc ⤸
                    eInput"/>
<bpel:to part="BioActivity4I" variable="outputMessage"/>
```

```
</bpel:copy>
</bpel:assign>

<bpel:invoke inputVariable="MOBYSHoundGetGenBankWhateverSeque ⤸
                           nceInput" ⤸
                           name="MOBYSHoundGetGenBankWhatever ⤸
                           Sequence" ⤸
                           operation="MOBYSHoundGetGenBankWha ⤸
                           teverSequence" ⤸
                           outputVariable="MOBYSHoundGetGenBa ⤸
                           nkWhateverSequenceOutput" ⤸
                           partnerLink="MOBYSHoundGetGenBankW ⤸
                           hateverSequencePL" ⤸
                           portType="BioActivity4: ⤸
                           MOBYSHoundGetGenBankWhateverSequen ⤸
                           cePortType"/>

<bpel:assign name="AS_Ending">
<bpel:copy>
<bpel:from part="body" ⤸
                variable="MOBYSHoundGetGenBankWhateverSequenc ⤸
                eOutput"/>
<bpel:to part="BioActivity4" variable="outputMessage"/>
</bpel:copy>
</bpel:assign>

</bpel:sequence>

<bpel:reply operation="callBioProcess" ⤸
                      partnerLink="BPELMyServicePL" ⤸
                      portType="tueb:BioProcessPortType" ⤸
                      variable="outputMessage">
<bpel:targets>
<bpel:target linkName="EndingLink"/>
</bpel:targets>
</bpel:reply>

</bpel:flow>

```

```
</bpel:process>
```

A.1.2 Example Process 2

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
BPEL Example No 2
-->
<bpel:process xmlns:BioActivity1="http://biomoby. ⤸
                                 org/AGILocusList- ⤸
                                 arabidopsis.org" xmlns: ⤸
                                 BioActivity2="http: ⤸
                                 //biomoby. ⤸
                                 org/getNASC_codebyAGI_locus- ⤸
                                 arabidopsis.info" xmlns: ⤸
                                 BioActivity3="http: ⤸
                                 //biomoby. ⤸
                                 org/getEMBLaccessionByNASCst ⤸
                                 ockCode-arabidopsis.info" ⤸
                                 xmlns:BioActivity4="http: ⤸
                                 //biomoby. ⤸
                                 org/MOBYSHoundGetGenBankWhat ⤸
                                 everSequence-bioinfo. ⤸
                                 icapture.ubc.ca" xmlns:aecf- ⤸
                                 xmlstring="http://docs. ⤸
                                 active-endpoints. ⤸
                                 com/activebpel/sample/custom ⤸
                                 Function/2006/09" xmlns: ⤸
                                 bpel="http://docs.oasis- ⤸
                                 open.org/wsbpel/2. ⤸
                                 0/process/executable" xmlns: ⤸
                                 bpelx="http://schemas. ⤸
                                 oracle.com/bpel/extension" ⤸
                                 xmlns:bpws="http://schemas. ⤸
                                 xmlsoap. ⤸
                                 org/ws/2003/03/business- ⤸
                                 process/" xmlns:sns="http: ⤸
                                 //org.apache.axis2/xsd" ⤸
                                 xmlns:tueb="http: ⤸
                                 //tuebigrid.sample.org" ⤸
                                 xmlns:xsd="http://www.w3. ⤸
                                 org/2001/XMLSchema" ⤸
                                 name="AGILocusList_to_MOBYSH ⤸
                                 oundGetGenBankWhateverSequen ⤸
                                 ce" targetNamespace="http: ⤸
                                 //tuebigrid.sample.org">
<bpel:import importType="http://schemas.xmlsoap.org/wsdl/" ⤸
                        location=".. ⤸
                        /wsdl/AAgetNASC_codebyAGI_locus- ⤸
                        arabidopsis.infoAA.wsdl" ⤸
                        namespace="http://biomoby. ⤸
                        org/getNASC_codebyAGI_locus- ⤸
                        arabidopsis.info"/>
<bpel:import importType="http://schemas.xmlsoap.org/wsdl/" ⤸
                        location=".. ⤸
                        /wsdl/AAMOBYSHoundGetGenBankWhateverS ⤸
                        equence-bioinfo.icapture.ubc.caAA. ⤸
                        wsdl" namespace="http://biomoby. ⤸
                        org/MOBYSHoundGetGenBankWhateverSeque ⤸
                        nce-bioinfo.icapture.ubc.ca"/>
<bpel:import importType="http://schemas.xmlsoap.org/wsdl/" ⤸
                        location=".. ⤸
                        /wsdl/AAgetEMBLaccessionByNASCstockCo ⤸
                        de-arabidopsis.infoAA.wsdl" ⤸
                        namespace="http://biomoby. ⤸
                        org/getEMBLaccessionByNASCstockCode- ⤸
                        arabidopsis.info"/>
<bpel:import importType="http://schemas.xmlsoap.org/wsdl/" ⤸
                        location="../wsdl/AAAGILocusList- ⤸
                        arabidopsis.orgAA.wsdl" ⤸
                        namespace="http://biomoby. ⤸
                        org/AGILocusList-arabidopsis.org"/>
<bpel:import importType="http://schemas.xmlsoap.org/wsdl/" ⤸
                        location="../wsdl/BioProcess.wsdl" ⤸
                        namespace="http://tuebigrid.sample. ⤸
                        org"/>
<bpel:partnerLinks>
<bpel:partnerLink name="AGILocusListPL" ⤸
                       partnerLinkType="BioActivity1: ⤸
                       AGILocusListServicePLT" ⤸
                       partnerRole="AGILocusListService"/>
<bpel:partnerLink myRole="BioProcess" name="BPELMyServicePL" ⤸
                         partnerLinkType="tueb: ⤸
                         BioProcessPLT"/>
<bpel:partnerLink name="MOBYSHoundGetGenBankWhateverSequenceP ⤸
                       L" partnerLinkType="BioActivity4: ⤸
                       MOBYSHoundGetGenBankWhateverSequenceSe ⤸
                       rvicePLT" ⤸
                       partnerRole="MOBYSHoundGetGenBankWhate ⤸
                       verSequenceService"/>
<bpel:partnerLink name="getNASC_codebyAGI_locusPL" ⤸
                       partnerLinkType="BioActivity2: ⤸
                       getNASC_codebyAGI_locusServicePLT" ⤸
                       partnerRole="getNASC_codebyAGI_locusSe ⤸
                       rvice"/>
<bpel:partnerLink name="getEMBLaccessionByNASCstockCodePL" ⤸
                       partnerLinkType="BioActivity3: ⤸
                       getEMBLaccessionByNASCstockCodeService ⤸
                       PLT" ⤸
                       partnerRole="getEMBLaccessionByNASCsto ⤸
                       ckCodeService"/>
</bpel:partnerLinks>
<bpel:variables>
<bpel:variable messageType="BioActivity3: ⤸
                           getEMBLaccessionByNASCstockCodeOut ⤸
                           put" ⤸
                           name="getEMBLaccessionByNASCstockC ⤸
                           odeOutput"/>
<bpel:variable messageType="BioActivity1:AGILocusListOutput" ⤸
                           name="AGILocusListOutput"/>
<bpel:variable name="tmp" type="xsd:string"/>
<bpel:variable messageType="BioActivity1:AGILocusListInput" ⤸
                           name="AGILocusListInput"/>
<bpel:variable messageType="BioActivity3: ⤸
