Institute of Architecture of Application Systems
University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diploma Thesis No. 2984

Development of an Integrated Database Architecture for a Runtime Environment for Simulation Workflows

Christoph Marian Müller

Course of Study:	Software Engineering
Examiner:	Prof. Dr.-Ing. Dimka Karastoyanova
Supervisors:	Dipl.-Inf. Peter Reimann, Dipl.-Math. Michael Reiter
Commenced:	July 29, 2009
Completed:	February 5, 2010
CR-Classification:	H.3.4, H.4.1, I.6.7

Contents

1 Introduction

Today, computer simulations are one of the cornerstones of scientific research. Computer simulations are utilized for various problems in physics, climate modeling, genomics, and earthquake prediction ([TDGS06], [GKC+09], [Har05], etc.). On the one hand, the methodology of simulations is applied to investigate theoretical hypotheses (cf. [Roh90]). On the other hand, simulations are utilized as a complement to experiments, or even as a substitute, e. g., car crash simulations.

Besides the steady increase of computing capabilities, scientific research faces a huge amount of data that has to be managed ([DG06], [GKC+09]). Furthermore, the scientific community has always tend to build problem specific solutions. Therefore, the reuse of simulation models and data in other projects, or even disciplines, is an issue.

One way to tackle these challenges are scientific workflows and – especially considered in this thesis – simulation workflows. The workflow technology originates from the business process world, offers modular loosely-coupled components, and facilitates service integration ([LR00], [BG06], [MSK+95], [WCL+05]). Moreover, modern workflow systems provide monitoring and administration functions for workflows as well as fault and compensation handling. Thereby, workflow management systems simplify the life-cylcle management of workflows essentially ([GDE+07]).

The scientific community has perceived the potential of workflow technology, since it allows the automation and repeatability of long running scientific processes. The workflow language BPEL (Business Process Execution Language) has been investigated and suggested to be applied for scientific purposes ([AMA06], [TDGS06]). In addition to that, multiple projects have begun to develop more or less generic environments for scientific workflows, such as Kepler[1], Taverna[2], and Microsoft Trident[3].

[1] `https://kepler-project.org/`
[2] `http://www.taverna.org.uk/`
[3] `http://research.microsoft.com/en-us/collaboration/tools/trident.aspx`

1.1 Motivation and Task of this Thesis

In the context of simulation workflows, data management is an important issue. Such workflows do not only have to cope with huge amounts of data, they also have to integrate heterogeneous data sources ([PHP+06]). [HG07] state that simulations spent a lot of time in pre-processing and post-processing of complex models. [Feh10] agrees to this and confirms that a lot of work on modeling and performing simulations is spent in data conversion tasks.

Database technology has been applied in some projects to address data conversion problems and ease data management for simulation models (e. g., [HG07]). Moreover, databases facilitate rich query languages through standardized interfaces and provide transactional access for concurrent simulation activities.

The Cluster of Excellence "Simulation Technology" (SimTech)[4] at the University of Stuttgart conducts research for computer simulations to tackle common issues in this domain. Especially the *Project Network 8* of SimTech investigates how workflow technology and integrated data management can support scientists in the development of simulation workflows. An example for the research of the project network is a solution how monolithic traditional simulation software can be integrated into a runtime environment for simulation workflows.

This thesis is situated in the SimTech context and is concerned with data management issues in control flow-oriented simulation workflows. The thesis investigates how database technology can be utilized to simplify typical data provisioning tasks in simulation workflows. Suggestions are collected and developed how a control flow-oriented workflow environment would benefit of database systems that are tightly integrated into the environment and accessible by workflow tasks. Accompanied by additional services, these database systems form an integrated database architecture. The thesis further suggests application fields that highlight how the integrated database architecture simplifies tasks related to data management or improves their performance.

The ideas developed in this thesis are applied to a runtime environment for simulation workflows based on BPEL as the workflow language and Apache ODE as the workflow engine. The integrated database architecture is conceptualized in general and developed for its application in this environment. The database architecture consists of multiple database systems to support the following requirements:

- Storage of the process information of Apache ODE.
- Storage of the runtime information of Apache ODE, such as events and auditing information.
- Import of external data into the database architecture in
  - relational and
  - XML based formats.

[4] `http://www.simtech.uni-stuttgart.de/`

- Storage of relational and XML based data in
  - external storage based databases and in
  - main memory based databases.

The main memory based databases are supported by the database architecture in order to provide a fast local storage for data-intensive simulation tasks and hence to improve their performance. [Rei08] already considered a local cache for performance improvements in BPEL workflows. This thesis prepares a possible realization of this approach. In order to provide necessary database systems for the integrated database architecture, candidates are collected, analyzed, and selected for their applicability during an evaluation.

The integrated database architecture respects the design of the student project SIMPL[5], which, amongst others, provides a framework to enrich the workflow language BPEL with data management activities. Since the SIMPL project is yet to be finished, the database architecture cannot rely on their implementation. Thus, the thesis provides an own approach to access databases from BPEL workflows.

In order to demonstrate the practical application of the integrated database architecture, this thesis extends example workflows provided by [Rut09]. The workflows utilize a developed Web Service framework, which abstracts the access to traditional simulation software. The extended example workflows show how to orchestrate the steps of traditional simulation software and how their input and output data can be provisioned. This data provisioning is important because it facilitates the seamless integration of such simulation software into simulation workflows.

1.2 Conventions for this Document

In this document, special symbols and font styles are applied. They are briefly introduced in this section.

Abbreviations are explained at their first occurrence. When an abbreviation is used again, the char ↗ indicates that it has already been introduced (e. g., BPEL↗). Appendix A.1 on page 157 contains the list of all abbreviations used in this document.

A DVD is distributed along with this thesis. When a file is mentioned that can be found on the DVD, it is indicated like this: `[DVD]/readme.html`. This HTML file contains a brief overview of the layout of the DVD and thereby helps an interested reader in navigating through the DVD.

Later in the thesis, requirements for the integrated database architecture are introduced and identified with a unique number. A requirement is afterwards referred to like this: ℝ **1**.

[5] SimTech: Information Management, Processes and Languages: `http://www.ipvs.uni-stuttgart.de/abteilungen/as/lehre/lehrveranstaltungen/studienprojekte/SS09/StuPro.SIMPL`

A `monospaced font` is used to indicate in-line code snippets, table names, and file names. Database table names are always spelled with uppercase letters. An example is table `PROTOTYPE.SIMULATION_FILE`.

When a figure or table contains a title, identifier, or term, it is referenced in the explanation text using an *emphasis font*. The same is applied for terms that are meant to be *highlighted*.

The following chapters describe the development of an integrated database architecture for a runtime environment for simulation workflows. Instead of the full title, mostly only *integrated database architecture* or *database architecture* are used as a reference. Moreover, the titles of some developed components start with the prefix *IDARES*, which is only an abbreviation for the title.

1.3 Legal Statements

The software developed during the work on this thesis is published under the Apache License, Version 2.0[6] including the exclusion of warranty and liability. The licenses of all third party libraries required by the software of this thesis remain unchanged.

The DVD distributed beside this document contains a virtual machine based on VMWare disk images. This virtual machine contains commercial software that requires the purchase of specific licenses:

- IBM DB2 Workgroup Server Edition 9.7 for Linux 32 bit and
- ChemShell 3.2

Since the virtual machine automatically starts the IBM DB2 database on system startup, a person has to hold both licenses even when only starting the virtual machine. In order to protect the disk images from unprivileged access, they are protected with a password. A person that passes the password on has to be sure that the receiver holds a valid license, too.

A major part of this thesis contains a database evaluation. The author of this thesis has performed the preparation and the actual evaluation with special diligence. Nevertheless, the author can not rule out that mistakes might have been made. Thus, the author excludes liability for the correctness and completeness of the whole evaluation and its findings.

1.4 Structure of this Document

The remainder of this document is structured as follows: Chapter 2 introduces the core concepts this thesis is build upon. In particular, the chapter provides background information about scientific and simulation workflows. Chapter 3 presents details on the integrated database architecture and collects requirements for its implementation. Furthermore, three scenarios are

[6] `http://www.apache.org/licenses/LICENSE-2.0`

described to develop specific application fields of the database architecture. An architecture of a runtime environment that contains the integrated database architecture is suggested in Chapter 4. Taking all requirements and the architectural environment into account, a database evaluation is performed in Chapter 5. All facts about the realization of the integrated database architecture are then described in Chapter 6. Chapter 7 summarizes the achievements of this thesis and mentions possible future work. The appendix can be found at page 157.

2 Background

This chapter contains background information needed for the understanding of the remaining thesis. Section 2.1 starts with a discussion about the service concept and Web Services as a prominent realization. Section 2.2 goes on with information about the roots of the workflow concept and gives an overview of the workflow language WS-BPEL↗. A major part of this chapter is Section 2.3 that considers scientific workflows, simulations, and provides demonstrative examples. A brief discussion about database related issues can be found in Section 2.4. Software tools and frameworks applied by the implementation of this thesis are presented in Section 2.5. A final discussion about possible database workflow integration mechanisms in Section 2.6 concludes Chapter 2.

2.1 Basics of Web Services

A number of trends in the business world have changed the design principles for software systems in the last decades. [WCL+05] see the highly dynamic markets and the rapidly changing environment as the key factors for that change. To tackle these issues, service orientation and process modeling have proven to be fundamental techniques. Not only the business world, but also the scientific community ([TDGS06]) has shown vast interest in these topics. This section provides a basic introduction to Web Services while Section 2.2 on page 10 covers workflows on the other hand.

The essential parts for this section originate from [WCL+05], [CCMW01], and [BHM+04]. Other sources are referred explicitly.

2.1.1 Services

To understand the basic idea of the term *service*, a commonly known metaphor is used. You can imagine a service as a fundamental supply, such as water or electric power. These services are always available, they are freely accessible, and provide a certain functionality. In addition to that, services can be combined in order to create a new functionality. Besides this ability to compose services, the example illustrates two more characteristics of services: we can search for services providing a special functionality, and we can investigate the details of the services. This metaphor is now transfered to IT services.

The term *service* within the IT domain is standing for an endpoint in the network that provides a specified functionality. To provide this functionality, it receives and replies messages of a

standardized format. Besides the functionality, a service may guarantee additional qualities, such as a maximal latency. Taking all these properties together, a service may be compared with a well defined application programming interface (API). On one side we find an application providing the API, on the other side a client making a request to this API.

The whole service concept has a number of advantages. The provider can offer a service without telling a client anything about its implementation. This is a very powerful concept, since it allows modifications of the implementation while neither having to adapt the interface specification nor the clients. This principle is also called *loose coupling* and facilitates a *technology abstraction*. That is why Web Services are the concept of choice when integrating legacy systems into modern system architectures. Moreover, having a given service definition implemented by two providers, it is possible for the clients to switch the providers easily or change to the endpoint address. This concept is known as *provider abstraction* and preludes the whole concept of a Service Oriented Architecture (SOA).

Service Oriented Architecture describes an architectural style that encompasses all of the concepts above. The key idea of a SOA is that a service provider *publishes* his service endpoint together with service meta information to a centrally accessible service registry (e. g., Universal Description, Discovery, and Integration, short UDDI). A requestor queries the registry to *find* a certain functionality with a given set of non-functional requirements (also called policies) and retrieves a collection of qualified service endpoints in return. Finally the requestor connects to one of the service providers and *binds* to it in order to use it. An optional component called Enterprise Service Bus (ESB) simplifies this whole publish/find/bind process for the requestor by selecting the most suitable service automatically.

2.1.2 Web Services

While the discussion in Section 2.1.1 appears to be abstract, *Web Services* can be understood as a realization of this services concept. In [HB04] the W3C↗ has proposed a definition of Web Services that summarizes its key aspects:

> *A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL↗). Other systems interact with the Web service in a manner prescribed by its description using SOAP[1]-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.*

The W3C has played a key role in the specification of Web Services and related concepts. After having made some adaptions to the initial proposals for WSDL version 1.0, the W3C has officially recommended WSDL version 2.0 ([CMRW07]). In this thesis we utilize version 1.1 of WSDL,

[1] Simple Object Access Protocol

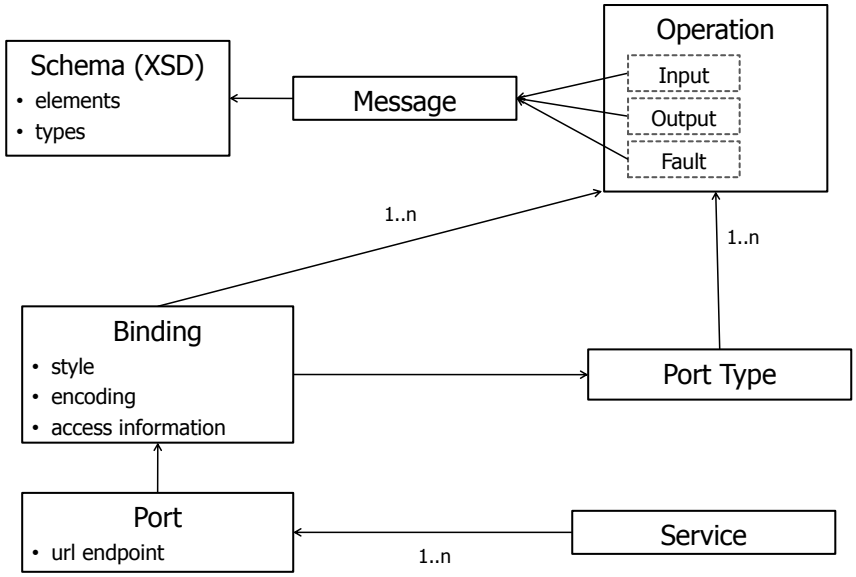


Figure 2-i: Parts of a WSDL file, version 1.1

which is compatible to BPEL↗. For this reason, the further discussion of WSDL is focused on this version only.

A Web Service defines a set of operations that can either be called synchronously or asynchronously. A synchronous call blocks the requestor while waiting for the response. When calling an asynchronous operation, the requestor includes an endpoint reference for itself in the request, calls the operation without being blocked, and goes on with other work. After the called operation has finished, it informs the requestor using the endpoint reference, which is therefore called callback.

[WCL+05] emphasize the strength that lies in WSDL: an industry wide accepted standard for a service oriented architecture. In the following, we briefly present the language structure WSDL. WSDL is an XML based language that specifies what the functionality of a Web Service is, describes how it can be accessed, and contains endpoint information where to access this Web Service. The parts of a WSDL file are shown in Figure 2-i. The following paragraphs explain these parts.

Schema: A WSDL file is XML based and therefore elements are used as a basis for message interchange. Either the schema can be defined within a `<types>` element within the WSDL file or it can be imported from external XSD↗-files. WSDL allows extensibility in that way that extensions can be defined within the WSDL file enabling non-XSD type systems.

Message: The `<message>` elements define the content of messages exchanged for requests, responses, and fault situations. They refer to the schema specified before and use its elements or types.

Operation: An `<operation>` defines a method by its name and the related request and response messages. In addition, an operation may define `fault` messages representing possible

exceptional situations. WSDL version 2.0 requires that an operation implements exactly one of the provided Message Exchange Patterns (MEP), such as *in-only* or *in-out*. While the *in-out* pattern represents synchronous operations, the *in-only* pattern is mostly used for asynchronous calls that do not expect an immediate response.

Port Type: A `<portType>` is a collection of correlated operations. These operations can provide access to the same entity, concern the same scenario, or play in the same scope.

Binding: A `<binding>` defines how a port type is offered to the client, which protocols are used for transportation, and how the messages have to be serialized. SOAP over HTTP is the most common transportation mechanism, as it allows to send structured and typed data via a widely used protocol. A helpful discussion on the appropriate binding styles and encodings for SOAP can be found at [But05]. Besides HTTP, other transports such as SMTP↗ or JMS↗ are possible, too. Finally, a binding specifies how an operation can be invoked, e. g., by defining a SOAP action or the HTTP `GET` method.

Port: A `<port>` is the sole part within a WSDL file mentioning the URL where the operations of a port can be accessed. Since a port depends on the used transportation mechanism, it refers to a specific binding and thus to a specific port type.

Service: The sole purpose for the `<service>` element is to encapsulate a number of ports (respectively endpoints).

The WSDL syntax uses the XML-namespace mechanism in order to reference named parts, such as a port name. In addition to that, extensions to WSDL are not build into the WSDL namespace directly. Bound extensions, e. g., SOAP or HTTP, define their own namespace and are used within the WSDL binding element. The same way, other Web Service extensions can be declared within a WSDL file – WS-Policy ([W3C07]) is just one example.

2.2 Basics of Workflows

As workflows were originally used in the domain of business processes, we take a look at this aspect first and then inspect WS-BPEL↗ as a workflow language within this section. A distinction to scientific and simulation workflows is covered in Section 2.3 on page 16.

The descriptions are based on [LR00] and [JE07], enriched by thoughts of [WCL+05]. Other sources are referred explicitly.

2.2.1 Business Processes and Workflows

In [Wor99] we find a suitable definition for the term *business process*:

> A set of one or more linked procedures or activities which collectively realize a business objective or policy goal, normally within the context of an organizational structure defining functional roles and relationships.

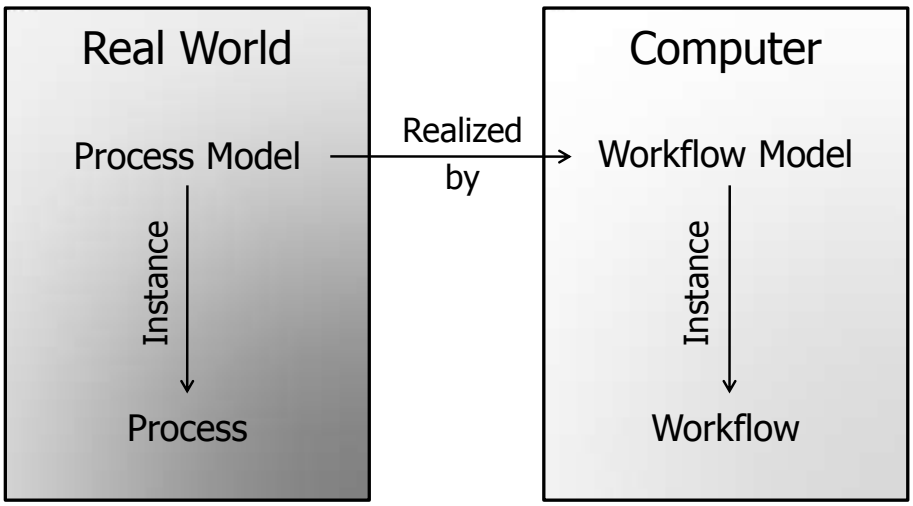


Figure 2-ii: Distinction between process, workflow, and their models; following [LR00]

Often such processes are repeated a number of times, with slight adaptions of the activities or other parameters. We can say they follow a common pattern that we call *process model* or *business process model*. A process may consist of steps that an employee is performing or that are executed by a computer software. These computer tasks can be represented by a *workflow* and, following [Wor99], we define the term *workflow* as:

> The execution of a process on a computer, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules.

As there are process models, there are also workflow models (see Figure 2-ii). The latter can be seen as the implementation of a business process model that can be interpreted and executed by a workflow management system or also called workflow engine. As there are steps in the process model that are directed to an employee, the workflow model can adopt these human tasks in some form or describe only a part of the process model. Based on the description of the model, one or more instances can be created. They represent a concrete execution based on the model.

To understand and study the aspects of workflows in more detail, [LR00] have proposed a three dimensional view of workflows that is illustrated by Figure 2-iii on the next page. The *process logic* dimension of a workflow describes which activities have to be performed in which order and having which dependencies on each other. If the process logic is based on Web Services, the term *orchestration* is often used to characterize this composition of services ([Pel03]).

The dimension *organization* defines who should perform a task. Typically a person is responsible for a workflow task, because he or she assigned to a certain role or job function within the organization. Besides the human responsibilities, a workflow management system might also perform a process step without human intervention.

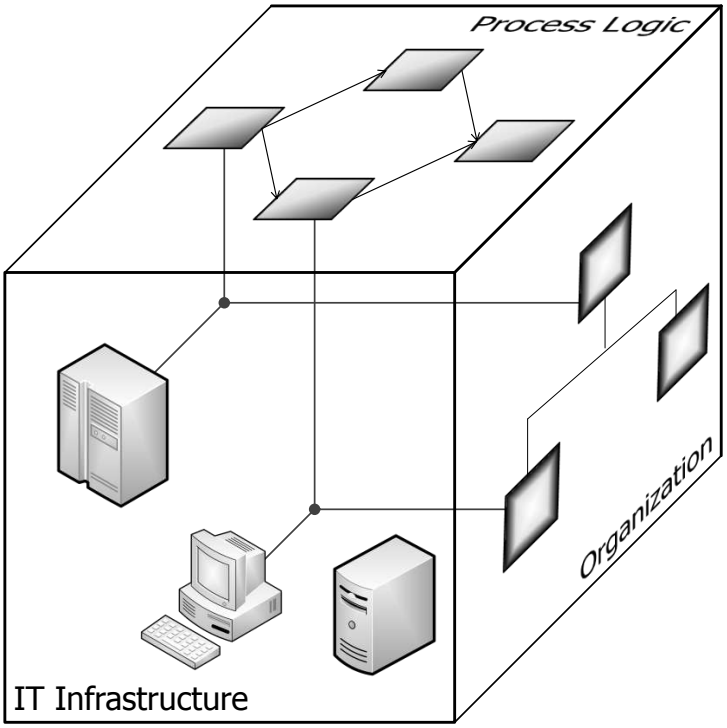


Figure 2-iii: The three dimensions of workflows by [LR00]

The third dimension defines, which IT resources are used to perform an activity. Therefore, the third dimension is called *IT Infrastructure*. The integration of databases into workflows, which is considered in this thesis, concerns exactly this dimension.

To define business processes, a number of meta models and languages have been developed. Some of them specify the control flow others are more concentrated on the data dependencies and model the data flow. Both perspectives have their advantages. In the course of this thesis we want to highlight WS-BPEL↗ that is described in the next section.

2.2.2 Web Services Business Process Execution Language Version

On the one hand, WS-BPEL↗ is a standardized language specifying the structure of a business process. That is why it is called "Business Process Execution Language". On the other hand, WS-BPEL is the language of choice for high level, control flow oriented composition of Web Services, also called orchestration.

WS-BPEL, or short BPEL, is an XML based workflow language whose current version 2.0 has been standardized by OASIS[2] in 2007 ([JE07]). From the beginning of its development, BPEL

[2]Organization for the Advancement of Structured Information Standards: `http://www.oasis-open.org/`

Listing 2.1 Syntax of a BPEL file: the process and scope element

```
<process>
  <extensions/>?
  <import/>*
  <partnerLinks/>?
  <messageExchanges/>?

  <variables/>?
  <correlationSets/>?

  <faultHandlers/>?
  <eventHandlers/>?

  activity
</process>
```

```
<scope>
  <variables/>?
  <partnerLinks/>?
  <messageExchanges/>?
  <correlationSets/>?

  <eventHandlers/>?
  <faultHandlers/>?
  <compensationHandler/>?
  <terminationHandler/>?

  activity
</scope>
```

was integrated into the whole Web Service concept (see Section 2.1.2). A BPEL process is the composition of services that are described via WSDL, while a BPEL process provides WSDL port types to other participants, too. BPEL processes extensively exploit a concept called "late service binding". This is done by binding the services via deployment descriptors at deploy time or even at runtime by exchanging service endpoints specified via WS-Addressing ([GHR06]). Moreover, BPEL supports long running synchronous and asynchronous communication to other Web Services.

To sustain long running workflows, the BPEL standard considers a lot of aspects related to recoverability and maintainability such as fault handling and a compensation mechanism. In addition to that, BPEL is based on a life cycle management that applies to each process instance from the creation to its termination, completion, or faulting (see [KKS+06]). In this context every activity is persistently managed in order to allow suspension of processes and resuming them later.

We now have a look at the most important syntax elements of the language and explain special aspects of BPEL at the most appropriate elements. Listing 2.1 shows the essential syntax of the two main environment elements `process` and `scope`.

The BPEL language applies the concept of scopes to apply the visibility and range of variables and handlers. Scopes can be recursively nested and used where ever an activity is valid. The root scope of a BPEL process is defined by the `process` element and has a slightly different syntax than the regular `scope` elements. The `process` element defines global imports to external schema definitions or WSDL files. BPEL supports a sophisticated extension mechanism, which makes it is possible to import extensions – e. g., new activity types – within a custom namespace and declare them in a BPEL file. An examples to highlight is *BPEL4People* ([AAD+07]) that describes human user interactions.

A `scope` can specify so called fault handlers in the `<faultHandlers>` element, which are executed when faults are caught within their range. Moreover, a `scope` may define a

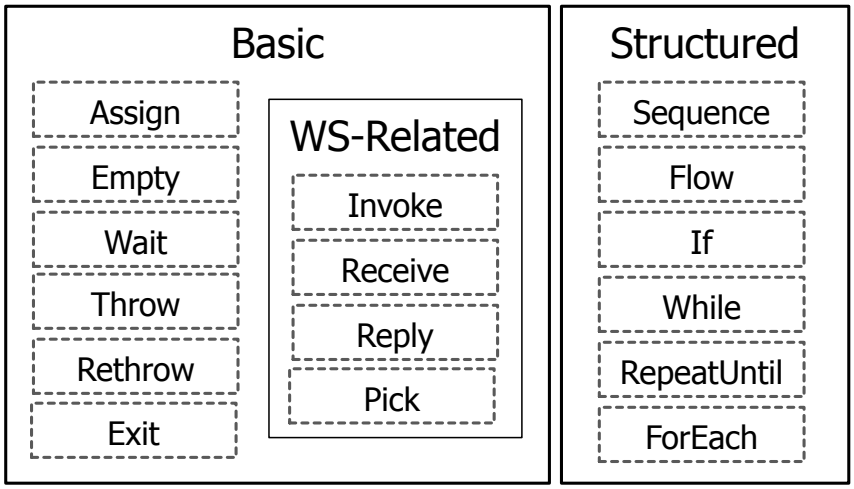


Figure 2-iv: WS-BPEL: Activity overview

`<compensationHandler>`, which tries to reestablish a valid state after a fault has happened in the scope. This is necessary when an activity within the scope has its own transactional semantic and is executed successful with a commit. As the committed transaction is durable, a reverse activity is needed, e. g., an order activity can be undone by sending a cancellation message. A `<terminationHandler>` can specify an additional activity, which is executed when the related scope is forcefully terminated.

In addition to handlers, scopes define variables that are very similar to their counterparts in the most common programming languages. They are named, only visible within the range of the scope, and they are typecasted. XSD elements and types can be used to specify the type of a variable. BPEL variables can also be typecasted with a WSDL message type. Variables can be used as input and output for Web Service calls or the various condition elements of BPEL activities. In order to access sub-elements or attributes within a structured variable, the language XPath 1.0 ([CD99]) is used within BPEL. The only restriction is that a variable must be initialized before its first usage, i. e., a structure complying with the declared schema type must be set before the variable is read the first time.

The main work of a process is done by activities. The activities can be subdivided into basic and structured activities as shown in Figure 2-iv. Basic activities describe elemental steps of the process behavior. Structured activities are activities that control how sub-activities are executed. They specify whether there is a condition deciding which sub-activities are executed, whether sub-activities are executed concurrently or sequentially, and how they depend on each other. Every activity – either basic or structured – can be used as a sub-activity.

For each BPEL workflow, there is at least one activity defining the entry point. This activity describes incoming messages or event situations that instruct the workflow engine to create a new instance of this workflow. The following paragraphs explain the most common activities briefly.

Receive: The receive activity represents the handling of an incoming WSDL message. For this purpose the activity defines a WSDL operation and optionally a port type, which is offered by the process. A concept called *partner links* helps to specify the sides of a bidirectional partner relationship a process participates in. When multiple instances of a process prepare to receive a message for the same WSDL port and operation, the workflow engine has to choose the instance the message is routed to. For this reason, the *correlation set* defines a part of the received message that serves as an identifier for the conversation. The receive activity may serve as an entry point of the process in order to create a new instance.

Reply: The reply activity is the counterpart of a receive. The same way it defines a WSDL operation, optionally a port type, the related partner link, and how to bind the response message. In order to preserve a bidirectional communication, the reply activity has to indicate the correlation set of its receive counterpart. When there are multiple possible receive-reply pairs for the same partner link and operation, so called *message exchanges* distinguish these pairs. The same is applied for other incoming message activities other than receives, such as pick or `<onEvent>` handlers.

Invoke: The invoke activity represents a Web Service call of another partner. An input variable or an equivalent `<toParts>` construct has to be provided and partner links and correlation sets have to be considered. If the invoke calls a synchronous Web Service, an output variable or an equivalent `<fromParts>` construct has to be specified. Otherwise, the invoke represents an asynchronous call and does not wait for a response.

Assign: The assign activity is used to update the content of variables. It allows a list of `<copy>` elements that each represent an assignment specification. Each assignment has a source, defined by a `<from>` element that can refer to a variable, a message part, an endpoint, a literal, or an XPath expressions for sub element selection or calculation. Except for the literal, the same references can be applied to the target of the assignment, defined by a `<to>` element. The assign activity has atomic behavior; i. e., either all assignments are performed or none.

Empty: The empty activity does nothing at all and is used as a dummy in various situations.

Wait: The wait activity can be used within a workflow to wait for a given period of time or until a deadline is reached.

Exit: Using the exit activity, a process instance is terminated instantly. No handlers are executed and all activities are ended immediately.

Pick: The pick activity is similar to the receive activity, but allows the specification of multiple events, including incoming messages or timer-based alarms. Each of event specifies an activity that is executed, when the event occurs. A pick activity waits for the all events, but only executes the activity of the first event. A pick activity may serve as an entry point of the process in order to create a new instance.

If: The structured activity if resembles a conditional branching statement. It contains a sequence of branches that each are activated by a condition defined by an XPath expression. Only

the first condition that evaluates to true activates its related branch. An optional `<else>` element introduces a branch if no condition has been true.

While: The structured activity while represents a loop whose condition is evaluated at the beginning of each iteration. If the condition is true, a sub-activity is executed and the loop condition is evaluated again. This is repeated until the condition is evaluated to false.

RepeatUntil: The structured activity repeatUntil represents a loop whose condition is evaluated at the end of each iteration. Thus, the sub-activity is executed at least one time.

ForEach: The structured activity forEach represents a loop with a counter variable that is incremented for each iteration. At the initialization of the forEach activity the start and the final value of the counter are determined. For each incrementation of the counter variable the sub-scope is executed. A special feature of the forEach activity is to run all sub-scopes concurrently.

Sequence: The structured activity sequence contains a list of sub-activities that are executed in that specified order.

Flow: The structured activity flow contains a set of sub-activities that build up an acyclic dependency graph. Therefore, the flow specifies named links between the activities and executes sub-activities by following the links. The flow concept is a very powerful technique to achieve concurrency within the process. Additional transition and join conditions enrich the link concept by evaluating boolean expressions for the activation of the links.

BPEL in the version 2.0 is utilized as the orchestration language for workflows in the scope of this thesis. For more details on the BPEL language specification, the reader is referred to [JE07].

2.3 Scientific Workflows and Simulations

After the workflow technology has evolved in the business world, the scientific community has adopted this concept in many ways. This section investigates motivations for using workflows in scientific domains and compares business and scientific workflows. As the task of this thesis covers a simulation workflow environment, we examine simulation workflows in detail. The section goes on with a brief introduction to finite element method. Afterwards simulation frameworks and examples for simulations are presented.

2.3.1 Scientific Workflows

The term *scientific workflow* contains the word `workflow`, which we have already defined in Section 2.2.1 on page 10. Hence, we start our discussion with a definition of the term derived from [LAB+06] and the workflow definition by [Wor99]:

A scientific workflow is:

(i) a formal description of a process for accomplishing a scientific objective, usually expressed in terms of tasks and their dependencies, as well as

(ii) the execution of this process on a computer using a workflow environment.

When analyzing this definition, we can conclude two requirements for scientific workflows. First, a scientific workflow needs an objective in the scientific rather than in the business world. This can be, for example, a complex calculation, a simulation, or a data analysis. Second, a scientific workflow is based on the process concept of linked tasks that collectively achieve a scientific objective. Based on this argumentation, many scientific projects claiming to be scientific workflows satisfy the first, but not the second requirement. Thus, neither monolithic software tools, which do not constitute coupled activities, nor scripted analyses that are not executed in a workflow environment represent scientific workflows in the understanding of this thesis.

Scientific workflows are applied for a great number of reasons we now investigate more deeply. [DG06] describe a dilemma of scientific research. On the one hand, there is a tremendous growth in the performance of computers, data storages, and other system elements, while on the other hand, its adoption by scientific projects is growing much slower. [DG06] conclude that a lot of scientific work, done in a sequence of manually performed activities today, would benefit of workflow technology in many ways. Workflows allow the automation of scientific processes and hence their repeatability – a cornerstone of the scientific method ([DG06]). [LAB+06] agree and state that, although the major task of scientific workflow engines is the execution of such processes, they also document the sequence of the performed activities and their output, which can facilitate provenance. This is another cornerstone that characterizes scientific work: achievements are reproducible and shared in the community.

Besides the increased performance of computers, there are other reasons for the application of scientific workflows. The amount of data for scientific analyses is growing rapidly. [GKC+09] note that during experiments with the Large Hadron Collider 1 PB/s of raw data is streaming from sensors into a grid of computer nodes for later analysis. [GLNS+05], [Blu], and [LAB+06] confirm this development, which is not only caused by data of sensors, but also by models for scientific simulations, which are typically getting more complex and powerful. Furthermore, the integration of distributed heterogeneous data sources are another challenge for scientific work today ([TDGS06]).

Workflow automation can tackle these challenges – but workflow environments are capable of much more. [LAB+06] address optimization and efficiency aspects to a scientific workflow environment as well as the accessibility and reuse of components across projects. [MSK+95] call attention to the trend that workflow environments have to integrate components that are getting more and more independent, by orchestrating them into a set of loosely-coupled collection of activities. Another trend is that capabilities such as transaction support and monitoring of workflow instances are adopted from business workflow environments ([BG06]).

But scientific workflows have many characteristics that are significantly different to business workflows, as [BG06] and [LAB+06] showed. In the scientific world the use of standards has been neglected for a very long time. It was not uncommon to reinvent technology – especially in the domain of workflows – specific to a current problem. This has led to a vast number of

tools, scripts, and frameworks in the scientific domain that are each suited for just small branches of domains, e. g., molecular calculations of proteins. In contrast, the business world has very determinedly aspired to fulfill technology standards. But with the development of inter-domain scientific workflow workbenches (cf. simulation frameworks in Section 2.3.4 on page 24), the scientific world is attempting to follow suit.

Furthermore, we can emphasize a reason for the differences between scientific workflows and business workflows, which is not a technological one: the exploratory nature of science ([DG06]). While business process developers have a given goal their processes have to achieve, scientists often tend to create workflows in a trial and error principle. They require a very flexible workflow environment where they can develop a prototype of the workflow very rapidly. Moreover, scientists like to pause running workflow instances, revise parameters or even the structure of the workflow, and resume its work. Furthermore, scientists are experts in their specific domain rather than experts in information technology. They have their own way of thinking and modeling, which is a reason why they prefer data flow oriented workflows. On the contrary, business developers favor high level programming, meaning modeling the control flow with attached data artifacts.

The last considered difference concerns the properties of the workflow instances. While business processes typically have a large number of short running instances, scientific workflows may only have one instance. Either the instance does not satisfy the scientist and he or she has to modify the process, or the instance has achieved his or her objective and no further instance is needed. In addition, scientific workflows are often running for a very long time, are highly concurrent, and incorporate multiple distributed data and computing nodes ([BG06], [PHP+06]). Especially the grid technology has proven itself as the distributed computing infrastructure of choice for large-scale, data-intensive simulations and analyses, as it allows a transparent and coordinated view on shared resources ([FKT01], [TDGS06], [JXW08]). In the context of such distributed systems, stateful Web Services are beneficial. Here, [BFA08] highlight the Web Services Resource Framework (short WSRF, see [OAS06]) as a prominent solution.

In the consequence, scientists have begun to adopt proven technology from the business world and extended it for their needs where required. [Slo06], [AMA06], and [WEB+06] are examples for this approach, as they have investigated the use of the workflow language BPEL for scientific workflows, too. Even suggestions for the integration into a grid architecture were made. Likewise, the SimTech project network 8[3] has, amongst others, dedicated its work to the research of data provisioning techniques, modeling and runtime environments in the context of simulation workflows.

We now take a brief look at the types of scientific workflows, described by [LAB+06]. Table 2-i on the facing page shows the four dimensions we use to describe different characteristics of scientific workflows. In a first differentiation, [LAB+06] distinguish *exploratory* workflows, which resemble the trial and error approach, and *production* workflows, which are executed on a regular

[3] SimTech PN 8: Integrated data management, workflow and visualization to enable an integrative systems science: `http://www.simtech.uni-stuttgart.de/forschung/projekte.php`

Dimension	Domain	
Sophistication	exploratory	productive
Orientation	science-oriented	resource-oriented
Load	data-intensive	compute-intensive
Program Flow	control flow	data flow

Table 2-i: Classification of scientific workflows based on four dimensions; sophistication and orientation follow [LAB+06]

basis with varied input. The latter are often used as sub-workflows for routine calculations. The next dimension, considers the way of workflow modeling. On the one hand, there are *science-oriented* workflows, whose activities represent high-level steps of a scientific method – each step is meaningful for the scientist. *Resource-oriented* workflows on the other hand have the sole purpose of handling and analyzing data resources directly. Here, the workflow orchestrates data movement and management tasks rather than knowing about their contribution to the superordinate scientific goal. [LAB+06] also distinguish *data-intensive* from *computation-intensive* workflows, this way focusing on the main load of the workflow during execution. The dimension of the model of programming, whether *control flow-oriented* or *data flow-oriented*, was added in this thesis for the sake of completeness.

As different as scientific workflows might be, it becomes more and more important to share proven artifacts within the scientific community. In this context, [LAB+06] highlight workflow descriptions as *valuable knowledge assets* and expects the growth of workflow reuse and the increasing importance of social sharing within the scientific community.

In this section, we have taken a brief look at scientific workflows in general and presented one possible classification. In the following section, we inspect simulations and simulation workflows, which are a subgroup of scientific workflows.

2.3.2 Simulations and Simulation Workflows

This section considers simulations that are carried out on a computer, inspects the related computational model, and classifies simulation workflows.

We start the discussion by looking for a pertinent definition of the term *simulation*. The first one is taken from [IEE90] and defines a simulation as:

(i) A model that behaves or operates like a given system when provided a set of controlled inputs.

(ii) The process of developing or using a model as in (i).

In this thesis, we agree to this definition, but understand the model of (i) as a *simulation model* and consider a simulation only as the process that uses this model. In contrast to (ii) the development of such models are no simulations in the narrow sense. In addition to this definition, we quote one from [Hum90], too:

> A computer simulation is any computer-implemented method for exploring the properties of mathematical models where analytic methods are unavailable.

This definition is just oriented towards mathematical models – a fact we shall extend later – but sets the appropriate focus, because we consider only computer-implemented simulations in this thesis. When comparing both definitions, it is apparent that simulations rely on the existence of a simulation model. A general model theory has been introduced by [Sta73], whose thoughts shall inspire the following paragraph.

The basic idea of modeling is to create an description of a real world object by *mapping* its attributes. A model might also represent an object that is not even real or that is a model itself. Models can have different forms: a mathematical formula, a technical drawing, or a photograph. The mapping defines three classes of attributes:

- *relevant* attributes are the same for the original and the model; e. g., the ratio of the metrics are the same for a car and its technical drawing model
- *omitted* attributes of the original, which are not mapped to the model; e. g., the color of the original car
- *superfluous* attributes of the model, which are added during the mapping process; e. g., the line thickness of the technical drawing

Models are created for a number of reasons – the most common are:

- *representation*: a model visualizes the original and can even be used to build the original
- *simplification*: a feature of the original can be explained or examined more easily when unnecessary attributes, and therefore complexity, are omitted
- *pragmatic orientation*: the model can substitute the original in a given context, because the original is not applicable, not attainable, or does not even exist

In the context of simulations, the *pragmatic orientation* and the *simplification* are the most common application fields of models. As [Hum90] states in his definition, a computer simulation model is typically a mathematical model. When looking at mechanics, the model is often specified by partially differential equations and systems of equations. Such models are called white or glass box models, as they are derived from known quantifiable relations between the *relevant* attributes. But a lot of systems cannot be modeled with perfect accuracy, because the relationship between the attributes are unknown or the model would be too complex. We call this type of models black box models. But as the simulation world is neither black nor white, we have to relativize this distinction: most models are somewhere between a perfectly mapped reality and an empirical model.

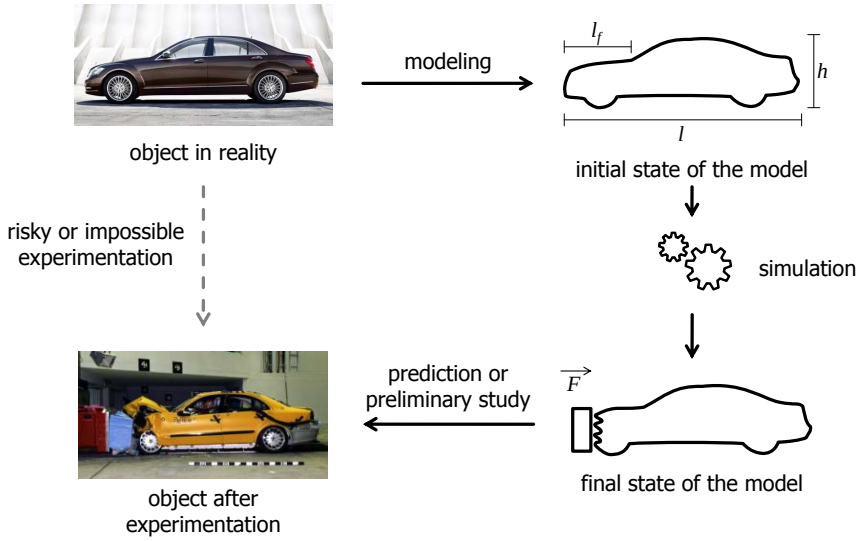


Figure 2-v: Car crash test simulation: bringing together model theory and simulation; following [LL07]; pictures originate from *mercedes-benz.de* and *carsarmored.org*

Typically, the first task in preparing a simulation consists of modeling the real world (see example in Figure 2-v). For many simulations, model creation involves multiple steps. The simulation can use the model to inspect its behavior under a given set of inputs. These inputs may represent environmental variables or a specific state of the original. During the simulation the state of the model and the model itself are put together to calculate a resulting state. Computer simulations often investigate the behavior of the model over the time scale and therefore calculate multiple intermediate states of the model. Using the final state of the model after the successful simulation, one can make predictions of the original object under the same set of inputs. Referring to the illustration in Figure 2-v again, we can predict how the form of the frame of the real car would have been deformed. Using simulations, we can minimize the number of expensive experiments or even avoid them entirely. In other cases, where we cannot experiment with the real object, a simulation is the only choice to study the object.

[Har05] lists possible functions of simulations and states that simulations are used as a:

1. technique: investigate details of a system, which cannot be done pragmatically in experiments (the system is too complex or the object is not available); example: black hole simulations in astrophysics
2. heuristic tool: the simulation technique is used to develop hypotheses, models, and even theories by extracting knowledge about a system derived in multiple simulation runs; example: simulating the dynamics inside physical particles to build up new theories about their bonding
3. substitute for an experiment: simulations in regions of a parameter space that are inaccessible by normal experiments; example: simulate a tax increasing by 100 %

4. tool for experimentalists: simulations can inspire and analyze experiments; example: simulate why the bus controller of a car might have crashed during an experiment
5. pedagogical tool: using simulations to teach students by letting them *play* within a simulated environment; example: SESAM[4]

We consider simulation workflows to be a subgroup of scientific workflows – a distinction that is not interpreted too strictly, because workflows for car crash simulations have their objectives in the business world, but are considered as simulation workflows, too. Having mentioned this, we use the term *simulation workflow* within this in the following sense:

> A simulation workflow is a *scientific workflow* that is based on a *simulation model* and calculates the behavior of this model for a given set of inputs.

As this statement relies on previously defined terms, we can directly discuss its consequences. A simulation workflow *in the narrow sense* is *science-oriented* (with the words of [LAB+06]), as its activities represent the simulation steps upon its model. But a lot of scientific simulations rely on massive data preparations prior to the simulation and on compute-intensive calculations of the steps during the simulation. These activities are usually delegated to other sub-workflows. For this reason, we consider such sub-workflows as simulation workflows *in the broader sense*, as their objective concerns the simulation indirectly.

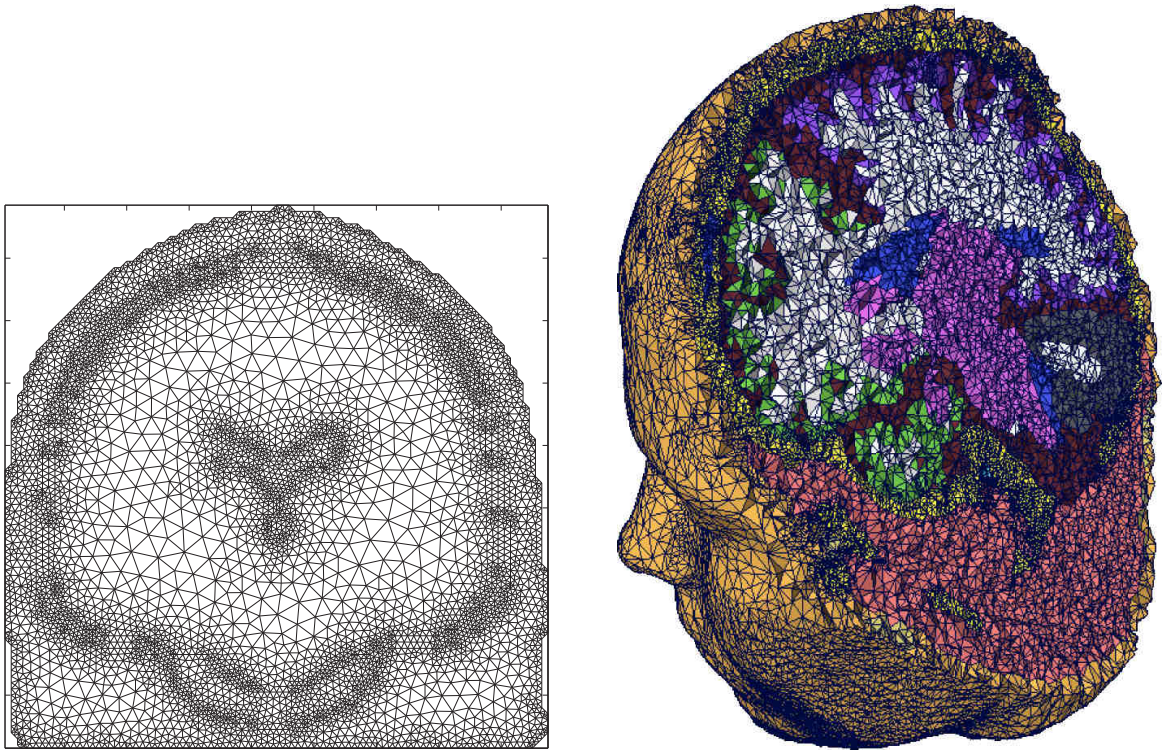
2.3.3 Finite Element Method

A lot of simulation models in engineering and physics are based on differential equations. Examples for such models are the kinematics of an automobile, the air flow on the wing of an airplane, and the heat transfer between gases. As the finite element method (short FEM) is used in several example workflows, we present a brief overview of its basics within this section.

In order to solve differential equations in simulation models, an analytical solution is very complex to find. Another approach to solve differential equations – especially partially differential equations (PDE) on spatial models – is the discretization of the problem. Instead of trying to solve equations on an unknown or very complex shape in the model, the shape is split into a finite number of simple geometric shapes – typically triangles for a two-dimensional model. They are called finite elements. As Figure 2-vi-a on the facing page depicts, in a two-dimensional model, we can create a grid – or sometimes called a mesh – of a finite number of nodes. When connecting three adjoining nodes of the grid with an edge, triangles appear. Figure 2-vi-b on the next page shows the same approach on a three-dimensional model using tetrahedrons.

Using these finite elements, we can now use numerical methods to make an approximation of the original model. We start by applying the equations of our problem to some or all of the nodes of our grid. A set of equations describes the *value* of a node at time t_i, where the value can be considered as the state of the model. Combining all equations into a system of equations, we

[4] Software Engineering Simulation by Animated Models:
`http://www.iste.uni-stuttgart.de/se/research/sesam/`



(a) FEM model of a two-dimensional coronal slice of the head using triangles [HVG+07]

(b) FEM model of an arbitrary cut through a tetrahedral meshed volume of the head [BSP+08]

Figure 2-vi: Two examples of a 2D and 3D finite element model of a human head

can make assertions for the values of the whole model at time t_i. The system of equations is represented as a matrix and is solvable with numerical methods. When applying the initial value of the model at t_0 to the system of equations, we can calculate the value of each node – and of the whole model – at t_1. This procedure can be iterated a number of times, until the end time t_n is reached.

An important issue that arises for each discretization, is the precision. If the discretization is very precise, a large number of nodes and finite elements are the consequence. This leads to a better approximation, but increases the computational effort to solve the equations. As the number of equations to solve increases with the product of the number of nodes and the number of discrete time steps, the effort tends to grow enormously with a finer grid.

Numerical methods provide an estimation of the precision of a finite element model by calculating the order of the error between the approximation and the original model. This approach can be used as a basis for different optimization techniques of the finite element method. If some nodes are more sensitive to the input, the grid of finite elements can be refined for these nodes, while the rest of the grid remains unadapted. This approach is illustrated by Figure 2-vi-a on this page, where the triangles in the middle and at the edge of the figure use a finer grid than the rest of the

triangles. This local refinement increases the quality of the approximation only where necessary and keeps the computational effort at an appropriate scale.

This section covered only an overview of the topic FEM to ease the understanding of some following simulation examples. The finite element method is accompanied by further techniques, such as finite volumes or finite differences. Moreover, adaptive algorithms that automatically estimate the order of the error and refine the mesh where the error could be too large cannot be discussed in detail within this thesis. The interested reader is referred to special literature on the topic, especially [ZTZ05].

2.3.4 Simulation Frameworks and Simulation Workflow Management Systems

Today's simulations are often based on software frameworks that have been developed for a specific application field, e. g., finite element method (FEM). These frameworks distinguish themselves by the way how simulations are developed by the scientist. Some frameworks depend on low level programming, but support the scientist with powerful libraries and structured templates. Other frameworks allow a high level visual modeling where the scientists define activities and link them together into a sophisticated graph-like representation.

As simulations tend to result in long-running computations, distributed programming is supported by more and more frameworks. Simulation frameworks offer different methods to split the workload into multiple processes on clustered or loosely coupled computer nodes and they also handle the communication between them. In this context, grid computing has become the state of the art technique to achieve reliable distributed computing in the scientific world. Hence, sophisticated frameworks implement grid support in any form.

Besides the fundamental features, some frameworks offer sophisticated tools for the administration and monitoring of running simulations. As more and more scientists become aware of the importance of the topic provenance, frameworks record rich auditing trails or even come with an own provenance tool chain. Finally simulation frameworks often let the scientist specify data import or export activities and provide domain specific visualization components.

In the following sections, some simulation frameworks are presented that either are used in this thesis or have importance in the world of scientific simulation today or possibly tomorrow. Work of [LAB+06] and [TMMF09] was found to be useful for the preparation of this section.

DUNE

DUNE 1.x[5] (Distributed and Unified Numerics Environment) is a software library that provides a number of modules for solving partial differential equations. For that purpose, the DUNE toolbox contains implementations of grid based techniques, such as finite elements, finite volumes, and

[5] http://www.dune-project.org/

finite differences (see Section 2.3.3 on page 22 for an overview of this topic). Besides, there are various solvers for linear equations as well as iterative solvers and modules for vector and matrix calculations. Finally a collection of parsers and readers for different file formats are contained in the library. The DUNE library supports various file formats for export, such as Visualization ToolKit[6] (VTK) and DUNE Grid Format (DGF) for finite element grids.

The core of DUNE is a C++ library that is supplied as source code. The creation of a new DUNE simulation project consists of the following steps:

1. Create a new module with a name and a given version.
2. Define dependencies to DUNE modules of the library that are needed by the simulation, e. g., a solver for linear equations.
3. If needed, implement custom solvers or helper modules, which implement abstract interfaces of DUNE.
4. Implement a main routine that uses the modules to describe the simulation steps.
5. Define and use file exporters for the result data, e. g., for a subsequent Visualization ToolKit import.
6. If needed, adapt or create the configuration file specifying the build sequence.
7. Compile the required DUNE modules, the own modules, and the main routine using the GNU build system[7].

The result of the compilation is an executable application that can perform the simulation steps. As the developers of DUNE have spend a great effort in efficiency, a lot of compiler optimization techniques are exploited. For that reason, an application created with DUNE is optimized for a single simulation problem.

On the downside, the slightest reason for an alteration of the simulation steps forces the scientist to repeat the whole adaption and compilation procedure. Thus, DUNE applications are monolithic and cannot be executed as a step by step simulation workflow. Therefore, it is difficult to use only parts of a DUNE application within a workflow, e. g., based on BPEL. Section 2.3.5 on page 29 suggested an approach how a DUNE application can be accessed via Web Services.

For more details on DUNE, the reader is referred to the DUNE project website `http://www.dune-project.org/` and the DUNE HOWTO ([BBD+09]), which contains a guide for the development of an example simulation using DUNE.

[6] `http://www.vtk.org/`
[7] `http://www.gnu.org/software/libtool/manual/automake/`

ChemShell

ChemShell 3.x[8] is a computational environment for the chemistry domain. ChemShell supports quantum mechanics (QM), molecular mechanics (MM) as well as hybrid quantum mechanics/-molecular mechanics (QM/MM) calculations.

ChemShell claims itself as a director that delegates time-consuming analyses to external tools with specialized implementations. While the external tools are, for example, responsible for energy evaluations, ChemShell handles the communication between these tools and the file handling. Therefore, ChemShell has integrated routines for file transformations, data handling, and visualization. Moreover, ChemShell can handle simple calculations and numeric problems itself.

The ChemShell framework is based on the scripting language TCL[9] (Tool Command Language). Hence, ChemShell provides an interactive shell where the scientist can enter commands, which are then interpreted and executed. A number of commands are offered by ChemShell – customized commands can be implemented in the programming languages C or Fortran. Using the interactive shell, the scientist can perform his or her simulation step by step and decide about the next command during the simulation.

In order to automate the simulation, a script can be used too. Such a script has to contain a fixed order of commands, which are executed one after another. When ChemShell is started with a script as parameter, it executes all commands automatically and shuts itself down after the last command.

Since ChemShell is based on simulations defined by scripts or user interactions, the ChemShell environment needs to be compiled only once. The simulation processing is performed by the TCL interpreter and hence no compilation of the simulation scripts is required. Only when new commands or other external modules are added, the ChemShell environment needs a new compilation. Given a set of fixed modules for a specific problem domain, the script representing the simulation is the only artifact to be altered.

For more details on the ChemShell toolbox, the reader is referred to the ChemShell project website: `http://www.chemshell.org/`.

Kepler

Kepler 1.x[10] is an open source scientific workflow management system with a wide support for various scientific and engineering disciplines. Kepler offers a Java-based graphical user interface (see Figure 2-vii on the next page) where the scientist can compose a workflow in a graph like manner. The nodes of the graph are typically data processing or data modeling activities, such as

[8] `http://www.chemshell.org/`
[9] `http://www.tcl.tk/`
[10] `https://www.kepler-project.org/`

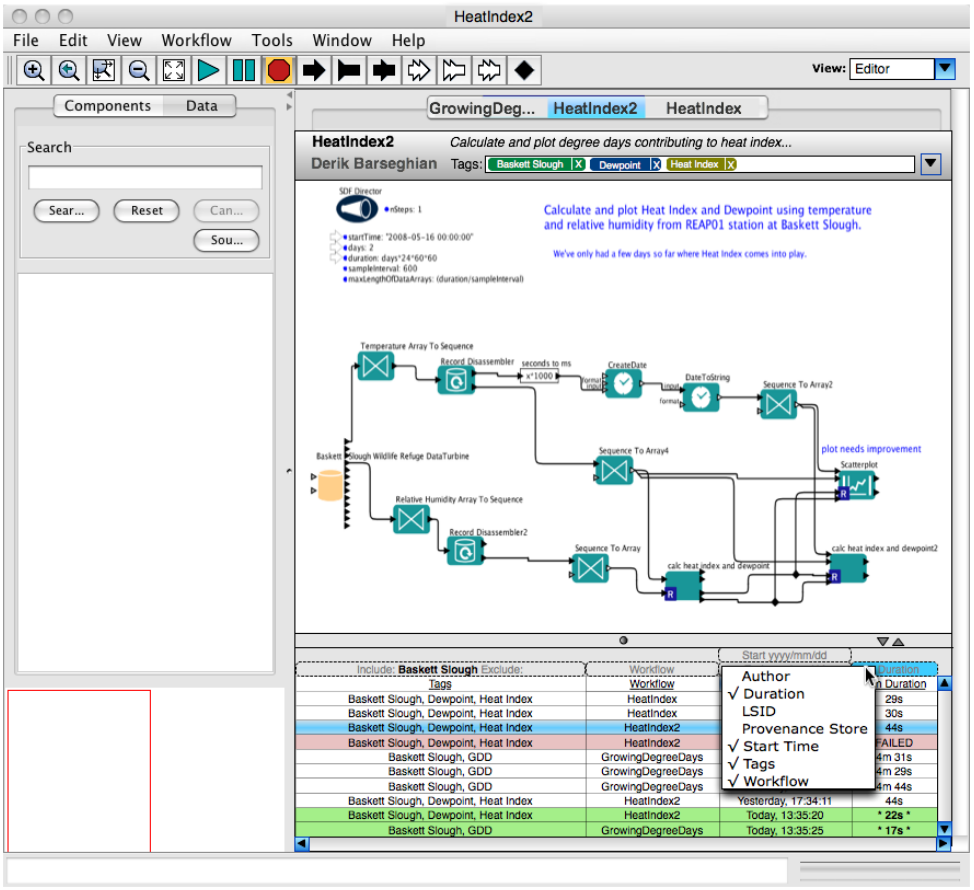


Figure 2-vii: Screenshot of the Kepler user interface; originates from *kepler-project.org*

data transformation and filtering, statistical algorithms, signal processing, or even geo-spatial data processing. The Kepler library contains a vast collection of such activities and provides them to the scientist through a sophisticated semantic search system. The edges in the graph mostly represent data channels, as Kepler workflows are modeled by their data-flow.

Each activity in a Kepler workflow can be assigned to one or multiple computer nodes for execution. The same way, data access can be routed to specific data stores. The Kepler software allows the execution of the workflow directly from the user interface. A Kepler Scientific Workflow (KSW) file contains all information that the workflow can be executed at any other Kepler system. The Kepler project motivates its users to share and reuse workflow artifacts, workflow data, and customized components within the Kepler community.

Taverna

The myGrid[11] group has developed a set of open source tools for typical scientific computing purposes. Taverna[12] has been created in this context – an open source tool for designing and executing workflows. Its desktop client is called *Taverna Workbench* and provides a graphical workflow editor where a scientist can manipulate workflow diagrams. Taverna workflows consist of data flows between *processors*. These processors are services (e. g., Web Services) or other executable components that each receive a set of input data and create a set of output data. The Taverna Workbench contains integrated context-specific views to find appropriate Web Services and other resources.

The software components *Taverna Engine* and *Taverna Server* are responsible for a remote execution of workflows, while Taverna Workbench can be used to start and manage the workflow execution. Taverna supports the workflow language SCUFL (Simple Conceptual Unified Flow Language) and a Taverna specific derivative. Moreover, Taverna emphasizes its data exporting capabilities into Microsoft Excel and into CSV↗ tables. Like Kepler, Taverna motivates its users to share their work using the *myexperiment*[13] platform.

Trident

Microsoft Trident[14] is a scientific workflow management system based on the Microsoft Windows Workflow Foundation[15]. It contains the *Trident Workflow Composer* as a client where the scientist can visually model activities into a workflow. The modeling is control flow-oriented, but Trident supports data-flow edges, too. The activities are pre-packaged basic and composite activities or originate from custom activity libraries or domain specific workflow packages. Custom activities can be implemented using *Microsoft Visual Studio* and the *.Net* framework. Trident includes a workflow catalog and also supports appropriate workflows from the *myexperiment* platform.

In order to execute a workflow, Microsoft Trident offers two approaches. The first approach is to compile and export a workflow into a simple workflow application. This application can be executed on an arbitrary system and even includes a graphical user interface to run and monitor the workflow. The second approach is to execute the workflow on the *Trident Runtime Services* – based on *Windows Workflow Foundation Runtime* –, which handle job scheduling, communication between components, and even High Performance Computing (HPC) Cluster integration. In this environment, the *Management Studio* can be used for workflow administration, workflow monitoring, and to examine workflow instances during and after their execution.

[11] http://www.mygrid.org.uk/
[12] http://www.taverna.org.uk/
[13] http://www.myexperiment.org/
[14] http://research.microsoft.com/en-us/collaboration/tools/trident.aspx
[15] http://msdn.microsoft.com/en-us/netframework/aa663328.aspx

2.3.5 Web Service Integration

Like the previous discussion has shown, many scientific simulations are currently modeled and executed using specific simulation frameworks. The simulation is then typically performed by monolithic simulation applications or via scripts. Hence, the seamless integration of such simulation applications into a superordinate workflow is very difficult. [Rut09] has developed a generic Web Service based framework to integrate simulation applications into BPEL processes. We have a brief look at its architecture in the following paragraphs.

The basic idea of the work by [Rut09] is to let BPEL processes orchestrate the activities of simulation applications. In consequence, the simulation application can be exploited for routine calculations, while the BPEL process is in charge of setting the input and launching the individual commands during the simulation. [Rut09] has implemented the framework for the simulation frameworks DUNE and ChemShell (see Section 2.3.4 on page 24).

A schema of the architecture of the generic Web Service framework is shown in Figure 2-viii on the next page. The architecture has two major aspects to consider. On the one hand, simulation applications have to be adapted so that they can be accessed from other processes. [Rut09] used a Web Service library to implement these extensions, e. g., for DUNE the *DUNE WS (internal)* provides a Web Service stub implemented in C++. With this Web Service, the simulation application can be directed to execute specific commands, such as to load a file storing a finite volume grid.

On the other hand, a simulation manager (*Generic Adapter*) implements the Web Service interface towards a BPEL process. It contains a *Program Manager* and an *Instance Pool* for all known simulation instances. The Web Service interface offers various methods, e. g., creation a new instance, extraction of the archives with the simulation applications, and starting of simulation applications. Special extension Web Services (e. g., *DUNE Web Service*) are responsible to provide operations that are individual for a certain simulation framework or a simulation context. An example is a method for DUNE to refine a finite volume grid. The framework contains eight WSDL↗ files in total, some of which are callback services. The latter is required to update the state of a simulation instance in the *Program Manager* asynchronously when a simulation step is finished.

2.3.6 Simulation Examples

This section presents three interesting examples for simulations and simulation workflows that are referenced later in this thesis. The examples cover a computational fluid dynamics problem using DUNE, a catalyze reaction based on ChemShell, and the Ecological Niche Modeling project using Kepler.

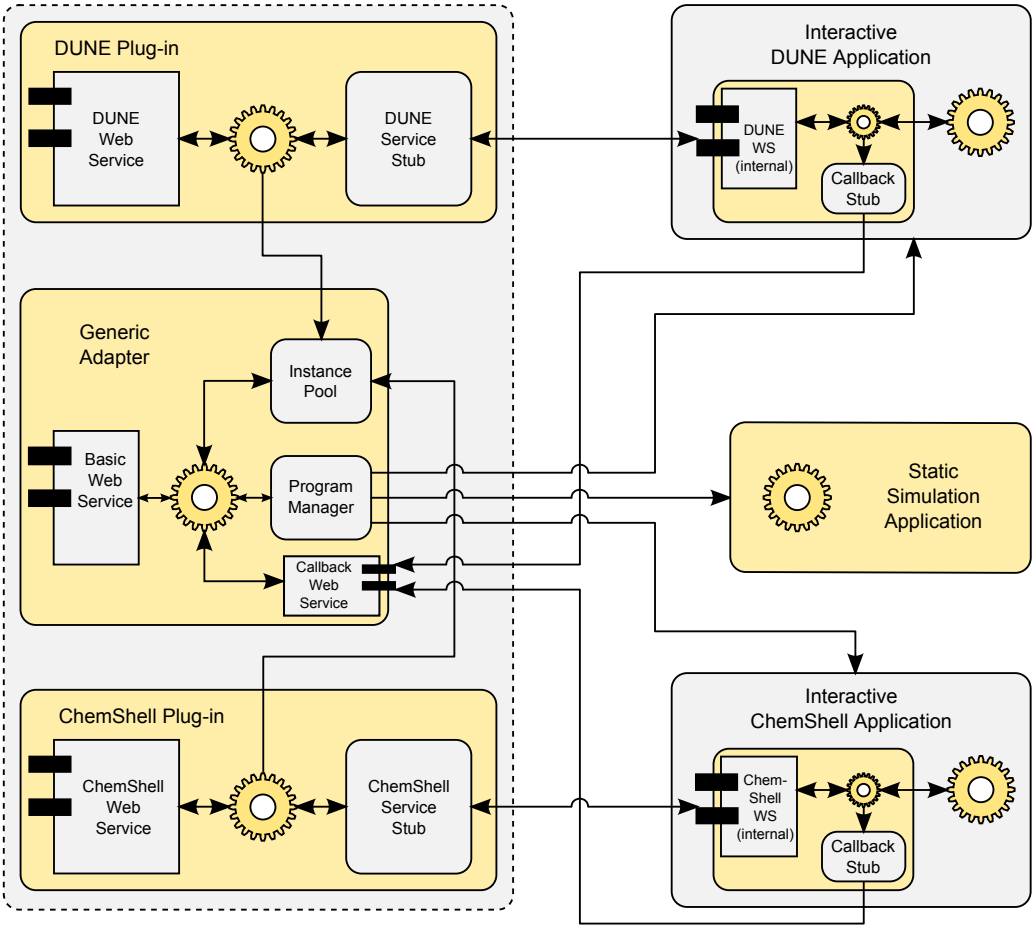


Figure 2-viii: Architecture of the Web Service framework by [Rut09]

Computational Fluid Dynamics Using DUNE

A number of example simulations using the DUNE framework (see Section 2.3.4 on page 24) can be found in the DUNE HOWTO ([BBD+09]). [Rut09] uses the example "Cell centered finite volumes" of this collection. As the on-hand thesis extends the work of [Rut09], the example is presented here as well.

The finite volume example simulates the diffusion of an ink drop into a box of water. The mathematical background is to solve the linear hyperbolic partial differential equation shown in Equation 2-i.

$$\frac{\partial c}{\partial t} + \nabla \cdot (uc) = 0 \quad \text{in } \Omega \times T \tag{2-i}$$

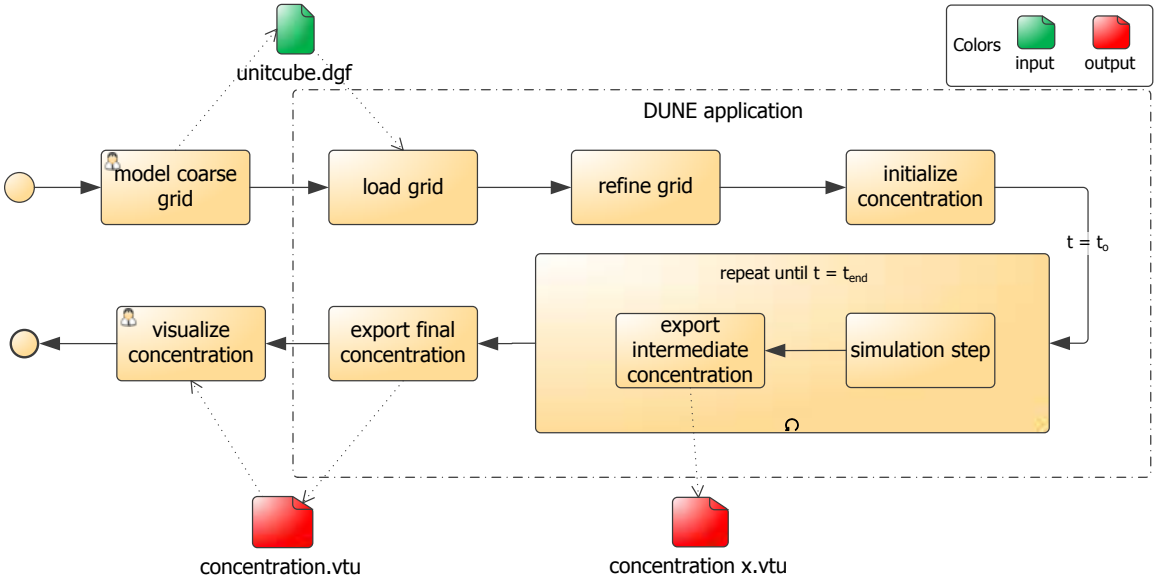


Figure 2-ix: "Cell centered finite volumes" example using DUNE

In this equation, $\Omega \subset \mathbb{R}^d$ is the domain, $T = (0, t_{end})$ is the time interval for the simulation, $c : \Omega \times T \to \mathbb{R}$ is the unknown concentration, and $u : \Omega \times T \to \mathbb{R}^d$ is the velocity field. The equation holds the initial condition shown in Equation 2-ii.

$$c(x, 0) = c_0(x) \quad x \in \Omega \tag{2-ii}$$

The boundary condition of the equation is shown in Equation 2-iii, where $v(y)$ represents the outer normal of the water boxes at a point $y \in \partial\Omega$, and $\Gamma_{in}(t)$ is the inflow boundary at time t.

$$c(x, t) = b(x, t) \quad t > 0, x \in \Gamma_{in}(t) = \{y \in \partial\Omega \mid u(y, t) \cdot v(y) < 0\} \tag{2-iii}$$

As we have a water box, we can define that $\Omega \subset \mathbb{R}^3$ is our domain. The concentration $c(x, t)$ is the ink concentration at point x and time t. The velocity field $u(x, t)$ represents the velocity of the ink at point x and time t.

This partial differential equation can now be solved using a finite element method. For details, the interested reader is referred to [BBD+09].

More important from our point of view are the activities of the simulation. As we have a DUNE application, all the activities are written in the programming language C++ using the DUNE classes for the grid model and the vector field. The overall simulation activities and their order are shown in Figure 2-ix.

In a first step, the scientist writes the *unitcube.dgf* – a grid specification file that specifies the *coarse-grained grid* for the finite element method. The *unitcube.dgf* contains the dimensions, the edges, and all the nodes representing the grid. A DUNE module *loads the grid* into a specific

grid typed object. In the following, the *grid is refined* by inserting intermediate nodes and corresponding edges into the initial grid. Using this approach, the scientist can model a coarse-grained grid manually, but uses a better approximation for the following simulation steps. The ink *concentration is initialized* for the time t_0.

The actual simulation is based on a time discretization so that each *simulation step* corresponds to a point in time. For each step of the simulation, the amount of the concentration at time t_{i+1} is calculated for each finite element on basis of the velocity field and the concentration at time t_i. *Intermediate concentrations are exported* for all steps. In the example, the iterated calculation ends when a given number of steps is reached. Finally, the simulation application *exports the final concentration* for each node of the grid into *concentration.vtu* file, which the scientist can visualize by using the Visualization ToolKit[16].

As the example is only based on a small grid and only a few number of simulation steps, it is not representative for all DUNE applications regarding the computational effort. Nevertheless, this example shows how DUNE simulations are used to simulate a finite element model and which input and output sources are touched by the application or the scientist.

The main C++ source file for the finite volumes example can be found in: `[DVD]/Frameworks/DUNE/dune-grid-howto-1.2.1-modified.tar.gz!finitevolume.cc`.

Catalyze Reaction Using ChemShell

Using the ChemShell framework (see Section 2.3.4 on page 26), hybrid quantum mechanics/molceular mechanics (QM/MM) simulations can be performed. In contrast to the DUNE example, typical QM/MM simulations in ChemShell do not apply an FEM↗ approach. Instead, the Schrödinger equation is used on an atom centered approach, which promises a higher efficiency than FEM. The solution for the Schrödinger equation is typically compute-intensive and data-intensive and delegated by ChemShell to external tools, such as Molpro[17] or GAMESS-UK[18]. ChemShell itself provides the molecular structure as input for these tools. In an iterated loop, ChemShell lets the tools solve the Schrödinger equation for the given molecular structure. With the result, the energy of and the forces between the atoms of the molecule can be derived. Thereby, ChemShell can perform the molecular dynamic calculation, which is a central task for the following example. For further details of QM/MM calculations, the interested reader is referred to [Atk01].

The following ChemShell example we consider, simulates the conversion of glutamate into methyl aspartate using the enzyme glutamate mutase as a catalyst. This chemical reaction is fundamental for life forms to gain energy and create specific carbon compounds. The example illustrates a typical hybrid QM/MM simulation without utilizing all its capabilities. As the chemical details of

[16] `http://www.vtk.org/`
[17] `http://www.molpro.net/`
[18] `http://www.cse.scitech.ac.uk/ccg/software/gamess-uk/`

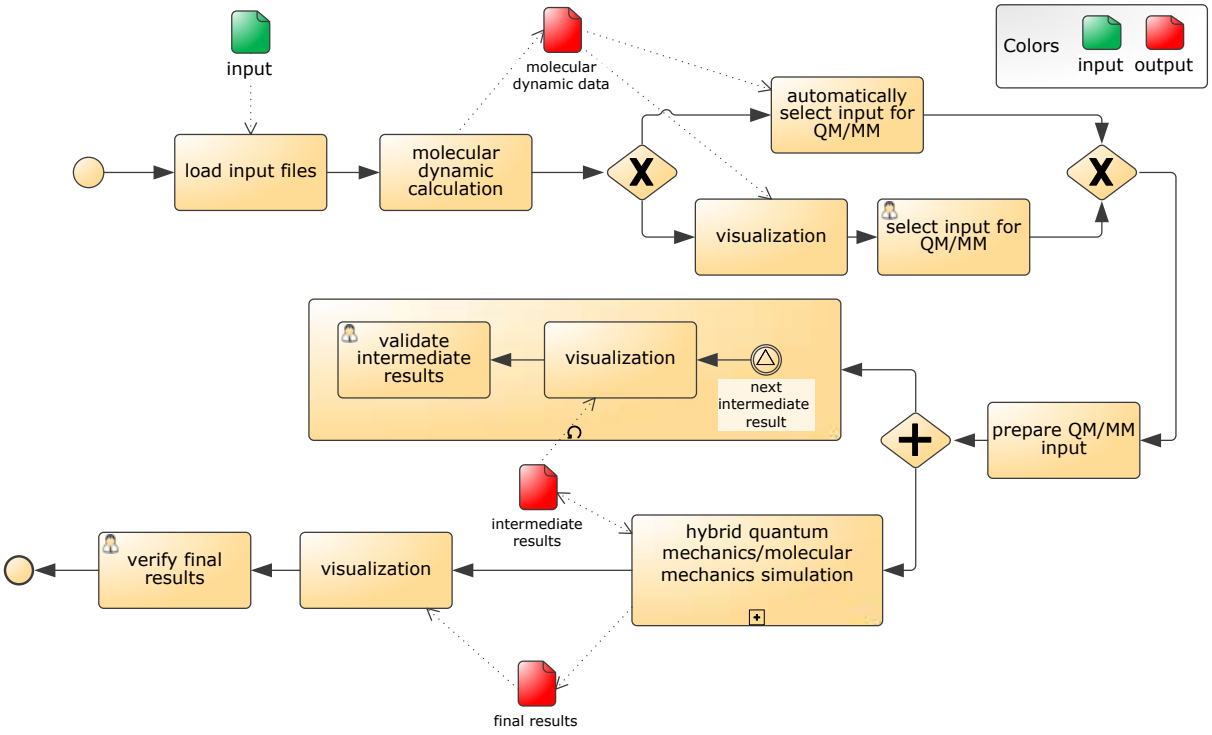


Figure 2-x: Schema of the example catalysis reaction using ChemShell

this reaction are out of scope of this thesis, we keep the following presentation at a more abstract view. Background information about this type of QM/MM approach can be found in [SVG+03].

The simulation example is shown in Figure 2-x. The simulation starts by *loading input files*, such as a protein database file that can be downloaded from the Protein Data Bank[19]. Afterwards, a classical *molecular dynamic calculation* (MD) is done as a preparatory step. Using these results, alternatively the scientist or an activity *selects the input* for the following simulation. Since the ChemShell workbench has an interactive mode, the scientist has the ability to interfere with the preparation at this time. To help the scientist in his decision, a *visualization* is done.

Additional files are processed and loaded into variables and *prepare the QM/MM input*. The *QM/MM simulation* utilizes external tools, while ChemShell is responsible for providing variables and allocate memory. In parallel cluster-based simulations, ChemShell manages the nodes of the cluster.