                           getEMBLaccessionByNASCstockCodeInp ⤸
                           ut" ⤸
                           name="getEMBLaccessionByNASCstockC ⤸
                           odeInput"/>
<bpel:variable messageType="tueb:callBioProcessInputMessage" ⤸
```

```
                                 name="inputMessage"/>
<bpel:variable messageType="BioActivity2: ⤸
                                 getNASC_codebyAGI_locusOutput" ⤸
                                 name="getNASC_codebyAGI_locusOutpu ⤸
                                 t"/>
<bpel:variable messageType="BioActivity4: ⤸
                                 MOBYSHoundGetGenBankWhateverSequen ⤸
                                 ceInput" ⤸
                                 name="MOBYSHoundGetGenBankWhatever ⤸
                                 SequenceInput"/>
<bpel:variable messageType="BioActivity2: ⤸
                                 getNASC_codebyAGI_locusInput" ⤸
                                 name="getNASC_codebyAGI_locusInput ⤸
                                 "/>
<bpel:variable messageType="tueb: ⤸
                                 callBioProcessOutputMessage" ⤸
                                 name="outputMessage"/>
<bpel:variable messageType="BioActivity4: ⤸
                                 MOBYSHoundGetGenBankWhateverSequen ⤸
                                 ceOutput" ⤸
                                 name="MOBYSHoundGetGenBankWhatever ⤸
                                 SequenceOutput"/>
</bpel:variables>
<bpel:flow>
<bpel:links>
<bpel:link name="Link1"/>
<bpel:link name="Link2"/>
<bpel:link name="Link3"/>
<bpel:link name="StartingLink"/>
<bpel:link name="EndingLink"/>
</bpel:links>

<bpel:receive createInstance="yes" ⤸
                                   name="ReceiveBioWFUserRequest" ⤸
                                   operation="callBioProcess" ⤸
                                   partnerLink="BPELMyServicePL" ⤸
                                   portType="tueb: ⤸
                                   BioProcessPortType" ⤸
                                   variable="inputMessage">
<bpel:sources>
```

```
<bpel:source linkName="Link1"/>
</bpel:sources>
</bpel:receive>

<bpel:sequence>

<bpel:targets>
<bpel:target linkName="Link1"/>
</bpel:targets>

<bpel:sources>
<bpel:source linkName="Link2"/>
</bpel:sources>

<bpel:assign name="AS_Beginning">
<bpel:copy>
<bpel:from part="BioActivity1" variable="inputMessage"/>
<bpel:to variable="tmp" />
</bpel:copy>
<bpel:copy>
<bpel:from>aecf-xmlstring:xmlStringToElement($tmp, ⤸
                                                  "AGI_LocusCode") ⤸
                                                  </bpel:from>
<bpel:to part="data" ⤸
                    variable="getNASC_codebyAGI_locusInput"/>
</bpel:copy>
<bpel:copy>
<bpel:from part="data" ⤸
                        variable="getNASC_codebyAGI_locusInput"/>
<bpel:to part="BioActivity2I" variable="outputMessage"/>
</bpel:copy>
</bpel:assign>

<bpel:invoke inputVariable="getNASC_codebyAGI_locusInput" ⤸
                                   name="getNASC_codebyAGI_locus" ⤸
                                   operation="getNASC_codebyAGI_locus ⤸
                                   " ⤸
                                   outputVariable="getNASC_codebyAGI_ ⤸
                                   locusOutput" ⤸
                                   partnerLink="getNASC_codebyAGI_loc ⤸
```

```
                           usPL" portType="BioActivity2:⤸
                           getNASC_codebyAGI_locusPortType"/>

</bpel:sequence>

<bpel:sequence>

<bpel:targets>
<bpel:target linkName="Link2"/>
</bpel:targets>

<bpel:sources>
<bpel:source linkName="Link3"/>
</bpel:sources>

<bpel:assign name="AS_Int2tmp">
<bpel:copy>
<bpel:from part="body" ⤸
                variable="getNASC_codebyAGI_locusOutput"/>
<bpel:to variable="tmp"/>
</bpel:copy>
<bpel:copy>
<bpel:from>aecf-xmlstring:xmlStringToElement($tmp, ⤸
                                             "NASC_code")<⤸
                                             /bpel:from>
<bpel:to part="data" ⤸
              variable="getEMBLaccessionByNASCstockCodeInput" ⤸
              />
</bpel:copy>
<bpel:copy>
<bpel:from part="data" ⤸
                variable="getEMBLaccessionByNASCstockCodeInpu⤸
                t"/>
<bpel:to part="BioActivity3I" variable="outputMessage"/>
</bpel:copy>
</bpel:assign>

<bpel:invoke inputVariable="getEMBLaccessionByNASCstockCodeIn⤸
                           put" ⤸
                           name="getEMBLaccessionByNASCstockC⤸
```

```
                                    ode" ⤸
                                    operation="getEMBLaccessionByNASCs ⤸
                                    tockCode" ⤸
                                    outputVariable="getEMBLaccessionBy ⤸
                                    NASCstockCodeOutput" ⤸
                                    partnerLink="getEMBLaccessionByNAS ⤸
                                    CstockCodePL" ⤸
                                    portType="BioActivity3: ⤸
                                    getEMBLaccessionByNASCstockCodePor ⤸
                                    tType"/>

</bpel:sequence>

<bpel:sequence>

<bpel:targets>
<bpel:target linkName="Link3"/>
</bpel:targets>

<bpel:sources>
<bpel:source linkName="EndingLink"/>
</bpel:sources>

<bpel:assign name="AS_Int2tmp">
<bpel:copy>
<bpel:from part="body" ⤸
                     variable="getEMBLaccessionByNASCstockCodeOutp ⤸
                     ut"/>
<bpel:to variable="tmp"/>
</bpel:copy>
<bpel:copy>
<bpel:from>aecf-xmlstring:xmlStringToElement($tmp, ⤸
                                                 "identifier")< ⤸
                                                 /bpel:from>
<bpel:to part="data" ⤸
                   variable="MOBYSHoundGetGenBankWhateverSequenceI ⤸
                   nput"/>
</bpel:copy>
<bpel:copy>
<bpel:from part="data" ⤸
```

```
                variable="MOBYSHoundGetGenBankWhateverSequenc ⤸
                eInput"/>
<bpel:to part="BioActivity4I" variable="outputMessage"/>
</bpel:copy>
</bpel:assign>

<bpel:invoke inputVariable="MOBYSHoundGetGenBankWhateverSeque ⤸
                           nceInput" ⤸
                           name="MOBYSHoundGetGenBankWhatever ⤸
                           Sequence" ⤸
                           operation="MOBYSHoundGetGenBankWha ⤸
                           teverSequence" ⤸
                           outputVariable="MOBYSHoundGetGenBa ⤸
                           nkWhateverSequenceOutput" ⤸
                           partnerLink="MOBYSHoundGetGenBankW ⤸
                           hateverSequencePL" ⤸
                           portType="BioActivity4: ⤸
                           MOBYSHoundGetGenBankWhateverSequen ⤸
                           cePortType"/>

<bpel:assign name="AS_Ending">
<bpel:copy>
<bpel:from part="body" ⤸
                variable="MOBYSHoundGetGenBankWhateverSequenc ⤸
                eOutput"/>
<bpel:to part="BioActivity4" variable="outputMessage"/>
</bpel:copy>
</bpel:assign>

</bpel:sequence>

<bpel:reply operation="callBioProcess" ⤸
                      partnerLink="BPELMyServicePL" ⤸
                      portType="tueb:BioProcessPortType" ⤸
                      variable="outputMessage">
<bpel:targets>
<bpel:target linkName="EndingLink"/>
</bpel:targets>
</bpel:reply>