The quantum mechanics simulation solves the Schrödinger equation, is typically very computation-intensive, and can take a long time. Moreover, the amount of temporary data is growing along with the simulation time. Hence, ChemShell stores *intermediate results* into files very often. These

[19] `http://www.rcsb.org/pdb/`

files have to be serialized and parsed a number of times, which requires much effort, as these files are very complex and large.

A tool *visualizes* the *intermediate results* so that the scientist can asynchronously *validate the intermediate results*. When the simulation is finished, a number of *final result* files are written. They contain all information needed to reproduce the reaction, e. g., the three dimensional coordinates of each atom and their movement during the reaction. The scientist can *visualize* the reaction by importing these files into an appropriate visualization tool, e. g., Visual Modular Dynamics[20] and thereby *verify the final results*.

Ecological Niche Modeling Using Kepler

An example for a data flow based simulation workflow is the *Ecological Niche Modeling* ([PHP+06]), which uses the Kepler workflow system (see Section 2.3.4 on page 26). Based on the analysis and synthesis of multidimensional data from laboratory experiments, satellite imagery, and related simulation models, the workflow simulates biodiversity evolution caused by geographic and ecological phenomena. In order to tackle the complexity of this simulation, a hierarchical decomposition into three sub-workflows is applied.

The major challenge for this workflow is the extraction of knowledge from considerably different models stored in many heterogeneous data stores. Hence, an extra preparation sub-workflow is responsible to load data from distributed topographic maps, climate layers, and climate predictions. For each data access, different interfaces and protocols are used through *EcoGrid*[21] – a data grid infrastructure for ecological research. After having retrieved the data, it has to be preprocessed and transformed, as different underlying models have to be homogenized into one model. The components of the Kepler workflow management system are intensively used for conversions and recalculations. The same way, specially implemented Web Services are utilized. Moreover, the preparation sub-workflow includes activities that convert textual files into binary representations, too.

For being very data-intensive, the preparation sub-workflow is carried out in a data grid architecture – just like the second modeling sub-workflow. The second workflow is trying to build up a model that can explain biodiversity evolution based on historical data. It is nearly impossible to specify the model manually because of its complexity. Instead, the model is iteratively derived from the data. A *Genetic Algorithm for Rule-set Production* (GARP) is used to build a sophisticated set of rules for the model. This is done in many iterations by selecting a small set of sample data, training the model with this data, and verifying the model on a disjoint testing sample by its predictive error. During the iterations, some models come out on top and are considered as the result of the second sub-workflow.

The third sub-workflow uses the best models created before and simulates the evolution of the biodiversity of different species for each model and each climate scenario available. Afterwards,

[20] `http://www.ks.uiuc.edu/Research/vmd/`
[21] `http://ecogrid.nchc.org.tw/`

maps are generated that contain the predicted population of the species. Annotations state the probabilities of the results. Finally, a *Receiver Operating Characteristic* (ROC) analysis is applied to evaluate the errors of the predictions. This statistic facilitates a validation of the predictive capability of the results. Finally, the results are stored through the EcoGrid interface and can be analyzed by scientists.

2.4 Database Issues

Before we consider specific database issues, we start with a brief introduction of database related terminology, based on [HR01], [KE04], and [Nar06].

A *database system* (DBS) consists of two parts: collections of data and software that controls access to the data. The collection of data is often referred to as *database* and organized in such ways that the data can be easily accessed, managed, and updated. Different *data models* can be applied to this storage, which define rules and operators how data from the real world can be represented in the database. The currently most prominent data model for databases is the relational model ([KE04]).

A *database management system* (DBMS) is the other part of a *database system* and handles the access to the database, maintains a catalog of databases, ensures consistency of the data, manages concurrent access, and applies security aspects. An application interacts with the DBMS using a *database language*. Structured Query Language (SQL) is the main representative for relational *database management systems*.

An important aspect of DBS is the *data independence* between the *database schemata*. Here, we follow the design standard by ANSI/SPARC[22]. The *conceptual schema* represents the result of a data modeling process – the model of the universe of discourse based on the provided data model of the database. The *internal schema* defines the physical storage structures the database uses to store the model of the conceptual schema. The *external schema* provides additional views on top of the conceptual schema to applications that interact with the DBMS.

In the remaining document, we do not put special emphasize on the fine distinction of the three terms and mostly apply the term *database system* or shorter *database*.

Throughout this thesis, the term *local cache* is used multiple times. We consider a cache as an intermediate storage of duplicated data that originates from another data store. The sole purpose of a cache is to avoid the repeated expensive retrieval of data. There are multiple ways to implement a caching mechanism – using a static retrieval of a subset of remote data or a more sophisticated dynamic approach (cf. [FCL97]). Furthermore, the cache can be implemented as a set of main memory pages or as a locally accessible database with a sophisticated query language.

[22] American National Standards Institute, Standards Planning And Requirements Committee

An example for the latter approach is the IBM solid DB[23], which can be utilized as a *Universal Cache* for remote disk based relational databases.

Subsequently, this section covers enhanced database related issues that are crucial for the understanding of the following discussions in this thesis. The reader gets an overview of main memory based databases, XML as data format, and special features of embedded Java databases.

2.4.1 Main Memory Databases

The traditional relational database management system (RDBMS↗) stores its data on disk, being a stable but relatively slow storage. The main memory and the processor cache allow a five orders of magnitudes faster access than external storage on disk. [HR01] describe this as the "access gap", which implicates a lot of concepts that try to minimize the disk access rate and to improve the memory hits.

Main memory databases (MMDB) benefit from all the advantages of main memory ([GMS92]). Besides the fast and direct access to the main memory, the database can utilize a random access to the memory, too. For this reason the data structures do not need to be designed for an optimized block-oriented retrieval or a sequential access. This allows the design of other index structures, such as a binary AVL tree (also called T-Tree) or a bucket-chained hash ([Bon02]). These index structures have proven to be more efficient in memory than traditional B-Trees would be.

Taking all the differences into account, the development of a MMDB requires a complete reconsideration of a lot of techniques that traditional disk based databases are build upon. The disk I/O cannot be used as the primary factor for a cost based query optimization ([Mit95]). The CPU utilization and the memory access are now the key factors, being fuzzy and hard to quantify ([Bon02]). Moreover, the design of a MMDB has to focus on the efficient use of the limited main memory, since there is a given upper bound, which disk based databases can easily outperform. As hashing can be space consuming, it is avoided in large scales. In addition, memory based databases tend to exploit the advantages of column-oriented table storage instead of traditional row based storage. This allows a compression based on the data type knowledge of a column and improves the performance of queries selecting nearly every, row but not every column, e. g., online analytical processing (OLAP) queries.

A downside of MMDBs is the volatile nature of main memory causing a violation towards the durability aspect of the atomicity, concurrency, isolation, and durability concept (short ACID). But techniques involving uninterruptible power supplies, battery backed memory, or log entries on a stable storage address this problem. In addition, main memory databases might be replicated on different computer nodes to ensure the persistence of the data after a crash of one node.

A performance benchmark illustrating the differences between main memory and disk based databases has been made by [Gra02]. They investigated 1) the disk based database db.linux, 2)

[23] `http://www-01.ibm.com/software/data/soliddb/`

the same database, but using a RAM drive, and 3) the main memory database eXtremeDB[24]. The results show that porting the disk based database on a RAM drive increases read and write access only by a factor of 4. But the eXtremeDB can even outperform this increased access times by a factor of 420 for writes and a factor of 4 for reads. [Gra02] sees the fundamentally different design and the obsolete cache as the key factors for this difference. This proves in a practical way that main memory databases have a complete different environment that requests certain adaptions.

2.4.2 XML Databases

With the rise of the World Wide Web and the usage of XML as the standard for document exchange, a new requirement towards databases arose. Databases have to efficiently store, transform, and process data based on the XML *data model*. The following discussion is based on [CRZ03] and [Sta03] and describes different types of XML databases. The types of XML databases can be distinguished by their internal representation of the XML data. The external database language is mostly based on XPath- or XQuery-based statements ([CD99], resp. [BCF+07]).

XML based data can be transformed into a relational representation – a format that traditional databases can handle best. As XML describes content and structure, a transformation approach seams practicable. But when looking at the details, it becomes clear that not every aspect of hierarchical based XML structures can be efficiently represented by a set of relational tables. Moreover, the request and response transformation requires complex shredding of the XML file and expensive joins among different relational tables to reconstruct the XML document.

Some database vendors of relational database systems favor a pragmatic mechanism: the creation of a new column type `XML`. The whole XML document is stored as an attribute value of a table row and can be accessed via additional XPath- or XQuery-based expressions in SQL. The advantage is obvious: SQL stays the state of the art query language for databases and XML data can be processed, too. Such databases are typically called *XML enabled*. Nevertheless, the straightforward storage of an XML file as a CLOB↗ does not benefit from a deeper analysis of the structure of the XML document. Hence, the query processor cannot rely on database techniques such as indexes or cost based query plans. In some cases the database has to parse the XML file for each query.

Taking all these limitations into account, XML databases have been developed aiming at the special needs of XML documents. They are often called *native XML databases*, because they solely focus on XML data. Native XML databases try to exploit all structural and content information to efficiently store the document and process requests. This leads to the construction of a new physical data model representing XML files and a special node identification mechanism. New logical operators and special join variants called Twig Joins ([MH06]) can be build upon these two techniques. At the end, native XML databases have achieved the same level of performance, feature richness, and ACID compliance as traditional relational databases have.

[24] `http://www.mcobject.com/extremedbfamily.shtml`

Besides the native XML databases, *hybrid databases* for both relational and XML data are advertised, too. Such databases implement two different storage models for relational and XML data, but provide a common *external schema* for both *data models*. Some hybrid databases were originally XML enabled databases, but replaced the storage of XML documents as CLOB values with a specialized XML storage.

2.4.3 Java Embedded Databases

In contrast to large scaled relational database management systems, a number of embedded databases have been developed for the Java market. At this point we want to highlight Apache Derby[25] (since Java 1.6 also known as Java DB) and H2 Database Engine[26]. They are relational databases entirely written in Java and supporting SQL queries.

These databases see their market as development databases and are not directly intended for productive environments. Nevertheless, they have great potential. Derby and H2 provide a feature called embedded mode. Instead of using a network connection between the application and the database, the Java application can run the database in the same Java Virtual Machine (JVM). The communication is faster and no network overhead occurs. For query-intensive applications this may result in shorter query execution times and faster query response load. In addition, the very tight integration of the database allows the sharing of resources by the application and the database engine. Moreover, these databases – aimed at the Java market – have a better support for Java data types and ease Java object serialization. Finally a native Java database can be deployed wherever the Java application is deployed – the integration effort is minimal ([Wya00]).

But native Java databases have downsides too. As they share the same JVM with the application, they both are more vulnerable. A crash of the application forces the database to crash too – a weakness considering the ACID compliance. Furthermore, both compete for JVM heap memory, which forces a thoroughly thought through memory management and configuration in order to avoid frequent memory exceptions. However, lightweight databases emphasize their support for scalability and concurrent access. These issues are investigated in performance benchmark tests by [Pol07] and [H209], while [STH06] and [MP07] try to master known deficiencies by tuning the Apache Derby database.

2.5 Tools and Frameworks

This section presents tools and frameworks used for the work on this thesis. A detailed list of tools and frameworks that are actually applied to the work on this thesis can be found in Section 6.1 on page 131.

[25] `http://db.apache.org/derby/`
[26] `http://www.h2database.com/`

2.5.1 Apache Tomcat

Apache Tomcat[27] 6.0 is a popular, open source implementation of a Java based web server. Its main purpose is the realization of a container for Java based Servlets[28] and Java Server Pages[29] (JSP). Apache Tomcat provides a HTTP↗ transport layer and allows SSL↗ encrypted communication, too.

Apache Tomcat provides Java-based extensions for its modules in order to allow customization for specific needs. As Tomcat is designed modular, it is possible to integrate only a sub-selection of the modules of Tomcat into other Java frameworks. Therefore, a lot of software uses Tomcat as container for help pages (e. g., Eclipse[30]) or to provide an administration web interface (e. g., the VMWare[31] Infrastructure Web Access). Other software exploits the HTTP transport capabilities of Tomcat in an embedded mode or eases its deployment by offering a ready-to-use Tomcat web application such as Apache Axis2 (see Section 2.5.2).

Apache Tomcat 6.0 is used in this thesis, because it is easy to install, is rich in features, and has an open source license. In addition to that, deployment plug-ins exist for the development environment Eclipse, which is also used for the work on this thesis.

2.5.2 Apache Axis2

Apache Axis2[32] 1.5 is an open source Web Service framework with a feature-rich set of tools. Axis2 is implemented in the programming languages Java and C. However, we just consider the Java variant in the context of this thesis.

The major application of Axis2 is the deployment of Web Services, whose functional logic is implemented in Java. In addition to that, Axis2 makes it possible to create Web Service clients written in Java. Axis2 supports WSDL↗ version 1.1 ([CCMW01]) and 2.0 ([CMRW07]) and various protocols for Web Service transport and messaging, including SOAP↗, HTTP↗, TCP↗, and JMS↗. Axis2 implements WS-Addressing ([GHR06]) and the interchange of WS-Addressing endpoints. On this basis, Axis2 client applications can invoke asynchronous server applications, which utilize a callback endpoint for the response.

Axis2 implements WSDL by mapping its message model to a Java representation. As there are different methods to represent XML-based SOAP messages as Java objects, Axis2 provides a set of data binding mechanisms. In order to ease the development of Java based Web Services using these data bindings, Axis2 has a set of code generation tools for the server and client side.

[27] http://tomcat.apache.org/
[28] http://java.sun.com/products/servlet/
[29] http://java.sun.com/products/jsp/
[30] http://www.eclipse.org/
[31] http://www.vmware.com/
[32] http://ws.apache.org/axis2/

A developer can start with the service definition of a WSDL file and use the code generation tool `WSDL2Java` to generate a server skeleton or respectively a client stub. He or she then can implement the generated Java interface for the server side, or respectively implement the usage of the stub on client side. Axis2 maps a WSDL port type to a Java interface, WSDL operations to Java methods, and the messages to a tree of Java objects. Exploiting the Jibx[33] framework, it is possible to define a custom data binding and reuse manually developed model classes for the message representation.

Developers who already have a Java interface can use the tool `Java2WSDL` to generate a derived WSDL file. Dependent on the chosen data binding mechanism, either self defined XSD↗ types or generated ones are used. Since no client stub or server skeleton is created this way, the `WSDL2Java` tool has to be used subsequently. The developer is also free to modify the generated WSDL file before using the `WSDL2Java` tool.

All in all, the generation tools are a powerful but, occasionally, a tricky matter. When using the `WSDL2Java` tool, errors that are not obvious in the WSDL or XSD files are not treated with meaningful failure messages, but rather with obscure `NullPointerExceptions`. In addition to that, the default Axis2 data binding (ADB), a very fast and yet simple XML to Java mapping, does not fully support all XML schema features. So a developer might end up having implemented a huge set of generated server skeletons and now has to switch the data binding mechanism, because the ADB tool chokes on sophisticated XSD constructs. Nevertheless, when handled with care and automated by using scripts, the generation tools are a great benefit for a Web Service developer.

Either way, a developer can use Axis2 to implement a server skeleton of a Web Service. He or she then has to add the obligatory Axis2 service description file: `services.xml`. This file specifies which services are implemented by which Java classes and which Message Exchange Patterns (in-out, in-only) each operation uses. Together with needed libraries and the compiled Java class files, the `services.xml` has to be packed in a compressed Axis2 service archive (AAR). This archive can be deployed into the Axis2 service folder by just copying it. The Axis2 hot deployment mechanism waits for new archives in the service folder and adds the defined Web Services instantly. After that, the Web Services of this archive can be used by clients.

The Axis2 server has two deployment modes. On the one hand, the binary Axis2 download contains a simple server that is used to listen for HTTP requests and provides the deployed Web Services. On the other hand, there is the Axis2 web archive. This more sophisticated package is intended to be deployed within a Servlet container such as Apache Tomcat. It uses a `SimpleHTTPServer` Servlet to intercept all requests to Web Services on the HTTP transport layer. A *Message Receiver* of the service skeleton processes the request, unmarshals SOAP messages, and calls the implemented Java method. The same way, in a synchronous scenario, the response message is passed through to the client. A feature which comes in handy, is the Axis2 web administration site of the web archive deployment. This site lists all deployed Web Services and links to their WSDL files.

[33] `http://jibx.sourceforge.net/`

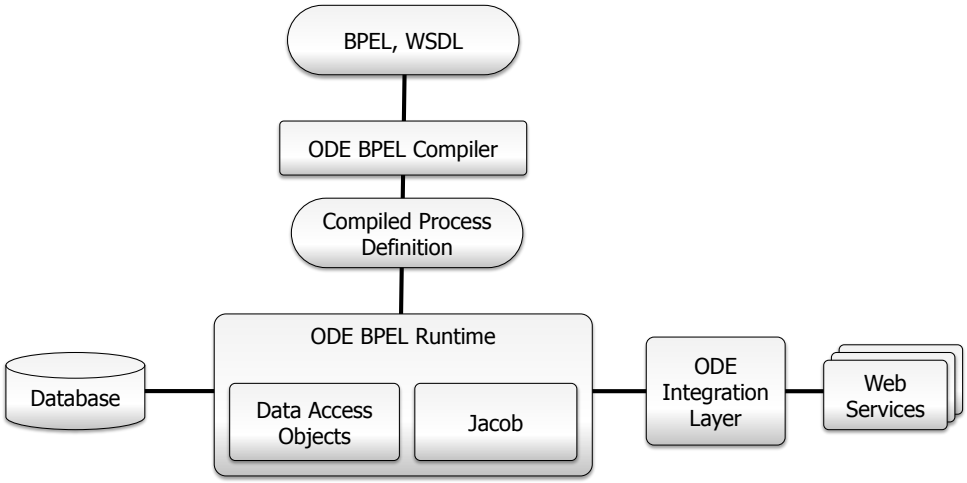


Figure 2-xi: ODE architecture overview following
`http://ode.apache.org/architectural-overview.html`

When using Axis2 as a Web Service client, generally no server modules are needed. The generated client stub just has dependencies to a collection of Axis2 libraries, including the Apache Commons HTTP Client for HTTP connections.

Axis2 1.5 is used in this thesis, because it is a powerful Web Service framework that supports all required Web Service standards, provides an open source license, and its code generators can be easily integrated into Apache Ant[34] build scripts.

2.5.3 Apache ODE

Apache ODE[35] stands for Orchestration Director Engine. Apache ODE 2.0 is an open source, Java based workflow engine for the workflow language BPEL↗ (see Section 2.2.2 on page 12). ODE makes it possible to deploy short and long running BPEL workflows, create instances and execute them in a persistent concurrent environment, and let the workflows communicate with local and remote Web Services. The architecture of ODE is illustrated in Figure 2-xi.

In order to unroll the architecture, we start with the deployment. The workflow designer has created a BPEL process and WSDL files that contain the implemented port type and related partner link types. Additionally an ODE deployment descriptor is required, which contains mappings for each partner link of the process to a WSDL port. The designer might have used the Eclipse BPEL Designer[36] for the BPEL process development. The workflow designer can now use

[34] `http://ant.apache.org/`
[35] `http://ode.apache.org/`
[36] `http://www.eclipse.org/bpel/`

the Eclipse BPEL Designer to deploy the resources or manually copy them in a hot deployment folder of Apache ODE.

Apache ODE starts the deployment process by compiling the BPEL process and creating an executable representation of the process. This process model contains resolved references to variables and partner links and is optimized for navigation during the workflow execution. The compiled process is popularized to the ODE BPEL runtime, which adds the process model to the collection of available processes. The main work of the BPEL runtime is to instantiate BPEL workflows on incoming messages or events, navigate through the process model, and execute the BPEL constructs by using their corresponding Java based implementations.

The BPEL runtime uses a Data Access Object (DAO) layer to mediate the interactions to an underlying data store. The DAO layer consists of a set of Java interfaces and requires an underlying implementation. In most deployment variants, a relational database is applied and an implementation based on the framework OpenJPA[37] is utilized for the object relational mapping (ORM). An alternative implementation based on Hibernate[38] is also provided by ODE. To ease deployment and configuration, the embedded Java database Apache Derby[39] is supplied with the Apache ODE release. As stated in [Les09], Apache Derby is not intended to be used in a productive environment. Therefore, the configuration file `ode-axis2.properties` can be adapted in order to change the database mode of ODE:

- EMBEDDED: Apache Derby (default)
- EXTERNAL: a database is provided by the application server represented by a `DataSource` object
- INTERNAL: a database referred by a JDBC connection URL

The data source is used as a transactional storage to efficiently save and retrieve:

- deployed processes
- running process instances
- partner links for Web Service communication
- the state of the instances; e. g., running, failed, terminated
- values of variables for each instance
- activities waiting for events or incoming messages
- a log for all events and messages
- the ODE job queue

[37] `http://openjpa.apache.org/`
[38] `https://www.hibernate.org/`
[39] `http://db.apache.org/derby/`

Besides the DAO layer, the ODE BPEL Runtime relies on another component: Jacob (short for Java Concurrent Objects). Jacob is responsible for persistent and concurrent execution of the activities of the BPEL workflows instances. Using Jacob, it is possible to execute concurrent activities with the needed isolation and durability requirements and allow the execution of an instance to be paused and to be resumed later. The latter is important if the workflow engine has restarted after a crash, or the workflow administrator has paused an instance manually.

Apache ODE is tightly bound to an *Integration Layer*, which can be seen as its container and the interface to outside communications. There are a number of possible integration layers:

- Axis2: Axis2 (see Section 2.5.2 on page 39) in web application mode, e. g., deployed in a Tomcat container
- Apache ServiceMix[40]: an open source implementation of an Enterprise Service Bus (ESB) that is build on top of Java Business Integration[41] (JBI) and provides functionality of a Service Oriented Architecture (SOA)
- Apache Tuscany[42]: an implementation of the Service Component Architecture[43] (SCA)
- JBoss Riftsaw[44]: an integration into the JBoss[45] application server, exploiting various JBoss modules, such as its Web Service stack

Apache ODE was build from the scratch with the design of a modular, extension based architecture. This fact can be seen at the abstract DAO layer or the various integration variants of ODE. Moreover, ODE offers useful XPath extensions, such as an implementation for XPath 2.0 ([BBC+07], ODE web site[46]) and XML node modification functions that resemble XQuery ([BCF+07]). However, the most important extension mechanism might by the implementation of BPEL extension activities. It is possible to define a new BPEL extension activity and implement it as a Java class. This is a powerful concept, because additional activities, such as database management features, can be seamlessly integrated into BPEL workflows. An example for this approach can be seen at the SIMPL framework (see Section 2.6 on the next page). Besides extension activities, ODE supports the implementation of custom extensible assign operations, too. They can be used as a customized data manipulation alternative to the `<copy>` elements in an assign activity.

All in all, Apache ODE is a reliable, extension-rich, and open source workflow engine for BPEL. Regarding the integration into the Java development environment Eclipse and the beneficial administration web interface for monitoring, ODE is chosen for the work on this thesis. We explicitly choose ODE 2.0 beta 2, because this release includes important bug fixes for the Axis2 integration and supports useful XPath 2.0 extensions when working with BPEL variables.

[40] `http://servicemix.apache.org/`
[41] `http://jcp.org/aboutJava/communityprocess/final/jsr208/index.html`
[42] `http://tuscany.apache.org/`
[43] `http://osoa.org/display/Main/Service+Component+Architecture+Home`
[44] `http://www.jboss.org/riftsaw`
[45] `http://www.jboss.org/`
[46] `http://ode.apache.org/xpath-extensions.html`

2.6 Database Workflow Integration

This section investigates different approaches how workflow environments allow their workflows to access databases. It assumes the existence of local or remote database management systems and does not present techniques for their architectural integration. As the context of this thesis lies on BPEL processes, we set focus on a related environment.

[VSS+07] see a great importance in the tight integration of databases into workflows. They see workflow technology as the key technology to cope with heterogeneous applications and data stores – not only in the business, but also in the scientific world. [VSRM08] inspect SQL support in workflow products and therefore investigate different database access techniques.

As workflow environments typically support Web Services in any kind, the encapsulation of database access using this technology seems natural. Web Services can implement more or less generic operations that provide access to an underlying database. This approach is known as *adapter* technology or as *data services*. Web Services can be implemented in any programming language and deployed close to the database. In an example request-response case, a data service receives a query, optionally pre-processes the query, sends it to the database, retrieves the result, materializes it into an XML representation, and returns it to the requestor. A request that just performs data insertion or insertions can be realized in a similar way, but would not need to return a result. In addition, a data service can implement functional logic for complex manipulations on the data. A simple way to establish security and authentication, besides a self-description of the provided service, are clear advantages of data services. The data service technique is often applied for centrally accessible information repositories, such as the DNA Data Bank of Japan[47]. In the context of grids of shared resources, the WSRF↗ can be applied to provide stateful resources as Web Services. Since the database architecture does not use grid technology, WSRF is not relevant for this thesis.

For scenarios with a lot of transactions and data traffic, the database access needs to be integrated tighter into the workflow system. In this context [VSRM08] investigate the direct definition of SQL data management activities in the workflow, which they call SQL inline support. In the context of BPEL, we can imagine a data management activity that has attributes for a database reference and an SQL statement. When the activity is executed, it executes the query to the database and stores a potential response into a BPEL variable. In contrast to data services, such activities do not have an overhead for Web Service calls and unmarshaling of responses, which can be severe when handling great sets of data. But an SQL inline support may even offer more functionality. The result of a query does not need to be materialized into a BPEL variable. Instead, the data management activity only keeps a pointer or reference to a database cursor within a BPEL variable. A `forEach` may than iterate over the cursor behind the BPEL variable and fetched the current row into BPEL variables. Only a minimum of materializations is required.

As SQL inline activities are syntactically integrated into the workflow language, e. g., BPEL, placeholders in queries can be used to reference variable values or load results into variables.

[47] `http://www.ddbj.nig.ac.jp/`

While data services typically offer only one specific functionality – e. g., insert a new DNA sequence – inline activities do not have this restriction. One can even specify data definition statements, e. g., to create a new SQL schema. This technique can be exploited when defining ETL (term in data warehouse context: Extraction Transformation Load of data) processes for large data preparation and analysis scenarios ([MMLD05]). Finally, two facts have to be noted. First, an inline support does not depend on a relational database in principle – XML and other databases can be integrated likewise. Second, the BPEL specification does not include data management activities – a fact that workflow engine vendors have partially handled by making proprietary language extensions. Hence, the workflow description is bound to this specific engine and cannot be easily shared within a community.

In the student project SIMPL[48] at the University of Stuttgart, a generic and extensible framework to integrate various data sources into BPEL processes is currently[49] conceptualized and implemented. This framework specifies data management activities based on the BPEL extension activities that are hence independent from the workflow engine. An actual implementation is offered for the workflow engine Apache ODE (see Section 2.5.3 on page 41). SIMPL facilitates the late binding of data sources by a form of data source registry. As this student project is running simultaneously to this thesis, its concepts have to be considered in this thesis, while its results cannot be exploited yet.

Apache ODE currently supports a technique called *External Variables with JDBC Mapping*[50]. This mechanism allows the binding of a BPEL variable to the content of a relational database table, accessed by using Java Database Connectivity (JDBC). The details of this binding have to be defined in an ODE deployment descriptor. At runtime a read access to the variable is interpreted as the selection of a row of the table, while a manipulation of the variable results in updating a row in the table. ODE implements a simple type mapping mechanism between the XML structure of the BPEL variable and SQL types of the table. All in all, the External Variables allow a simple approach in integrating SQL tables. Nevertheless, the approach lacks on supporting non-relational databases and allows neither data definitions nor set oriented retrieval, which is very important when analyzing data.

To sum this section up, various integration techniques exist – each of them has advantages and disadvantages. The prototype implementation of this thesis has to integrate databases into BPEL too. A decision for an approach can be found in Section 4.3 on page 76.

[48] SimTech: Information Management, Processes and Languages: `http://www.ipvs.uni-stuttgart.de/abteilungen/as/lehre/lehrveranstaltungen/studienprojekte/SS09/StuPro.SIMPL`

[49] Winter semester 2009/2010

[50] `http://ode.apache.org/external-variables-jdbc-mapping.html`

3 Requirements Analysis

Established software engineering starts with collection of requirements and their specification ([LL07]). When having noted all requirements for an application or system, the following design and implementation can be achieved in a more target-oriented way. In addition to that, the outcome of the work can be verified by inspecting the fulfillment of the requirements.

Some requirements for the integrated database architecture can be directly derived from the task description for this thesis. These requirements are clear and verifiable and discussed in detail in Section 3.1 on the following page.

In addition, more specific requirements have to be retrieved, as the conceptualized database architecture shall be applied for various simulation use cases in future and has to support their needs, too. As this thesis is oriented at control flow based simulation workflows using the workflow language BPEL, we investigate general requirements for this environment in Section 3.2 on page 50. A literature research supports this investigation. In order to find requirements directly concerning databases, we inspect scientific projects – especially simulations – that integrate data sources of any kind. As there are currently few public simulation workflow examples using BPEL, we take a look at other simulation environments, too. During the research, we are particularly interested in:

- direct integrations of databases into workflow environments
- projects that favor databases over other data sources and reason their motivation for using them
- ideas how to optimize data integration into workflows by using databases
- challenges and answers in the context of data provisioning for simulation workflows

We discuss these issues in Section 3.3 on page 51. The result of this discussion are requirements that originate from existing projects, but may be too abstract for their further use in this thesis. Thus, we develop three scenarios for simulation workflows to which we apply these requirements and inspect possible improvements in using the integrated database architecture. The scenarios are presented in Section 3.4 on page 54.

Requirements for the integrated database architecture are phrased in a way so that it is obvious for the reader what a requirement is and what not. To further simplify the management of the requirements, we add the symbol $\mathbb{R}$ followed by a unique number for the requirement to a sentence or paragraph that introduces a requirement. In addition to that, all requirements are listed in Appendix A.2 on page 160.

3.1 Requirements by the Task of the Thesis

This sections analyses the task description of this thesis and extracts requirements for the database architecture. Demarcations are stated when it is ambiguous whether a concept has to be implemented or whether it has just to be respected.

The task of this thesis is the conceptualization of an integrated database architecture for a simulation workflow environment. Hence, the database architecture has to be deployed locally to the workflow engine and has to respect their environment $\mathbb{R}$ **1**. A plan for the architectural integration has to be developed that shows the communication between the workflow engine and the database and the deployment of the database architecture.

The database architecture has to be integrated into the workflow engine Apache ODE. Thus, the integration plan has to argue which adaptions to the workflow engine are required and how its configuration has to be adapted $\mathbb{R}$ **2**. In addition to Apache ODE, the database architecture has to be applicable to other workflow environments for BPEL processes in theory and therefore a description of the architecture independent of the workflow engine is required, too $\mathbb{R}$ **3**.

The integration of the database architecture into Apache ODE has to be implemented $\mathbb{R}$ **4**. In this context a runtime environment using a portable virtual machine has to be created as a reference implementation of the database architecture $\mathbb{R}$ **5**.

For the usage of the database architecture within a BPEL process, the implementation of the SIMPL↗ project shall be used (for SIMPL see Section 2.6 on page 44). Therefore, the architectural design of the SIMPL project has to be respected $\mathbb{R}$ **6**. As the database architecture shall be utilized by the SIMPL runtime, a discussion about this integration is required, too. The implementation of the SIMPL runtime is not be finished in time to be used by this thesis. Hence, prototypes and test workflows for the integrated database architecture require a slim component for database accesses in BPEL workflows $\mathbb{R}$ **7**. Therefore, services have to be implemented that provide:

- A generic query interface for relational retrieve, insert, update, and delete statements.
- A generic query interface for XQuery statements.
- Import and export of data from and to external relational and XML databases $\mathbb{R}$ **8**.
- Import and export of data in a CSV↗ formatted file into and from a relational table $\mathbb{R}$ **9**.

The heart of the database architecture consists of one or more databases, which are utilized as data stores. It is required that the database architecture provides storage for relational data $\mathbb{R}$ **10** as well as data of an XML-based format $\mathbb{R}$ **11**. Performance improvement is an important goal of the database architecture. Therefore, it has to provide not only databases based on external storage $\mathbb{R}$ **12**, but also main memory databases $\mathbb{R}$ **13**.

As the relational storage often has to answer SQL based analysis requests, it has to support basic features of SQL, such as projections, selections, joins and aggregations $\mathbb{R}$ **14**. Moreover, the relational storage has to support primary and foreign keys. Likewise, the database architecture has to handle basic XQuery ([BCF+07]) requests to its XML storage $\mathbb{R}$ **15**.

Another major use case for the database architecture is the storage of the variables of BPEL workflow instances, which Apache ODE currently stores in an Apache Derby database $\mathbb{R}$ **16**. Therefore, it has to be investigated which requirements originate from the relational schema of the Apache ODE DAO↗ layer and how the database architecture can be used by this layer. Further optimization ideas might be discussed in this context. Moreover, ODE produces auditing data that has to be stored in the database architecture $\mathbb{R}$ **17**.

As a convention, from now on we distinguish the data to be stored by the integrated database architecture into:

- static *process information*: version, identifier, and meta data of a process Apache ODE stores in a database (Apache ODE keeps the actual process model that represents the BPEL workflow only in memory)
- dynamic *instance model*: variables and states of the activities managed by Apache ODE during the workflow execution
- *runtime information*: all events, auditing data, and messages Apache ODE stores for recording purposes
- *application data*: data needed by the simulation workflow itself, e. g., a finite element grid

Although the four distinguished terms are associated with Apache ODE, they are easily transferable to other workflow engines.

In order to satisfy the requirements above, one or more database systems have to be applied. Figure 3-i on the following page shows five application types of databases needed by the database architecture. The database architecture has to provide external and main memory based storage for relational as well as for XML application data. In addition, a database for the static *process information*, the dynamic *instance model*, and the *runtime information* of Apache ODE is required. This database is intended to be a relational external storage database, because the default DAO implementations map Java objects to a relational database and we require full ACID↗ support, which only external storage based databases implement sufficiently (for a detailed discussion see Chapter 4 on page 71). Since this database has to satisfy different requirements, it may but has not to be equal to the relational external storage based database for the application data. A database evaluation has to find an appropriate database system for each database application type $\mathbb{R}$ **18**.

Some general requirements can be addressed to the selected database systems already at this point. The license of the database has to be considered – proprietary systems can be suggested, but open source alternatives have to be presented, too $\mathbb{R}$ **19**. By that way, we can select the best database in our opinion, but also mention cost-free alternatives.

The integrated database architecture is intended for simulation workflow environments that are utilized by scientists. For that reason, the database architecture has to be installed without specific database experiences, or a clear step-by-step installation guide has to be provided $\mathbb{R}$ **20**. This requirement is important, although a working runtime environment is supplied by this thesis, too. The database architecture has to support platforms compatible with Apache ODE – this

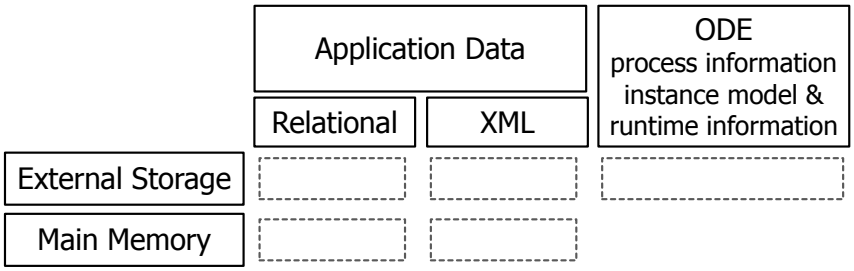


Figure 3-i: Overview of the application types of databases required by the integrated database architecture

forces platform support of the database architecture for Microsoft Windows systems and Linux based systems, as well as 32 bit and 64 bit support $\mathbb{R}$ **21**. Moreover, the database systems should provide a digital manual and user friendly software tools that are helpful for a user working with the database architecture $\mathbb{R}$ **22**.

The database architecture shall be exploited by simulations. Hence, more concrete requirements are addressed to the database architecture and its databases in the subsequent sections. In order to show the application of the database architecture, at least one prototype for a simulation workflow has to be implemented $\mathbb{R}$ **23**. In addition, test workflows, which do not need to represent simulation use cases, have to be applied to demonstrate other features of the database architecture that are not required by the prototype(s) $\mathbb{R}$ **24**.

3.2 Requirements due to the Simulation Workflow Environment

In this section, we have a look at projects in the simulation and workflow domain that mention general requirements to their workflow environment. Thereupon, we can investigate requirements that indirectly affect the database architecture because of its relation to the environment. These requirements can mostly be applied to the part of the database architecture that is responsible for the *process information* and to the *instance model*.

Scientific workflows are often long-running ([SKDN05]), while the accessed resources are partially fragile and unreliable. For this reason, the workflow environment has to cope with these circumstances and support failure handling, compensations, and instance resuming ([AMA06]). To do that, the workflow engine has to keep knowledge of all activities – whether they have failed or finished successfully. At the same time, scientific workflows tend to be very computation-intensive and exploit the technique of parallelism ([PHP+06]). Hence, the workflow engine has to access the states of the activities in many parallel transactions. Therefore, we can derive requirements for the database architecture: support for a performant access to the instance model as well as full ACID↗ support for the state transitions of the activities. $\mathbb{R}$ **25**

Model complexity and exploratory modeling are other challenges [PHP+06] address to typical simulation workflows. Their solutions involve hierarchical decomposition and loose coupling

of modular components – two aspects that affect the database architecture. While multiple hierarchical workflows directly cause a large workflow model as well as a large instance model, more communication arises at runtime, which leads to a lot of runtime information. The database architecture has to be able to keep and access the model in a performant way $\mathbb{R}$ **26**. As history and provenance have become important issues for scientific workflows ([DCBE+07], [MI06], [Slo06]), the runtime information becomes an important data base. The database architecture has to support sophisticated retrievals for the data stores related to runtime information. Scientists might even use this information in an enriched form for runtime investigations and to look at intermediate results as [Tan07] suggests. Moreover, a mechanism has to be provided to drop runtime information that is not needed anymore $\mathbb{R}$ **27**.

[SKDN05] suggest a closer integration of the computing capabilities of databases within workflow environments. Although we do not follow the proposed workflow techniques, we agree with others that it is beneficial to let the database perform some data manipulations directly, instead of implementing the logic in an external form. For this reason, special calculation capabilities of databases should be investigated: spatial transformations, date time calculations, string operations, etc. Moreover, the capability to implement stored procedures in any form is useful for the database(s) storing the application data. $\mathbb{R}$ **28**.