```

```
</bpel:flow>

</bpel:process>
```

A.1.3 Example Process 3

```

<?xml version="1.0" encoding="UTF-8"?>
<!--
BPEL-Example no 3
-->
<bpel:process xmlns:BioActivity1="http://biomoby. ⤸
                                 org/AGILocusList- ⤸
                                 arabidopsis.org" xmlns: ⤸
                                 BioActivity2="http: ⤸
                                 //biomoby. ⤸
                                 org/getNASC_codebyAGI_locus- ⤸
                                 arabidopsis.info" xmlns: ⤸
                                 BioActivity3="http: ⤸
                                 //biomoby. ⤸
                                 org/getEMBLaccessionByNASCst ⤸
                                 ockCode-arabidopsis.info" ⤸
                                 xmlns:BioActivity4="http: ⤸
                                 //biomoby. ⤸
                                 org/MOBYSHoundGetGenBankWhat ⤸
                                 everSequence-bioinfo. ⤸
                                 icapture.ubc.ca" xmlns:aecf- ⤸
                                 xmlstring="http://docs. ⤸
                                 active-endpoints. ⤸
                                 com/activebpel/sample/custom ⤸
                                 Function/2006/09" xmlns: ⤸
                                 bpel="http://docs.oasis- ⤸
                                 open.org/wsbpel/2. ⤸
                                 0/process/executable" xmlns: ⤸
                                 bpelx="http://schemas. ⤸
                                 oracle.com/bpel/extension" ⤸
                                 xmlns:bpws="http://schemas. ⤸
                                 xmlsoap. ⤸
                                 org/ws/2003/03/business- ⤸
```

```
                                  process/" xmlns:sns="http: ⤸
                                  //org.apache.axis2/xsd" ⤸
                                  xmlns:tueb="http: ⤸
                                  //tuebigrid.sample.org" ⤸
                                  xmlns:xsd="http://www.w3. ⤸
                                  org/2001/XMLSchema" ⤸
                                  name="AGILocusList_to_MOBYSH ⤸
                                  oundGetGenBankWhateverSequen ⤸
                                  ce" targetNamespace="http: ⤸
                                  //tuebigrid.sample.org">
<bpel:import importType="http://schemas.xmlsoap.org/wsdl/" ⤸
                        location=".. ⤸
                        /wsdl/AAgetNASC_codebyAGI_locus- ⤸
                        arabidopsis.infoAA.wsdl" ⤸
                        namespace="http://biomoby. ⤸
                        org/getNASC_codebyAGI_locus- ⤸
                        arabidopsis.info"/>
<bpel:import importType="http://schemas.xmlsoap.org/wsdl/" ⤸
                        location=".. ⤸
                        /wsdl/AAMOBYSHoundGetGenBankWhateverS ⤸
                        equence-bioinfo.icapture.ubc.caAA. ⤸
                        wsdl" namespace="http://biomoby. ⤸
                        org/MOBYSHoundGetGenBankWhateverSeque ⤸
                        nce-bioinfo.icapture.ubc.ca"/>
<bpel:import importType="http://schemas.xmlsoap.org/wsdl/" ⤸
                        location=".. ⤸
                        /wsdl/AAgetEMBLaccessionByNASCstockCo ⤸
                        de-arabidopsis.infoAA.wsdl" ⤸
                        namespace="http://biomoby. ⤸
                        org/getEMBLaccessionByNASCstockCode- ⤸
                        arabidopsis.info"/>
<bpel:import importType="http://schemas.xmlsoap.org/wsdl/" ⤸
                        location="../wsdl/AAAGILocusList- ⤸
                        arabidopsis.orgAA.wsdl" ⤸
                        namespace="http://biomoby. ⤸
                        org/AGILocusList-arabidopsis.org"/>
<bpel:import importType="http://schemas.xmlsoap.org/wsdl/" ⤸
                        location="../wsdl/BioProcess.wsdl" ⤸
                        namespace="http://tuebigrid.sample. ⤸
                        org"/>
```

```
<bpel:partnerLinks>
<bpel:partnerLink name="AGILocusListPL" ⤸
                       partnerLinkType="BioActivity1: ⤸
                       AGILocusListServicePLT" ⤸
                       partnerRole="AGILocusListService"/>
<bpel:partnerLink myRole="BioProcess" name="BPELMyServicePL" ⤸
                         partnerLinkType="tueb: ⤸
                         BioProcessPLT"/>
<bpel:partnerLink name="MOBYSHoundGetGenBankWhateverSequenceP ⤸
                       L" partnerLinkType="BioActivity4: ⤸
                       MOBYSHoundGetGenBankWhateverSequenceSe ⤸
                       rvicePLT" ⤸
                       partnerRole="MOBYSHoundGetGenBankWhate ⤸
                       verSequenceService"/>
<bpel:partnerLink name="getNASC_codebyAGI_locusPL" ⤸
                       partnerLinkType="BioActivity2: ⤸
                       getNASC_codebyAGI_locusServicePLT" ⤸
                       partnerRole="getNASC_codebyAGI_locusSe ⤸
                       rvice"/>
<bpel:partnerLink name="getEMBLaccessionByNASCstockCodePL" ⤸
                       partnerLinkType="BioActivity3: ⤸
                       getEMBLaccessionByNASCstockCodeService ⤸
                       PLT" ⤸
                       partnerRole="getEMBLaccessionByNASCsto ⤸
                       ckCodeService"/>
</bpel:partnerLinks>
<bpel:variables>
<bpel:variable messageType="BioActivity3: ⤸
                           getEMBLaccessionByNASCstockCodeOut ⤸
                           put" ⤸
                           name="getEMBLaccessionByNASCstockC ⤸
                           odeOutput"/>
<bpel:variable messageType="BioActivity1:AGILocusListOutput" ⤸
                           name="AGILocusListOutput"/>
<bpel:variable name="tmp" type="xsd:string"/>
<bpel:variable messageType="BioActivity1:AGILocusListInput" ⤸
                           name="AGILocusListInput"/>
<bpel:variable messageType="BioActivity3: ⤸
                           getEMBLaccessionByNASCstockCodeInp ⤸
                           ut" ⤸
                           name="getEMBLaccessionByNASCstockC ⤸
                           odeInput"/>
<bpel:variable messageType="tueb:callBioProcessInputMessage" ⤸
                           name="inputMessage"/>
<bpel:variable messageType="BioActivity2: ⤸
                           getNASC_codebyAGI_locusOutput" ⤸
                           name="getNASC_codebyAGI_locusOutpu ⤸
                           t"/>
<bpel:variable messageType="BioActivity4: ⤸
                           MOBYSHoundGetGenBankWhateverSequen ⤸
                           ceInput" ⤸
                           name="MOBYSHoundGetGenBankWhatever ⤸
                           SequenceInput"/>
<bpel:variable messageType="BioActivity2: ⤸
                           getNASC_codebyAGI_locusInput" ⤸
                           name="getNASC_codebyAGI_locusInput ⤸
                           "/>
<bpel:variable messageType="tueb: ⤸
                           callBioProcessOutputMessage" ⤸
                           name="outputMessage"/>
<bpel:variable messageType="BioActivity4: ⤸
                           MOBYSHoundGetGenBankWhateverSequen ⤸
                           ceOutput" ⤸
                           name="MOBYSHoundGetGenBankWhatever ⤸
                           SequenceOutput"/>
</bpel:variables>

<bpel:sequence>

<bpel:receive createInstance="yes" ⤸
                             name="ReceiveBioWFUserRequest" ⤸
                             operation="callBioProcess" ⤸
                             partnerLink="BPELMyServicePL" ⤸
                             portType="tueb: ⤸
                             BioProcessPortType" ⤸
                             variable="inputMessage">
</bpel:receive>

<bpel:assign name="AS_Beginning">
<bpel:copy>
```

```
<bpel:from part="BioActivity1" variable="inputMessage"/>
<bpel:to variable="tmp" />
</bpel:copy>
<bpel:copy>
<bpel:from>aecf-xmlstring:xmlStringToElement($tmp, ⤸
                                                    "AGI_LocusCode") ⤸
                                                    </bpel:from>
<bpel:to part="data" ⤸
              variable="getNASC_codebyAGI_locusInput"/>
</bpel:copy>
<bpel:copy>
<bpel:from part="data" ⤸
                variable="getNASC_codebyAGI_locusInput"/>
<bpel:to part="BioActivity2I" variable="outputMessage"/>
</bpel:copy>
</bpel:assign>

<bpel:invoke inputVariable="getNASC_codebyAGI_locusInput" ⤸
                          name="getNASC_codebyAGI_locus" ⤸
                          operation="getNASC_codebyAGI_locus ⤸
                          " ⤸
                          outputVariable="getNASC_codebyAGI_ ⤸
                          locusOutput" ⤸
                          partnerLink="getNASC_codebyAGI_loc ⤸
                          usPL" portType="BioActivity2: ⤸
                          getNASC_codebyAGI_locusPortType"/>
<bpel:assign name="AS_Int2tmp">
<bpel:copy>
<bpel:from part="body" ⤸
                variable="getNASC_codebyAGI_locusOutput"/>
<bpel:to variable="tmp"/>
</bpel:copy>
<bpel:copy>
<bpel:from>aecf-xmlstring:xmlStringToElement($tmp, ⤸
                                                    "NASC_code")< ⤸
                                                    /bpel:from>
<bpel:to part="data" ⤸
              variable="getEMBLaccessionByNASCstockCodeInput" ⤸
              />
</bpel:copy>
```

```
<bpel:copy>
<bpel:from part="data" ⤸
               variable="getEMBLaccessionByNASCstockCodeInpu ⤸
               t"/>
<bpel:to part="BioActivity3I" variable="outputMessage"/>
</bpel:copy>
</bpel:assign>

<bpel:invoke inputVariable="getEMBLaccessionByNASCstockCodeIn ⤸
                          put" ⤸
                          name="getEMBLaccessionByNASCstockC ⤸
                          ode" ⤸
                          operation="getEMBLaccessionByNASCs ⤸
                          tockCode" ⤸
                          outputVariable="getEMBLaccessionBy ⤸
                          NASCstockCodeOutput" ⤸
                          partnerLink="getEMBLaccessionByNAS ⤸
                          CstockCodePL" ⤸
                          portType="BioActivity3: ⤸
                          getEMBLaccessionByNASCstockCodePor ⤸
                          tType"/>

<bpel:assign name="AS_Int2tmp">
<bpel:copy>
<bpel:from part="body" ⤸
               variable="getEMBLaccessionByNASCstockCodeOutp ⤸
               ut"/>
<bpel:to variable="tmp"/>
</bpel:copy>
<bpel:copy>
<bpel:from>aecf-xmlstring:xmlStringToElement($tmp, ⤸
                                           "identifier")< ⤸
                                           /bpel:from>
<bpel:to part="data" ⤸
             variable="MOBYSHoundGetGenBankWhateverSequenceI ⤸
             nput"/>
</bpel:copy>
<bpel:copy>
<bpel:from part="data" ⤸
               variable="MOBYSHoundGetGenBankWhateverSequenc ⤸
```

```
                eInput"/>
<bpel:to part="BioActivity4I" variable="outputMessage"/>
</bpel:copy>
</bpel:assign>

<bpel:invoke inputVariable="MOBYSHoundGetGenBankWhateverSeque⤸
                           nceInput"⤸
                           name="MOBYSHoundGetGenBankWhatever⤸
                           Sequence"⤸
                           operation="MOBYSHoundGetGenBankWha⤸
                           teverSequence"⤸
                           outputVariable="MOBYSHoundGetGenBa⤸
                           nkWhateverSequenceOutput"⤸
                           partnerLink="MOBYSHoundGetGenBankW⤸
                           hateverSequencePL"⤸
                           portType="BioActivity4:⤸
                           MOBYSHoundGetGenBankWhateverSequen⤸
                           cePortType"/>

<bpel:assign name="AS_Ending">
<bpel:copy>
<bpel:from part="body"⤸
                variable="MOBYSHoundGetGenBankWhateverSequenc⤸
                eOutput"/>
<bpel:to part="BioActivity4" variable="outputMessage"/>
</bpel:copy>
</bpel:assign>

<bpel:reply operation="callBioProcess"⤸
                      partnerLink="BPELMyServicePL"⤸
                      portType="tueb:BioProcessPortType"⤸
                      variable="outputMessage">
<bpel:reply>

</bpel:sequence>

</bpel:process>
```

A.2 HOBBES: Compiled BPEL

The following listing is an example of incomplete BPEL code compiled by HOBBES during an ongoing editing session:

```
<bpel>
  <process name="sample" targetNamespace="http://tuebigrid.⤸
                sample.org" ext:createTargetXPath="yes"⤸
                xmlns="http://docs.oasis-open.org/wsbpel/2.⤸
                0/process/executable" xmlns:sns="http://org.⤸
                apache.axis2/xsd" xmlns:bpws="http://schemas.⤸
                xmlsoap.org/ws/2003/03/business-process/"⤸
                xmlns:bpelx="http://schemas.oracle.⤸
                com/bpel/extension" xmlns:wsa="http:⤸
                //schemas.xmlsoap.org/ws/2004/03/addressing"⤸
                xmlns:abx="http://www.activebpel.⤸
                org/bpel/extension" xmlns:sref="http://docs.⤸
                oasis-open.org/wsbpel/2.0/serviceref" xmlns:⤸
                ext="http://www.activebpel.⤸
                org/2006/09/bpel/extension/query_handling"⤸
                xmlns:aecf-xmlstring="http://docs.active-⤸
                endpoints.⤸
                com/activebpel/sample/customFunction/2006/09"⤸
                xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <extensions>
      <extension mustUnderstand="yes" namespace="http://www.⤸
                                activebpel.⤸
                                org/2006/09/bpel/extension/qu⤸
                                ery_handling"/>
    </extensions>
    <partnerLinks/>
    <variables/>
    <eventHandlers>
      <onAlarm>
        <until>9-11-3T0:0:0</until>
        <scope>
          <variables/>
          <flow>
            <targets/>
          </flow>
        </scope>
        </onAlarm>
    </eventHandlers>
    <flow name="root">
        <targets/>
        <links>
            <link name="Link1256829383691"/>
            <link name="Link1256829388526"/>
            <link name="Link1256829381863"/>
            <link name="Link1256829321517"/>
            <link name="Link1256829324475"/>
            <link name="Link1256829385775"/>
            <link name="Link1256829344429"/>
            <link name="Link1256829346555"/>
        </links>
        <while>
            <targets>
                <target linkName="Link1256829346555"/>
            </targets>
            <sources>
                <source linkName="Link1256829388526"/>
            </sources>
            <condition/>
        </while>
        <sequence>
            <targets>
                <target linkName="Link1256829321517"/>
            </targets>
            <sources>
                <source linkName="Link1256829381863"/>
            </sources>
        </sequence>
        <while>
            <targets>
                <target linkName="Link1256829344429"/>
            </targets>
            <sources>
                <source linkName="Link1256829385775"/>
            </sources>
            <condition/>
        </while>
    <sequence>
      <targets>
        <target linkName="Link1256829324475"/>
      </targets>
      <sources>
        <source linkName="Link1256829383691"/>
      </sources>
      <assign>
        <targets/>
      </assign>
      <invoke partnerLink="null"  operation="null">
        <targets/>
      </invoke>
      <if>
        <targets/>
      </if>
      <assign>
        <targets/>
      </assign>
      <wait>
        <targets/>
      </wait>
    </sequence>
    <receive partnerLink="null"  createInstance="yes">
      <targets/>
      <sources>
        <source linkName="Link1256829321517"/>
        <source linkName="Link1256829344429"/>
        <source linkName="Link1256829324475"/>
        <source linkName="Link1256829346555"/>
      </sources>
    </receive>
    <reply partnerLink=""  portType="">
      <targets>
        <target linkName="Link1256829383691"/>
        <target linkName="Link1256829388526"/>
        <target linkName="Link1256829381863"/>
        <target linkName="Link1256829385775"/>
      </targets>
    </reply>
```

```
        </flow>
    </process>
</bpel>
```

Bibliography

[AAA+07] Alexandre Alves, Assaf Arkin, Sid Askary, Charlton Baretto, Ben Bloch, Francisco Curbera, Mark Ford, Yaron Goland, Alejandro Guizar, Neeklatantan Kartha, Canyang Kevin Liu, Rania Khalaf, Dieter König, Mike Marin, Vinkesh Mehta, Satish Thatte, Danny van der Rijn, Prasad Yendluri, and Alex Yiu. Web Services Business Process Execution Language Version 2.0. OASIS standard, chairs: Diane Jordan and John Evdemon, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf, April 2007. last accessed: 24/10/2007.

[AAD+a] Ashish Agrawal, Mike Amend, Manoj Das, Mark Ford, Chris Keller, Matthias Kloppmann, Dieter König, Frank Leymann, Ralf Müller, Gerhard Pfau, Karsten Plösser, Ravi Rangaswamy, Alan Rickayzen, Michael Rowley, Patrick Schmidt, Ivana Trickovic, Alex Yiu, and Matthias Zeller. WS-HumanTask specification, v1.0. Proposal. Adobe, BEA, Oracle, Active Endpoints, IBM, SAP. June 2007. http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel4people/WS-HumanTask_v1.pdf, last accessed: 03/02/2010.

[AAD+b] Ashish Agrawal, Mike Amend, Manoj Das, Mark Ford, Chris Keller, Matthias Kloppmann, Dieter König, Frank Leymann, Ralf Müller, Gerhard Pfau, Karsten Plösser, Ravi Rangaswamy, Alan Rickayzen, Michael Rowley, Patrick Schmidt, Ivana Trickovic, Alex Yiu, and Matthias Zeller. WS-BPEL Extension for People specification, v1.0. Proposal. Adobe, BEA, Oracle, Active Endpoints, IBM, SAP. June 2007. http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel4people/BPEL4People_v1.pdf, last accessed: 23/10/2008.

[ACD+03] Tony Andrews, Francisco Curbera, Hitesh Dholokia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovi, and Sanjiva Weerawearana. Business Process Execution Language for Web Services version 1.1. http://download.boulder.ib.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf, May 2003. last accessed: 24/10/2007.

[ADD+03] Gabrielle Allen, Kelly Davis, Konstantinos N. Dolkas, Nikolaos D. Doulamis, Tom Goodale, Thilo Kielmann, Andrè Merzky, Jarek Nabrzyski, Juliusz Pukacki, Thomas Radke, Michael Russell, Ed Seidel, John Shalf, and Ian Taylor. Enabling applications on the grid: A gridlab overview. *International Journal of High Performance Computing Applications: Special Issue on Grid Computing: Infrastructure and Applications*, 15(4):449–466, November 2003.

[AG] Software AG. Alignspace homepage. `http://www.alignspace.com`, last accessed 31/12/2009.

[Alo] G. Alonso. Bioopera: Grid computing in virtual laboratories. In *ERCIM News*, No.45 - April 2001.

[bas06] RFC 4648: The Base16, Base32, and Base64 Data Encodings, 2006. `http://tools.ietf.org/html/rfc4648`, last accessed 17/11/09.

[BBPP99] I. Bomze, M. Budinich, P. Pardalos, and M. Pelillo. The maximum clique problem. In D.-Z. Du and P. M. Pardalos, editors, *Handbook of Combinatorial Optimization*, volume 4. Kluwer Academic Publishers, Boston, MA, 1999.