Most scientist have only a limited information science and especially database background ([PHP+06]). The database architecture can tackle this issue by pointing out tool support for the encapsulated databases. These tools should help the scientist in managing database schema and database storage as well as showing potential solutions for certain issues. In addition to that, [PHP+06] emphasize the constant modifications of scientists to their environment. Here, tools can support the scientist too by easing data migration, data import, data export, and backups.

Scientific workflows often exploit distributed computing on the basis of grids or clusters in order to handle complex calculations and data-intensive transformations. The database architecture does not need to support such an environment, as it focused on local data provisioning only.

3.3 Requirements due to Potential Database Applications in Simulations

In this section, we take a look at concrete utilizations of databases in simulation and scientific workflows. Thereupon, we can create general ideas for the usage of the integrated database architecture. Moreover, we can derive abstract requirements due to potential database applications.

[GLNS+05] state that scientific research and computer simulations create vast data stores that require new scientific methods for analysis and organization. [GLNS+05] see databases as an essential part of the solutions for this challenge and give a number of reasons. Modern database systems provide a database schema, which implies some *semantic information* of the data and hence simplifies the organization and structuring of data. Moreover, [GLNS+05] emphasize that most relational database systems support SQL as their query language and SQL supports a powerful semantic algebra, perfectly suited for analysis. Finally, default database *design principles* are beneficial for data organization, such as the view concept and data independence. The latter

represents the concept that databases hide their internal physical storage of the data from client requestors and can therefore decide the optimal storage strategy without affecting the clients.

A lot of simulation projects use customized data storages which are specially implemented, whereas [NSGS+05] show that the utilization of current database technology would have great advantages. First, database systems do not need to be implemented, since they already exist. Therefore, scientist can spend their time on modeling the simulation instead of writing special data stores. Second, as database systems rely on standards (e. g., SQL and XQuery), they are *interchangeable*. The same simulation might be executed on a small embedded database for prototype runs and on a high-end database system in productive mode. Third, the usage of standards facilitates the utilization of a vast set of *tools* for databases, e. g., database schema modeling tools. To sum this discussion up, the database architecture guarantees a *structured storage* for most kinds of simulation data, provides some semantic information due to its database schema, and relies on standard based database languages.

An important issue regarding data stores is *optimization*. Scientists want to write request to the data store – or even get the requests generated – and do not want to think about optimization. [NSGS+05] see a great opportunity for databases in this context. Database vendors have spent a huge effort in creating indexes and optimizing queries. A utilization of these techniques seems appropriate: while scientists can concentrate on writing queries for any analysis question, the database is responsible for their efficient execution. Furthermore, [HG07] state that good *scalability* and *self-management* features of databases are more issues where the database can take responsibility from the scientist. We aggregate these thoughts into a requirement and address it to the database architecture. ℝ **29**

Besides the structured storage and optimization aspects, databases support *transactions* and guarantee *ACID*↗ compliance – a major aspect for concurrent and reliable data processing. When multiple simulation activities in a workflow access a database object concurrently, transformations have to be performed in an atomic and isolated manner. When an activity makes changes to application data, a database ensures its consistency at the end of the transaction and can fail the activity if an assertion is violated. Although most databases support transactions and the ACID concept, we require the inspection of these aspects during the database evaluation. ℝ **30**.

In the following paragraphs, we investigate possible applications for the database architecture in workflow environments. We start with discussing the application as a *local cache*. [FCL97] present different techniques to establish a local cache and investigate its principles. The basic idea of a local cache is to retrieve a set of data from a remote data store and keep to it near to its application instead of sending all requests to the remote data store. A local cache can be realized by a software layer which selects physical database pages to be kept locally. An alternative solution is to manually fetch remote tables into a local temporary table, before or when accessing it for the first time. While data services (see Section 2.6 on page 44) try to execute data-intensive calculations near to the data and utilize the remote storage, a local cache minimizes the access rate to the remote storage, decreases the network traffic, and exploits the resources of clients. Following [Rei08], we list questions that help to decide which approach is suited better:

1. How much of the remote data is touched during the data analysis (or calculation)?

2. How much data is returned by the analysis as a response to the client?
3. How often does the workflow repeat the analysis?

If only a small part of the remote data is touched, a large ratio of this data is returned, and the analysis is repeated a great number of times, a local cache is the better alternative. Otherwise, a data service, realizing a function shipping to the remote database, is the better choice.

Another application of a local cache arises at compute-intensive simulations. [DMS+06] have stored intermediate and final data products after their creation into a "catalog" and *reuses* them from that cache. Thereby, some activities of the simulation workflow can be omitted increasing the average execution duration of the workflow.

When considering current simulations and simulation workflows, a lot of file handling is involved for input, output, and intermediate results. [HG07] have investigated this issue in the domain of finite element method. Their suggestion was to store the finite element grid into a database and support subsequent calculations with the database, too. They state that the pre- and post-processing steps benefit from the database alternative. Moreover, intermediate file parsing and serializations are omitted, as all the data handling is done by the database. Scalability, concurrency, and data management aspects are clear advantages of databases over file based data storage. Concrete requirements to the database architecture regarding this context can be found in Section 3.4.1 on page 56.

Besides the large amount of data simulation workflows are accessing, the heterogeneity of the distributed and partially unreliable data sources is another challenge ([DG06], [PHP+06]). In order to build up simulation models and perform the simulation, different data repositories have to be accessed (cf. Ecological Niche Modeling example in Section 2.3.6 on page 34). Often a set of preparatory activities is responsible to extract, transform, normalize, and load the data into an own data storage. This technique is called ETL (Extraction Transformation Load) process in data warehouse jargon. In [MMLD05] the workflow concept and ETL processes have been compared and a combination of both technologies has been proposed. The synergy can be a BPEL workflow enriched by SQL activities and performing the ETL activities. Another software based solution for the integration of heterogeneous data are data integration frameworks, which have been evaluated by [BMR09]. Either way, the integrated database architecture can serve as the target of the data integration phase and provide a reliable and homogeneous view of the data. Concrete requirements to the database architecture regarding this context can be found in Section 3.4.3 on page 64.

When working with databases, scientist often have problems to adopt their programming model to appropriate database capabilities. [GLNS+05] called this the impedance mismatch and investigated issues scientists have with databases. Some of these issues have already been discussed before – we want to consider data types challenges now. [GLNS+05] mentioned that scientists have difficulties when trying to store N-dimensional arrays, time-sequences, or spatial data into a relational table. In order to address these challenges, we demand support for the following techniques by the database architecture ($\mathbb{R}$ **31**):

- support time and time-sequence data types

- support Binary and Character Large Objects (BLOB and CLOB)
- support custom scalar `DOMAIN` – data types based on default SQL data types with additional assertions
- support the definition of customized structured data types (respectively record types)

On basis of these requirements, the database architecture can provide customized data types that resemble the programming model of the scientist.

[HG07] proposed that for some use cases XML annotations are very useful to enrich relational representations of simulation model elements in order to add useful attributes or even history and provenance data. As the structure of an *XML attachment* is not bound to the relational schema, its content can be dynamically modified and enriched during calculations and simulations. Therefore, we address a requirement to the database architecture that XML attachments at relational records have to be supported in any kind. $\mathbb{R}$ **32**

To sum up this section, we state that the database architecture can be a great benefit for a runtime environment for simulation workflows. We have found application fields of the database architecture, such as a local cache and the data storage for data preparation workflows. Moreover, we have addressed additional requirements to manifest the database architecture as a scalable, optimized, transactional, and self-managing data storage within the simulation environment. Finally, we have thought about techniques to simplify the adoption of the database architecture by scientists.

3.4 Workflow Scenarios and Use Cases

The previous sections have presented general requirements to a simulation workflow environment and considered various database integration aspects. In order to structure these requirements and prioritize them, we introduce three scenarios within this section. The scenarios are based on several simulation examples. For each scenario, the corresponding examples look similar in their coarse-grained structure and utilize similar data integration techniques. Some of the examples have already been introduced in Section 2.3.6 on page 29 – additional examples are described within this section.

The purpose of the scenarios is to establish an abstract view on different types of simulation workflows. On this basis, we can illustrate how the integrated database architecture can be applied to simulation workflows resembling a common scenario. In the following, these *application fields* of the integrated database architecture can be used to collect concrete requirements and derive criteria for the subsequent database evaluation (see Chapter 5 on page 81). Moreover, we argue about benefits simulation workflows have when using the database architecture. In addition to the application fields, we can discuss possible improvements when the proposed database architecture is integrated into the simulation workflow environment. Therefore, we can exploit various ideas, listed in Section 3.3 on page 51, e. g., the motivation for a local cache database. For all scenarios we address *alterations* in order to integrate the database architecture.

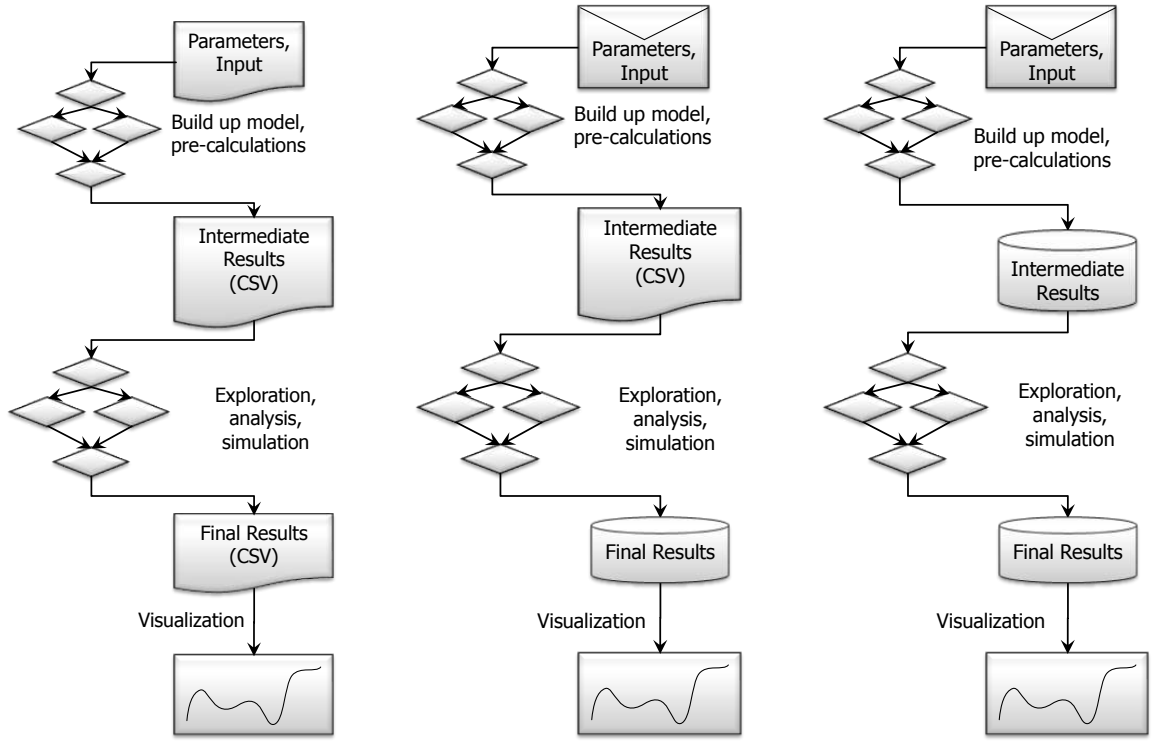
The scenarios describe their simulation workflow environment and apply the scientific workflow classification as shown in Table 2-i on page 19. By highlighting specific aspects of the environment, we can conclude basic requirements to database architecture and its databases.

The focus of the scenario discussion lies on their data provisioning aspects. We investigate which kinds of data sources are used in particular: databases, XML, CSV, or binary based files. In this context, it is important how the workflow activities access the data sources and how they process the data. Some workflows use services for data provisioning, others directly pass files in a data flow from one activity to another. If the workflow is just orchestrating activities, it is crucial to examine whether the data is transfered to the orchestrating workflow and back again by value or whether pointers or other techniques are used to transfer the data by reference. In order to suggest improvements, the scenario descriptions estimate the size of the data sources and the amount of data that is shipped from the data source to the workflow activities and vice versa.

Furthermore, we take a look at the read and write behavior of the activities. For some scenarios, the activities are part of a pipes and filter design ([GS93]) that leads to buffers that are filled by a process and are used as input by its successors. Other data flow oriented workflows may create temporary files that are written once by an activity and read once or multiple times by future activities until they are not needed anymore. If databases are accessed by activities, some databases might only be used for retrieval, others might only be used for storing the results, and some might be accessed by both read and write activities concurrently. A scenario states, which of the patterns is the case.

Finally, transactional behavior and requirements of the data integration is described for each scenario. Some scenarios might not use transactions and only rely on a final result stored in a persistent storage. Other scenarios might use transactions to guarantee isolation and concurrent access. Some examples rely on the durability of intermediate results to allow the recovery of failed activities using this persistent state. Atomicity is another possible requirement of some scenarios. When an activity alters the state of the simulation model in a non-atomic scope, it may cause severe inconsistency. Moreover, compensation mechanisms might be exploited by scenarios (see Section 2.2.2 on page 12 for a brief discussion of the compensation mechanism in BPEL workflows).

As outlined before, the aim of this section is to present application fields of the database architecture in the context of the scenarios. The application fields might arise by replacing current data sources with others, or by introducing new databases to the workflows. In this context, it is important that the semantics of the simulation workflow are not altered and that new requirements have to be discussed in detail. As a result, the description of each scenario clearly distinguishes between the current state and the proposed modification.



(a) current simulation

(b) simulation activities orchestrated by a workflow

(c) simulation re-implemented as a workflow with intermediate results in a database

Figure 3-ii: Schema of a simulation that is typical for Scenario I

3.4.1 Scenario I – Calculations and Equation Solvers

Description and Requirements

The first scenario resembles the usage of a simulation framework such as DUNE or ChemShell (see Section 2.3.4 on page 24). Simulations based on such simulation frameworks either use scripts or a programming language to describe the activities forming a simulation process. Therefore, their development is control flow oriented. As discussed in Section 2.3.1 on page 16, we do not consider these simulations as workflows, because they do not orchestrate loosely coupled activities.

The simulations of Scenario I are often used for a repeated simulation of a fixed application context, and so their structure is rarely touched and only the input is changed in the respective executions. Following the classification of [LAB+06] (see Table 2-i on page 19), such simulations are productive, science-oriented, and compute-intensive. The extensive use of files for input, temporary data, and final results is characteristic for this type of simulations. Figure 3-ii-a on this page shows a schema of a simulation that fits in Scenario I.

As the schema depicts, the input for the first activity is typically a set of files that contain parameters and the initial simulation model or a structured description to develop a model. A set of activities has to parse these files, which may be CSV↗ files, of an proprietary structured ASCII format, or even binary files. Using the given input, data structures representing the simulation model are build up in memory. Afterwards, the model undergoes a transformation and refinement process.

Simulations in this scenario tend to call commands or modules as their activities that do not always share the simulation model in a common memory. For that reason, intermediate files are often produced that represent the data passed from one activity to the next. In order to keep the model small in memory, some activities might store their intermediate results into files, too. Not until a future activity needs this information, the files are parsed again.

When the preparation of the simulation model is finished, the simulation is started. The simulation framework may parallelize the work to manage compute-intensive calculations, but the model and the simulation processing are often kept in memory. Activities that represent the simulation processing frequently alter the state of the simulation model (cf. discussion in Section 2.3.2 on page 19). These intermediate states are recorded into files after given time intervals. When the simulation is finished, the final state of the simulation model is written to a file and a special variant for visualization purposes is exported. The scientist can use the final state and the visualization to investigate the course of the simulation and its findings.

When analyzing the data provisioning for Scenario I, we can summarize that it is all done by using files. In order to prepare the following *alterations*, we can distinguish:

1. Files that contain input data and are only read.
2. Files that contain temporary data. These files are created, filled with data, read, maybe updated, and potentially dropped.
3. Files that contain intermediate logging information or the final result, which are only written during the simulation.

As said before, the format of the files is often specifically designed for the purpose of the simulation domain and requires specific parsing and serialization techniques for which the simulation frameworks provide modules. As file processing of this kind is not transactional, we have no requirements according to atomicity or durability. Moreover, each file is only written by one activity at a time. Multiple concurrent activities might read the file afterwards in an non-isolated manner. The size of the files and their number might vary for different instances of Scenario I. If the duration of a simulation grows, there might be hundreds of files with gigabytes of data in total.

Alterations and Improvements

The first idea of an alteration is the continuation of the work of [Rut09]. Therein it was shown how simulations that fall in Scenario 1 can be integrated into a workflow based on BPEL by

using Web Services. With the result of this work it is possible to orchestrate the activities of such simulations from the start to the end. To retrieve the final results, a Web Service to get the content of result files has been implemented. Our first alteration of such simulations is the provision of the result data using a database. The integration of the simulation into a workflow is illustrated in Figure 3-ii-b on page 56.

This alteration does not require the modification of the simulation activities, but requires an additional activity that parses the final results and loads it into a database. The superordinate workflow can use the results in the database for further analysis itself, or it can propagate a database reference to subsequent activities. This approach is not possible by just using references to the result files – the following activities might not even touch the same computer node.

The requirements for the database architecture for this application field cover parsing the specific result data and storing the parsed data. For the parsing, individual extensions of the Web Service interface of [Rut09] have to be developed. Moreover, specific database layouts have to be designed that can capture the specific result data. The database architecture has to support extraordinary column types if needed. $\mathbb{R}$ **33**

The load of the data represents the insertion of a large number of rows into typically few tables with potential referential constraints. Therefore, a requirement for the database architecture is the fast processing of so called *bulk loads*, while guaranteeing referential constraints for the result data. If needed, special consistency constraints have to be checked by the database architecture, too. $\mathbb{R}$ **34**

As the major intention of the storage of result data in a database is to ease its further processing, typical *analysis queries* have to be supported by the database architecture. Hence, full table scans retrieving the model are a requirement as well as selective retrieval, by primary or foreign key predicates. This represents the case when an activity only needs a small subset of the results for its work. Furthermore, the result data might be split into several tables and table joins are required to reestablish a meaningful view.

As there have been no requirements concerning transactions, the only requirement for the database architecture is the atomicity of the data load. Either all result data is load into the database architecture or none. In order to allow parallel analysis of the result data, the database architecture has to support the concurrent full or selective retrieval within an appropriate time.

The next idea for an alteration concerns the input data for simulations of Scenario I. In order to call a simulation with a given set of input data, the database architecture is be applied, too. This approach supports superordinate workflows that have a set of preparatory activities that assemble or collect the input for our simulation. After the preparation, the database contains the input as raw data or a structured representation of it. The database architecture has then the task to serialize the data into files and to provide them as the input for the simulation workflow. All further requirements to the database architecture for the fulfillment of this alteration are covered by the discussion before.

Simulations of Scenario I spend a great effort in serializing intermediate results and parsing them again and again. For this purpose, the last alteration is the utilization of the database architecture

for the intermediate data, too, or even to hold the model entirely as [HG07] have suggested for finite element grids. This means that the activities would directly use the database for storage and retrieval and would not parse and serialize files anymore. The simulation would have a structured and reliable storage for temporary data as well as for durable data. The modified schema for Scenario I is shown in Figure 3-ii-c on page 56.

The last alteration requires a redesign of the simulation frameworks, it is not the intention of this thesis to implement this approach. The purpose of this discussion is to lead to the following suggestion: when a scientist intends to create a new simulation workflow that falls into Scenario I and he or she discovers that a vast amount of time would be spent with serialization and parsing, it would be a better choice to rely on a simulation framework that stores intermediate data in a database.

We have shown how the database architecture can be used to store input and output data of simulation activities. This approach improves the orchestration of the activities of such simulations into a workflow, because the preliminary data provisioning and the subsequent data analysis benefit of advantages of the databases. As the database architecture is integrated into the simulation workflow environment, the access to this input and output data is simplified. Furthermore, we have made a suggestion to exploit a database storage for intermediate results, too, and thereby to avoid time-consuming file serializations and parsing.

Within the work of [Rut09], two simulation frameworks have been extended and two example workflows have been used to demonstrate the usage of the implemented Web Service interface. As they fall into Schenario I and are suited for a prototype, we demonstrate the application of the database architecture for these example workflows. The following two use cases of Scenario I present the examples accompanied with particularized requirements.

The use cases originate from simulation frameworks applied in related SimTech projects. By improving their integration into workflows, we make a further step towards a simulation workflow environment where simulations based on these frameworks can be orchestrated beside other simulation activities. The database architecture is utilized to store input and output data and thereby simplifies the loosely coupling of simulation activities.

Use Case I – Computational Fluid Dynamics Using Dune

For the first use case, we consider the DUNE example simulating ink dropping into water. This simulation is based on a computational fluid dynamics approach and FEM (see Figure 2-ix on page 31).

For this use case, we implement a prototype that stores the initial finite element grid in the database architecture. As the ASCII↗ file for the input of the use case is very small in size (only 193 B), it is not worth to be split into tables and rows. Thus, a requirement for the database architecture is the storage of arbitrary text files using CLOBs (Character large object, a default database column type).

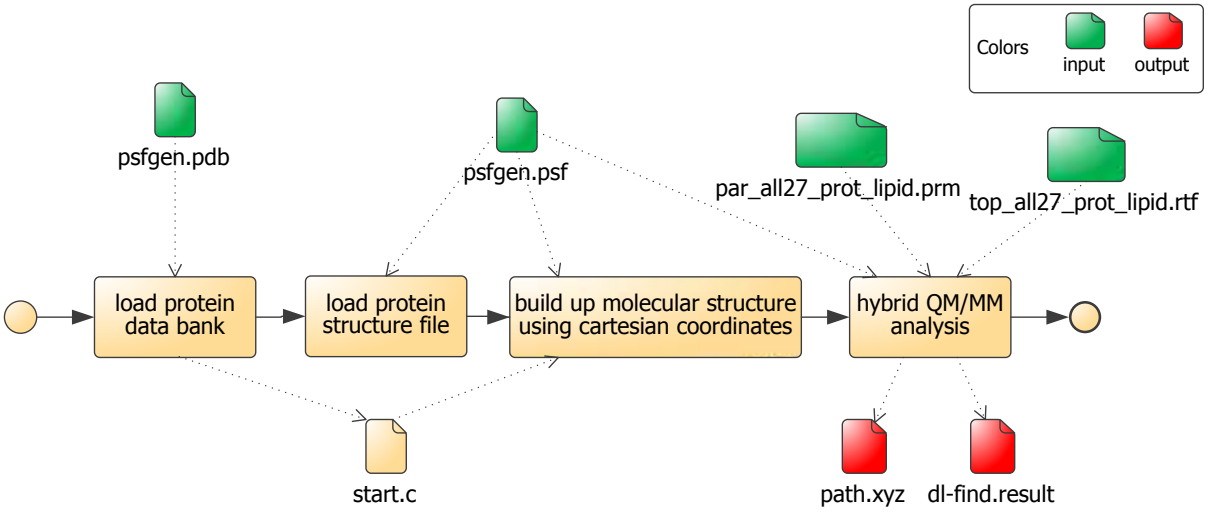


Figure 3-iii: Simple ChemShell example with input and output references

The intermediate and final concentration of the ink is currently stored into VTU files. They are the input for a visualization tool and based on a XML format. The database architecture has to store these files in its XML database to ease its utilization in potential further analysis and visualization activities. As these VTU files contain a typical finite element grid, the database architecture demonstrates its application as a relational storage for result data by storing the grid in its relational storage, too. A parser and a relational schema have to be developed for this approach.

Use Case II – Catalysis Reaction Using ChemShell

The second use case is based on a simple ChemShell example shown in Figure 3-iii. In contrast to the sophisticated catalysis example (Section 2.3.6 on page 32), this simulation does not require a cluster-based environment and has already been integrated into the Web Service framework by [Rut09]. Hence, it is applicable as a database architecture prototype.

The example ChemShell simulation is based on a hybrid quantum mechanics/molecular mechanics calculation where different input files are involved:

- A protein data bank[1] file (PDB) that contains ATOM records describing the atom residues of the protein and their coordinates.
- A protein structure file (PSF) that specifies the bonds and angles between the atoms of the protein. This file is linked to ATOM descriptions of the PDB file.

[1] http://www.rcsb.org/pdb/

- A topological file (RTF) containing additional information about all important types of residues. This file is reused for multiple different simulations.
- A parameter file (PRM) containing additional constants that are needed to simulate pairs, triples, or quads of atoms (e. g., equilibrium values and spring constants). This file is reused for multiple different simulations.

A task for the database architecture prototype is to store all this information in the database architecture and to implement an appropriate data provisioning activity preceding the first simulation activity. In order to demonstrate the relational storage using the database architecture, the PDB file is stored in the relational database using a table. Additionally, the PSF, RTF, and PRM files are stored using CLOBs↗.

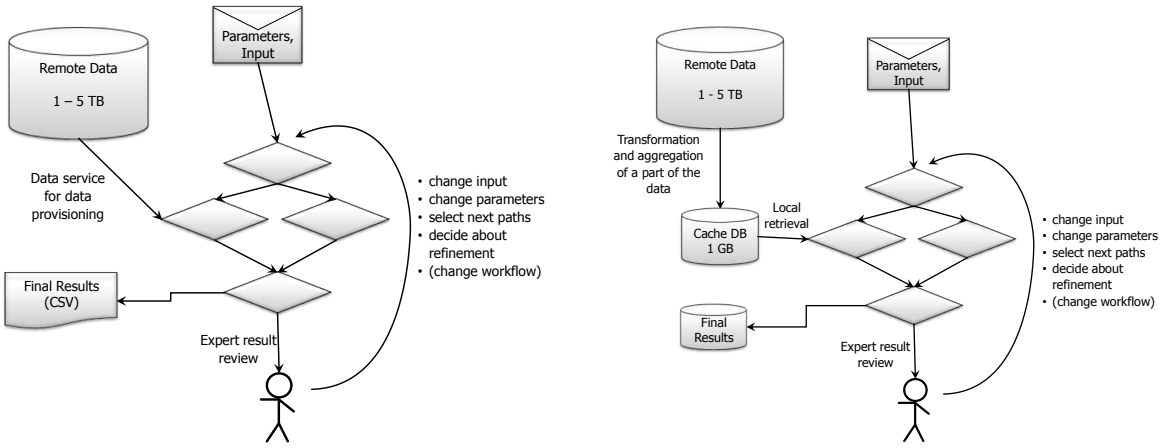
After the hybrid simulation, the simulation has created twelve intermediate and final result files. For the sake of simplicity, we just load the `dl-find.result` file into the database architecture. This file contains the three dimensional coordinates of each atom of the simulated protein and their connections after the simulation. This information can be well represented by a relational database model. Referential constraints between the atom records must be supported by the database architecture.

3.4.2 Scenario II – Large Remote Simulation Models

Description and Requirements

In molecular and genome research, a number of public databases have emerged in the last decade that contain a huge collection of centrally shared chemical data to describe proteins, enzymes, gene sequences, and other compounds. [BKT00] show an example collection of a few of these databases. One of these databases is GenBank, which contains nucleotide sequences for more than 300,000 organisms and is publicly available ([BKML+09]). A sample workflow using GenBank is presented by [CCbD06]. The workflow downloads gene sequences from GenBank, builds up a local gene sequence model, creates alignments between the sequences, and refines them. Afterwards, a workflow activity derives a phylogenetic tree of the refined sequences. A scientist is involved in verifying the quality of the results. The whole process is repeated until the quality is satisfying. This simulation workflow illustrates the shape of Scenario II.

The schema for Scenario II is depicted by Figure 3-iv-a on the next page. The workflows in this scenario are executed by a script or a simulation framework and started with a set of input data or messages. Another input originates from a remote data storage – e. g., a database – and has to be retrieved using data services. Data services allow a standardized access to the remote data and might provide additional functional logic (cf. Section 2.6 on page 44). As the GenBank example has shown, the simulation needs this remote data to build up the local simulation model and enrich this model with additional parameters. After the first iteration of the simulation on this model has been finished, a scientist is involved. He or she validates the quality of the simulation results, changes the parameters, or even alters the structure of the simulation workflow by changing the order of the activities.



(a) current data provisioning using data services **(b)** optimized data provisioning using a local cache

Figure 3-iv: Schema of a workflow that is typical for Scenario II

The simulation is repeated a number of times in order to improve the simulation model and the output. Each time, identical or new information has to be retrieved from the remote storage. This retrieval may cause high data traffic. When the simulation has build up a satisfying model, a final result is written to a set of CSV files. These files contain the final simulation model and potentially provenance information. The provenance information describes the history of all iterations of the simulation and all decisions that have led to the final simulation model.

Another example falling into Scenario II concerns simulations in the automobile industry. Manufacturers have spent a great effort in creating huge, almost realistic models of their cars that are used for simulations. These models can be applied for predictions of the kinematic behavior of a car or to simplify the development of test cases (cf. SimTech project "A Framework to Capture Digital and Physical Test Cases" [2]). As the models are precise and have thousands of parameters, they often have a size in the magnitude of terabytes. When a developer wants to simulate how the car acts in specific circumstances, he or she typically only needs a part of the gigantic model. For example, the developer only needs information about the frame of the car, tires, and dampers to simulate the car driving on a bumpy road. He or she then might vary the damper rate and repeat the simulation until he or she has optimized the shock absorption by the dampers.

Summing up the requirements for the data integration, we can state that there is a storage of data in the magnitude of terabytes. The simulation workflow only needs a part of this data repeatedly. The access to the data is only retrieval. Adjustments to the data may be aggregated and deferred after the last iteration of the simulation is finished, because not until the last iteration the best simulation model has been created. The data set actually needed by the workflow is estimated with 1 GB. The data provisioning is done by specific data services, such as Web Services, a SQL connection, or by accessing files (the GenBank service provides the gene sequences utilizing

[2] http://www.simtech.uni-stuttgart.de/forschung/abstracts/Antrag_Lange,_Mitschang_Abstract.pdf

an FTP↗ interface). Either way, the requestor expects to receive a consistent snapshot of the remote simulation model, which might force transactional behavior on the side of the remote data source.

For the result data, typically multiple `CSV` files are created. These `CSV` files share a common schema that is split into multiple relations. The trace data has the form of a log with records for each decision or intermediate state. The files are only written by activities and no transactional requirements are applied here, too.

Alterations and Improvements

The repeated requests to the remote storage might cause a performance lack for workflows that fit in Scenario II. Therefore, we propose the implementation of a local cache ℝ **35**. For this approach, we use thoughts of [Rei08], who considered the usage of a temporary cache within the BPEL engine, and of [FCL97], who investigated client-server cache consistencies. The adapted schema of a Scenario II workflow is shown in Figure 3-iv-b on the preceding page. The main motivation for the local cache lies in the reduction of the data traffic by keeping the data local (see discussion of advantages for a local cache in Section 3.3 on page 51).

The local cache allows the simulation workflow to be independent from the remote data services during the simulation. The database architecture can keep the data of the local cache in the best suited form – relational or XML based, regardless whether the remote data storage already uses databases or simply files. Hence, the storage of the local cache in a database enables the usage of integrated database workflow activities on the local cache, e. g., using the SIMPL BPEL data management extensions (see Section 2.6 on page 44). On basis of this tighter database integration, various optimization ideas of [VSS+07] can be applied after the alteration.

In order to load the data needed for the simulation, the database architecture has to be instructed to retrieve the data once. As only the workflow modeler can decide which data the simulation needs, he or she has to add one or more preparatory activities to the whole simulation workflow. These activities are responsible to fill the cache once before the data of the local cache is needed for the first time. The data loading can be realized by transferring data from a relational database to the local cache or by parsing the response of data services. Moreover, the retrieved data can be the subject of basic normalizations and aggregations. The database architecture has to be able to load such a big amount of data at best time performance and guarantee referential and other constraints afterwards. If there are multiple data sources involved, a concurrent load into the local cache shortens the time needed for this data load. Thus, the concurrent insertion into different relational tables or XML documents has to be supported by the database architecture, too. (cf. ℝ **34** on page 58)

As the local cache is mostly used for retrieval during the simulation activities, we address no transactional requirements to it. But, it is queried in parallel. To provide better performance for these – potentially compute-intensive – requests, we apply the in-memory database of the database architecture for this application field. At this point, we cannot place requirements to the

data structures or the potential queries on the local cache. Hence, we demand that the database has a good time and memory performance in a generic retrieval oriented benchmark. ℝ **36**

An important point concerning the in-memory aspect is the total size of the local cache. [Rei08] does not consider in-memory databases for his temporary cache and assumes that there is always enough storage space to store the provisioned data. However, the data capacity is limited when utilizing an in-memory database. The performance of in-memory databases decreases tremendously when touching swapped memory pages or they do not allow swapping at all. For this reason, the database architecture has to support the scientist in estimating how much memory is totally required by the local cache. Another solution might be the support for efficiently swapping seldom used pages by the in-memory database.

For being only temporary, the data of the local cache is deleted when the simulation has finished all related calculations. The database architecture has to support this by easing the deletion of whole schemata or XML document collections. Moreover, the database architecture has to free the related main memory and delete possible logging records on the disk eventually. ℝ **37**

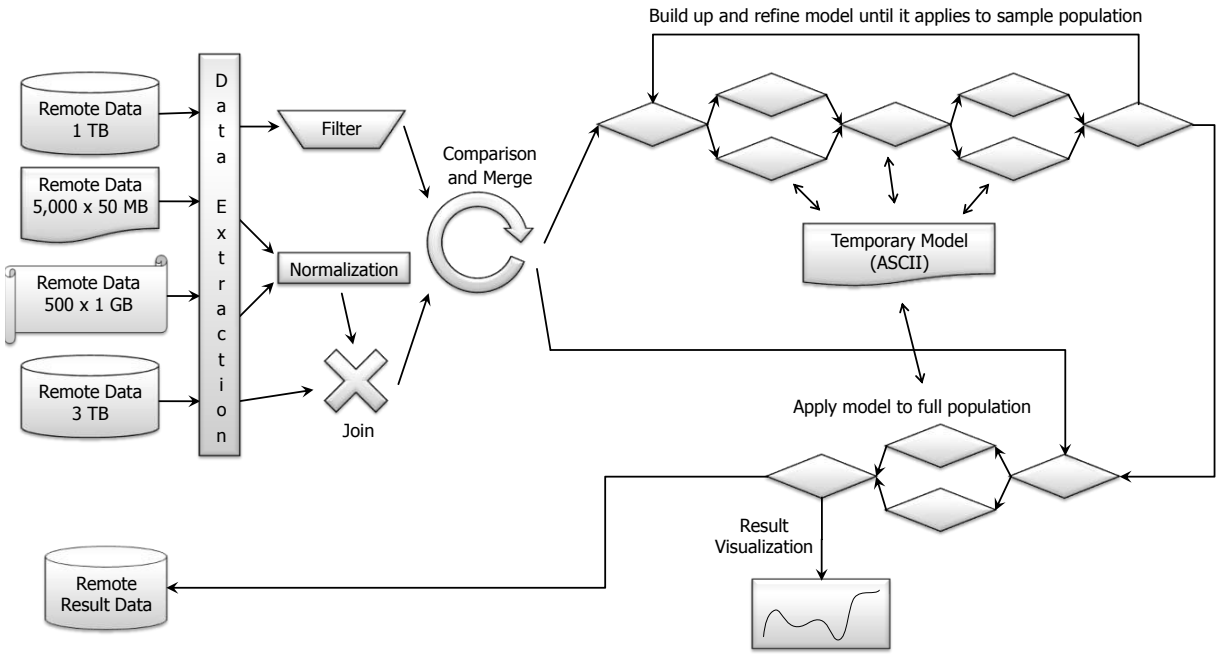
If the workflow makes adaptions to the simulation model, the workflow might propagate the whole simulation model or the alterations to the remote storage after the simulation has finished. Hence, the database architecture has to support this kind of synchronization with remote SQL or XML databases. To realize this requirement, the database architecture might exploit existing features of the involved database systems or might instead implement a customized approach. At least, the database architecture has to provide a method to perform a load of all or some relations (respectively XML documents) into a remote storage. A customized approach does not need to implement a sophisticated synchronization approach, e. g., logging modified records or calculating a difference or delta update. ℝ **38**

Another alteration addresses the result data. As shown in Figure 3-iv-b on page 62 we intend to store the result data into the database architecture. The workflow modeler can decide which database of the database architecture is used as target of the result data – the database of the local cache or another. The relational representation of the result CSV data provides a better structure and guarantees referential and other constraints. Moreover, the storage of the result data in the database architecture eases the usage of the results by further activities. For this application field, the same requirements as described in Scenario I are applied.

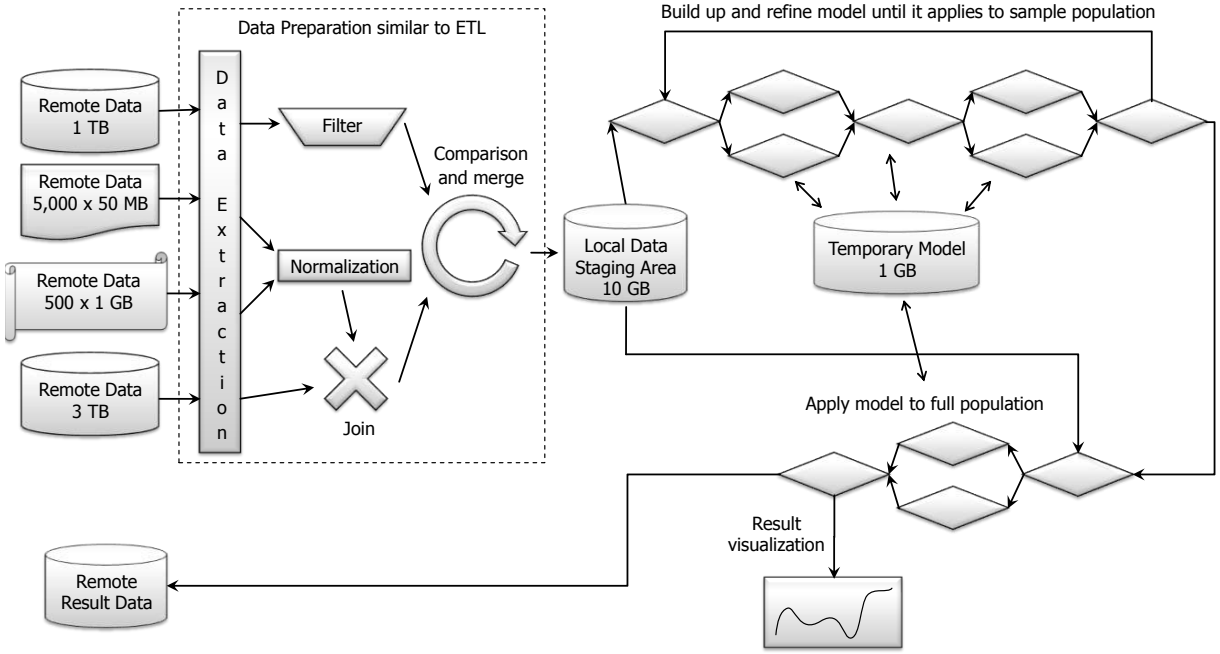
3.4.3 Scenario III – Data Provisioning From Large Heterogeneous Sources

Description and Requirements

The idea for Scenario III originates from the Kepler example simulating biodiversity of species (see Section 2.3.6 on page 34). This workflow is an example for workflows having the typical data provisioning phase preceding the actual simulation. Figure 3-v-a on the next page shows a schema of a workflow that is typical for Scenario III.