[BCMS07] E. Bartocci, F. Corradini, E. Merelli, and L. Scortichini. BioWMS: a web-based Workflow Management System for bioinformatics. *BMC Bioinformatics*, 8 Suppl 1, 2007.

[BDG+93] Adam Beguelin, Jack Dongarra, Al Geist, Robert Manchek, and Keith Moore. Hence: A heterogeneous network computing environment. Technical report, Knoxville, TN, USA, 1993.

[BDG+94a] Adam Beguelin, Jack Dongarra, G. A. Geist, Robert Manchek, Keith Moore, Peter Newton, and Vaidy Sunderam. HeNCE: A Users' Guide Version 2.0, 1994. `http://www.netlib.org/hence/hence-2.0-doc-html/hence-2.0-doc.html`, last accessed 27/08/2009.

[BDG+94b] Adam Beguelin, Jack J. Dongarra, George Al Geist, Robert Manchek, and Keith Moore. HeNCE: a heterogenous network computing environment. *Scientific Programing*, 3(1):49–60, 1994.

[BGH+08] A. Brinkmann, S. Gudenkauf, W. Hasselbring, A. Hoeing, O. Kao, H. Karl, H. Nitsche, and G. Scherp. Employing WS-BPEL Design Patterns for Grid Service Orchestration using a Standard WS-BPEL Engine and a Grid Middleware. In *Crakow Grid Workshop'08 Proceedings*, 2008.

[BK73] Coen Bron and Joep Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, 16(9), 1973.

[Bla07] M. Brian Blake. Decomposing Composition: Service-Oriented Software Engineers. *IEEE Software*, 24(6):68–77, 2007.

[bpma] Business Process Modeling Notation (BPMN) Version 1.2. OMG Available Specification, `http://www.omg.org/spec/BPMN/1.2/PDF`, last accessed 28/10/2009.

[bpmb] Business Process Modeling Notation, V1.1. OMG Available Specification, `http://www.omg.org/spec/BPMN/1.1/PDF`, last accessed 28/10/2009.

[BPSA02] W. Bausch, C. Pautasso, R. Schaeppi, and G. Alonso. BioOpera: Cluster-aware Computing. In *Proceedings of the 4th IEEE International Conference on Cluster Computing*, September 2002.

[Bro78] Frederick P. Brooks, Jr. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1978.

[BRS99] James Begole, Mary Beth Rosson, and Clifford A. Shaffer. Flexible Collaboration Transparency: Supporting Worker Independence in Replicated Application-sharing Systems. *ACM Transactions on Computer-Human Interaction*, 6(2):95–132, 1999.

[BSR96] A. J. Bonner, A. Shrufi, and S. Rozen. Labflow-1: A database benchmark for high-throughput workflow management. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, 1996.

[BvH08] Adam Barker and Jano van Hemert. Scientific Workflow: A Survey and Research Directions. In Roman Wyrzykowski et al., editor, *Seventh International Conference on Parallel Processing and Applied Mathematics, Revised Selected Papers, volume 4967 of LNCS*, pages 746–753. Springer, 2008.

[Car] Jorge Cardoso. Control-flow Complexity Measurement of Processes and Weyuker's Properties. In Ardil and Cemal, editors, *6th International Conference on Enformatika*, volume 8, pages 213–218. International Academy of Sciences.

[Car05] Jorge Cardoso. Evaluating the Process Control-Flow Complexity Measure. In *2005 IEEE International Conference on Web Services (ICWS 2005)*, pages 803–804. IEEE Computer Society, 2005.

[Car07] Jorge Cardoso. Complexity Analysis of BPEL Web Processes. *Software Process: Improvement and Practice*, 12(2):35–49, 2007.

[CG06] Sébastien Carrere and Jérôme Gouzy. REMORA: a pilot in the ocean of BioMoby web-services. *Bioinformatics*, 22(7):900–901, 2006.

[CM07] Marcus Christie and Suresh Marru. The LEAD Portal: a TeraGrid gateway and application service architecture: Research Articles. *Concurrency and Computation : Practice and Experience*, 19(6):767–781, 2007.

[CML04] T. Clark, S. Martin, and T. Liefeld. Globally distributed object identification for biological knowledgebases. *Briefings in Bioinformatics*, 5(1):59–70, March 2004.

[CMNR06] J. Cardoso, J. Mendling, G. Neumann, and H.A. Reijers. A Discourse on Complexity of Process Models. In J. Eder and S. Dustdar, editors, *BPI'06 - Second International Workshop on Business Process Intelligence, In conjunction with BPM 2006, LNCS 4103*, page pp.115–126. Springer-Verlag, Berlin, Heidelberg, 2006.

[DF08] Susan B. Davidson and Juliana Freire. Provenance and scientific workflows: challenges and opportunities. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1345–1350, New York, NY, USA, 2008. ACM.

[DGST09] Ewa Deelman, Dennis Gannon, Matthew Shields, and Ian Taylor. Workflows and e-Science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25(5):528–540, May 2009.

[dKGlL$^+$04] Remko de Knikker, Youjun Guo, Jin long Li, Albert K. H. Kwan, Kevin Y. Yip, David W. Cheung, and Kei-Hoi Cheung. A web services choreography scenario for interoperating bioinformatics applications. *BMC Bioinformatics*, 5:25, 2004.

[Dra96] Thomas Drake. Measuring software quality: A case study. *IEEE Computer*, 29(11):78–87, 1996.

[Dre09] Alexander Dreiling. Gravity – Collaborative Business Process Modelling within Google Wave, 2009. SAP Community Network, `http://www.sdn.sap.com/irj/scn/weblogs;jsessionid=%28J2EE3417700%29ID0831279350DB00207143003258434961End?blog=/pub/wlg/15618`, last accessed 17/12/2009.

[dRGS08] David de Roure, Carole Goble, and Robert Stevens. The Design and Realisation of the ^{m}yExperiment Virtual Research Environment for Social Sharing of Workflows. *Future Generation Computer Systems*, 2008.

[DSS$^+$] Ewa Deelman, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G. Bruce Berriman, John Good, Anastasia Laity, Joseph C. Jacob, and Daniel S. Katz. Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems. *Scientific Programming Journal*, 13(3):219–237.

[EBC$^+$05] Wolfgang Emmerich, Ben Butchart, Liang Chen, Bruno Wassermann, and Sarah Price. Grid Service Orchestration Using the Business Process Execution Language (BPEL). *Journal of Grid Computing*, 3(3-4):283–304, September 2005.

[ECM99] Standard ECMA-262: ECMAScript Language Specification, 3rd Edition, 1999. Ecma International, Geneva, Switzerland. `http://www.ecma-international.org/publications/standards/ecma-262.htm`, last accessed 16/11/2009.

[EG89] C. A. Ellis and S. J. Gibbs. Concurrency Control in Groupware Systems. *SIGMOD Record*, 18(2):399–407, 1989.

[EJL$^+$03] Johan Eker, Jörn W. Janneck, Edward A. Lee, Jie Liu, Xiaojun Liu, Jozsef Ludvig, Stephen Neuendorfer, Sonia Sachs, and Yuhong Xiong. Taming Heterogeneity—The Ptolemy Approach. *Proceedings of the IEEE*, 91(1), 2003.

[EKA$^+$08] Tommy Ellkvist, David Koop, Erik Anderson, Claudio Silva, and Juliana Freire. Using Provenance to Support Real-Time Collaborative Design of Workflows. In *Second International Provenance and Annotation Workshop (IPAW 2008)*. Springer, 2008.

[ESCR07] Onyeka Ezenwoye, S. Masoud Sadjadi, Ariel Carey, and Michael Robinson. Grid service composition in bpel for scientific applications. In *Proceedings of the International Conference on Grid computing, high-performAnce and Distributed Applications (GADA'07)*, Vilamoura, Algarve, Portugal, November 2007. (accepted for publication.).

[Fie00] Roy Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, 2000. University of California, Irvine.

[FKT01] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computer Applications*, 15(3):200–222, 2001.

[fle] Adobe Flex Developer Center. `http://www.adobe.com/devnet/flex/`. last accessed 24/10/2007.

[FN00] Norman E. Fenton and Martin Neil. Software metrics: roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pages 357–370, New York, NY, USA, 2000. ACM.

[Fow99] Martin Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[FPT] T. Fahringer, S. Pllana, and J. Testori. Teuta: Tool Support for Performance Modeling of Distributed and Parallel Applications. In *International Conference on Computational Science; Tools for Program Development and Analysis in Computational Science*.

[Fre] `http://freefluo.sourceforge.net/`, last accessed 26/08/2009.