(a) current structure of the workflow



(b) ETL↗ based data provisioning and local database for temporary data

Figure 3-v: Schema of a workflow that is typical for the Scenario III

In order to build up the simulation model for simulations falling into Scenario III, the data provisioning activities have to access a huge number of heterogeneous remote data sources using different protocols. Inspired by the descriptions of [TDGS06], we can make up a list:

- XML↗ files
- CSV↗ tables
- Structured ASCII↗ files
- Binary files
- Data Services through a UDDI↗
- Arbitrary files using FTP↗
- Grid Data Services

In addition to loading data from different resources, the data is based on different schemata and has to be normalized. Such normalizations might involve the transformation of values into a common unit or scaling of numerous values. The names of identifiers (e. g., for columns) have to be unified and values have to be transfered into the same granularity; e. g., a text value might contain a listing of sub-values and has to be split. All in all, this phase requires a huge effort of parsing files and restructuring the data of different models into a basis for the following simulation. Usually not all of the remote data is needed – often data is aggregated, filtered by certain predicates, or some properties are not necessary. Hence, the total amount of data needed by the simulation is only a fraction of the data of the remote source. For the estimated 5 TB of remote data in Figure 3-v-a on the preceding page, we might get about 10 GB as input for our simulation. We call this input data D.

The second phase of our Scenario III is spent by training simulation models S. This might be necessary, because we only can develop a description of our domain of discourse manually without knowing the parameters or we can only guess what the relationships between the modeled entities are. [PHP+06] use information of a sample partition of D in order to derive relationships, annotate facts, refine parameters, and create simulation model S_1. The prediction capability of model S_1 is verified using one or more sample partitions of D. This process is repeated a number of times, enriched by ideas of a genetic algorithm. In each iteration the current simulation model S_i is modified and verified until a better model S_{i+1} is found. Not until the predictions are within a given error range, the second phase ends with the best simulation model S_b found. Not all workflows that resemble Scenario III have to use such a genetic algorithm, too. We have just described the approach of [PHP+06] in order to derive typical data usage patterns.

We have compute-intensive and data-intensive calculations and comparisons in the second phase. They are performed by a lot of activities running concurrently and may query all data of $\{D, S_0, \ldots, S_{i-1}\}$. The approach might even develop multiple competing simulation models S^j parallel that all have to be verified. This results in a lot of files used for the intermediate models

and verification outcomes, as not all of this information can be kept in memory. We estimate the total size of all simulation models in Equation 3-i:

$$\begin{aligned} n &= \text{number of parallel models} \\ b_j &= \text{number of iterations until best model is found for } S^j \\ \text{total size of simulation model} &= \sum_{j=1}^{n} \left(\sum_{i=1}^{b_j} sizeOf\left(S_i^j\right) \right) \leq 1 \text{ GB} \end{aligned} \tag{3-i}$$

The third phase is very compute-intensive, because it represents the actual and detailed simulation using the best simulation models $S^j_{b_j}$. There might be a time discretization for the simulation and, for each step, data from D and the simulation models $S^j_{b_i}$ are touched by multiple activities. We do not make further assumptions for the methods of the simulation, but expect the calculation of multiple subsequent simulation states for each of the best simulation model.

The final result for workflows fitting into Scenario III consists of the final simulation states for each simulation model $S^j_{b_j}$ and the simulation models themselves. These results are exported in a remote database for further analysis. Moreover, a scientist investigates the outcome of the final simulation states for each $S^j_{b_i}$. To do that, the scientist can either use exported visualization file or start a data analysis for the exported data.

Besides the Kepler example, the SimTech↗ project "Model reduction in flexible multi-body dynamics" at the University of Stuttgart ([Feh09], [FFHE09]) can be classified into Scenario III in a wider sense. The geometry of bodies is imported from a Computer-aided design (CAD) tool and discretized using the finite element method. The finite model is the input of the reduction process and very complex due to its many nodal degrees of freedom ($\gg 10,000$). The finite elements are stored in a lot of files that resemble the *Remote Data Sources* of Figure 3-v-a. The model reduction now has the task to reduce the degrees of freedom and keep the most relevant ones. A combination of Krylov-subspaces and frequency weighted Gramian matrices forms the basis of the reduction technique. Within an optimization loop, an adaptive learning technique is applied. A greedy search algorithm repeatedly calculates snapshots of all degrees of freedom, which resembles the *model refinement loop* of Figure 3-v-a. Again and again the output of the finite element discretization has to be analyzed, which could be interpreted as the *Filtering* and *Comparison* activities of Figure 3-v-a. After the optimization loop, additional post-processing activities improve the snapshot selection and store the reduced finite element model. [Feh10] states that currently the process contains a lot of manually performed steps and can therefore not yet be considered as a workflow in our sense. Nevertheless, the orchestration of the steps may be possible based on the Web Service framework by [Rut09].

Alterations and Improvements

When looking at the first phase of Scenario III, we are reminded of the ETL↗ discussion of Section 3.3 on page 51. Figure 3-v-b on page 65 highlights the related extraction, transformation, and load steps. The implementation of the ETL-like process as a BPEL workflow is the subject of our first improvement for Scenario III.

Workflows falling into Scenario III can use the SIMPL framework (see Section 2.6 on page 44) in order to model such ETL processes. As SIMPL is intended to use a generic data source model, the integration of both relational and XML databases is possible. Parsers for the CSV and ASCII files can be provided by specific SIMPL Data Access Services (DAS). In this context, the database architecture concerns the target data storage for this ETL process. In a data warehouse system, so called *data staging areas* are used to load the data from various data sources in a temporary storage and to serve as a medium for all transformation and aggregation activities. This fact forces a whole set of requirements.

The database architecture has to provide a storage that can keep about 10 GB of data at a time. Typical ETL↗ tasks (cf. [WK07]) have to be supported, such as:

- bulk loading of relations
- table content manipulations
- data movements
- data type conversions
- string, date, and measure conversions
- data aggregation
- data accumulation
- data consistency checks
- duplicate elimination

In data warehouse systems the aggregated data is typically moved or copied from the data staging area into a specific storage (e. g., a data mart) for further OLAP purposes. Within the database architecture, we can also use the database that represents the data staging area for the queries of the second phase of the simulation. The data loaded in the data provisioning phase must be kept in a durable storage, as its preparation costs a huge effort and the data can be reused by multiple simulation workflows sharing the same data D. In order to allow optimizations during the ETL processing, no requirements to durability and consistency are forced *during this phase*. However, the database architecture has to guarantee the durability and consistency of model D after the end of the first workflow phase. As we can neither provide example data nor queries for such a data staging area, the database has to be evaluated using generic retrieval based benchmarks.

[Feh10] states that for the project “Model reduction in flexible multi-body dynamics” many manually performed data conversion steps have to be performed to adopt the output of the finite element method. When applying the database architecture, the ETL tasks *bulk loading*, *content manipulations*, and *duplicate elimination* can be modeled as workflow activities in order to automate the conversion.

The second alteration regards the intermediate simulation models that are originally stored in multiple temporary files. As discussed in detail in Scenario I, we intend to replace the usage of files by a temporary simulation model stored in the database architecture. In addition to the

requirements stated in Scenario I, we add the support for parallel activities that each retrieve, insert, or update the data of S_i or retrieve data of $\{D, S_0, \ldots, S_{i-1}\}$. The database has to guarantee isolation for these activities, while implementing intelligent locking mechanisms in order to avoid blocking requests in that phase.

The last alteration is more a design suggestion that arises from the usage of the database as a fast and intelligent storage. As the second phase is very compute-intensive, we use the idea of [DMS+06] to reuse calculated intermediate results. When compute-intensive calculations have been finished, they might store their arguments and results in the database architecture. The latter serves as a cache for frequently calculated results and can support activities in that phase by using the cached results instead of calculating them each time on their own.

3.5 Summary of the Application Fields

During the inspection of Scenario I to III we found the following application fields for the integrated database architecture:

1. Store input data of simulation workflows and provide them.
2. Store result data of simulation workflows to ease the integration in superordinate workflows and simplify the access of subsequent analyses and visualizations.
3. Avoid serialization and parsing activities by using databases as a structured and reliable storage for intermediate results.
4. Load only a partition of a remote storage into a local cache and increase the query throughput by magnitudes.
5. Serve as a data staging area for data-intensive ETL↗ workflows and subsequent retrievals on this storage.

Accompanied by the description of the requirements, these application fields lead to the following design of the database architecture in Chapter 4 on page 71 and serve as a basis for the development of evaluation criteria in Chapter 5 on page 81.

4 Simulation Workflow Database Architecture

The database architecture specified in the previous chapter is to be integrated in a simulation workflow environment (cf. $\mathbb{R}$ **1** on page 48). This chapter describes the architectural aspects of this integration.

We start the discussion with a coarse grained view on a possible logical architecture of a simulation workflow environment in Section 4.1. Thereupon, we describe the relevant components of an environment on the basis of Apache ODE (see Section 2.5.3 on page 41) and SIMPL (see Section 2.6 on page 44) in Section 4.2 on the next page. This environment is demanded by the task of this thesis and is used in other current and future projects in the SimTech context (e. g., [Rut09]). A presentation of the integration of the database architecture into this workflow environment is another part of this section.

Furthermore, Section 4.3 on page 76 describes the components used in the prototype of the database architecture, as the SIMPL framework is currently developed and can not be utilized yet. Finally, a brief discussion in Section 4.4 on page 78 shows how the concept of the database architecture can be applied by other simulation workflow environments.

4.1 Logical Architecture

The intention of this section is to describe a possible logical architecture of an environment for simulation workflows. Currently the development of such an architecture in the SimTech context is in progress, but not yet published. Therefore, we use thoughts of [LLF+09] and the architectural descriptions of the "Trident Workbench" by [Mic08]. We continue with a brief description of four layers of a possible logical architecture and mention related frameworks applied in the context of this thesis (see framework overview in Section 2.5 on page 38).

The *Operational Layer* consists of heterogeneous data sources, remote services, and tools. This layer is explicitly separated from the remaining workflow management system in order to state its independence. The other layers only depend on interfaces or protocols provided by remote data sources (e. g., SQL) or services (e. g., specified via WSDL↗).

The *Task Management Layer* serves as an integrator of the resources of the *Operational Layer* for the other layers. The *Data Management* component abstracts the heterogeneous data sources by providing a common view and common data types to the other components. A thought through design of the *Data Management* component is crucial, since its degree of abstraction has to be traded against the achievable degree of performance and throughput. This component is further

responsible for the access to and movement of data and the efficient execution of related activities – even in a distributed environment. The SIMPL framework serves as the *Data Management* component in our environment. The *Task Management* component is responsible for executing various tasks during the execution of a simulation workflow, e. g., data transformation activities, remote service calls, event handling, etc. During the execution of a workflow, the *Provenance* component receives necessary meta data to build up a provenance model. In addition, this component is responsible for the storage, archival, search, and visualization of this provenance model.

The *Workflow Management Layer* contains the *Workflow Engine* component, which instantiates workflows, schedules their activities, and executes all necessary work. The *Workflow Engine* delegates a major part of the work to the *Data Management* component and the *Task Management* component. Moreover, the *Workflow Engine* is coupled to a *Workflow Monitoring* component, which observes the status of each running workflow instance and provides techniques in the presence of failures. Apache ODE is used as a *Workflow Engine* within this thesis. In addition, Apache ODE is tightly integrated into Apache Axis2 to perform Web Service related activities. Hence, Apache Axis2 can be seen as a part of the *Task Management* component, too.

The *Presentation Layer* represents the graphical user interface for the scientist. On the one hand, it contains modeling and designing tools to create workflows and ease their deployment. The Eclipse BPEL Designer is an example for such a *Workflow Modeler* component. On the other hand, the *Presentation Layer* provides a user interface for *Visualization & Presentation* to the scientists so that they can manage data sources, monitor and administrate workflows, and investigate provenance issues. Apache ODE already contains a web-based tool for simple monitoring and administrating tasks. The SIMPL framework provides a user interface for managing the data source registry.

4.2 Components of the Architecture

Now, we look at the components of the simulation workflow environment more closely. In contrast to the previous section, we only consider components that are relevant to data integration into simulation workflows, especially database integration. A complete overview of these components and their interactions is depicted in Figure 4-i on the next page. The arrows in this figure represent call relations rather than data transfer directions. The background colors in this figure represent:

- already existing frameworks and data sources
- components provided by the SIMPL framework
- generic Web Service Interface realized by [Rut09]
- components implemented in this thesis

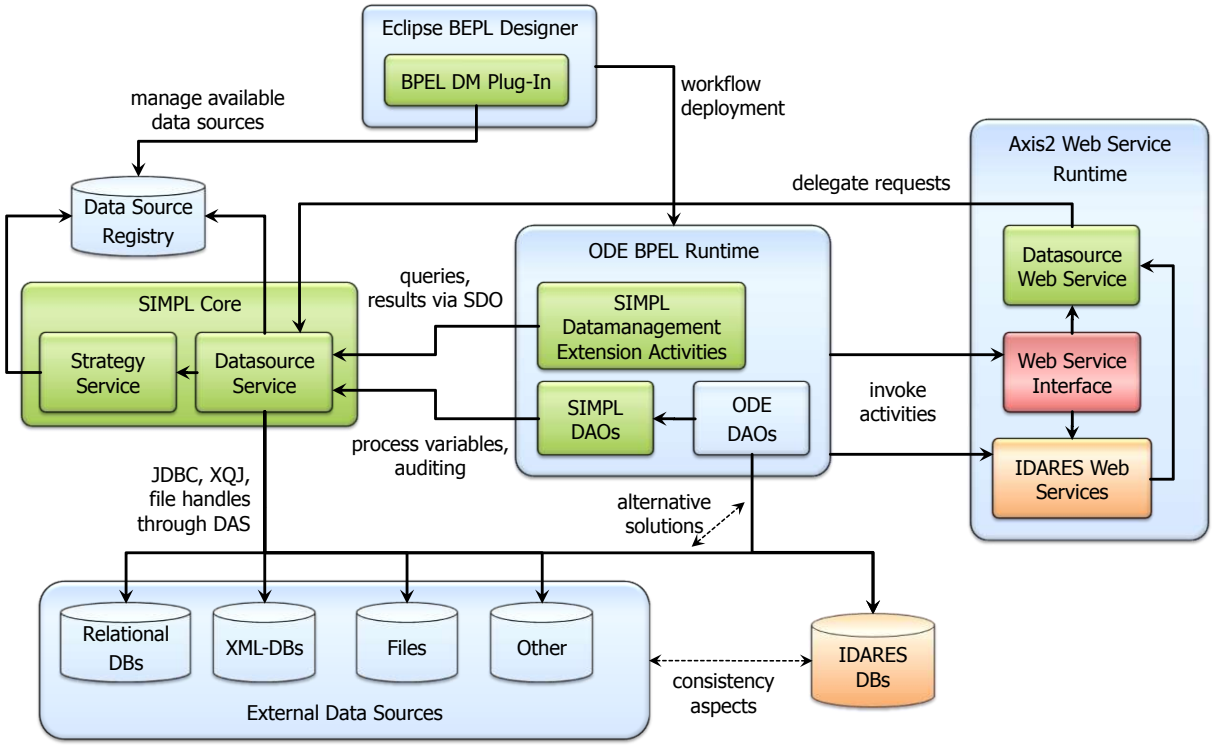


Figure 4-i: Components for data management in the simulation workflow environment

The heart of the simulation workflow environment is the *Apache ODE BPEL Runtime*. It is responsible for the execution of simulation workflows and their life-cycle management. As Section 2.5.3 on page 41 has shown, there are multiple ways to deploy Apache ODE. In the context of this thesis, we need neither features of an application server nor functionality of a SOA↗, but want to capable to interact within a SOA. Therefore, we rely on a simple web application deployment within Apache Tomcat. This variant of Apache ODE is tightly integrated into the *Apache Axis2 Web Service Runtime*. Axis2 is utilized in our architecture as the component where Web Service operations can be deployed.

As ℝ **6** on page 48 demands, the SIMPL framework is to be exploited for data integration into the workflow environment. The following description of the architecture is based on version 1.5 of the SIMPL design document ([Stu09]). The *SIMPL Core* is deployed within the Tomcat Runtime as a runtime extension to the Apache ODE web application. The SIMPL Core contains the *Datasource Service* that is responsible for the access to all data sources. The *Datasource Service* utilizes adapters that realize Data Access Services (DAS). These adapters abstract from heterogeneous data sources and provide a common access pattern. They are applied to guarantee extensibility for various data sources and query languages (e. g., SQL↗ via JDBC↗ or XQuery via XQuery API for Java (XQJ) [OC09]). The *Datasource Service* resolves specified data sources using the *Data Source Registry*, which is implemented as a UDDI↗. The *Strategy Service* can optionally utilized to apply a late binding of data sources.

The *SIMPL Datamanagement Extension Activities* are extension activities according to the BPEL specification. This means that they allow the definition of data management activities within BPEL workflows. These activities are implemented in Java and deployed within the *ODE BPEL Runtime*. *SIMPL Datamanagement Extension Activities* are only wrappers for data management operations to be executed by the data sources and delegate their queries to the *Datasource Service* in the *SIMPL Core* directly. According to the type of the data management activity, a result might be returned and marshaled into a BPEL variable or only a reference (e. g., to a cursor over an SQL result set) might be returned.

Another extension in the *ODE BPEL Runtime* are the *SIMPL DAOs*↗. The *ODE DAOs* are a set of interfaces used as an abstraction layer for the persistent storage of runtime model and variables of BPEL workflows in ODE. The job of the *SIMPL DAOs* is to implements these interfaces and to redirect the storage of runtime model and variables to an arbitrary data source managed by the *SIMPL Core* (see Section 2.5.3 on page 41 for details on the DAO mechanism).

The *Workflow Modeling* component introduced in Section 4.1 on page 71 is represented by the *Eclipse BPEL Designer*. Enriched with a *BPEL DM Plug-In* it serves as a graphical simulation workflow modeling tool, including the modeling of data management activities. Furthermore, the *Eclipse BPEL Designer* supports a deployment to the *ODE BPEL Runtime*, and the *BPEL DM Plug-In* provides a user interface to manage available data sources. Data source descriptors are stored directly in the *Data Source Registry*. These descriptors are used during the workflow modeling and are resolved at workflow runtime by the *Datasource Service*.

The external access to the *SIMPL Core* is realized by Web Services, whose implementation is deployed in the *Apache Axis2 Web Service Runtime*. For example, there is a *Datasource Web Service* that accepts data management requests through SOAP↗ messages. The *Datasource Web Service* internally delegates requests to the *Datasource Service* in the *SIMPL Core* by directly calling the appropriate Java class. The corresponding deployment issues to allow this direct access are described in [Stu09].

Besides the SIMPL framework, the simulation workflow environment provides the generic *Web Service Interface* developed by [Rut09], too. As described in Section 2.3.5 on page 29, the Web Service framework provides operations to start, execute, and interact with dynamic simulations. These operations are specified in WSDL↗ and deployed in the *Apache Axis2 Web Service Runtime*. The simulation workflows in the *Apache ODE BPEL Runtime* can invoke these operations using the default `invoke` activities of BPEL. The *Web Service Interface* is deployed in the same JVM↗ and accessed through endpoints on the *localhost*.

The integrated database architecture provides potentially multiple *IDARES*↗ *databases* to realize the storage of relational and XML data (ℝ **10**, ℝ **11** on page 48), based on external storage and main memory (ℝ **12**, ℝ **13** on page 48). These databases are installed and started on the same computer node as the *Apache ODE BPEL Runtime*. Therefore, the databases are close to the workflow runtime, can be integrated into the runtime more easily, and can be accessed without causing network traffic (cf. ℝ **1** on page 48). As stated before, all data sources are integrated into the simulation workflow environment using the SIMPL framework. Hence, the *IDARES databases* have to be registered in the *Data Source Registry* of SIMPL. The data source descriptors specify

the capabilities of the *IDARES databases* and that they allow a fast, local access to relational or respectively XML based data.

A workflow modeled in BPEL can access the *IDARES databases* through *SIMPL Datamanagement Extension Activities*. The activities reference the appropriate *IDARES database* through their descriptor (the SIMPL project has not yet published the details for this referencing technique). The *Datasource Service* is responsible to resolve the references and accesses the *IDARES databases*. The *Datasource Service* is written in Java and needs a driver to access the databases. In the case of an SQL based database, a JDBC↗ driver in the form of a `JAR` file is provided in the `lib` folder of Apache ODE. Apache Tomcat loads the JDBC driver for the ODE web application and thus for the *SIMPL Core*. As the SIMPL project has not yet published details on their implementation of the integration of XML databases, we expect that a kind of database specific driver is required too.

When concerning the *application data* of the workflow, the integration of the *IDARES databases* is completely abstracted through the SIMPL *Datasource Service*. Hence, no adaptations to the workflow language BPEL are required. All in all, the requirement ℝ **2** on page 48 to integrate the *IDARES databases* into Apache ODE is satisfied by this approach.

The SIMPL framework has already presented a technique to redirect the storage of the process information, runtime model, and variables of Apache ODE to another database. We utilize the related *SIMPL DAOs* in order to apply one of the relational *IDARES databases* for this storage (we call this database *ODE model db*). Since the SIMPL framework implements the *ODE DAO* interfaces completely, the *ODE model db* simply has to realize the SQL schema that is requested by Apache ODE and has to be configured as the target of the *SIMPL DAOs*. As a result, the process information, instance model, the variables, the messages, and the event data of Apache ODE are stored in the database architecture (fulfilling ℝ **16** and ℝ **17** on page 49).

Although we cannot investigate details of the *SIMPL DAOs* yet, we have to raise the question of performance at this point. Requirement ℝ **26** on page 51 states that large runtime models of Apache ODE have to be stored, kept, and accessed. When looking at the architecture, we can conclude that a request for a runtime model entity (e. g., the value of a variable) has to pass the *SIMPL DAO* layer, the *Datasource Service*, and the relational DAS↗ adapter. Moreover, the result has to be mapped from relational data accessed via JDBC into a common model provided by the DAS↗ and into the Java objects representing the *ODE DAOs*. Although this approach might cope with large data, the effort of mapping and abstracting data is relatively high. Since the database for the variables of the *ODE BPEL Runtime* does not change often, a fixed configuration setup integrating the *ODE model db* might be more appropriate. When the SIMPL framework is implemented, a performance evaluation might investigate this issue. The same applies for requirement ℝ **35** on page 63, which demands the application of a database as a fast, local cache. The degree of abstraction via the *Datasource Service* might decrease the overall performance of the local cache.

Another important issue at this point is the extendability of BPEL. As other projects in the SimTech context probably depend on domain or simulation specific BPEL activities, a number of extensions might be implemented. If an extension has, for example, the form of a *new extension activities* or a *new extension assign operation*, an implementation of the required interface of ODE will be

provided. Hereby, the implementation can use the current form of the *ODE DAOs* or implement additional DAOs for other runtime model objects (e. g., provenance data). However, a *new extension activity* can be seamlessly integrated into the *ODE DAO* mechanism and does not even know whether a database of the integrated database architecture is applied underneath or not. The *ODE DAO* mechanism serves as a perfect abstraction layer at this point.

New functionality might also be provided by additional Web Services that that are deployed within the simulation workflow runtime. Their implementation might access data sources through the SIMPL *Datasource Web Service*. Due to the fact that the *Datasource Service* serves as an abstraction layer too, the new functionality is decoupled from all databases. All in all, the database architecture supports future extensions to the simulation workflow environment and is applicable in their context too.

Besides the *IDARES databases*, a set of services is provided due to the work on this thesis. These services are implemented as Web Services and deployed in the *Axis2 Web Service Runtime*. The services are:

1. An *ODE model db* helper that provides operations to clean information from the runtime model and to perform other ODE specific tasks (cf. ℝ **27** on page 51).
2. A service to assist with the local cache. It contains operations to synchronize between remote databases and the local cache (cf. ℝ **38** on page 64) and to drop data from the local cache (ℝ **37** on page 64).
3. Extensions to the *Web Service Interface* in the context of the prototype implementation (see Section 4.3 for details).

The SIMPL framework can be utilized to load data from heterogeneous data sources into the local cache and back again by modeling a corresponding set of BPEL activities. The local cache helper service provided by this thesis is rather intended for a fast copy of data from a relational database to the local cache where performance matters. As the *Datasource Service* always applies a mapping to a common data model on basis of Service Data Objects (SDO), unnecessary and expensive mappings are applied. A straightforward row by row copy from a remote table to the local cache is in most cases less resource intensive and therefore faster.

4.3 Components of the Prototype Architecture

Due to the fact that the SIMPL framework is yet to be implemented, we need a temporary simulation environment for the following purposes:

1. evaluation of databases within their target environment (ℝ **18** on page 49)
2. implementation of a prototype to demonstrate the application of the integrated database architecture (ℝ **23** on page 50)
3. testing and demonstrating features of the database architecture not utilized in the prototype (ℝ **24** on page 50)

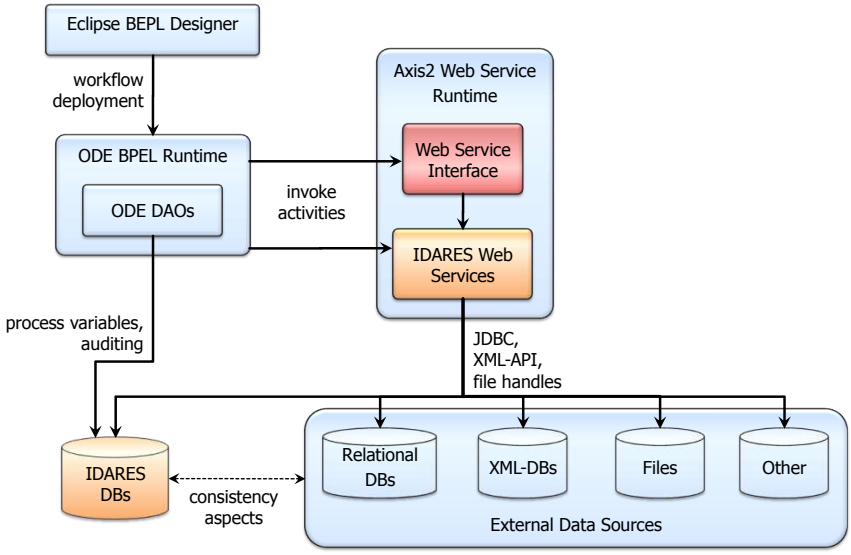


Figure 4-ii: Components required for evaluation and prototyping

Figure 4-ii depicts the components of the prototype architecture.

The heart of the prototype architecture is the Apache *ODE BPEL Runtime* too. Apache ODE is also deployed in web application mode into an Apache Tomcat container. To access databases in this environment, we use a combination of `DataSources` and Web Services, which is explained in the following paragraphs.

We do not implement a database registry, but rather use the Apache Tomcat resource descriptor mechanism to manage database connections. The connection settings for each registered database are specified in the XML based configuration file `context.cml`, including connection URL, user name, password, and additional connection properties. Each database connection has a unique name for further references. Apache Tomcat implements a connection pool to these databases and provides a `DataSource` object for each configured connection. A web application – or Web Service – can query these `DataSource` objects using a UDDI↗. When a `DataSource` object is received, it can be used to create JDBC↗ `Connection` objects. Therefore, a JDBC↗ driver has to be provided in the Apache Tomcat `lib` folder for each registered database.

Instead of providing a slim version of an extension activity to access databases from BPEL workflows, we implement a utility Web Service that handles generic SQL and XQuery based database requests (see *IDARES Web Services* in Figure 4-ii). The Web Service is to be called by BPEL `invoke` activities and is implemented in Java. The utility Web Service receives the unique name of the target database in the request message, retrieves the appropriate `DataSource` object through the UDDI, and creates a JDBC `Connection` object. The `Connection` object is used to execute the SQL (or respectively XQuery) request. The Web Service transforms the result of the request into a simple XML based structure. This approach provides a simple and yet flexible

way to access databases via queries from BPEL workflows (cf. ℝ **7** on page 48). Besides the Web Service operation that provides access to databases, we also implement import and export operations for relational (cf. ℝ **8** on page 48) and CSV based data (cf. ℝ **9** on page 48).

The prototype of this thesis has to load and store input and result data for the DUNE and ChemShell examples (cf. ℝ **33** on page 58). Therefore, we need Web Service operations that can parse and serialize the relevant files. These operations are integrated into the current form of the *Web Service Interface* by [Rut09] and also share common functionality with the *IDARES Web Services*. The adapted *Web Service Interface* is to be deployed in the *Axis2 Web Service Runtime* too. We bundle the *Web Service Interface* and the *IDARES Web Services* to a single Axis2 Archive (AAR) in order to allow direct internal access between the Java classes. This avoids unnecessary Web Service calls and the associated marshaling between Java objects and their XML representations. All Web Services can be accessed from the *ODE BPEL engine* via localhost endpoints.

4.4 Portability to Other Workflow Environments

ℝ **3** on page 48 demands that the integrated database architecture has to consider the integration into other simulation workflow environments too. Since we have started the architectural discussion with the description of an architecture of such an environment (see Section 4.1 on page 71), we now can apply our thoughts on its basis.

The integrated database architecture is more a set of concepts and ideas than a tightly coupled, ready-to-use component. Hence, its applicability to other simulation workflow environments just requires the adoption of the concepts of the integrated database architecture. Any simulation workflow environment that resembles the architectural layers we discussed in Section 4.1 can be used to adopt the concepts.

The following steps are required for the integration of the database architecture into other workflow environments:

1. The workflow environment has to provide a technique to access databases from within simulation workflows. Section 2.6 on page 44 has already presented some possible techniques – most simulation workflow environments already have a *Data Management* component as described in Section 4.1. Since the SIMPL framework has a core that is independent from the workflow engine and just requires a JVM↗, it might be applicable too.
2. A number of databases have to be evaluated for their applicability. If platform and workflow engine are similar to the environment presented in this thesis, a part or the whole database evaluation of Chapter 5 on page 81 can be adopted.
3. The database(s) of the integrated database architecture have to be installed and configured on the same computer node that also keeps the workflow engine.
4. The database(s) have to be registered at a database registry or a connection pool so that the *Data Management* component knows them.

5. One database of the database architecture can store the runtime model and variables of the workflow engine and can provide storage of the *Provenance* component. If the workflow engine already has a productive database applied for this purpose, it can be investigated whether this database is also applicable for other application fields of the database architecture.

To sum the discussion up, we can state that the principles and ideas of the database architecture can be adopted by other workflow environments. Nevertheless, each simulation environment has different requirements to data integration aspects and relies on different architectural decisions. Therefore, the components described in Section 4.2 on page 72 represent just one possible realization of the database architecture, and their composition might not be transferable to all other simulation environments.

5 Database Evaluation

To fulfill the general requirements of the task of the thesis (see Section 3.1 on page 48), the requirements by the scenarios (see Section 3.4 on page 54), and to realize the components proposed in the architectural design (see Section 4.2 on page 72) several database techniques are needed. Hence, we have to select one ore more available database systems and evaluate their applicability for the integrated database architecture.

The evaluation approach applied in this thesis consists of four successive steps. In Section 5.1 the evaluation schema is presented. Following in Section 5.3 the databases being short-listed are described briefly. Additionally a discussion argues why other database system are not listed. The evaluation schema is applied in Section 5.4, followed by the final selection in Section 5.5.

5.1 Evaluation Schema

In order to compare the short-listed databases, we contrive a schema that evaluates which databases are most applicable for the database architecture proposed before. The schema can be seen as a catalog of evaluation criteria that cover different aspects. While most criteria are derived from the requirements directly, others are added for completeness. Every criteria states its motivation and the related requirements. The criteria are structured hierarchically, have a name, and contain instructions how to apply them. Finally a criterion may specify how the degree of applicability can be quantified and mapped to a scale. For non obvious scales a criteria states which value range is optimal. For some criteria, the inspection of a list of features is required, which can be recorded and compared in the form of a check list.

5.1.1 General Information and Database Management

Criterion 1: License and Costs This criterion asks for the license of the database system. If the license is a kind of open source license, a remark about the copyleft[1] is required. If otherwise a database has a commercial license and has to be purchased, a price estimate has to be noted. Special offers for scientific applications might be noted too.

[1] If a software is published with license having a copyleft note, all modifications, additions, and derivative works using this software have to be distributed under the same license.

This criterion is motivated by ℝ **19** on page 49 and introduced because open source database systems are preferable. Nevertheless, this criterion is considered as a recommendation rather than a absolute necessary condition.

Criterion 2: Platform This criterion examines supported platforms of the database system. This includes supported operating systems as well as support for 32 bit and 64 bit. If the database has additional hardware or software requirements, they have to be stated too. Furthermore, a Java based database driver (preferably a JDBC↗ driver) has to be supported by the database. The criterion considers the compatibility of the driver to the workflow environment.

This criterion is motivated by ℝ **21** on page 50. Its intention is to proof for a database system that it is applicable where the typical platform of the runtime environment for simulation workflows is working. This criterion constitutes a necessary condition.

Criterion 3: Installation and Integration The database systems have to be installed on the runtime environment for their evaluation. This criterion requests a short description of the installation procedure. Especially the duration of the installation, the installation size, and the required user experience have to be stated. Moreover, this criterion investigates whether there is a installation wizard and a installation manual. These issues are stated in ℝ **20** on page 49.

In addition, this criterion investigates all aspects that consider the actual integration into the workflow environment based on Apache ODE. All important configuration aspects for this integration have to be noted and evaluated for their impact.

Criterion 4: Administration and User-Friendliness Besides the installation, the scientist working with the integrated database architecture might be involved in database management activities. This criterion investigates provided administration tools, web front-end, and assistance in tracing errors. Moreover, the database system manual and further user documentation is considered by this criterion. This criterion is motivated by ℝ **22** on page 50.

5.1.2 Features

The database architecture needs database stores for relational and XML based data (cf. ℝ **10**, ℝ **11** on page 48). The following criteria examine the features of a database in this context. If a database does not support a feature (e. g., an XML database does not support relational storage) the criterion is not applied. Instead a short note is made.

Criterion 5: SQL Features This criterion examines basic relational features of the database as demanded by ℝ **14** on page 48. The requested features include support for SQL↗ queries and support for the following language elements:

- queries for retrieval with projection, selection, join, aggregation, and sub-queries
- queries to modify and delete database objects
- data definitions including constraints and assertions
- triggers
- views
- user management and authorization

Compliance with a specific version of the SQL standard is preferred by this criterion. Moreover, a database specific SQL language reference, including abnormalities of the SQL dialect, can be referenced.

Another part of the investigation of this criterion considers supported data types. The application of this criterion has to investigate whether the database system provides default SQL data types, especially BLOB↗ and CLOB↗ types. Furthermore, ℝ **31** on page 53 has demanded that the relational storage provides date and time types, customized `DOMAIN` types, and customized structured data types.

Moreover, the criterion inspects which calculations can be carried out by the database system. Apart from date and time based calculations, the database system might provide geo-spatial or further statistic functions. In addition to that, ℝ **28** on page 51 demands support for customized stored procedures.

ℝ **32** on page 54 has presented an idea to enrich relational entities with XML annotations. Hence, the criterion has to investigate, whether the database system may support such an approach.

Criterion 6: XQuery Features This criterion applies to databases that support XQuery request to their XML data. As ℝ **15** on page 48 requires support for XQuery ([BCF+07]) requests, the criterion examines the database for XQuery support, in particular:

- import of XML documents and document collections
- manipulations and dropping of XML documents
- `For Let Where Order Return` (FLWOR) expressions
- XPath 1.0 or 2.0 compliance in expressions
- respect XML namespace declarations in queries

The same way a relational database should support stored procedures, the XML database should support the definition of customized modules, especially customized functions. The criterion has to investigate this capability.

Criterion 7: Transactions and ACID This criterion investigates the ACID↗ compliance of the database. In addition, the criterion has to investigate the locking techniques that are used to ensure the isolation aspect of ACID. Since some databases just support table locking instead of row locking, the number of concurrent modifications is limited to just one. This criterion is motivated by ℝ **25** on page 50 and ℝ **30** on page 52 that address ACID aspects to the database architecture.

Criterion 8: Cache Consistency An application field of the database architecture is the provisioning of data to a local cache (see ℝ **35** on page 63). In order to synchronize data from remote data stores, the evaluated database might already provide a helpful feature. Hence, this criterion inspects the database for any kind of technique that can be exploited for cache consistency.

5.1.3 Performance

An important requirement for all data integration activities is performance. In the context of databases the term *performance* not only covers the execution time of database requests, but also resource utilization. In order to evaluate the short-listed databases in this context, we apply a performance evaluation. On the one hand, we have to apply a sufficient number of different tests to make an objective conclusion. On the other hand, we have to limit the number of performance tests to an amount that is realizable within the context of this thesis.

One part of the performance tests are based on database features stated by Criterion 5 and Criterion 6. Thus, we not only investigate if a database supports these SQL features, we also apply them. The data for the performance tests are partially taken from the simulation context (e. g., [CDR+07]). Furthermore, we apply generic benchmarking tests like TPC-H ([Tra08]) for relational databases and XMark ([SWK+01]) for XML databases. These tests resemble retrieval oriented benchmarks as demanded in ℝ **36** on page 64.

We assume that the benchmarks are representative for the simulation context and hence their results are significant for the subsequent database comparison, because:

- The benchmark queries contain typical analysis request as used in Scenario II (see Section 3.4.2 on page 61).
- The data size resembles the order of magnitude we described; e g., the relational benchmark has a size of 0.4 GB and the temporary model of Scenario III (see Section 3.4.3 on page 64) was estimated with 1 GB.
- The benchmark investigates bulk loading CSV data as demanded by ℝ **34** on page 58.
- The benchmark contains complex queries that require sophisticated query optimization techniques (cf. ℝ **29** on page 52).
- The relational benchmark data contains values of various data types, like `DECIMAL`, `INTEGER`, `CHAR`, `VARCHAR`, `DATE`, and `DATETIME`.

- A subset of the tests uses data from [CDR+07], which contains an FEM↗ grid and can be seen as typical for the simulation context.