[FSF+06] Thomas Friese, Matthew Smith, Bernd Freisleben, Julian Reichwald, Thomas Barth, and Manfred Grauer. Collaborative Grid Process Creation Support in an Engineering Domain. In *Proceedings of High Performance Computing - HiPC 2006, 13th International Conference*, 2006.

[FT02] Roy T. Fielding and Richard N. Taylor. Principled design of the modern web architecture. *ACM Transactions on Internet Technology*, 2(2):115–150, 2002.

[Gar05] Jesse James Garret. Ajax: A New Approach to Web Applications, 2005. Blog Entry, `http://www.adaptivepath.com/ideas/essays/archives/000385.php`, last accessed 26/10/2009.

[GDE+07] Yolanda Gil, Ewa Deelman, Mark Ellisman, Thomas Fahringer, Geoffrey Fox, Dennis Gannon, Carole Goble, Miron Livny, Luc Moreau, and Jim Myers. Examining the Challenges of Scientific Workflows. *IEEE Computer*, 40(12):24–32, 2007.

[GEO] The GEO 600 Project. `http://geo600.aei.mpg.de/`, last accessed: 19/08/2009.

[GGKK03] Ananth Grama, Anshul Gupta, George Karypsis, and Vipin Kumar. *Introduction to Parallel Computing, 2nd Edition*. Addison Wesley, 2003.

[GGW+07] Andrew Gibson, Matthew Gamble, Katy Wolstencroft, Tom Oinn, and Carole Goble. The data playground: An intuitive workflow specification environment. In *E-SCIENCE '07: Proceedings of the Third IEEE International Conference on e-Science and Grid Computing*, pages 59–68, Washington, DC, USA, 2007. IEEE Computer Society.

[GGW+09] Andrew Gibson, Matthew Gamble, Katy Wolstencroft, Tom Oinn, Carole Goble, Khalid Belhajjame, and Paolo Missier. The data playground: An intuitive workflow specification environment. *Future Generation Computer Systems*, 25(4):453–459, 2009.

[GHCM09] Thilina Gunarathne, Chathura Herath, Eran Chinthaka, and Suresh Marru. Experience with adapting a WS-BPEL runtime for eScience workflows. In *GCE '09: Proceedings of the 5th Grid Computing Environments Workshop*, pages 1–10, New York, NY, USA, 2009. ACM.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison-Wesley Professional, January 1995.

[GHS95] D. Georgakopoulos, M. Hornick, and A. Sheth. An overview of workflow management: From process modelling to workflow automation infrastructure. *Distributed and Parallel Databases*, 3:119–153, 1995.

[GM08] Tristan Glatard and Johan Montagnat. Implementation of turing machines with the scufl data-flow language. In *CCGRID '08: Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*, pages 663–668, Washington, DC, USA, 2008. IEEE Computer Society.

[GRS98] Nathan Goodman, Steve Rozen, and Lincoln D. Stein. The labflow system for workflow management in large scale biology research laboratories. In *Proceedings of the 6th International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 69–77, 1998.

[GRSS98] N. Goodman, S. Rozen, L. D. Stein, and A. G. Smith. The labbase system for data management in large scale biology research laboratories. *Bioinformatics*, 14(7):562–574, 1998.

[GS07a] P. Gordon and C Sensen. A pilot study into the usability of a scientific workflow construction tool. Technical Report CPSC Technical Report 2007-874-26. Technical report, University of Calgary, Sun Center of Excellence for Visual Genomics, 2007.

[GS07b] Paul M.K. Gordon and Christoph W. Sensen. Seahawk: moving beyond HTML in Web-based bioinformatics analysis. *BMC Bioinformatics*, 8(208), 2007.

[Hav05] Michael Havey. *Essential Business Process Modeling*. O'Reilly Media Inc., 2005.

[HB08] Markus Held and Wolfgang Blochinger. Collaborative bpel design with a rich internet application. In *CCGRID '08: Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid*, pages 202–209, Washington, DC, USA, 2008. IEEE Computer Society.

[Her07] James D. Herbsleb. Global Software Engineering: The Future of Socio-technical Coordination. In *FOSE '07: 2007 Future of Software Engineering*, pages 188–198, 2007.

[Hol95] David Hollingsworth. The workflow reference model, 1995. Published by the Workflow Management Coalition.

[HWD05] Thomas B. Hodel-Widmer and Klaus R. Dittrich. Concept and prototype of a collaborative business processing environment for document processing. *Data & Knowledge Engineering*, 52(1):61–120, 2005.

[Ign06] Claudia-Lavinia Ignat. *Maintaining Consistency in Collaboration over Hierarchichal Documents*. PhD thesis, Swiss Federal Institute of Technology, Zürich, 20006.

[IN] C.-L. Ignat and M.C. Norrie. Grouping in collaborative graphical editors. In *CSCW'04: Proceedings of the 2004 ACM conference on Computer Supported Cooperative Work*, pages pp. 447–456.

[IN06] C.-L. Ignat and M.C. Norrie. Draw-together: Graphical editor for collaborative drawing. In *Proceedings of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work*, pages 269–278, 2006.

[JKB08] I. Janciak, C. Kloner, and P. Brezany. Workflow enactment engine for wsrf-compliant services orchestration. In *GRID '08: Proceedings of the 2008 9th IEEE/ACM International Conference on Grid Computing*, pages 1–8, Washington, DC, USA, 2008. IEEE Computer Society.

[KBS04] Dirk Krafzig, Karl Banke, and Dirk Slama. *Enterprise SOA: Service-Oriented Architecture Best Practices*. Prentice Hall, 2004.

[KDF96] Péter Kacsuk, Gábor Dózsa, and Tibor Fadgyas. Designing parallel programs by the graphical language grapnel. *Microprocess. Microprogram.*, 41(8-9):625–643, 1996.

[KDF97] P. Kacsuk, G. Dozsa, and T. Fadgyas. A graphical programming environment for message passing programs. In *PDSE '97: Proceedings of the 2nd International Workshop on Software Engineering for Parallel and Distributed Systems*, page 210, Washington, DC, USA, 1997. IEEE Computer Society.

[KKL$^+$] Matthias Kloppmann, Dieter Koenig, Frank Leymann, Gerhard Pfau, Alan Rickayzen, Claus von Riegen, Patrick Schmidt, and Ivana Trickovic. WS-BPEL Extension for People – BPEL4People. White Paper. IBM, SAP. July 2005. `http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel4people/BPEL4People_white_paper.pdf`, last accessed: 23/10/2008.

[KLM$^+$97] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-Oriented Programming. In *Proceedings of the European Conference on Object-Oriented Programming - ECOOP*, pages 220–242, 1997.

[KSW06] Edward Kawas, Martin Senger, and Mark D. Wilkinson. Biomoby extensions to the taverna workflow management and enactment software. *BMC Bioinformatics*, 7:523+, November 2006.

[Ley06] Frank Leymann. Choreography for the grid: towards fitting bpel to the resource framework. *Concurrency and Computation: Practice and Experience*, 18(10):1201–1217, 2006.

[LHC$^+$06] Qiang Lu, Pei Hao, Vasa Curcin, Weizhong He, Yuan-Yuan Li, Qing-Ming Luo, Yi-Ke Guo, and Yi-Xue Li. Kde bioscience: platform for bioinformatics analysis workflows. *J. of Biomedical Informatics*, 39(4):440–450, 2006.

[Lig95] Peter Liggesmeyer. A set of complexity metrics for guiding the software test process. *Software Quality Journal*, 4(4):257–273, 1995.

[LLO08] Anders Lanzén, Anders Lanzén, and Tom Oinn. The Taverna Interaction Service. *Bioinformatics*, 24(8):1118–1120, 2008.

[LM] Catherine Letondal and Wendy E. Mackkay. Participatory Programming and the Scope of Mutual Responsibility: Balancing scientific, design and software commitment. In *Participatory Design Conference*, pages 31–44. ACM.

[LMS09] Anna-Lena Lamprecht, Tiziana Margaria, and Bernhard Steffen. From Bio-jETI Process Models to Native Code. In *Proceedings of the 2009 14th IEEE International Conference on Engineering of Complex Computer Systems*, volume 0, pages 95–101, Los Alamitos, CA, USA, 2009. IEEE Computer Society.

[LR00] Frank Leymann and Dieter Roller. *Production workflow: concepts and techniques*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000.

[LWMB09] Bertram Ludäscher, Mathias Weske, Timothy McPhillips, and Shawn Bowers. Scientific workflows: Business as usual? In Umeshwar Dayal, Johann Eder, Jana Koehler, and Hajo Reijers, editors, *7th Intl. Conf. on Business Process Management (BPM)*, LNCS 5701, Ulm, Germany, 2009.