[HSW08], [Pol07], and [Law05] presented database related evaluations and benchmarks. Their work inspired the performance tests in this thesis with useful thoughts, too. For the evaluation of the databases we use the prototype architecture already presented in Section 4.3 on page 76. The details of the evaluation infrastructure and the *IdaresDBEvaluation* workflow are described in Section 5.2 on page 91. The following paragraphs describe three evaluation criteria, each containing a set of performance tests.

Criterion 9: Performance of Relational Databases This criterion is applied to databases that support SQL requests to their relational data store. The criterion consists of a list of test queries that have to be performed by the database. There are two subsets of performance tests. The first subset originates from a benchmark, the second subset is based on a FEM scenario.

The basis for the first subset of the performance tests is the TPC-H benchmark version 2.8.0 ([Tra08]). TPC-H is an SQL benchmark developed by the Transaction Processing Performance Council[2]. TPC-H is based on a decision support scenario of the business world. Since the decision support queries resemble typical requests for Scenario II and Scenario III, the application of the TPC-H benchmark is adequate for our domain. TPG-H provides a set of 8 tables and 22 queries. The queries reproduce business questions and resemble ad-hoc queries. They exploit a vast set of SQL operators and selectivity constraints. Therefore, they not only apply a benchmark, but also verify the support for certain SQL features (cf. ℝ **14** on page 48).

The amount of data for the TPC-H benchmark can be scaled. The minimal factor 1.0 represents a set of data with a total amount of about 1.0 GB, a factor of 10.0 represents 10.0 GB. The TPC-H download provides a generator tool that creates both CSV↗ data files and SQL query files.

For the application in this thesis, we choose a scaling factor of 1.0 and shrink the largest table to one sixth part of its size. The adaptation is necessary, because although the raw data size is estimated with about 1.0 GB, the databases need much more space to store the data besides additional indexes. As the size of this data should fit in main memory for the test of the main memory databases, a smaller data size was chosen for all databases. This approach eases the subsequent comparison of the databases. However, the discussion in Section 5.1.3 on the facing page has shown that the reduced data size of about 0.4 GB is yet representative. Table 5-i on the next page shows an overview of the tables for this benchmark including the row counts and size estimations. A detail analysis of the table sizes can be found in `[DVD]/Documents/Stuff/TPCH.xlsx`.

The basis for the second subset of relational performance tests is a published FEM grid by [CDR+07]. The grid is downloadable from a related workshop homepage[3] and has the form of three CSV files:

[2] `http://www.tpc.org/`
[3] `http://www.iws.uni-stuttgart.de/co2-workshop/`

Table	Row count	Column count	Size per record	Total size
REGION	5	3	96	479
NATION	25	4	108	2,700
PART	200,000	9	126	25,189,800
SUPPLIER	10,000	7	140	1,402,400
PARTSUPP	800,000	5	139	110,888,000
CUSTOMER	150,000	8	154	23,139,000
ORDERS	1,500,000	9	99	147,765,000
LINEITEM	1,000,000	16	89	88,510,000
$\sum$				396,897,379

Table 5-i: Tables of the TPC-H benchmark; the size values are an estimation of the raw data size in bytes, respecting the actual data types

- `vertices_johansen.dat`: contains 62,800 points of a three dimensional grid and assigns identifiers; size: 2.6 MB.
- `properties_johansen.dat`: contains additional attributes for each point; size: 3.71 MB.
- `elements_johansen.dat`: contains 54,756 finite elements in the form of a hexahedron, where each element refers to exactly 8 points of the vertices file; size: 2.78 MB.

The application of this criterion involves the execution of 75 SQL tests. These tests are selected to cover a great number of SQL operators, selectivity aspects, and statement types. Some tests contain just one operator or predicate, others are more complex. The following enumeration gives an overview of the tests. Alongside a brief description, a motivation is stated to show the significance for the comparison of the databases. This criterion contains:

1. A test that creates the schema and all tables (cf. Table 5-i).
2. Tests that import data from CSV files into tables. These tests measure performance for bulk loading data (cf. ℝ **34** on page 58).
3. Tests that define primary keys, foreign keys, and additional indexes. We create these access paths after the data import has been finished in order to accelerate the previous import. This is a common database technique applied when huge amounts of data are to be loaded into a database.
4. Tests that retrieve the number of rows in a table in various ways. Databases implement different techniques to answer these requests – some have to touch all rows, others do not.
5. Tests that request a table scan, which retrieve all columns and all rows. For the synchronization (ℝ **38** on page 64) and the data provisioning aspect of the prototype (cf. ℝ **33** on page 58) all rows of a table have to be retrieved.

6. Tests that request a predicated retrieval of a subset of the rows of a table, either all or only a projection of its columns. Common analysis queries do not request a full table scan, but rather a predicated subset of the data, often including the sorting by a column.
7. Tests that request a join between tables, using a referential index column or a non-index column. In order to minimize the degree of replication of logical data, relations are split into multiple tables (cf. the description of normal forms, e. g., [Cod71]). These tables are mostly linked by foreign key references. The original view on the logical data is reestablished by applying a join on the related tables.
8. Tests with aggregations. Typical analysis requests not only join tables, but also aggregate the results over multiple dimensions and calculate minimum, maximum, or sum of so called fact values. Moreover, Scenario III in Section 3.4.3 on page 64) applies the database architecture as its data staging area – an application field where aggregations are often used.
9. Tests that utilize multiple techniques mentioned before in one query: complex subqueries, predicated selection, column projections, sorting, joins, and aggregations. These tests are addressing the optimization capabilities of the databases, requested in ℝ **29** on page 52.
10. Tests that simulate concurrent access to a table: read only, read and write, and insert only. These tests, in particular, address the application of the database as the *ODE model database*, which stores the ODE variables and the runtime model. Furthermore, when used as a concurrently accessed storage for intermediate data, the database has to answer concurrent accesses at good performance (cf. Scenario III again).
11. Tests on the FEM↗ model load the CSV files (cf. ℝ **34** on page 58) and simulate typical retrieval operations on the FEM grid.
12. Tests that drop all tables and their data. This is required when a temporary model or local cache is not needed anymore (cf. ℝ **37** on page 64).

All tests and their associated queries are listed in Appendix A.3.1 on page 162.

Criterion 10: Performance of XML Databases The XML database performance evaluation is based on *The XML Benchmark* project, also called XMark ([SWK+01]). XMark describes a benchmark to evaluate retrieval performance of XML stores and query processors. [AM06] have surveyed five XML benchmark projects and attested XMark a well design and a rational selection of queries based on the XQuery 1.0 standard ([BCF+07]). This is why we apply XMark to our XML databases, which all support XQuery to a certain degree.

The description of the requirements and scenarios in Chapter 3 on page 47 mention neither typical XML schemata nor data sizes of XML data in the context of simulation workflows. For the performance evaluation of the XML databases we imagine two applications:

- The intermediate or final result is represented by relational data, enriched by XML based attachments, e. g., provenance data (cf. ℝ **32** on page 54). On the one hand, queries may retrieve the whole attachment for a relational record or on the other hand they might use the XML attachments to find qualifying relational records.
- The final result of a simulation has the form of a large XML file that is to be investigated by a scientist. Typical analysis queries are used to count a number of elements that qualify for a predicate, to find the best element amongst several candidates, or to aggregate certain attributes.

For these two applications, the XMark benchmark provides a set of relevant queries and representative data.

The XMark project provides a data generator tool for both Linux and Windows, which can create the reference data. An example invocation of the generator shows the following listing:

```
./xmlgen-linux.bin -f 0.1 -s 600 -o scale-0.1_split-600/content.xml
```

The output of the data generator can be scaled with a factor, provided by the option `-f`. A factor of 1.0 refers to XML data with a size of about 100 MB; a factor of 0.1 to about 10 MB. Besides the scaling, the data generator lets the output be split into multiple XML files, specified with the option `-s`. When the option `-s` is not given, a single XML file is produced. When used, the option `-s` specifies the maximal number of XML elements per XML document. In this case, the data generator splits the data into multiple XML files. For more details on the data generator, the reader is referred to the website of the XML benchmark project[4]. The domain of discourse of the reference data concerns auctions, bidders, and sold items. These elements are mapped to a tree based XML structure and enriched with additional references between sub-trees of the structure.

For the XML performance evaluation, we generate two XML files with reference data of different scales: 0.02 and 0.1, both without splitting. The files have a size of 2.3 MB, respectively 11.2 MB and represent the two applications we mentioned before: a collection of XML based attachments and a large XML result file. For each of the two XML files, we apply a set of tests. Thereby, we can investigate, how a database behaves with increased data size. We apply the following tests:

- An initial test loads the XML document into the database. As XQuery 1.0 does not define a standardized function to achieve this, we have to apply database specific functions here.
- Tests that utilize database specific queries to retrieve information about stored XML documents and document collections. These methods are important to investigate the XML document storage when only the XQuery API is applicable.
- Tests that count all nodes or all nodes of a subtree. These tests investigate how efficiently the database can provide an overview of the element count.

[4] `http://www.xml-benchmark.org/generator.html`

- Tests that retrieve a full subtree of the original XML file. These tests investigate how efficiently the database can reestablish the original structure of the document after it has been shredded.
- Tests that apply XPath queries with single or multiple constraints to the elements on the path.
- Tests that utilize FLWOR↗ statements in different syntactical variants.
- Tests that join different sub-trees by their references. Hereby, we reveal the performance of the join techniques the databases implement.
- Tests that join different sub-trees by arbitrary values and produce large intermediate results.
- Tests that create a new XML structure based on XQuery element construction and complex queries with hundreds of elements in the result sequence. These queries investigate the generation of reports having an arbitrary structure that might differ from the documents stored in the database.
- Tests that apply a full text search to the text nodes. These tests are important, since XML documents are often utilized to contain textual annotations (ℝ **32** on page 54). A full text search might find objects whose XML annotations contain known keywords.
- A final test that drops the XML document only utilizing database specific XQuery functions. This test is necessary to investigate cleanup capabilities of the databases.

Besides these retrieval oriented tests, tests that manipulate the XML document(s) are needed too. Unfortunately, the three XML databases provide very different approaches to manipulate documents via their APIs. A benchmark that investigates the performance for manipulating XML documents would not be representative and would hence not be appropriate for a comparison. Therefore, we only apply retrieval oriented tests to the XML databases.

All evaluation queries are listed in Appendix A.3.2 on page 170.

Criterion 11: Application as ODE Model DB In order to evaluate the application of a database as the *ODE model db*, we configure the DAO↗ layer of Apache ODE to use the database and then execute a representative workflow. An example workflow has been created in the scope of this thesis to demonstrate the application of the integrated database architecture. Background and implementation of this workflow are explained in detail in Section 6.6.4 on page 147. At this place, we discuss why we apply this workflow for the performance evaluation to potential *ODE model dbs* in the context of simulation workflows.

The example workflow utilizes a genetic algorithm and is therefore distantly related with simulation workflows like the Ecological Niche Modeling (see Section 2.3.6 on page 34). Although the example workflow does not represent a simulation itself, it shows typical characteristics of simulation workflows. The example workflow calls multiple local Web Services and therefore creates a lot of messages and events that have to be handled by Apache ODE. This circumstance challenges the database, because a lot of small event entries and messages have to be stored

efficiently. Moreover, the workflow has about 50 activities that Apache ODE has to navigate over. Although 50 activities are not many, we address $\mathbb{R}$ **26** on page 51, which demands that large process and runtime models can be stored, kept, and accessed. The example workflow has a `ForEach` loop whose body is executed concurrently. Thereby, we investigate compliance with $\mathbb{R}$ **25** on page 50 that demands ACID support for the runtime model in the presence of concurrently executed activities. As stated in [Les09], the default *ODE model db*, Apache Derby, often raises exceptions due to lock timeouts. We investigate this issue during the evaluation.

A last fact that underpins the utilization of the example workflow is its flexibility. The request message contains a number of parameters that have effect on the execution of the example workflow. The parameters `loopCount` and `findBetterNeighborsCount` define the number of repetitions of the main loop, respectively the genetic loop – this can be used to create different workflow instances with varied execution times. A parameter `neighborsCount` effects the number of neighbor individuals that are generated during a loop. Their fitness is executed simultaneously and therefore the parameter correlates with the degree of concurrency of the workflow. By varying these parameters, we can address different requirements to the evaluated *ODE model db*.

The environment for this criterion is the same as for the other two performance criteria. However, for this criterion the example workflow is deployed in Apache ODE and the evaluated database is configured to be used by the ODE DAO layer. The following tests are performed, where the duration is measured for each test:

- Apache ODE is started with a clear database.
- The example workflow is executed consecutively with the following parameters:
  - `loopCount` = 40; `findBetterNeighborsCount` = 0; `neighborsCount` = 1
  - `loopCount` = 8; `findBetterNeighborsCount` = 4; `neighborsCount` = 2
  - `loopCount` = 8; `findBetterNeighborsCount` = 6; `neighborsCount` = 10
- The example workflow is executed with concurrently running instances:
  - `loopCount` = 40; `findBetterNeighborsCount` = 0; `neighborsCount` = 1; 2 parallel instances
  - `loopCount` = 8; `findBetterNeighborsCount` = 4; `neighborsCount` = 2; 6 parallel instances
- Apache ODE is shutdown after the last instance is finished.
- Apache ODE is restarted.

A simple Java application is used as a test harness, as the *IdaresDBEvaluation* workflow does not support concurrent executions of workflows at this time. Besides the measurement of the duration, the application of this criterion has to investigate idiosyncrasies of the databases that ease or hinder their implementation as *ODE model db*. This especially concerns the integration into the DAO layer of Apache ODE.

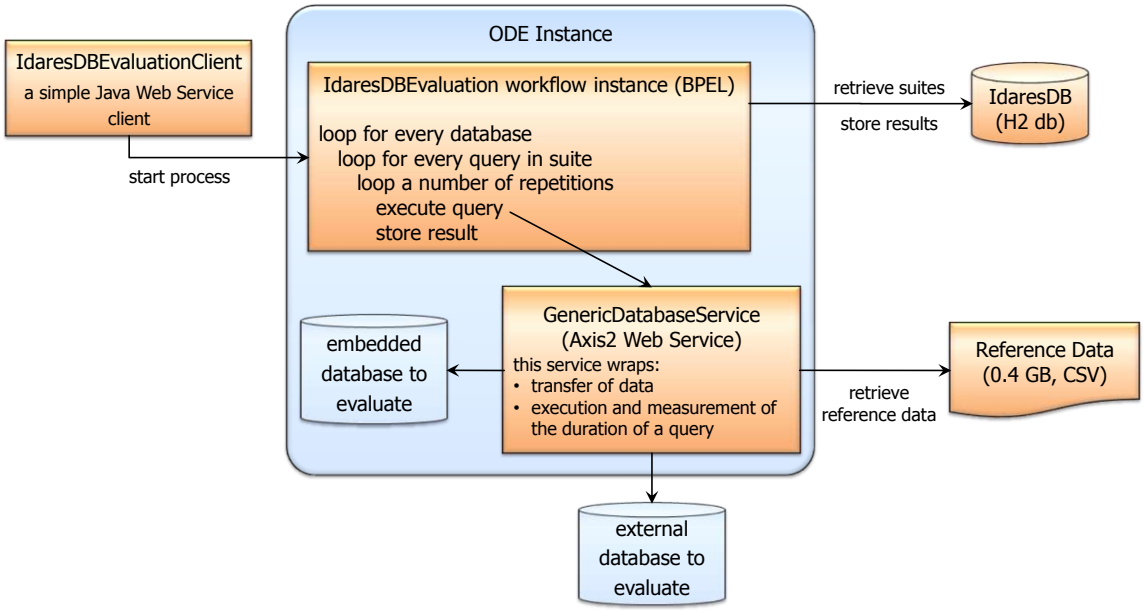


Figure 5-i: Evaluation infrastructure for performance measurement

5.2 Evaluation Infrastructure

In order to test the integration of the short-listed databases and apply their performance evaluation, an evaluation environment is being developed in the scope of this thesis. Section 4.3 on page 76 already described the components of the prototype architecture, we also use for the evaluation. This section presents the whole performance evaluation approach and describes the components more detailed. An overview of the evaluation infrastructure is shown in Figure 5-i. We discuss each of the components before describing the evaluation process later in this section.

As mentioned before, the performance evaluation is based on tests and queries. In order to manage these tests, we keep information of the tests in an embedded database. Thereby, the tests can be easily modified, refined, and queried. Moreover, the result of the performance evaluations can be stored in the database too. Powerful SQL↗ queries can be used to analyze the evaluation runs. The embedded Java based H2 Database[5] is utilized for this purpose. H2 Database is a lightweight, but yet feature reach, SQL compliant database. Its CSV↗ export mechanism can be exploited to generate reports that can be further analyzed with any kind of spreadsheet software. In comparison with Apache Derby, H2 Database is less strict when modifying the database schema – a fact that is very useful when altering the database schema over time. The H2 Database containing the performance evaluation model is labeled *IdaresDB* in Figure 5-i. This database is

[5] `http://www.h2database.com/`

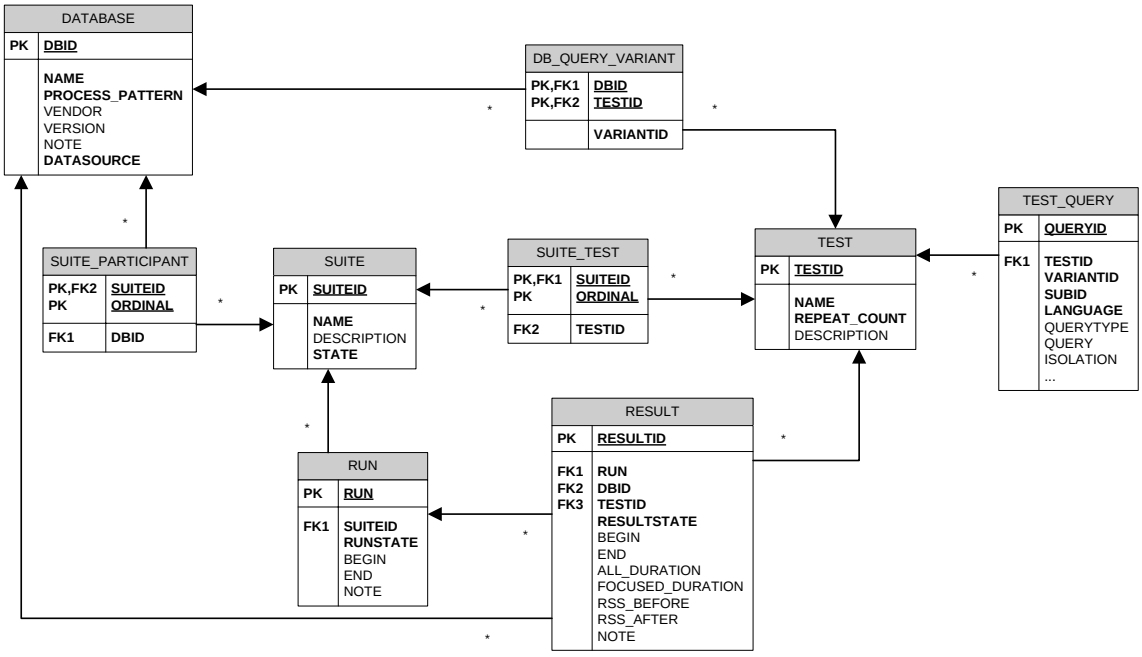


Figure 5-ii: Database relations for the performance evaluation model

integrated into the Apache Tomcat container by defining a *DataSource* with the name *IdaresDB* (cf. discussion of *DataSource* in Section 4.3 on page 76).

To represent all necessary entities, we created a database schema that is shown in Figure 5-ii. A `TEST` has a `NAME` and a list of `TEST_QUERY`s. During the performance evaluation all queries of a test are to be executed in the order of their `SUBID`. The `TEST_QUERY` with $\texttt{SUBID} = 0$ is called the *focused query*. The duration is to be measured for all `TEST_QUERY`s of a `TEST` and for the `focused query` separately. This allows the modeling of preparation queries ($\texttt{SUB\_ID} < 0$) and cleanup queries ($\texttt{SUB\_ID} > 0$) before and after the *focused query*. Furthermore, a `TEST` has an attribute `REPEAT_COUNT` in order to repeat all `TEST_QUERY`s a number of times.

The `LANGUAGE` of a `TEST_QUERY` can be:

SQL subsumes `SELECT`, `UPDATE`, or `DELETE` query or a Data Definition Language statement (DDL).

XQUERY represents an XQuery statement executed by a JDBC driver.

XMLDB represents an XQuery statement executed by an XML:DB driver.

COPY_CONTENT copies all rows or only a subselection of them from a source table to a table in the evaluated database.

IMPORT_CSV parses and imports data from a CSV file into a table in the evaluated database. We utilize a Java CSV parser to achieve this, instead of utilizing individual operations of the databases. This is necessary, since not every database supports the idiosyncrasies of CSV files we use.

APP_SIM_WS stands for Application Simulation Web Services. It refers to Web Service implementing the WSDL↗ operation `simulateApplications`, which simulates concurrent access to the database. The WSDL definition can be found in Appendix A.4.2 on page 188.

Besides the entity `TEST`, Figure 5-ii on the preceding page shows the entity `DATABASE`. It contains information about a database including the `NAME` and the `DATASOURCE` reference. Different databases often apply different SQL dialects. Hence, the performance evaluation model allows the definition of multiple variants of `TEST_QUERY`s for a `TEST`. The entity `DB_QUERY_VARIANT` defines which variant of a `TEST_QUERY` is used by a `DATABASE`. If none is specified, the `TEST_QUERY` with the default variant `0` is used. The purpose of the entity `SUITE` is the definition of a collection of `TEST`s that are applied to a collection of `DATABASE`s. The attribute `ORDINAL` of `SUITE_TEST` and `SUITE_PARTICIPANT` define the order of the `TEST`s or respectively of the `DATABASE`s within a `SUITE`.

A performance evaluation run is recorded in the entity `RUN`. Moreover, for each pair of `TEST` and `DATABASE`, a `RUN` records the outcome of the performance evaluation in the entity `RESULT`. If the `TEST` was executed without errors, the duration of the `focused query` and the overall duration is saved. Moreover, the Resident Set Size (RSS) of the related database process[6] before and after the execution of the `TEST` is stored. If otherwise an error occurred, the `RESULT` contains the error message in its `NOTE`.

The detailed data definition for this performance evaluation model can be found in `[DVD]/Sources/SQL/Evaluation/evaluation.ddl.sql`. The same folder contains the SQL script `evaluation-analysis.sql`, which contains useful queries to analyze the outcome of an evaluation `RUN`.

The execution logic of the performance evaluation is modeled as a BPEL process (see *IdaresDB-Evaluation* in Figure 5-i on page 91). Since the environment of the thesis utilizes the workflow technology, it seems natural to model the evaluation of the databases the same way. As the implementation of a production like BPEL workflow was an good method to get to know the workflow environment, it seems justified that the implementation of a workflow caused slightly more effort than the development of a Java based program would have caused. Due to the fact that the performance evaluation workflow can be started by just calling a Web Service operation, it is also possible to integrate the performance evaluation workflow in a superordinate workflow. Likewise, the BPEL process can serve as a basis for all kinds of evaluations in future. With slight modifications of the relational database evaluation model and the BPEL workflow itself, performance measurements of arbitrary Web Services are imaginable. The source of the BPEL workflow can be found in `[DVD]/Sources/Projects/Idares/IdaresDBEvaluation.bpel`[7].

Before describing the logic of the evaluation workflow, we briefly consider the remaining components shown in Figure 5-i on page 91. Deployed in the Apache Axis2 Web Service engine there is the *GenericDatabaseService*. This is a utility Web Service that facilitates various database related operations. It is part of the *IDARES Web Services* presented in the prototype architecture in

[6] Resident Set Size is the portion of the memory of a process that is kept in the RAM.
[7] The source file was not attached in the appendix of this thesis because it has got a length of about 2,500 lines.

Section 4.3 on page 76. The *GenericDatabaseService* not only performs the database access to the underlying JDBC↗ driver, it also takes the time for database requests. This is an important design issue, because the implementation of the *GenericDatabaseService* can determine when a database request starts and when it ends. If the evaluation process, which calls the *GenericDatabaseService*, would stop the time, the measured duration would not only contain the database request but also the duration for Web Service communication. In addition to the time measurement, the *GenericDatabaseService* determines the RSS↗ of the database process before and after database activities. To realize this, a Java based wrapper for the Linux command *ps*[8] has been implemented. This wrapper adds up the RSS values for all processes that match a given regular expression. This is necessary because some database systems start multiple processes with different names.

The last component in Figure 5-i on page 91 that requires explanation is the *IdaresDBEvaluationClient*. This is a Java based program to start an evaluation run. It contains Web Service clients for the *GenericDatabaseService* and the *IdaresDBEvaluation* workflow. The Web Service stubs have been generated with help of the Apache Axis2 generator tool. The client uses dialogs to interact with the user, while its output is printed in the shell.

Before describing the evaluation process, we state the steps that are needed to prepare an evaluation run:

- The database to evaluate has to be started.
- This database has to be configured as a *DataSource* for Apache Tomcat.
- The *IdaresDB* has to be started.
- The evaluation `SUITE` has to be created. The tests have to respect the dialect of the database to evaluate.
- Apache Tomcat has to be started. This step usually takes a while, since Apache ODE and the Apache Axis2 web applications have to be started. Moreover, the *IdaresDBEvaluation* workflow has to be compiled by Apache ODE.

In order to ensure the comparability of the evaluation results, the same circumstances have to be guaranteed for each database. Therefore, we just evaluate a single database within an evaluation `SUITE`. After the execution of a `SUITE` has been finished, the evaluation infrastructure is shutdown and can be prepared for the next database.

The whole process of a performance evaluation run is shown in Figure 5-iii on page 96. The process is modeled using the Business Process Modeling Notation v. 1.1 (short BPMN, see [OMG09]) standard. The evaluation process is started by invoking the *IdaresDBEvaluationClient*. The client calls the *GenericDatabaseService* and retrieves information of all evaluation `SUITE`s currently stored in the *IdaresDB*. The client provides a list for the user, where he or she can select a *SUITE* that shall be executed. The client calls the Web Service operation of the *IdaresDBEvaluation* workflow, which instructs the ODE BPEL runtime to create a new instance of this workflow.

[8] `http://linuxmanpages.com/man1/ps.1.php`

In a preliminary step, the *IdaresDBEvaluation* workflow retrieves information of the SUITE and checks that the SUITE exists in the database and that it currently has the state *ready*. To address the SUITE, its *suiteID* has been received in the request message of the workflow. If the SUITE is applicable, a new entry for the entity RUN is created in the database. This RUN is further referenced by its *runID*. With the initialized variables *suiteID* and *runID* the child scope *execute run* of the workflow is executed (see Figure 5-iii-b on the next page).

The scope *execute run* starts with two parallel activities that retrieve information about all DATABASEs and all TESTs of the related SUITE. Both results are ordered by their SUBID and stored into variables. Subsequently two nested forEach loops iterate over each pair of DATABASE and TEST. With the loops, the process continues with retrieving the DATABASE specific variants of the TEST_QUERIES for the current TEST. Afterwards the request message for the evaluation TEST is prepared. As there are different LANGUAGEs of TEST_QUERIES, there are different WSDL↗ ports that handle the requests. The *IdaresDBEvaluation* workflow selects the appropriate WSDL↗ port and hence the required structure of the request message. A further nested loop is executed as often as the REPEAT_COUNT of the TEST has specified. For each iteration the evaluation request message is sent to the *GenericDatabaseService* or another WSDL service synchronously. This service encapsulates the evaluation test, executes all queries to the underlying database, and takes the time. The *IdaresDBEvaluation* workflow receives the duration of the test and creates a new RESULT entry. In the case of an error, a note is attached to the RESULT entry and its RESULTSTATE is set to *failed*.

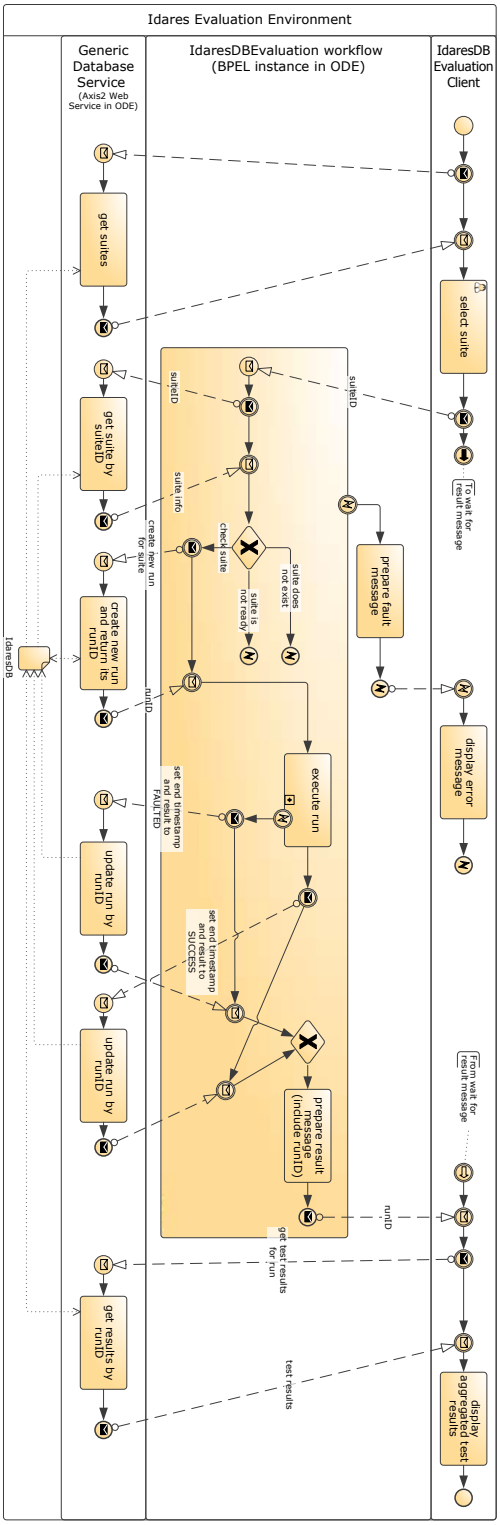
When the scope *execute run* is finished, the logic continues in the main scope. The end time of the current RUN is set and the *runID* is returned back to the *IdaresDBEvaluationClient*. The client, now knowing that the run has been finished, requests a short statistic from the *IdaresDB*. Thereby the user knows how many tests where successful and how many failed.

The whole evaluation infrastructure is installed on a virtual machine. Besides the software for the workflow environment, all databases to be evaluated are installed on the virtual machine, too. Thus, the evaluation infrastructure can be easily ported to another computer. Moreover, the virtual machine defines all properties of the environment and hence the evaluation runs are reproducible. The virtual machine is distributed with this thesis and described in detail in Section 6.7 on page 149.

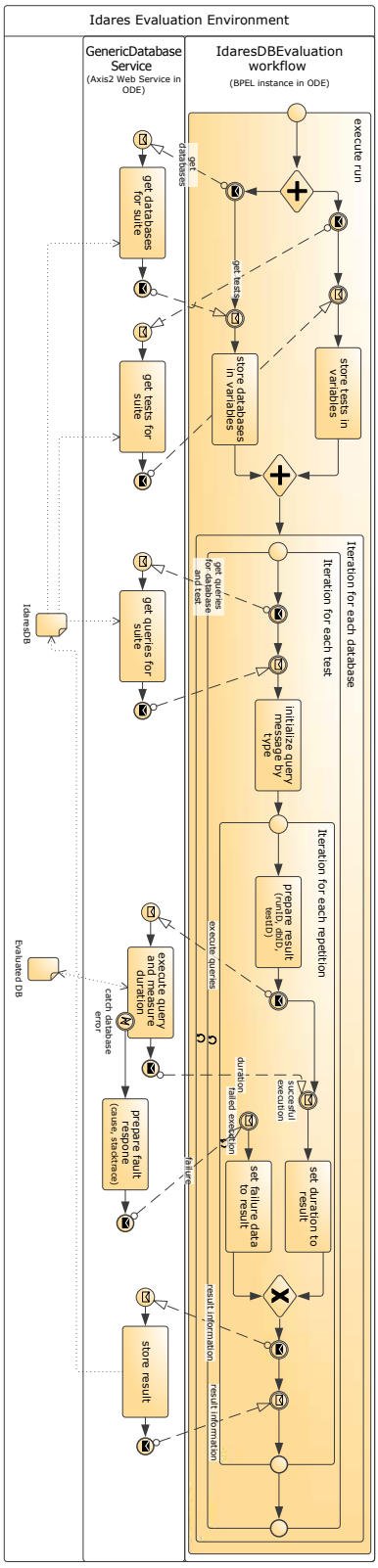
The department “Applications of Parallel and Distributed Systems” provided a personal computer for the execution of the performance evaluation. Since the evaluation results correlate with its performance, we mention the characteristics of the system at this point:

Processor	Intel Core 2 Quad Q9300 @ 2.5 GHz, four cores
Memory	3.25 GB Ram
Hard Drive	232.7 GB and RAID 1 mirroring
Operating System	Microsoft Windows XP Professional, Service Pack 3, 32 bit

The evaluation infrastructure is installed on Debian Linux as operating system. Linux was chosen, because it is free and open source. Therefore, no complications through license conditions occur.



(a) the main process



(b) the sub process *execute run*

Figure 5-iii: BPMN schema of the *IdaresDBEvaluation* workflow

Moreover, the package manager of Linux was beneficial for the installation of some databases that require additional libraries. Although the databases have to support Windows systems too, a second evaluation on a Windows virtual machine was not performed. It would have caused to much effort and the performance criteria are just one part of the evaluation.

5.3 Database presentation

Considering the requirements presented in Section 3.1 on page 48, different database techniques are needed. Figure 3-i on page 50 already showed an overview categorized by database type and language. To find the most appropriate database system for each database technique, possible candidates are presented in this section and a short list is created.

The following paragraphs contain all candidates taken into consideration. If a candidate is short-listed, the database presentation states the *evaluated version* of the candidate database. If a candidate is not short-listed, a brief argumentation discusses the reasons for the exclusion. The list of candidates is primarily based on a brief database market survey with focus on the required database techniques. Some inspiration for possible XML databases are taken from [HSW08]. They already investigated XML databases for the storage of XML artifacts in a SOA↗. As only a manageable number of databases can be evaluated in this thesis, the conditions for creating the short list are the following:

- the databases each have to support Criterion 2 on page 82, requiring support for Windows and Linux systems
- the databases have to be cost-free or provide a trial license with minor limitations; open source databases with a permissive copy left statement are preferred
- all five required database techniques have to be covered
- databases that support multiple database techniques properly (e. g., an external storage based database for relational and XML data) are favored
- diversity on the short list is approved

First all short-listed databases are presented, than the discarded candidates. Within this order, the databases are sorted by their name.

Apache Derby

Version: 10.5.3.0

Project URL: `http://db.apache.org/derby/`

Download URL: `http://apache.prosite.de/db/derby/db-derby-10.5.3.0/db-derby-10.5.3.0-bin.tar.gz`

Notes: Apache Derby is an open source Java based SQL database. Hence, Derby supports all platforms that are supported by the JVM↗. Apache Derby has a small installation size of about 2 MB and yet supports a rich subset of the ANSI↗ SQL standard. It supports transactions, concurrent access, and even triggers. Apache Derby has two deployment modes: client server mode and an embedded mode. The deployment of Apache Derby is very simple: only a reference of the appropriate `JAR` file in the Java classpath has to be added and a JDBC↗ URL has to be defined. Hence, Apache Derby is often utilized as testing database or shipped with other software framework, e. g., Apache ODE. Apache Derby is even part of the Sun Java Development Kit 6.0 under the name *Java DB*.
We use Apache Derby as a reference database, because it is already part of Apache ODE.

eXist-db

Version: 1.4.0 rev10440

Project URL: `http://exist.sourceforge.net/`

Download URL: `http://sourceforge.net/projects/exist/files/Stable/1.4/eXist-setup-1.4.0-rev10440.jar/download`

Notes: eXist-db is an open source database system for XML storage and management that is entirely written in Java. eXist-db is a native XML database (see classification in Section 2.4.2 on page 37) and therefore offers data storage capabilities that are optimized for XML data. Even XML specific indexes are supported by the database. All in all, the eXist-db project emphasizes the very efficient XML processing of the database.
eXist-db supports the standards XQuery 1.0 ([BCF+07]) and XPath 2.0 ([BBC+07]). An application can send its requests to eXist-db using a HTTP based interface (e. g., through REST, SOAP, or XMLRPC [Win03]) or the XML database specific interface XML:DB ([Sta01]) facilitating the XML:DB XUpdate language ([LM00]). The version 1.4 of eXist-db even contains text analysis extension by the means of the text search engine library Apache Lucene[9].
Since there are a vast number of free XML databases on the market, we selected eXist-db because a previous evaluation by [HSW08] declared eXist-db as their winner. Moreover, eXist-db is written in Java and is therefore easy to integrate into our already Java based workflow engine.

H2 Database Engine

Version: 1.2.121

Project URL: `http://www.h2database.com/`

[9] `http://lucene.apache.org/java/docs`

Download URL: `http://www.h2database.com/h2-2009-10-11.zip`

Notes: H2 is an Open Source embedded relational database written entirely in Java. H2 is extreme feature rich for its size (1.2 MB): it supports various connection modes (embedded, client server, mixed mode), full ACID conformity, encrypted databases, and a cost based optimizer. Besides the various table types, H2 offers in-memory tables too.
We choose the H2 Database for both embedded and client server mode in order to have direct comparison to the reference database Apache Derby. Moreover, the installation of the H2 Database into the simulation workflow environment seems very simple – an issue that is required by ℝ **20** on page 49.

IBM DB2

Version: DB2 Workgroup Server Edition 9.7 for Linux with pureXML

Project URL: `http://www-01.ibm.com/software/data/db2/linux-unix-windows/edition-workgroup.html`

Notes: IBM DB2 is one of the major commercial relational database systems. It addresses large scale and highly concurrent application fields. IBM DB2 advertises its self-tuning memory management and other autonomic features to minimize database administration tasks. Moreover, IBM DB2 provides a number of administration tools, even with graphical user interface.
IBM DB2 supports ANSI↗ SQL and, in the current version, a vast set of extensions. One of them is called *pureXML* and refers to the capability of managing XML data. Therefore, DB2 supports an XML data type that can be used in column specifications. Requests to the database may have the traditional form of SQL `SELECT` queries or may define XQuery statements. DB2 also allows queries that mix relational and XML requests. An integration in Java is done via a JDBC↗ driver that accepts both kinds of queries.
IBM DB2 is purchased in various editions for Windows and Linux. As the staff of the related departments of this thesis has access to commercial editions of IBM DB2, the Workgroup Server Edition was chosen for the evaluation. Otherwise the free alternative IBM DB2 Express-C would have been evaluated.

MonetDB SQL

Vendor: Centrum Wiskunde & Informatica

Version: monetdb5-sql 2.34.2-20091210

Project URL: `http://monetdb.cwi.nl/SQL/index.html`

Download URL: `http://monetdb.cwi.nl/downloads/Debian/`

Notes: MonetDB is an open source database engine that implements a main memory data storage. MonetDB supports two server versions: one for relational storage accessed through ANSI SQL and one for XML based storage accessed through XQuery. As MonetDB was designed as a main memory database from the beginning of the project, it exploits all its advantages and claims to realize a significant speed improvement when compared with traditional external storage based databases.
The MonetDB server implements an abstract column based data storage layer and can realize different logical data models thereupon. Besides SQL and XQuery, MonetDB provides the MonetDB Assembly Language (MAL) that is more intended for testing and developing purposes than for productive retrievals.
We choose MonetDB SQL for the short list, since its application as a local cache seems encouraging and its results in the TPH-H benchmark[10] are amazing.

MonetDB XQuery

Vendor: MonetDB XQuery

Vendor: Centrum Wiskunde & Informatica

Version: monetdb-xquery 0.34.2-1

Project URL: `http://monetdb.cwi.nl/XQuery/index.html`

Download URL: `http://monetdb.cwi.nl/downloads/Debian/`

Notes: The MonetDB XQuery database system is the same as the MonetDB SQL (see Section 5.3 on the preceding page). The database server has just to be started with another configuration.
As the MonetDB XQuery is the only free in-memory XML database found during the market survey, it has to be added to the short list.

MySQL Cluster

Vendor: Oracle (Sun, MySQL)

Version: 5.1.37 NDB, 7.0.8 cluster

Project URL: `http://www.mysql.com/products/database/cluster/`

Download URL: `http://dev.mysql.com/downloads/mirror.php?id=375080`

[10] `http://monetdb.cwi.nl/SQL/Benchmark/TPCH/`

Notes: The MySQL database is a popular relational database system that supports a broad subset of ANSI↗ SQL. MySQL is published under a dual license that allows its adoption both in open source and commercial projects.
In this thesis we are evaluating the MySQL Cluster – a cluster of MySQL data nodes that provide in-memory data storage. The MySQL Cluster advertises high availability and high performance capabilities. This is achieved by splitting the overall data set into disjoint subsets and by storing each of these subsets on multiple *Network DataBase* (NDB) nodes. Common MySQL Server nodes are responsible to receive requests, connect to the NDB nodes and perform the request. An additional management node is used to automatically administrate all nodes in the cluster.
We apply the MySQL Cluster because its performance and in-memory support seem promising.

Apache Xindice

Project URL: `http://xml.apache.org/xindice/`

Notes: Apache Xindice is yet another open source native XML database. Apache Xindice supports the XML:DB XUpdate language for updates and can be accessed by the XML:DB API or the XML-RPC API.
Apache Xindice has been compared with other XML databases in [HSW08]. Xindices has shown to have poorer performance and less features than eXist-db that is short-listed. Hence, we drop Apache Xindice.

ENEA Polyhedra in-memory DBMS

Project URL: `http://www.enea.com/Templates/Product____27039.aspx`

Notes: Polyhedra IMDB is a commercial in-memory relational database system. Polyhedra IMDB guarantees ACID compliance, supports SQL request, and even provides a JDBC↗ driver. Polyhedra is intended to be used in an embedded mode, but it is applicable in client server scenarios too.
Polyhedra IMDB may be an excellent candidate for the local cache, since it promises high-speed query processing and full ACID support. Unfortunately the software only provides a commercial license and its evaluation version is time-limited. Therefore Polyhedra is not short-listed.

HSQLDB

Project URL: `http://hsqldb.org/`

Notes: HSQLDB (formerly known as Hypersonic SQL database) is an embeddable Java based relational database. HSQLDB supports subsets of the ANSI SQL standard, client server and embedded mode, and in-memory and external storage based tables.
On the downside, HSQLDB lacks on support for ACID transaction – only the current development version supports all four isolation levels of the SQL standard. Moreover, HSQLDB is intended for small data sizes, as its default mode uses a simple SQL query logging mechanism in order to persists the current data. On a restart of the database all previously performed SQL statements have to be re-execute to build-up the model. As we have already considered H2 and Apache Derby as embeddable Java based databases, we exclude HSQLDB from the short list.

IBM solidDB

Project URL: `http://www-01.ibm.com/software/data/soliddb/`

Notes: IBM solidDB is a commercial relational in-memory database. IBM advertises the broad usage of solidDB for high-speed data processing and mission critical applications. IBM solidDB provides SQL access and a JDBC↗ driver. The database guarantees persistence and maintains two instances for the sake of data replication and therefore availability. Moreover, solidDB features distributed transactional storage and can partition in two phase commit scenarios. IBM solidDB also has an edition that provides a *Universal Cache* for underlying disk based relational databases. Therefore, solidDB provides a synchronization mechanism. These features seem perfect for the application field as a local cache. Unfortunately, the database only has a commercial license and does not provide any kind of evaluation version. Therefore, we have to discard IBM solidDB.

McObject ExtremeDB

Project URL: `http://www.mcobject.com/extremedbfamily.shtml`

Notes: McObject ExtremeDB is yet another commercial database that provides in-memory storage capabilities. The database system provides an SQL based interface to the in-memory storage and even an XML extension. ExtremeDB supports Windows and Linux platforms, but is oriented on embedded platforms and real time systems. Furthermore, McObject just offers a commercial license besides an evaluation release. Hence, ExtremeDB is not short-listed.

Microsoft SQL Server

Project URL: `http://www.microsoft.com/sqlserver/`

Notes: SQL Server is a commercial database from Microsoft not only oriented at the business market. The scientific workflow workbench Microsoft *Trident* also uses SQL Server as a possible data storage component. SQL Server is a database for traditional relational data, but also all other kinds of structured, semi-structured, and unstructured documents, including a native storage for XML data. SQL Server provides T-SQL as request language – a modified dialect of traditional SQL. The database is fully ACID compliant and supports optimistic concurrency control protocols too. The enterprise edition of SQL Server also provides sophisticated services for data warehousing and business intelligence.
All in all, SQL Server would be an interesting competitor for IBM DB2. There even is a scaled down, free edition of SQL Server: SQL Server Express Edition. Unfortunately, SQL Server only supports Microsoft Windows as operating system and has therefore to be discarded.

MySQL Server Community Edition

Vendor: Oracle (Sun, MySQL)

Project URL: `http://dev.mysql.com/downloads/mysql/`

Notes: The MySQL Server is yet another popular and free relational database system. It supports SQL and many other features known from relational database systems: stored procedures, triggers, views, and full ACID support. In the version 5.1 enhanced XML capabilities where integrated. However, [HSW08] have shown that the XML support is less competitive. Hence, we do not evaluate the MySQL Server for the same reason as for PostgreSQL: MySQL would only be evaluated for its disk based relational storage.

Oracle Berkeley DB Java Edition

Project URL: `http://www.oracle.com/database/berkeley-db/je/index.html`

Notes: Oracle Berkeley DB Java Edition is the Java implementation of the Oracle Berkeley DB. Berkeley DB JE is an open source embedded database written in Java. Berkeley DB JE provides a key value pair based view on the stored data. It neither provides an SQL nor an XQuery based API. Nevertheless, Berkely DB JE has great potential as a database tightly coupled to an application that requires high throughput and concurrency.
Berkeley DB JE does not fit plainly in any of our required database techniques and therefore is not short-listed. However, Berkeley DB JE fulfills conditions that would qualify it to serve as the *ODE model db*. This approach would require the re-implementation of the entire ODE DAO↗ layer based on storing key value pairs. As this approach is out of the scope of this thesis, we can just regard this idea as a suggestion for future projects.

Oracle Berkeley DB XML

Project URL: `http://www.oracle.com/database/berkeley-db/xml/index.html`

Notes: Oracle Berkeley DB XML is an open source XML database that facilitates XQuery requests to the XML data store. The database is written in C and based on Oracle's embeddable Berkeley DB and other third party software. Berkeley DB XML claims to provide a fast and efficient data storage and retrieval besides known database features like transactions and replication.
We do not put Berkeley DB XML onto the short list, although [HSW08] have attested a nearly as good performance as eXist-db. In order to minimize the evaluation effort, this database was used as a backup but not applied during the evaluation.

Oracle TimesTen In-Memory Database

Project URL: `http://www.oracle.com/technology/products/timesten`

Notes: Oracle TimesTen is a relational database that is based on a memory-optimized storage. Oracle emphasizes its instant responsiveness and a high throughput. TimesTen requires that the whole data model fits into the main memory. Using optimized data structures and access algorithms, TimesTen exploits the advantages of an in-memory data storage. TimesTen guarantees durability due to transaction logging and database checkpoints on disk. The database supports standard SQL queries to the relational storage.
Nevertheless, the default free license only grants the right to use the database for developing, testing, prototyping, and demonstrating purposes. Therefore, we exclude the database from the short list.

PostgreSQL

Project URL: `http://www.postgresql.org/`

Notes: PostgreSQL is a popular, open source, relational database system. The database system provides SQL access to its data store and many powerful features, such as Multi-Version Concurrency Control (MVCC). Since version 8.3, PostgreSQL even supports an XML column type and XPath based queries. [HSW08] have put PostgreSQL at the 6^{th} place out of 7 regarding its XML functionality. Hence, we discard the XML storage of PostgreSQL for this evaluation. Although the database system seems promising and has a permissive license, we would only exploit its disk based relational storage. Since we want to minimize the number of databases for the integrated database architecture, we do not put PostgreSQL on the short list.

Quilogic SQL/XML-IMDB

Project URL: `http://www.quilogic.cc/`

Notes: Quilogic SQL/XML-IMDB is a commercial in-memory database that supports both relational storage with an SQL interface and XML storage with an XQuery interface. QuiLogic Technologies emphasizes the performant query processing and the multi-threading transactional support. Hence, this database seems perfect for the application as a local cache for both relational and XML data.
Unfortunately, Quilogic SQL/XML-IMDB only provides a commercial license and a time limited evaluation version. Moreover, Quilogic SQL/XML-IMDB heavily dependents on the Windows platform and does not support Linux systems. All in all, we have to drop Quilogic SQL/XML-IMDB, too.

webMethods Tamino

Vendor: Software AG

Project URL: `http://www.softwareag.com/corporate/products/wm/tamino/`

Notes: webMethods Tamino XML Server is a data server for all types of structured and unstructured data. Its server application core contains functions to process, generate, and exchange XML documents. Tamino XML server is based upon a native XML data store, which is accessible via HTTP based DOM (Document Object Model), JDOM, and SAX (Simple API for XML) APIs and SOAP↗ services. The server not only supports XQuery as request language, also various text retrieval services are provided. The server supports different types of XML indexes for elements and attributes and even offers transactions.
All in all, Tamino XML Server is more than a database – and hence more than we look for. Moreover, the product is commercial and only supports Microsoft operating systems and enterprise versions of Linux distributions.

XML Transaction Coordinator (XTC)

Vendor: TU Kaiserslautern

Project URL: `http://wwwlgis.informatik.uni-kl.de/cms/dbis/projects/xtc/`

Notes: The XML Transaction Coordinator (short XTC) is another native XML database that facilitates a storage model directly addressing XML data requirements. XTC provides an XQuery API and supports multi-user access through concurrency control mechanisms. The database supports various XML specific indexing techniques, is transaction oriented on typical XML processing activities and offers self-tuning techniques.
Although the features of XTC seem promising, the current version 0.12.4 is only a beta release. Apart from various scientific papers, the website does provide a *Quick Start Guide* as the only documentation or manual. For these reasons XTC is not short-listed.

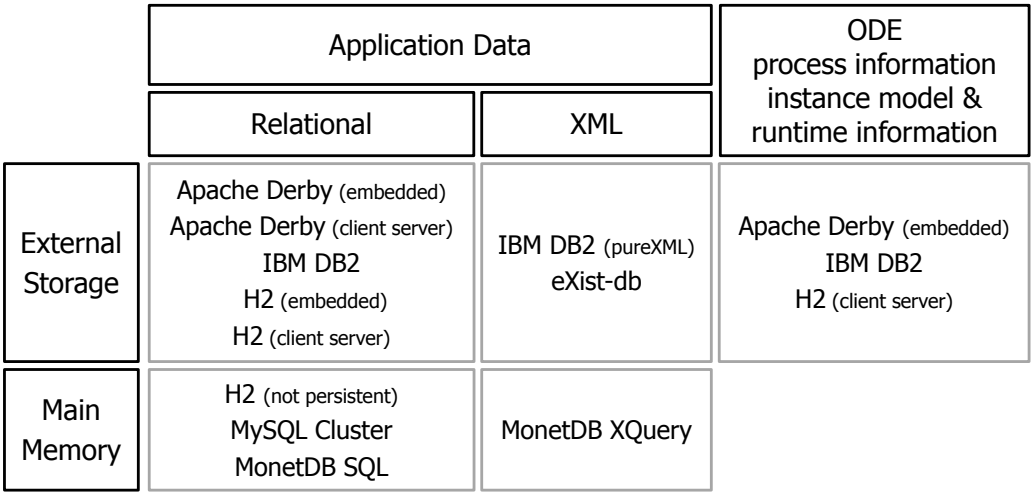


Figure 5-iv: Short list of the evaluated databases and classification of their data storage (external storage, main memory) and their application (relational data, XML data, *ODE model db*)

After all database systems have been briefly introduced, Figure 5-iv now shows the candidates being short-listed. The IBM DB2 database system is applied to both relational and XML based external storage. Section 2.4.3 on page 38 has shown that an embedded mode might have advantages when looking at performance, but might result in competition for resources in the same JVM↗. Therefore, the embeddable databases Apache Derby and H2 Database are applied in their embedded mode and in their client server mode.

5.4 Database Evaluation and Results

In this section, we apply all eleven criteria of Section 5.1 on page 81 to the five database candidates presented in Section 5.3 on page 97. Some databases are evaluated for multiple application fields, as already shown in Figure 5-iv. Some criteria evaluate general database properties and are applied just once for each database. Other criteria are questioning features of a database regarding a specific application field (e. g., XML storage) and may hence be applied multiple times for a database. The exact versions of the database candidates are already stated in the database presentation (see Section 5.3).

Most criteria request the investigation of specific database features or properties, but do not provide a scale for the evaluation. For these criteria, we introduce an ordinal scale[11] representing the degree of fulfillment. We prefer an ordinal scale, because most of the evaluation is done on

[11]The ordinal scale is a scale type that defines a level of measurement based on a total order of the scale values. This scale can be regarded as a ranking that is not based on measurement, hence the values can not be used for further calculations, such as a mean.

subjective investigations and an interval scale would allow unreasonable calculations upon the values of the scale. We introduce the following four degrees of fulfillment:

- *Totally fulfilled*: This is the best rank and means a total fulfillment of the criteria with only some minor flaws.
- *Mostly fulfilled*: This degree is given to candidates that mostly fulfill a criterion, but there is one major flaw that causes the devaluation. The database will serve well for the related application field, although there is potential for improvement.
- *Partly fulfilled*: This rank defines that more than one major requirement is only partially fulfilled by a candidate. This is the weakest rank where a database might be applied for an application field, even with difficulties.
- *Poorly fulfilled*: This is the weakest rank and stands for a limited or no fulfillment of a criterion. Each major requirement is not or only partially fulfilled. A database with this rank can not be applied to the related application field.

In the following paragraphs, we apply the actual evaluation criterion by criterion for every database. Thereby, we allow a better comparison between the databases for a given criterion. Moreover, we can adjust the degrees of fulfillment for a criterion more easily, as we can consider the accomplishments of each database.

5.4.1 Criterion 1 – License and Costs

Apache Derby is available under Apache License, version 2.0[12]. This is an open source license without any copyleft. Only a copyright and a disclaimer are required when redistributing Apache Derby with own software. Therefore, we assign the rank *totally fulfilled*.

IBM DB2 is a commercial product and has to be purchased. The IBM online shop charges € 493.85 for an "Authorized User License" and € 14,042.00 for a "Processor Value Unit License" (effective January 14, 2010) for the IBM DB2 Workgroup Server Edition, which is evaluated. Both licenses include twelve months IBM support. Nevertheless, IBM also provides an *Academic Initiative Program*[13] and, as mentioned before, the department related to this thesis already has a license for this database. IBM DB2 fulfills this criterion *partly*, since its license is complex to acquire and only cost-free within the department.

[12] `http://www.apache.org/licenses/LICENSE-2.0`
[13] `http://www.ibm.com/developerworks/university/data`

MySQL Cluster is offered under the GNU Public License (GPL)[14]. This license contains a copyleft note, so that software using MySQL products have to be published under GPL too. Moreover, MySQL products are available by a commercial license, too. However, no prices are stated for the MySQL Cluster. As the database system is available by GPL, which requires a copyleft, we assign a *mostly* fulfilled.

MonetDB (SQL and XQuery) is published under the Monet Public License[15], which is a derivate of the open source Mozilla Public License (MPL). The MPL is an open source license with only a weak copyleft. Therefore, we assign rank *totally fulfilled*.

eXist-db is published under the open source license GNU Lesser General Public License (LGPL)[16]. This license only has a weak copyleft note and therefore eXist-db receives a *totally fulfilled*.

H2 Database is published under the open source licenses MPL and Eclipse Public License (EPL)[17]. As H2 Database does not require a copyleft, it *totally* fulfills this criterion.

5.4.2 Criterion 2 – Platform

Apache Derby is based on Java technology and is hence well suited for our simulation workflow environment. The JVM↗ is provided for Windows, Linux, and many other operating systems and both 32 bit and 64 bit systems. Moreover, the embedded Apache Derby variant is tightly build around its JDBC↗ driver and therefore natively processes SQL queries without network traffic. Thus, both variants of Apache Derby *totally fulfill* the criterion.

IBM DB2 comes in many editions and variants, including Windows and Linux, both 32 bit and 64 bit. Nevertheless, a specific edition has to be purchased, so a later changing of the platform is not easily possible. During the installation on a Linux system, the package `libaio1`[18], which enables userspace to use Linux kernel asynchronous I/O system calls, was required and installed. IBM DB2 provides different JDBC↗ drivers. The *Type 2 JDBC driver* calls methods of a local native implementation of a DB2 client and is hence thought to be more performant. The *Type 4 JDBC driver* is entirely implemented in Java and serves for local and remote database connections. All in all, we assign DB2 the rank *mostly fulfilled*, as the changing of the platform requires another edition and might invalidate the purchased license.

[14] `http://www.gnu.org/licenses/gpl-2.0.html`
[15] `http://monetdb.cwi.nl/Development/Legal/MonetDBLicense-1.0/index.html`
[16] `http://www.gnu.org/copyleft/lesser.html`
[17] `http://opensource.org/licenses/eclipse-1.0.php`
[18] `http://packages.debian.org/sid/libaio1`

MySQL Cluster has different downloads for Windows and Linux, both 32 bit and 64 bit. Similar to the IBM DB2 system, MySQL Cluster has different downloads for different platforms. Nevertheless, a license is not dependent on the platform and therefore the platform can be switched. MySQL provides *Connector/J* – a generic JDBC driver for all versions of MySQL database systems. All in all, we assign the rank *totally fulfilled* to MySQL Cluster.

MonetDB (SQL and XQuery) is implemented in C++ and the download page provides installers for Windows and Linux, both 32 bit and 64 bit. A generic JDBC driver for the SQL and XQuery variant are available too. Considering its integration, MonetDB *totally fulfills* this criterion.

eXist-db is implemented in Java and therefore applicable to all required platforms. The eXist-db contains an XML:DB driver ([Sta01]) for local and remote database access. As this driver is solely intended for XML retrieval it is not compatible with the other JDBC drivers. Hence, eXist-db can not be simply integrated into Apache Tomcat using the *DataSource* mechanism. Nevertheless, its integration into the Java based workflow environment is very easy to accomplish. To sum the discussion up, we rate eXist-db with *mostly fulfilled*.

H2 Database is written in Java and is therefore fully capable to be integrated into the workflow environment. Similar as Apache Derby, H2 Database provides a single `JAR` file with the JDBC driver and the database implementation – hence it *totally fulfills* this criterion.

5.4.3 Criterion 3 – Installation and Integration

Apache Derby As Apache Derby just consists of a bundle of `JAR` files, the installation is quite simple. For the embedded mode, the `derby.jar` (2.4 MB) has to be copied into the `lib` folder of Apache Tomcat. As Apache ODE already uses an Apache Derby database, it is crucial to delete its `derby-10.4.1.3.jar` to avoid version conflicts. The definition of a *DataSource* has to define a JDBC URL[19] with an absolute path, like `jdbc:derby:/media/idares-evaluation/db-data-storage/DerbyEvaluation/embedded-eval;create=true`.

For the client server mode, similar steps have to be performed. Instead of `derby.jar`, the `derbyclient.jar` (0.5 MB) with the client JDBC driver has to be added to the classpath folder of Tomcat. The JDBC URL refers to a localhost server and a database name: `jdbc:derby://localhost:1527/client-eval`. To start an Apache Derby server on the same machine a utility Java class has to be called the following way:

[19] A JDBC URL specifies the database driver alias and further connection properties, like the database schema name. Every database vendor may have its own JDBC URL patterns.

```
java -Xms128M -Xmx500M
     -Dderby.system.home=/media/idares-evaluation/db-data-storage/
        DerbyEvaluation/
     -Dderby.storage.pageCacheSize=98304
     -cp derby.jar:derbynet.jar
     org.apache.derby.drda.NetworkServerControl start -p 1527
```

This Apache Derby server is accepting requests on localhost port 1527. Moreover, the folder for the database data has to be set via a system property `derby.system.home`. The requested database `client-eval` by the client lies relatively to this folder. During the evaluation, a script has been used to start the Apache Derby server: `[DVD]/Sources/Tools/derbyServerStart.sh`.

An issue with integrating Apache Derby is memory management. Apache Derby allows to set the size of its database cache with the system property `derby.storage.pageCacheSize`. The size is measured in 5 kB memory pages, hence 98,304 pages are about 384 MB cache. Furthermore, the maximum memory of the JVM↗ has to be increased using the `Xmx` directives. This topic is very complicated, as Apache Derby does not limit its memory consumption within the limit of the 384 MB cache, but increases its memory usage in special cases. During the performance evaluation, peaks arise when indexes are created and too low JVM memory settings result in an exception: `OutOfMemoryError - JavaHeapSpace`. Besides this exception on the server side, client requests simply freeze forever in such circumstances. The performance evaluation does not continue until the Apache Derby process is killed manually.

Therefore, it is very crucial to find a memory limit for Apache Derby that is sufficient for the highest peak but already low enough to not affect other running processes. This issue gets more complicated when Apache Derby is used in embedded mode within the same JVM as Apache Tomcat. In this scenario, the memory limits not only apply to the evaluated Apache Derby database, but also for Apache Tomcat, Apache ODE, and the other Apache Derby database utilized by Apache ODE.

All in all, the actual installation and integration of Apache Derby is very simple and documented well[20]. Nevertheless, the memory issues are very problematic and result in a *mostly fulfillment* of this criterion.

IBM DB2 The downloaded installation files of DB2 have the size of nearly 1 GB. The installation requires additional Linux packages, which have to be installed before. The installation itself is performed using a step-by-step wizard. The wizard leads experts and non-experts through the installation. All options in the wizard are explained and additional help is provided. The wizard performs the following tasks:

- copy program files to `/opt/ibm/db2/V9.7/`
- create a new database instance

[20] `http://db.apache.org/derby/docs/dev/getstart/getstart-single.html#cgsinsta`

- create a new user profile for the database instance and install database tools for this user
- create a new user profile for database management activities and install database tools for this user
- adjust environment variables and add DB2 to the startup scripts

The installation with the chosen components has a total size of about 0.95 GB. After a system reboot, ten additional processes are started on the virtual machine consuming about 300 MB of main memory. Since these processes compete with others during the evaluation, they have to be shutdown and are not to be started after a system reboot automatically. With `db2iauto -off db2inst1` the autostart of the database processes was disabled. A manual start of the DB2 database server is performed by the owner of the database instance with the simple command `db2start`.

Another step of the installation involves the creation of a new DB2 database container. This task can be performed using the *DB2 Control Center* – a graphical tool for all database management activities. This task finishes the installation of the DB2 database system. The online documentation provides a chapter that describes and helps with the installation[21]

The integration of the IBM DB2 as a *DataSource* into Apache ODE requires two `JAR` files in the `lib` folder: `db2jcc.jar` and `db2jcc_license_cu.jar`. The JDBC connection URL looks like this: `jdbc:db2://localhost:50000/IDARESEV`.

All in all, the complexity of the installation is an issue for DB2 and leads us to a *mostly fulfillment* of the criterion.

MySQL Cluster The installation file for the MySQL Cluster has a size of 207.8 MB. There is no installation – the downloaded file needs just to be extracted. The installed files have a size of about 582 MB. Contrary to this simple installation, the subsequent configuration is very complicated, since multiple configuration files have to be written. The MySQL Cluster documentation[22] provides all necessary background information, but no set of template configuration scripts. Scripts are provided by Severalnines.com[23] for different clustering scenarios and systems. The downloadable package contains:

- configuration files for all involved nodes
- scripts to start and shutdown the nodes
- a script to create an initial empty database

[21] `http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp?nav=/2_0`
[22] `http://dev.mysql.com/doc/refman/5.1/en/mysql-cluster-configuration.html`
[23] `http://www.severalnines.com/sandbox/`

These scripts have to be altered for our integration into the workflow architecture. We can not afford a replicated *Network DataBase* (NDB) node, because the main memory consumption would duplicate, too. So we create an environment with one management process, one NDB process, and one MySQL deamon process – all started on the local machine.

Another part of the MySQL Cluster configuration contains the definition of memory limits. As we want to exploit the main memory storage of data by the database, we have to estimate how much memory the database is to consume during the evaluation. Therefore, a data analysis has to be done: how many rows with which data types have to be kept by the database. The MySQL documentation[24] helps in estimating this data size. The result estimation can be investigated in `[DVD]/Documents/Stuff/TPCH.xlsx`.

Finally, the database has to be integrated into Apache ODE. Therefore, we copy the `mysql-connector-java.jar` into the `lib` folder of Apache Tomcat and define a JDBC URL: `dbc:mysql://localhost:3306`.

When summing up the difficult configuration during the installation and the complicated memory estimation, we can state that the installation does not satisfies the requirement for simple and intuitive installation. Therefore, we conclude that MySQL Cluster only *partly* satisfies this criterion.

MonetDB (SQL and XQuery) The installation of the MonetDB under the Linux operating system is very easy. MonetDB provides Debian packages that can be installed with a common package manager. To achieve this, simply the package source `http://monetdb.cwi.nl/downloads/Debian/` has to be added to the lookup list of the package manager. The package manager automatically installs and configures the database, when the packages `monetdb5-server` (for MonetDB SQL) and `monetdb4-server` (for MonetDB XQuery) are installed. The installation for both servers has a size of about 36 MB. The configuration files are not as complex as for the other database systems and only the data directory path is altered.

Both servers can be started automatically on client requests, when the *Merovingian* process is running in background. For the evaluation, we used the manual startup scripts instead:

```
mserver5 --dbname IdaresEvaluation --set mapi_port=50000
   --dbinit="include sql;"
Mserver --dbname XQueryEvaluation --set mapi_port=50001
   --dbinit="module(pathfinder);"
```

The first line starts the MonetDB server with SQL support, the second starts the MonetDB server with the pathfinder module for XQuery support. Unfortunately the MonetDB Server version 5 does not yet support XQuery request and therefore two different MonetDB servers have to be installed. Data directories for the defined database are automatically created during startup.

[24] `http://dev.mysql.com/doc/refman/5.1/en/faqs-mysql-cluster.html#qandaitem-23-10-1-4`

For both MonetDB variants the same `JAR` file is dropped to the `lib` folder of Apache Tomcat: `monetdb-jdbc.jar`. Additionally, two JDBC URLs are configured as *DataSources*: `jdbc:monetdb://localhost:50000/IdaresEvaluation?language=sql` for MonetDB SQL and `jdbc:monetdb://localhost:50001/XQueryEvaluation?language=xquery` for MonetDB XQuery.

The JDBC driver for the MonetDB XQuery does accept XQuery statements instead of SQL statements and always returns a `ResultSet` with one column and a row for each XML node in the XQuery result. Unfortunately, the JDBC driver does not support reading character streams piecewise from the server. That is why large XML documents are received as huge `String` objects, which can result in huge memory leaks. The XRPC (short for XQuery Remote Procedure Calls; see [ZB07] for the proposal) and its provided Java API might serve as a better driver for the MonetDB XQuery, but the documentation[25] states that this API uses old modules on the server side and therefore may decrease performance.

All in all, we can conclude a very uncomplicated installation besides a JDBC support that is not completely implemented. Hence, we assign the level *mostly fulfilled* for this criterion.

eXist-db The installer of eXist-db is a Java based installation wizard, which is performing the full installation. The installer has a size of about 41 MB, the installed files have a size of about 112 MB. Per default, eXist-db is started as a separate server utilizing *jetty WebServer*[26]. A further note is important at this point: eXist-db also supports an embedded deployment as a web application within Apache Tomcat. When considering the issue about embedded databases and memory management described for Apache Derby, we disregard this deployment variant.

In the `conf.xml` of eXist-db we reconfigure the path of the data directory of the database for the evaluation. eXist-db provides a number of configuration entries concerning memory limitations: `cacheSize`, `collectionCache`, etc. We do not touch the default values, as they seem already in a correct magnitude for the evaluation. In the `startup.sh` script of eXist-db we set the server port to 8888, since the default value 8080 collides with Apache Tomcat. The eXist-db server is started with the `startup.sh` script and is then ready to accept requests. Per default, eXist-db already creates an empty XML root collection for XML documents that are imported.

For the integration into Apache ODE, we use the *XML:DB* API ([Sta01]). eXist-db provides `JAR` files for the API and its implementation for the client side. An additional manual[27] describes the usage of the XML:DB API for eXist-db. Unfortunately, the documentation does not mention which `JAR` files are exactly required by a database client application. Apart from this restriction, the further integration was very simple.

As there are no issues concerning installation or integration for eXist-db, we see a *total fulfillment* of this criterion.

[25] `http://monetdb.cwi.nl/XQuery/Documentation/Using-XRPC-from-Java.html`
[26] jetty is a web application container similar to Apache tomcat; see `http://jetty.codehaus.org/jetty/`
[27] `http://exist-db.org/devguide_xmldb.html`

H2 Database The installation and integration of H2 Database is very similar to Apache Derby. H2 Database just provides a single `h2.jar` (1.2 MB) for the database kernel and the client JDBC driver. We copy this file into the `lib` folder of Apache Tomcat and configure three *DataSources* with the following JDBC URLs:

- embedded mode: `jdbc:h2:file:/media/idares-evaluation/db-data-storage/H2Evaluation/idares-eval-embedded;CACHE_SIZE=393216`
- client server mode: `jdbc:h2:tcp://localhost:9093/idares-eval-client;CACHE_SIZE=393216`
- main memory mode: `jdbc:h2:mem:idares-eval-mem;DB_CLOSE_DELAY=-1`

For the embedded mode we define an absolute file path where the data can be stored. For the client server mode, we define a localhost tcp connection. In both cases the size of the database cache is defined by the property `CACHE_SIZE` in kB (about 384 MB). The third JDBC URL is intended for the in-memory evaluation of H2. No server or path is given, as H2 does not persist the data. The property `CLOSE_DELAY=-1` specifies that the main memory data storage is only dropped when Apache ODE is shutdown.

The H2 server can be started with the following command:

```
java -Xms128M -Xmx600M
     -cp h2.jar
     org.h2.tools.Server -tcp -baseDir /media/idares-evaluation/
        db-data-storage/H2Evaluation
```

We use the script `[DVD]/Sources/Tools/h2ServerEvaluation.sh`

The same memory discussion made for Apache Derby can be applied for the H2 Database too. It is extremely difficult to find an appropriate upper memory limit for the virtual machine running the H2 kernel. Moreover, on `OutOfMemoryErrors` the H2 server might get stuck too and the client waits forever for its reply.

All in all, we assign the rank *mostly fulfilled* to H2 Database this criterion.

5.4.4 Criterion 4 – Administration and User-Friendliness

Apache Derby Apache Derby provides a command line tool to interact with a database: `ij`. Moreover, Apache Derby provides a Java Servlet based web application when used within an application server. Nevertheless, a graphical JDBC↗ database tool like SQuirreL SQL[28] is sufficient for default database management interactions. Apache Derby automatically creates the directory structure for a new database and does not require an administration task for that. All core database management activities have to be performed using SQL statements, e. g., for hot backup a stored procedure has to be called.

[28] `http://squirrel-sql.sourceforge.net/`

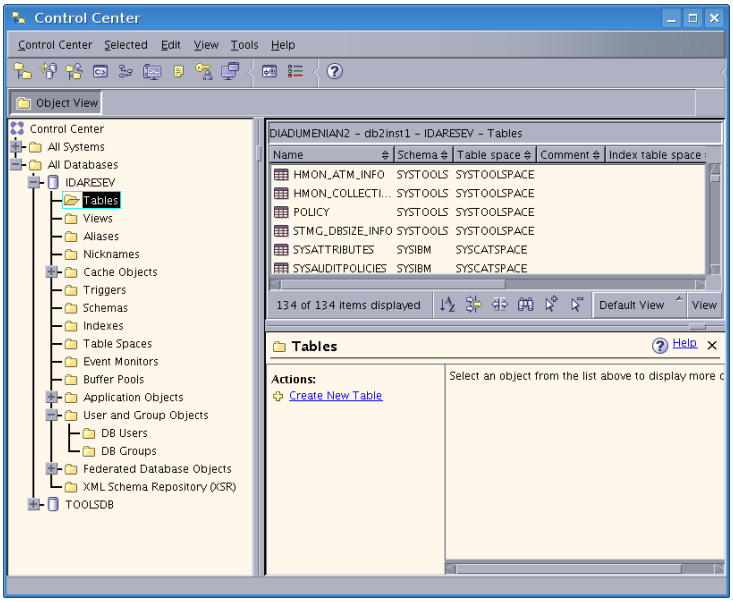


Figure 5-v: A screenshot of the DB2 Control Center

For the documentation a reference manual, an administration guide, and a tools guide are provided as online HTML websites. The information are described very detailed but understandable for the unexperienced user, too. Database exceptions are presented with an SQL error code and a brief error message. In most cases the error can be corrected with this information.

All in all, there is a good documentation, but no graphical administration interface. Hence, we assign *mostly fulfilled* for this criterion.

IBM DB2 IBM DB2 is a database system with many components and processes. Hence, a minimal experience is required to administrate a DB2 instance. IBM provides a vast set of graphical tools to administrate the database system in many ways. Figure 5-v shows the *DB2 Control Center*, where new databases can be created, the database system can be started and shutdown, backups can be planned, and the state of the database system can be investigated. The DB2 Control Center is very intuitive, but provides too many commands and choices for the unexperienced user.

For security reasons, DB2 is installed with different roles: a *database instance owner* and a *database administrator*. When used at a multi-user workflow environment, these roles are justifiable. In a single user desktop environment, this is an annoying issue, as the appropriate role has to be chosen to start or manage the database.

Exceptions and error messages are returned by the DB2 system with a cryptic set of error codes, like:

```
SQLCODE: -104, SQLSTATE: 42601, SQLERRMC=CURSOR;OF;ASC,
   DRIVER=3.53.95
```

In such cases the online manual has to be considered to investigate what the current out of about 3,000 error means. On the upside, the error descriptions not only contain possible reasons for an error, but also proposed solutions. The whole online documentation[29] provided for IBM DB2 is commendable and not only contains topics about administration issues, but also useful background information.

Balancing the pros and cons, we assign the rank *mostly fulfilled* to DB2 for this criterion.

MySQL Cluster With the default installation the MySQL cluster does not contain a graphical administration tool, but solely a set of shell based programs. In addition, MySQL provides the *MySQL Workbench* as front end for SQL development and database administration. The administration part of the MySQL Workbench is very powerful and provides tools for role management, database creation, data backup, and system status observation.

The online documentation for the MySQL server is very comprehensive and provides a separated chapter for the MySQL Cluster, too. Hence, MySQL Server *totally fulfills* this requirement.

MonetDB (SQL and XQuery) MonetDB does not provide any kind of administration tools. A started MonetDB server provides a shell interface, but the documentation states "Although this window comes with an interactive prompt, you should (unless you know what you are doing) keep this window minimized" [30]. Hence, all administration tasks have to be performed via SQL statement. The documentation provides an adequate SQL and XQuery reference but only a less detailed description for server configuration and administration. To sum it up, we assign the rank *mostly fulfilled* to MonetDB for this criterion.

eXist-db As the default deployment of eXist-db utilizes a web server, this server is utilized for administration tasks too. When logged into the administration part of the web pages, the administrator can investigate the health of the XML database system, create and drop collections, and install additional modules. The documentation for eXist-db contains a tutorial and a very comprehensive library for the description of all supported XQuery functions. The documentation is clear and understandable for less experienced users too. Hence, we assign the rank *totally fulfilled* to eXist-db for this criterion.

H2 Database The default H2 download provides a web based *H2 Server Console* to interact with any kind of database system through a JDBC↗ driver. Although this approach seems nice, its features are limited and do not support typical administration tasks for the database. Hence, all administration activities have to be performed using SQL statements. SQL error messages in the

[29] `http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp?topic=/com.ibm.db2.luw.container.doc/doc/c0052964.html`

[30] `http://monetdb.cwi.nl/SQL/Documentation/Start-and-Stop.html`

H2 Server Console console can be analyzed with a simple click on a help button and the related documentation entry is presented. The same can be achieved with the *Error Analyzer*[31].

The online documentation of H2 is clear, but not very detailed. On the upside, the online documentation is easy to navigate and provides helpful syntax diagrams for the SQL reference. Nevertheless, the non-existent graphical user interface forces us to see the criterion only *mostly fulfilled*.

5.4.5 Criterion 5 – SQL Features

This criterion only applies to relational databases. When looking at SQL support, the following features are supported by all relational databases:

- `SELECT` with column projection and `WHERE` predicates
- `INNER JOIN` and `OUTER JOIN`
- `GROUP BY`
- `ORDER BY`
- sub queries in `FROM` and `WHERE` clause
- `SCHEMA`s or `CATALOG`s as super structure
- data types: `SMALLINT` (16 bit), `INTEGER` (32 bit), `BIGINT` (64 bit), `FLOAT` (32 bit), `DOUBLE` (64 bit), `DECIMAL`, `CHAR`, `VARCHAR`, `BLOB`, `CLOB`, `DATE`, `TIME`, `TIMESTAMP`

For the other SQL features, we present a comparison in the following table. The sign “X” is used in a cell to indicate that a database supports a feature, “–” is used otherwise. When the documentation is not clear about a feature, we use the sign “?”.