[McC76] T.J. McCabe. A Complexity Measure. *IEEE Transactions on Software Engineering*, 2(1):308–320, 1976.

[MKS08] T. Margaria, C. Kubczak, and B. Steffen. Bio-jETI: a service integration, design, and provisioning platform for orchestrated bioinformatics processes. *BMC bioinformatics*, 9 Suppl 4, 2008.

[MLHJR04] Audrius Meskauskas, Frank Lehmann-Horn, and Karin Jurkat-Rott. Sight: automating genomic data-mining without programming skills. *Bioinformatics*, 20(11):1718–1720, 2004.

[NB92] Peter Newton and James C. Browne. The code 2.0 graphical parallel programming language. In *ICS '92: Proceedings of the 6th international conference on Supercomputing*, pages 167–177, New York, NY, USA, 1992. ACM.

[OAF$^+$04] Tom Oinn, Matthew Addis, Justin Ferris, Darren Marvin, Martin Senger, Mark Greenwood, Tim Carver, Kevin Glover, Matthew R. Pocock, Anil Wipat, and Peter Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004.

[OAS06] OASIS. OASIS Reference Model for Service Oriented Architecture 1.0. `http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf`, October 2006. last accessed 09/03/2009.

[OGA$^+$06] Tom Oinn, Mark Greenwood, Matthew Addis, M. Nedim Alpdemir, Justin Ferris, Kevin Glover, Carole Goble, Antoon Goderis, Duncan Hull, Darren Marvin, Peter Li, Phillip Lord, Matthew R. Pocock, Martin Senger, Robert Stevens, Anil Wipat, and Chris Wroe. Taverna: lessons in creating a workflow environment for the life sciences: Research articles. *Concurrency and Computation : Practice and Experience*, 18(10):1067–1100, 2006.

[O'R] Tim O'Reilly. What Is Web 2.0 - Design Patterns and Business Models for the Next Generation of Software. `http://www.oreillynet.com/lpt/a/6228`. last accessed 11/06/2008.

[Ost87] L. Osterweil. Software processes are software too. In *ICSE '87: Proceedings of the 9th international conference on Software Engineering*, pages 2–13. IEEE Computer Society Press, Los Alamitos, CA, USA, 1987.

[Pau08] Cesare Pautasso. BPEL for REST. In *7th International Conference on Business Process Management (BPM08)*, pages 278–293, Milan, Italy, September 2008.

[PFT$^+$04] S. Pllana, T. Fahringer, J. Testori, S. Benkner, and I. Brandic. Towards an UML Based Graphical Representation of Grid Workflow Applications. In *The 2nd European Across Grids Conference*, January 2004.

[PHL09a] Hoang M. Phung, Doan B. Hoang, and Elaine Lawrence. A Novel Collaborative Grid Framework for Distributed Healthcare. In *9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 514–519, 2009.

[PHL09b] Hoang M. Phung, Doan B. Hoanga, and Elaine Lawrence. A Front-End for collaborative task planning on the grid. In *International Symposium on Collaborative Technologies and Systems*, pages 292–299, 2009.

[PQF] S. Pllana, J. Qin, and T. Fahringer. Teuta: A Tool for UML Based Composition of Scientific Grid Workflows. In *1st Austrian Grid Symposium*. Schloss Hagenberg, Austria, December 2005, `http://www.par.univie.ac.at/~pllana/papers/pqf_ags05.pdf`, last accessed 26/08/2009.

[PZL08] Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. RESTful Web Services vs. Big Web Services: Making the Right Architectural Decision. In *17th International World Wide Web Conference (WWW2008)*, April 2008.

[RBB$^+$07] Paolo Romano, Ezio Bartocci, Guglielmo Bertolini, Flavio De Paoli, Domenico Marra, Giancarlo Mauri, Emanuela Merelli, and Luciano Milanesi. Biowep: a workflow enactment portal for bioinformatics applications. *BMC Bioinformatics*, 8(1), 2007.

[RGB$^+$08] David De Roure, Carol Goble, Jitenn Bhagat, Don Cruickshank, Antoon Goderis, Danius Michaelides, and David Newman. myExperiment: Defining the Social Virtual Research Environment. In *Proceedings of the 4th IEEE International Conference on eScience*, pages 182–189, 2008.

[RGS07] David De Roure, Carole Goble, and Robert Stevens. Designing the myexperiment virtual research environment for the social sharing of workflows. In *E-SCIENCE '07: Proceedings of the Third IEEE International Conference on e-Science and Grid Computing*, pages 603–610, Washington, DC, USA, 2007. IEEE Computer Society.

[RKO+03] Anthony Rowe, Dimitrios Kalaitzopoulos, Michelle Osmond, Moustafa Ghanem, and Yike Guo. The discovery net system for high throughput bioinformatics. *Bioinformatics*, 19(Suppl. 1):i225–i231, 2003.

[RR07] Leonard Richardson and Sam Ruby. *RESTful Web Services*. O'Reilly Media Inc., Sebastopol, California, USA, 2007.

[RSG95] S. Rozen, L. Stein, and N. Goodman. Labbase: A database to manage laboratory data in a large-scale genome-mapping project. *IEEE Engineering in Medicine and Biology*, 14:702–709, 1995.

[Say05] Robert Sayre. Atom: The Standard in Syndication. *IEEE Internet Computing*, 9(4):71–78, 2005.

[SBL+86] M. Stefik, D. G. Bobrow, S. Lanning, D. Tatar, and G. Foster. WYSIWIS revised: Early Experiences with Multi-User Interfaces. In *CSCW '86: Proceedings of the 1986 ACM conference on Computer-supported cooperative work*, pages 276–290, 1986.

[SBL+08] A. A. Shah, D. Barthel, P. Lukasiak, J. Blascewicz, and N. Krasnogor. Web & Grid Technologies in Bioinformatics, Computational Biology and Systems Biology: A Review. *Current Bioinformatics*, 3(1):10–31, 2008.

[SC02] C. Sun and D. Chen. Consistency maintenance in real-time collaborative graphics editing systems. *ACM Transactions on Computer-Human Interactions*, 9(1):1–41, 2002.

[Sch99a] August Wilhelm Scheer. *ARIS Business Process Modeling*. Springer Verlag, 1999.

[Sch99b] Marc-Thomas Schmidt. The evolution of workflow standards. *IEEE Concurrency*, 7(3):44–52, 1999.

[SE98] Chengzheng Sun and Clarence Ellis. Operational Transformation in Real-time Group Editors: Issues, Algorithms, and Achievements. In *CSCW '98: Proceedings of the 1998 ACM conference on Computer supported cooperative work*, pages 59–68, 1998.

[SHS+04] S. P. Shah, D. Y. He, J. N. Sawkins, J. C. Druce, G. Quon, D. Lett, G. X. Zheng, T. Xu, and B. F. Ouellette. Pegasys: software for executing and integrating analyses of biological sequences. *BMC Bioinformatics*, 5, April 2004.

[SJZ+98] Chengzheng Sun, Xiaohua Jia, Yanchun Zhang, Yun Yang, and David Chen. Achieving Convergence, Causality Preservation, and Intention Preservation in

Real-time Cooperative Editing Systems. *ACM Transactions on Computer-Human Interactions*, 5(1):63–108, 1998.

[SK] Gergely Sipos and Peter Kacsuk. Maintaining Consistency Properties of Grid Workflows in Collaborative Editing Systems. In *Proceedings of the 2009 Eigth International Conference on Grid and Cooperative Computing*, pages 168–175.

[SK05] Gergely Sipos and Peter Kacsuk. Multi-grid, Multi-user Workflows in the P-GRADE Portal. *Journal of Grid Computing*, 3(3-4):221–238, 2005.

[SKS$^+$08] Carlos Scheidegger, David Koop, Emanuele Santos, Huy Vo, Steven Callahan, Juliana Freire, and Cláudio Silva. Tackling the Provenance Challenge one layer at a time. *Concurr. Comput. : Pract. Exper.*, 20(5):473–483, 2008.

[Slo06] Aleksander Slominski. On using BPEL extensibility to implement OGSI and WSRF Grid workflows. *Concurrency and Computation: Practice and Experience Special Issue: Workflow in Grid Systems*, 18(10):1229 – 1241, 2006.

[SRG94] L. Stein, S. Rozen, and N. Goodman. Managing laboratory workflow with labbase. In *Proceedings of the 1994 Conference on Computers in Medicine (CompMed94)*. World Scientific Publishing Company, 1994.

[SRG03] Robert D. Stevens, Alan J. Robinson, and Carole A. Goble. myGrid: personalised bioinformatics on the information grid. *Bioinformatics*, 19(Suppl. 1):i302–i304, 2003.

[ST98] David B. Skillicorn and Domenico Talia. Models and languages for parallel computation. *ACM Computing Surveys*, 30(2):123–169, 1998.

[STT81] K. Sugiyama, S. Tagawa, and M . Toda. Methods for Visual Understanding of Hierarchical System Structures. *IEEE Transactions on Systems, Man and Cybernetics*, 11(2):109–125, 1981.

[STW$^+$04] R. D. Stevens, H. J. Tipney, C. J. Wroe, T. M. Oinn, M. Senger, P. W. Lord, C. A. Goble, A. Brass, and M. Tassabehji. Exploring Williams–Beuren syndrome using myGrid. *Bioinformatics*, 20(1):303–310, 2004.

[sub] SubEtha-Edit. `http://www.codingmonkeys.de/subethaedit/`. last accessed 24/10/2007.

[SV96] Munindar P. Singh and Mladen A. Vouk. Scientific workflows: Scientific computing meets transactional workflows. In *Reference Papers of the NSF Workshop on Workflow and Process Automation in Information*

Systems: State-of-the-art and Future Directions, May 1996. Position paper. http://www.csc.ncsu.edu/faculty/mpsingh/papers/databases/workflows/sciworkflows.html, last accessed 27/08/2009.

[TCH$^+$05] Francis Tang, Ching Lian Chua, Liang-Yoong Ho, Yun Ping Lim, Praveen Issac, and Arun Krishnan. Wildfire: distributed, Grid-enabled workflow construction and execution, 2005.

[TDGS06] Ian J. Taylor, Ewa Deelman, Dennis B. Gannon, and Matthew Shields. *Workflows for e-Science: Scientific Workflows for Grids*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[TH07] Maik Thiele and Dirk Habich. *Orchestrierung datenintensiver Prozesse: Einsatz von BPEL in der Genexpressionsanalyse*. VDM Verlag Dr. Müller, 2007.

[The08] The BioMoby Consortium. Interoperability with Moby 1.0 — It's better than sharing your toothbrush! *Briefings In Bioinformatics*, 9(3), 2008.

[TLD$^+$07] Oliver Thomas, Katrina Leyking, Florian Dreifus, Michael Fellmann, and Peter Loos. Serviceorientierte Architekturen: Gestaltung, Konfiguration und Ausführung von Geschäftsprozessen. Technical report, Institut für Wirtschaftsinformatik, Saarbrücken, January 2007.

[TMMF09] Wei Tan, Paolo Missier, Ravi Madduri, and Ian Foster. Building Scientific Workflow with Taverna and BPEL: A Comparative Study in caGrid. pages 118–129, 2009.

[TS07] Abhishek Tiwari and Arvind K. T. Sekhar. Review Article: Workflow based framework for life science informatics. *Computational Biology and Chemistry*, 31(5-6):305–319, 2007.

[vdAtHW03] W.M.P. van der Aalst, A.H.M. ter Hofstede, and M. Weske. Business Process Management: A Survey. In W.M.P. van der Aalst, A.H.M. ter Hofstede, and M. Weske, editors, *International Conference on Business Process Management (BPM 2003)*, pages 1–12. Springer-Verlag, Berlin, 2003. volume 2678 of Lecture Notes in Computer Science.

[Vog] Jürgen Vogel. *Consistency Algorithms and Protocols for Distributed Interactive Applications*. Dissertation, Universität Mannheim, 2004. http://madoc.bib.uni-mannheim.de/madoc/volltexte/2004/671/, last accessed 08/07/2008.

[W3C04a] W3C. XML Schema Part 0: Primer Second Edition. `http://www.w3.org/TR/xmlschema-0/`, last accessed: 23/10/2008., 2004.

[W3C04b] W3C. XML Schema Part 1: Structures Second Edition. `http://www.w3.org/TR/xmlschema-1/`, last accessed: 23/10/2008., 2004.

[W3C04c] W3C. XML Schema Part 2: Datatypes Second Edition. `http://www.w3.org/TR/xmlschema-2/`, last accessed: 23/10/2008., 2004.

[W3C07a] W3C. Soap version 1.2 part 1: Messaging framework (second edition). `http://www.w3c.org/TR/soap12-part1/`, last accessed: 23/10/2008., 2007.

[W3C07b] W3C. Soap version 1.2 part 2: Adjuncts(second edition). `http://www.w3c.org/TR/soap12-part2/`, last accessed: 23/10/2008., 2007.

[W3C07c] W3C. Soap version 1.2 part 2: One-way mep(second edition). `http://www.w3c.org/TR/soap12-part3/`, last accessed: 23/10/2008., 2007.

[WABA03] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.

[Wan03] Ian Wang. Triana conditional looping tutorial, February 2003. `http://www.trianacode.org/docs/looping%20tutorial.pdf`, last accessed: 19/08/2009.

[WCL$^+$05] Sanjiva Weerawarana, Francisco Curbera, Frank Leymann, Tony Storey, and Donald F. Ferguson. *Web Services Platform Architecture*. Prentice Hall PTR, 2005.

[WD09] Neil Ward-Dutton. The time is right for collaborative BPM, 2009. `http://communities.softwareag.com/ecosystem/communities/alignspace/2009/CollaborativeBPMNWD.html`, last accessed 03/02/2010.

[Wey88] E. J. Weyuker. Evaluating Software Complexity Measures. *IEEE Trans. Softw. Eng.*, 14(9):1357–1365, 1988.

[Whi07] Jim Whitehead. Collaboration in Software Engineering: A Roadmap. In *FOSE '07: 2007 Future of Software Engineering*, pages 214–225, 2007.

[Wil06] Mark Wilkinson. Gbrowse Moby: a Web-based browser for BioMoby Services. *Source Code for Biology and Medicine*, 1(4), 2006.

[WL02] Mark D. Wilkinson and Matthew Links. BioMOBY: an open-source biological web services proposal. *Briefings In Bioinformatics*, 3(4):331–341, 2002.

[WM96] Arthur H. Watson and Thomas J. McCabe. NIST Special Publication 500-235: Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric. Technical report, National Institute of Standards and Technology, 1996.

[WMTL06] P.E. Weiseth, B.E. Munkvold, B. Tvedte, and S. Larsen. The wheel of collaboration tools: a typology for analysis within a holistic framework. In *CSCW '06: Proceedings of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work*, pages 239–248. ACM, New York, NY, USA, 2006.

[WSA06] Web Services Addressing 1.0 - Core, May 2006. `http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/`, last accessed 12/10/2009.

[WSB06] Web Services Base Notification 1.3 (WS-BaseNotification), 2006. `http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.pdf`, last accessed 12/10/2009.

[WSR06a] Web Services Resource 1.2 (WS-Resource), April 2006. `http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf`, last accessed 12/10/2009.

[WSR06b] Web Services Resource Lifetime 1.2 (WS-ResourceLifetime), April 2006. `http://docs.oasis-open.org/wsrf/wsrf-ws_resource_lifetime-1.2-spec-os.pdf`, last accessed 12/10/2009.

[WSR06c] Web Services Resource Properties 1.2 (WS-ResourceProperties), April 2006. `http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-04.pdf`, last accessed 12/10/2009.

[WSS06] Web Services Service Group 1.2 (WS-ServiceGroup), April 2006. `http://docs.oasis-open.org/wsrf/wsrf-ws_service_group-1.2-spec-os.pdf`, last accessed 12/10/2009.

[WST06] Web Services Topics 1.3 (WS-Topics), 2006. `http://docs.oasis-open.org/wsn/wsn-ws_topics-1.3-spec-os.pdf`, last accessed 12/10/2009.

[WWDA02] P. Wohed, W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede. Pattern-Based Analysis of BPEL4WS. Technical report, Queensland University of Technology, Brisbane, 2002. QUT Technical report, FIT-TR-2002-04.

[xpa99] XML Path Language (XPath) 1.0, 1999. `http://www.w3c.org/TR/xpath`, last accessed 08/09/2009.

[XSc] `http://www.ebi.ac.uk/~tmo/mygrid/XScuflSpecification.html`, last accessed 30/09/2008.

[Yah] Yahoo Pipes. `http://pipes.yahoo.com`. last accessed 03/04/2008.

[YB05] Jia Yu and Rajkumar Buyya. A Taxonomy of Workflow Management Systems for Grid Computing. *Journal of Grid Computing*, 3(3-4):171–200, September 2005.

[zNS05] Michael zur Muehlen, Jeffrey V. Nickerson, and Keith D. Swenson. Developing web service choreography standards: the case of REST vs SOAP. *Decision Support Systems*, 40(1):9–29, 2005.