SQL Feature	Derby	IBM DB2	MySQL	MonetDB	H2
`PRIMARY KEY`	X	X	X	X	X
`FOREIGN KEY`	X	X	–	X	X
`(UNIQUE) INDEX`	X	X	X	X	X
`UNION`	X	X	X	X	X
`INTERSECT`	X	X	–	–	X
`EXCEPT`	X	X	–	X	X
`CHECK CONSTRAINT`	X	X	–	–	X
`WITH RECURSIVE`	–	X	–	–	–
`TEMPORARY TABLE`	X	X	–	X	X
`VIEW`	X	X	X	X	X

[31] `http://www.h2database.com/html/sourceError.html`

SQL Feature	Derby	IBM DB2	MySQL	MonetDB	H2
MATERIALIZED VIEW	–	X	–	–	–
TRIGGER	X	X	X	X	X
PROCEDURE / FUNCTION	X (Java)	X	X	X	X (Java)
BOOLEAN TYPE	–	–	X	X	X
ENUM TYPE	–	–	X	X	–
UUID↗ TYPE	–	–	–	–	X
DOMAIN type	–	–	–	–	X
structured TYPE	–	X	–	X	–
XML extensions	X	X	X	X	–
spatial extensions	–	X	X	X	–
load from CSV via SQL	X	X	X	–	X
export into CSV via SQL	X	X	X	–	X
authorization	X	X	X	?	X

Table 5-ii: SQL feature comparison of the evaluated relational databases

As the comparison of the relational databases shows, all databases support a common set of features. However, some databases do not support major features, which has an effect on their degree of fulfillment:

- Apache Derby does not support DOMAINs or other customized structured TYPEs. Since ℝ **31** on page 53 required customized types Derby only *mostly fulfills* the requirement.
- IBM DB2 shows its feature-richness and hence *totally fulfills* the requirement.
- MySQL Cluster does not support temporary tables, customized types, and advanced set operations. FOREIGN KEY constraints can be defined, but are ignored. Hence, it only *mostly fulfills* the requirement.
- MonetDB SQL does not support the import or export relational data from or into CSV files and does not provide a documentation about its authorization mechanism. Therefore it *mostly fulfills* the requirement.
- H2 Database is an astonishing feature-rich database for being an embedded database with only 1.2 MB size. Nevertheless, H2 does neither XML columns nor XPath queries and hence only *mostly fulfills* the requirement, since ℝ **32** on page 54 required support for XML based annotations to relational records.

5.4.6 Criterion 6 – XQuery Features

This criterion is only applied to databases we evaluate for their XML data storage.

IBM DB2, MonetDB XQuery, and eXist-db support XQuery to a certain degree. eXist-db is officially XQuery 1.0 ([BCF+07]) compliant and published its rating in the XML Query Test Suite (short XQTS)[32]: 99.4 % of the test queries are working as demanded. MonetDB XQuery is officially XQuery 1.0 conform, but does not provide results of compliance tests. IBM DB2 with pureXML does officially not comply with the XQuery standard, although its proprietary version has the same syntactical language elements. The basic XQuery function support seems satisfying for all three databases and we concentrate on the other requirements of this criterion for further comparison:

XQuery Feature	**IBM DB2**	**MonetDB XQuery**	**eXist-db**
single XML import	`IMPORT` command	`pf:add-doc`	`xmldb:store`
multiple XML import	–	with a set of functions possible	`xmldb:store-files-from-pattern`
XML export	`EXPORT` command	–	`file:serialize`
`For Let Where Order Return`	X (XQuery 1.0)	X (XQuery 1.0)	X (XQuery 1.0)
manipulation / deletion	XQuery updates equal to XQUF[33]; SQL `UPDATE` or `DELETE`	XQUF; **only** for updatable collections	XQUF; XML:DB Update
namespace support	X	X	X
validation on import	X	–	X
validation on demand	`XMLVALIDATE`	–	–
XSLT[34] support	`xsl:value-of`	–	`transform:transform`
indexes	for nodes, text, and attributes	automatically created on accessed text and attribute values	only text indexes
customized functions	–	X	X

Table 5-iii: XQuery related feature comparison of the evaluated XML databases

To sum up the comparison, we conclude the following degrees of fulfillment:

- IBM DB2 supports nearly every requirement for an XQuery based XML database, but is not compliant to the XQuery 1.0 standard. Therefore, DB2 only *mostly fulfills* this criterion.

[32] `http://www.w3.org/XML/Query/test-suite/`
[33] XQuery Update Facility: `http://www.w3.org/TR/2006/WD-xqupdate-20060711/`
[34] Extensible Stylesheet Language Transformation

- MonetDB XQuery supports all major requirements except the validation and therefore only *mostly fulfills* this criterion.
- eXist-db supports all major requirements we mentioned and provides an enormous set of additional XQuery functions, too. Hence, it *totally fulfills* this criterion.

5.4.7 Criterion 7 – Transactions and ACID

Apache Derby implements full ACID compliance with all four JDBC↗ isolation levels[35]. Nevertheless, Apache Derby does not support row level locking and hence receives a *mostly fulfilled*.

IBM DB2 has full ACID compliance, implements all four JDBC isolation levels and provides sophisticated row level locking too. In order to keep the list of locked objects small, DB2 even supports a lock escalation mechanism, which transforms too many row locks into a table lock when a given limit of row locks is reached. All in all, DB2 *totally fulfills* this criterion.

MySQL Cluster claims to be fully ACID compliant with one exception. In a MySQL cluster environment a commit results in a distributed update among all involved Network DataBase (NDB) nodes. Nevertheless, each of the nodes does not guarantee to flush the modifications into persistent logs on disk. Since we only configure one NDB node for our evaluation, MySQL Cluster does not guarantee durability as understood in the ACID concept. Furthermore, MySQL Cluster only supports the JDBC isolation level `READ COMMITTED`. Therefore, we have to degrade MySQL Cluster to *mostly fulfilled*.

MonetDB (SQL and XQuery) The documentation of the MonetDB server does not clearly mention the ACID compliance of MonetDB. But the documentation[36] mentions "ACID hooks", "full transaction control and recovery", and "Write-Ahead-Logging" [37] (WAL). We interpret these facts as: MonetDB implements some exceptional cases around ACID for performance reasons, but basically guarantees ACID compliance. The documentation moreover mentions that MonetDB only supports the highest JDBC isolation level `SERIALIZABLE`. The performance evaluation has shown that MonetDB regularly crashes when too many concurrent requests have to be isolated. Hence, MonetDB SQL *mostly fulfills* this criterion.

When considering the XML storage, MonetDB even requires that document management commands and XQuery updates are made in different transactions. Our harness for the evaluation revealed that MonetDB XQuery does not even allow a JDBC `commit()` at the end of a transaction. Instead, the JDBC driver complains that "`COMMIT` is not an XQuery compliant expression". This

[35] the JDBC isolation levels are counterparts to the SQL isolation levels; see: `http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Connection.html#TRANSACTION_READ_COMMITTED`
[36] `http://monetdb.cwi.nl/MonetDB/Documentation/MonetDB-Con_0027s.html`
[37] `http://monetdb.cwi.nl/MonetDB/Version4/Documentation/monet/index.html`

obvious bug forces us to use auto commit for the MonetDB XQuery JDBC driver. All in all, we only assign a *partly fulfills* to MonetDB XQuery.

eXist-db The eXist-db roadmap[38] mentions that currently eXist-db does not support transactional scopes to the client. All read activities bypass the internal transactional system and only an asynchronous thread synchronizes the main memory cache with the persistent storage on the disk. Moreover, the current XML:DB API implementation does not even provide transactional commands on the client side. Summing up these facts, eXist-db does *poorly fulfill* this requirement.

H2 Database is fully ACID compliant, implements *Multi-Version Concurrency Control* (MVCC), and even has support for *Two Phase Commit* (2PC). Moreover, H2 supports row level locking, too. All in all, we assign the rank *totally fulfills* to H2 Database.

5.4.8 Criterion 8 – Cache Consistency

This criterion does not evaluate databases, but investigates whether a database provides features that are useful for implementing a local cache. Only the three databases are stated that provide such an exploitable feature.

IBM DB2 comes with a lot of tools and additional programs. One of them is DB2 *High Availability Disaster Recovery*[39] (HADR). HADR is used to synchronize data manipulations from a master database node to a slave database node. This is done by a proprietary protocol between two DB2 database nodes. This feature might be exploitable to let the local cache automatically synchronize manipulation to an external database, where the data originates from.

eXist-db [McC09] shows how XQuery functions of eXist-db can be used to export a collection of documents with an additional last modification timestamp. On basis of these timestamp it can be decided, which documents have been changed since a given last synchronization date. [McC09] performs the actual synchronization with an Apache Ant[40] script.

H2 Database provides a feature called *Linked Tables*. This feature transparently integrates an external table via a JDBC connection into the local schema. This feature may be exploited to directly use an `INSERT INTO SELECT` SQL statement for the synchronization.

[38] `http://exist-db.org/roadmap.html`
[39] `http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp?topic=/com.ibm.db2.luw.admin.ha.doc/doc/c0021002.html`
[40] `http://ant.apache.org/`

5.4.9 Criterion 9 – Performance of Relational Databases

The criterion is applied to five databases with eight deployment variants in total. Unfortunately, the H2 Database in in-memory mode does not even succeed in loading the initial set of reference data and is therefore discarded. Appendix A.6.1 on page 201 shows the measured durations for all databases and all groups of tests. The criterion requested the execution of 75 tests. Some tests, e. g., all TPCH import tests, are aggregated together into a test group, resulting in 33 distinctive test groups. Some retrieval tests are repeated a number of times. We apply the arithmetic mean to these values. As a result, we have duration measured in milliseconds of the form $D_{d,t} \in \{\text{fault}\} \cup \mathbb{R}^+,\ d \in \{1,\ldots,7\}\ ,\ t \in \{1,\ldots,33\}$, where d is the index for the database and t the index of the test group.

These durations are now analyzed for further evaluation. The fully detailed analysis can be found in `[DVD]/Documents/Stuff/Relational Results.xlsx` – at this place, we rather present the approach and the final analysis result. First, we calculate the median[41] for each test group: $M_t = median(\{D_{d,t}\})$. A median gives us an impression about the average duration without being sensitive to extreme values as the arithmetic mean would be. Second, we set each duration $D_{d,t}$ in relation to the related median: $Rel_{d,t} = \frac{D_{d,t}}{M_t}$. These figures allow us to inspect anomalies: values less than 0.5 indicate that a database performed a test group in a duration less than half of the median, values greater than 5.0 indicate that a database performed bad in a test group, because its duration took longer than 5 times of the median. In a third step, we calculate the rank for each database per test group: $Rank_{d,t} = 1 + |\{D_{i,t} : D_{i,t} < D_{d,t}\}|$. In a forth step of the analysis, we assign a virtual rank 8 to faulty executions. The result is the $AdjRank_{d,t} \in \{1,\ldots,8\}$:

$$AdjRank_{d,t} = \begin{cases} 8 : D_{d,t} = \text{fault} \\ Rank_{d,t} : else \end{cases}$$

In order to allow a simpler comparison of the values, we investigate five test sets. Therefore, we further aggregate the test groups that are logical related and apply to similar applications of the database. The test sets are:

- *Import* subsumes tests for bulk loading data into the database.
- *Simple* contains simple tests that just retrieve all rows of a table or realize a predicated retrieval.
- *TPC-H* are six complex TPC-H benchmark queries.
- *Concurrent* are five tests that concurrently manipulate, retrieve, and delete data.
- *CO_2* are queries upon the FEM↗ model by [CDR+07].

For each test set, we calculate the arithmetic mean of the corresponding $AdjRank_{d,t}$.

[41] The median is the number that divides an ordered group of values into an upper an lower half.

How often...	Derby C/S	Derby Em	IBM DB2	MySQL	MonetDB	H2 C/S	H2 Em
best	1	3	7	0	16	5	1
better than median	15	9	21	7	20	14	10
twice as good as median	1	4	12	1	16	4	5
worse than or equal as median	17	21	12	22	7	19	23
worst	0	3	3	7	0	7	4
faulty	1	3	0	4	6	0	0

(a) Summary of the evaluation for the relational databases and all 33 test groups

Test Set	Derby c/s	Derby em	IBM DB2	MySQL	MonetDB	H2 c/s	H2 em
Overall	3.85 (3)	4.67 (6)	3.12 (1)	5.15 (7)	3.33 (2)	4.18 (4)	4.33 (5)
Import	4.25 (4)	5.00 (5)	2.00 (1)	6.75 (7)	2.00 (1)	5.00 (5)	3.00 (3)
Simple	3.75 (5)	3.67 (3)	3.50 (2)	3.67 (3)	2.42 (1)	5.42 (6)	5.58 (7)
TPC-H	5.00 (6)	5.83 (7)	4.50 (4)	4.50 (4)	1.83 (1)	2.83 (2)	4.00 (3)
Concurrent	2.60 (2)	6.00 (5)	3.00 (3)	6.60 (6)	7.40 (7)	2.00 (1)	3.20 (4)
$CO2_2$	4.29 (4)	4.43 (5)	1.86 (1)	6.71 (7)	2.71 (2)	5.00 (6)	3.43 (3)

(b) Performance evaluation results as arithmetic means of $AdjRank_{d,t}$ for the relational databases; the figures in brackets indicate the rank of a database for a test set; c/s = client server mode, em = embedded mode

Table 5-iv: Performance results for the relational databases

A summary of the performance results is shown in Table 5-iv-a on the current page. The $AdjRank_{d,t}$ are shown in Table 5-iv-b on this page. Both tables are considered the major outcome of this criterion.

Besides the $AdjRank_{d,t}$ values, we present special issues we had with applying the databases. A general topic considers the dialects of the databases. Especially queries with date or time calculations and substring functions often needed database specific variants. A query for the CO_2 example was designed to use a huge intermediate result set produced by a `UNION` operator. Unfortunately only IBM DB2 successfully recognized a key column within and was able to execute the query in excellent time. For the all other databases, a temporary table had to be introduced to bring the execution duration down to an acceptable value. The following paragraphs mention database specific notes.

Apache Derby As stated at the results for Criterion 3, we had to set the heap size of the JVM↗ very high to avoid `OutOfMemoryErrors`. This issue was not investigated during the bulk loading tests, but at the tests that created the foreign keys. The memory limits were 700 MB for the server mode and 1,300 MB for embedded mode. Unfortunately, we had to deactivate the TPCH query 22 for both Apache Derby variants, as the calculation took more than 7 hours. Moreover, the embedded variant showed a strange exception for two concurrent tests: `EmbedSQLException: Conglomerate could not be created. SQLState(XSCH4) ErrorCode(20000)` that could not be eliminated.

IBM DB2 When considering the attempts to achieve a clean performance run, IBM DB2 is the best candidate. This database was the first, where all 75 tests performed within an acceptable time frame. We applied the `RUNSTATS` command for DB2 that analyzes the data values and internally calculates statistics to improve the access paths for subsequent queries. We added the caused durations to the test set *Import*. Unfortunately, IBM DB2 seems to have problems with some queries. Especially the query relying on substring calculations took about 3,000 times as long as the median. We assume that a specific configuration tuning would improve the performance by magnitudes. All in all, IBM DB2 ist the best database when considering the average $AdjRank$ for all queries, for the test set *Import*, and the test set *CO_2*.

MySQL Cluster During the preparation and execution of the test queries, MySQL Cluster showed a lot of idiosyncrasies. Special table definition scripts were created that contain the suffix `ENGINE=ndbcluster` to specify that a table is to be stored in the main memory of the cluster. Moreover, MySQL does not allow schema prefixes for index names. Finally, MySQL Cluster performed worst the most times and had a lot of queries that took much too long or simply caused exceptions. This is especially true for a CO_2 query that took more than 5 hours, while the median lies at 19 seconds. This is also seen in the average $AdjRank$ for all queries, where MySQL take by far the last place.

MonetDB SQL The database with the most crashes was MonetDB SQL. This was not due to too much memory consumptions, but simply because of bugs in the database that caused the operating system to throw *Segmentation Faults*. For example a CO_2 query was syntactically well formed but crashed MonetDB SQL every time when executed. On the upside, MonetDB performed best in 16 of the 33 tests groups and was never worst (when crashes are ignored).

H2 Database There where nearly no special query variants necessary for H2 Database. An interesting fact is that H2 performed better in client server mode than in embedded mode – a contrary to the discussion made in Section 2.4.3 on page 38. Moreover, H2 was the best database for the concurrent queries and second for the TPC-H queries. The H2 SQL syntax provides a short command `ANALYZE` that is the counterpart to the `RUNSTATS` command of IBM DB2. We applied this command after loading the data.

The in-memory mode of H2 Database was not even able to load the TPC-H data into the memory. We applied the highest maximal heap of about 2,2 GB RAM for Apache ODE with embedded H2 database, but this amount was not sufficient. An `OutOfMemoryError` was the cause. Therefore, we discarded the in-memory mode of H2 for all further tests.

How often...	IBM DB2	MonetDB XQuery	eXist-db
best	7	12	19
better than median	7	12	19
twice as good as median	3	10	9
worse than or equal as median	28	26	19
worst	13	18	4
faulty	3	0	0

(a) Summary of the evaluation for the XML databases and all 38 tests

Scope	IBM DB2	MonetDB XQuery	eXist-db
Overall	2.32 (3)	2.16 (2)	1.61 (1)
Only scale 0.02	2.21 (2)	2.37 (3)	1.47 (1)
Only scale 0.1	2.42 (3)	1.95 (2)	1.74 (1)

(b) Performance evaluation results as arithmetic mean of $AdjRank_{d,t}$ for the XML databases; the figures in brackets indicate the rank of a database within a given scope

	IBM DB2	MonetDB XQuery	eXist-db
Slowdown	8.83 (2)	3.27 (1)	10.81 (3)

(c) Average slowdown for the XML databases between the smaller and the larger XML file; the figures in brackets indicate the rank of a database

Table 5-v: Performance results for the XML databases

5.4.10 Criterion 10 – Performance of XML Databases

This criterion is applied to three databases. The criterion defines 19 XQuery based tests, which are applied for two differently sized XML documents. Hence, we have 38 tests in total. Appendix A.6.2 on page 202 shows the detailed duration results for these tests. The retrieval oriented tests where applied multiple times and the arithmetic mean of the durations is stated for these tests. Table 5-v-a on the current page shows a summary of this performance evaluation. We also apply the $AdjRank_{d,t}$ approach for the XML performance evaluation, as described in Section 5.4.9 on page 122. Table 5-v-b shows the arithmetic mean of the $AdjRank_{d,t}$ for each database and different scopes.

As we have executed the exact same queries for two different sizes of XML documents, we can calculate the resulting slowdown. We define the slowdown as $SlowDown_{d,t1,t2} = \frac{D_{d,t1}}{D_{d,t2}}$, where $t1$ and $t2$ are tests with the same query, but $t1$ on the XML document with scale 0.02 and $t2$ with scale 0.1. Therefore, we expect values for the slowdown of about 5.0, if the durations scale linear with the increased XML document size. Table 5-v-c on the current page depicts the average slowdown for all 19 tests.

In contrast to the relational benchmark, the XML benchmark does not require a lot of individual adaptions of the queries. The compliance to the XQuery standard is very convincing for the three databases. Nevertheless, we have to define three different query variants anyway, because the XQuery standard does not specify how the XML documents are addresses within

Listing 5.1 Relational table to store XML documents for DB2

```
1 CREATE SCHEMA XMLEVAL;
2 CREATE TABLE XMLEVAL.COLLECTION (
3   ID      INTEGER NOT NULL PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
4   CONTENT XML NOT NULL
5 );
```

the collection. As a consequence, each database uses a different parametrization of the method `fn:collection(path)` to retrieve to root path of all XML documents.

IBM DB2 supports XML files as columns of relational tables. Therefore, we create a table just for the XML evaluation (see Listing 5.1). The XML documents of this table can then be addressed with the XQuery method `db2-fn:xmlcolumn('XMLEVAL.COLLECTION.CONTENT')`. Moreover, DB2 requires the keyword `XQUERY` at the begin of each XQuery statement to distinguish them from SQL statements. All in all, DB2 showed a poor performance in the XML performance evaluation. DB2 needed in 13 of 38 times the longest duration and one XQuery statement took more than 7 hour before it was canceled manually. Moreover, DB2 does not support the declaration of customized methods and hence fails to perform one of the chosen XMark queries. On the upside, IBM DB2 only required 624 MB RAM at peak, which is about half of the size of other two candidates.

MonetDB XQuery is an in-memory database and therefore we expect a great performance. Unfortunately, MonetDB XQuery only shows the best performance 12 of 38 times. Likewise, MonetDB only takes the second place of the average overall $AdjRank$ in Table 5-v-b. On the upside, MonetDB XQuery totally complies with the XQuery statement from XMark and shows the by far lowest slowdown in Table 5-v-c.

eXist-db is the clear winner of the XML performance evaluation. It executes half of the queries in the lowest duration and is very easy to configure. Moreover, we did not alter any memory settings and used eXist-db just as it was installed. On the downside, eXist-db utilizes 1.2 GB main memory at peak – nearly as much as the in-memory database MonetDB XQuery does. Moreover, we have to emphasize that eXist-db shows an average slowdown of about 10.8. eXist-db even has slowdown values of about 49.3 at maximum, which indicates that these queries scale quadratic with the size of the XML document. This can be interpreted as the result of missing indexes of eXist-db. Without an index, eXist-db may have to perform some queries with nested loop joins.

5.4.11 Criterion 11 – Performance of the ODE Model DB

This criterion is applied to three external storage based relational database systems that are candidates for the *ODE model db*. The results of the evaluation for this criterion are stated in

How often...	Apache Derby embedded	DB2	H2 Client
best	5	0	3
better than median	5	0	3
twice as good as median	3	0	0
worse than or equal as median	3	8	5
worst	1	5	2
faulty	0	0	0

Table 5-vi: Summary of the evaluation for the *ODE model db* and all 8 tests.

Appendix A.6.3 on page 203. All tests where executed twice and the arithmetic mean was applied to their durations. Table 5-vi shows a summary of the evaluation results.

When looking at the bare figures, the durations Apache ODE needs for a start and a shutdown seems not to be affected by the database. The durations are very close to each other for all three databases.

Apache Derby (embedded mode) seems to be the clear winner of this evaluation. When using as model db, Apache ODE executes the example workflow within the shortest duration. For the three tests where Apache Derby was not the best candidate, the H2 Database showed only slightly shorter durations. All in all, in our test environment, Apache Derby is the best candidate for the *ODE model db*.

The setup of Apache Derby as *ODE model db* did not cause any effort, since it is configured as the default database.

IBM DB2 is integrated into Apache ODE by setting configuration parameters in `ode-webapp/WEB-INF/conf/ode-axis2.properties`. We configure Apache ODE to use an `EXTERNAL` *DataSource* provided by Apache Tomcat and Hibernate as DAO↗ layer. The attempt to use OpenJPA as DAO layer failed due to a not fully supported OpenJPA dialect for the DB2 JDBC driver. Unfortunately, the access to DB2 via Hibernate did not succeed on the first attempt either. A bug in the JDBC driver provided by the DB2 9.7 installation forces us to alter the Hibernate Mapping Files (HBM)[42]. Thereby, the faulty retrieval of generated key values via the JDBC driver is omitted and instead an SQL `SEQUENCE` is utilized to generate unique numbers.

However, the performance of the DB2 in the evaluation tests for this criterion is poor. DB2 shows the worst performance in 5 of 8 times and required 2 to 11 times of the duration of Apache Derby for the actual workflow executions. At this point we can not distinguish, whether this is the fault of a misconfiguration, the fact that Hibernate is used instead of OpenJPA, a lack of performance of IBM DB2, or a combination of these three issues.

[42] See bug reports `https://jira.jboss.org/jira/browse/JBPAPP-2408` and `http://www-01.ibm.com/support/docview.wss?uid=swg1IC61590`

H2 Database (Client mode) is configured into Apache ODE by setting configuration parameters in `ode-axis2.properties` too. We configure Apache ODE to use an `EXTERNAL` *DataSource* provided by Apache Tomcat and OpenJPA as DAO layer.

The performance of the H2 Database in client mode is obviously poorer than of Apache Derby in embedded mode. In average, H2 Database requires about three times the duration as Apache Derby does.

Note: obviously, the evaluation of the DB2 database does not apply the same conditions as the other two evaluation do. An evaluation in future might investigate this issue further and either apply a fixed JDBC driver for DB2 or evaluate Hibernate for the other two databases too.

5.5 Selection

After all databases have been evaluated according to the eleven criteria, it is now possible to select the databases that are to be applied for the database architecture. In other words, the task of this section is to study the database short list again (cf. Figure 5-iv on page 106) and argue which is the most appropriate database system for each cell of the short list. Before beginning with the argumentation, we present an overview of the evaluation results in Table 5-vii on the facing page.

We start with the selection of an external storage based relational database. The candidates are: IBM DB2, Apache Derby, and H2 Database. We choose IBM DB2 because it shows the best overall performance in the relational benchmark and the best set of features. As open source alternative, both competitors are possible. Derby and H2 show similar performance, while H2 has less faulty tests and *totally fulfills* the ACID requirement. Hence, H2 Database is recommended as open source alternative to DB2.

As external storage based XML database, we have the candidates IBM DB2 and eXist-db. Here, we also suggest IBM DB2. Although DB2 shows a weaker performance in the XML tests, we are then able to exploit its sophisticated combination of relational and XML data store (favored by $\mathbb{R}$ **32** on page 54). Moreover, IBM DB2 provides a transactional API, which is a lack of eXist-db. Since eXist-db is the only competitor, we recommend it as open source alternative to IBM DB2. At this point, we emphasize that we left out other promising candidates of [HSW08] for our evaluation. A future evaluation might, for example, investigate Oracle Berkeley DB XML that claims to provide full ACID compliance.

As the in-memory based relational database, we have the candidates MonetDB SQL and MySQL Cluster. Here, we recommend MonetDB SQL. The MySQL Cluster seems to be intended for clustered applications only and shows the weakest overall performance in the evaluation (even when considering the external storage based databases). Hence, we discard MySQL Cluster for the application as an in-memory database anyway. Unfortunately, the recommendation of MonetDB SQL comes with the warning that the evaluated version of the database seems to be

Database \ Criterion	1) License/Costs	2) Platform	3) Integration	4) Administration	5) SQL Features	6) XQuery Features	7) Transactions	9) Relational Performance	10) XML Performance	11) ODE Model DB
Apache Derby client	++	++	+	+	+		+	3		
Apache Derby embedded	++	++	+	+	+		+	6		1
DB2	o	+	+	+	++	+	++	1	3	3
MySQL Cluster	+	++	o	++	+		+	7		
MonetDB SQL	++	++	+	+	+		+	2		
MonetDB XQuery	++	++	+	+		+	o		2	
eXist-db	++	+	++	++		++	–		1	
H2 Database client	++	++	+	+	+		++	4		2
H2 Database embedded	++	++	+	+	+		++	5		
H2 Database memory	++	++	+	+	+		++	–		

Table 5-vii: Overview of the results of the database evaluation; for each criteria regarding a degree of fulfillment the following signs are used: ++ *totally fulfilled*, + *mostly fulfilled*, o *partly fulfilled*, – *poorly fulfilled*; for a performance criteria the rank of the overall $AdjRank$ is stated; an empty cell indicates that a database was not evaluated for this criterion

partially instable and causes irregular crashes. Hence, we suggest a further survey for potential in-memory based relational database candidates. In addition, we can suggest that the IBM DB2 can serve as a replacement for MonetDB SQL, too. DB2 shows good performance in most tests and is the most stable candidate. However, especially for complex queries MonetDB SQL mostly shows top durations, some with remarkable dominance. All in all, we recommend MonetDB SQL as database for the local cache in cases where its application as main memory database seems promising or is explicitly required.

MonetDB XQuery is the only database that supports an in-memory XML data storage and is therefore recommended for this application in the integrated database architecture. However, MonetDB XQuery does not exhibit a clear performance advantage compared with the two external storage based XML databases (i. e., IBM DB2 and eXist-db). Therefore, its purpose as a local cache for XML data can be doubted.

The selection of an *ODE model db* is the most difficult one. On the one hand, Apache Derby in embedded mode shows the best performance for all five tests that execute the example workflow. On the other hand, we want to find a more suitable replacement for Apache Derby as [Les09] has

suggested. The approach to use DB2 as a substitute seems tempting, since we already apply this database in the database architecture and are thereby minimizing the total number of selected databases. This is an important issue for maintenance and administration. Apart from the weak performance of IBM DB2 in the *ODE model db* performance evaluation test, the overall relational performance of DB2 is convincing. Moreover, DB2 shows more stability than Apache Derby in the other performance evaluations, which is important for long running simulation workflows. Hence, we suggest IBM DB2 as *ODE model db* with the strong suggestion to investigate the issues stated in Section 5.4.11 on page 126. Until a further investigation, Apache Derby in embedded mode is recommended as open source alternative to DB2, because it outperforms H2 Database in this context.

6 Implementation and Prototypes

All necessary implementation details of the database architecture can be found in this chapter. We start with a summarization of the used software and frameworks. Then we consider details of the installation of the runtime environment for simulation workflows. In the subsequent sections, we take a look at the specification and the implementation of related Web Services and their operations. The chapter continues with the description of the two prototype applications of the database architecture and the example workflow. Finally, the runtime environment is briefly described.

6.1 Used Software and Frameworks

For the realization of the integrated database architecture, various software frameworks where used. The following Table 6-i on the next page lists all software that is part of the integrated database architecture and software that was just used during the development. The table contains information about the license of the software. This is important because the license of the integrated database architecture has to be compatible to the licenses of the software that is part of the integrated database architecture.

Name	**Version**	**License**	**Application/Purpose**
Software that is part of the database architecture			
Apache Axis2	1.5.1	Apache License 2.0	Web Service server environment
Apache Commons Collections	3.2.1	Apache License 2.0	utility framework for collection activities
Apache log4j	1.2.15	Apache License 2.0	logging framework
Apache ODE	2.0-beta2	Apache License 2.0	BPEL workflow runtime engine
Apache Tomcat	6.0.20	Apache License 2.0	Java web application container
Debian Linux	5.0.3	GPL	Operating system for the runtime environment
Hibernate	3.2.4.GA	LGPL 2.1	data type handling between database and Java
IBM DB2 32bit	9.7	Proprietary	Relational and XML database
JiBX	1.2.1	JiBX License, BCEL License, XPP3 License	Java XML mapping

Name	Version	License	Application/Purpose
MonetDB SQL	2.34.2	Monet Public License	Relational in-memory database
MonetDB XQuery	0.34.2-1	Monet Public License	XML in-memory database
OpenCSV	2.1	Apache License 2.0	CSV library
SLF4J	1.5.2	MIT license	logging facade
Software that is part of the prototypes or the example workflow			
Apache Commons Math	1.2	Apache License 2.0	mathematical calculations in the example workflow
ChemShell	3.2	Proprietary	ChemShell use case
DUNE	1.2.1	GPL	finite volume use case
Software used during the development			
Apache Ant	1.7.1	Apache License 2.0	build scripts
Apache Axis2	1.5.1	Apache License 2.0	Web Service generation tools
Eclipse BPEL designer	0.4.0	EPL 1.0	visual BPEL editor
Eclipse Java EE	Galileo	EPL 1.0	integrated development environment
Eclipse SQL Explorer	3.5.1	LGPL	SQL development environment
JiBX	1.2.1	JiBX License, BCEL License, XPP3 License	Java XML mapping generation
JUnit	4.6	CPL 1.0	test case development and execution
MayaStudios GeneralDevTools	rev. 99	GPL	Apache Ant extensions
SQuirreL SQL Client	3.0.3	LGPL	SQL development environment

Table 6-i: Software required by the database architecture and used during its development

Moreover, the integrated database architecture is dependent on a Sun Java Runtime Envrionment (JRE) at least version 1.5 (Binary Code License, short BCL).

6.2 Installation and Configuration of the Environment

This section covers all important details about the realization of a runtime environment for simulation workflows based on Apache ODE and the integrated database architecture. For the

integrated database architecture, the recommended database systems are applied (cf. Section 5.5 on page 128). A ready-to-use version of this environment is distributed with this thesis using a virtual machine (see Section 6.7 on page 149).

The operating system of the distributed runtime environment is a Debian Linux. The following installation and configuration descriptions are specific to this operating system. However all software components that are part of the database architecture can be ported to other operating systems, e. g., Windows. Hence, the following descriptions can be seen as an installation plan and they are transferable to future installations of the runtime environment. In the following paragraphs, installation paths are mentioned that refer to this virtual machine. The following listing gives an overview of paths that are mentioned.

```
+ /media/idares-prototype
| + datastore
| | + idares
| | + monetSQL
| | + monetXQuery
| | + ode-model
| + deployed-processes
| + deployed-services
| + tomcat-ode
| | + bin/catalina.sh
| | + conf
| | | + context.xml
| | | + log4j.xml
| | + lib
| | + logs
| | + webapps
| |   + axis2
| |   + ode/WEB-INF
| |     + conf
| |     + lib
| + WebServiceInterface
|   + archives
|   + instances
|   + export
+ /opt/ibm/db2/V9.7
+ /usr/lib
  + MonetDB4
  + MonetDB5
```

A required part of the workflow environment is Apache Tomcat that serves as web application container for Apache Axis2 and Apache ODE. The default binaray distribution of Apache Tomcat 6 is installed in `/media/idares-prototype/tomcat-ode`. Apache ODE is installed as web application `ode` and Apache Axis2 is installed as web application `axis2`. We decide to install a separate Axis2 web application for two reasons. First, [Les09] stated that although Apache ODE is based in the core on Axis2, it should not be used to deploy web service server skeletons there. Second, we use Apache Axis2 in version 1.5 for our server and client development of Web

Services. Unfortunately, Apache ODE is based on Axis2 1.4. Hence, our Web Service skeletons can not be deployed in Apache ODE for compatibility reasons.

The configurations of Apache Tomcat, Axis2, and ODE are customized the following way. A password is given to the tomcat *admin* and *manager* user in `conf/tomcat-users` (for logins and passwords see Section 6.7 on page 149). The same way, the default password is altered for the *admin* of both Axis2 and ODE for security reasons. The startup script `catalina.sh` of Apache Tomcat is altered and the following lines are added:

```
1 JAVA_OPTS="$JAVA_OPTS -Xms128M -Xmx500M"
2
3 JAVA_OPTS="$JAVA_OPTS -Dde.simtech.wsi.prefs.dir.archives=/media/
    idares-prototype/WebServiceInterface/archives"
4 JAVA_OPTS="$JAVA_OPTS -Dde.simtech.wsi.prefs.dir.instances=/
    media/idares-prototype/WebServiceInterface/instances"
5 JAVA_OPTS="$JAVA_OPTS -Dde.simtech.wsi.prefs.dir.export=/media/
    idares-prototype/WebServiceInterface/exports"
6
7 JAVA_OPTS="$JAVA_OPTS -Dlog4j.configuration=file://
    $CATALINA_BASE/conf/log4j.xml"
```

The first line increases the memory limit of the virtual machine for Apache Tomcat. Since both Axis2 and ODE are deployed, this higher memory limit avoids `OutOfMemoryErrors`. The second, third, and fourth line define required properties for the Web Service Interface of [Rut09].

The fifth entry of the `catalina.sh` listing concerns the logging framework. We use Apache log4j[1] to handle the entire logging within the *IDARES Web Services*. Moreover, Axis2 and ODE rely on log4j too. In order to make our implementation independent from log4j, we use an additional logging facade called slf4j[2]. The customized central configuration file `log4j.xml` is used instead of web application specific `log4j.properties` files, which Axis2 and ODE use per default. Our approach allows us to customize the form of logging entries centrally, decide which components are how verbose, and delegate the logging entries of different components to different files. The sophisticated management of logging entries is important in a multi-component software and extremely beneficial when locating errors. The log files are stored in the sub folder `tomcat-ode/logs`.

For Apache Axis2 and Apache ODE, we further have to reconfigure timeout settings. This is due to the fact that the default configuration of Axis2 prohibits long running ($> 30s$) synchronous Web Service calls. Although long running Web Service calls can also be implemented asynchronously, we do not want to force this time restriction. The time configurations are placed into `tomcat-ode/webapps/ode/WEB-INF/conf/ode.endpoint`. The folders for *hot deployment* are `/media/idares-prototype/deployed-processes` for BPEL↗ processes and `/media/idares-prototype/deployed-services` for Web Service skeletons.

[1] `http://logging.apache.org/log4j/index.html`
[2] `http://www.slf4j.org/`

Some JAR↗ files have to be added or removed from the classpath to allow the integration of necessary Java libraries. The `log4j.jar` is removed from both Axis2 and ODE and added to the `libs` folder of Tomcat. This is due to the fact that we want to use log4j with a central configuration. Five Hibernate JAR files are added to the `lib` directory of ODE: `hibernate3.jar`, `ehcache-1.2.3.jar`, `cglib-2.1.3.jar`, `dom4j-1.6.1.jar`, `asm.jar`. This is done to allow the usage of the Hibernate implementation of the ODE DAO layer for IBM DB2 as *ODE model db*. Finally, we add JAR files with the JDBC drivers for IBM DB2 V9.7 (`db2jcc.jar`, `db2jcc_license_cu.jar`) and MonetDB into the `lib` folder of Apache Tomcat.

The database systems are installed as described in the evaluation chapter (see Section 5.4.3 on page 109). DB2 is installed in the default path `/opt/ibm/db2/V9.7`. Both versions of MonetDB are installed with the package manager to `/usr/lib/MonetDB5`, respectively `/usr/lib/MonetDB4`. The path `/media/idares-prototype/datastore` serves as a parent folder for all database data.

For IBM DB2 we create a database *IDARES* that serves as storage for relational and XML *application data* (cf. Section 3.1 on page 48) and a database *ODE* for the *ODE model db*. In order to derive a database schema for the *ODE model db*, using DB2 as data storage and Hibernate as DAO layer, we utilize the Hibernate tool `hbm2ddl`. This tool uses Hibernate Mapping Files (HBM) to generate table definition statement including primary and foreign keys. The invocation script of the tool can be found in `[DVD]/Sources/Tools/HibernateGen/build-hibernate.xml`, the resulting database schema in `[DVD]/Sources/SQL/ODE/ODE-DB2-hibernate.ddl.sql` and `ODE-DB2-job.ddl.sql` in the same folder. The latter defines a table where ODE stores its jobs without using a DAO, but rather accessing the database via JDBC directly.

In order to respect ℝ **27** on page 51, a mechanism has to be provided to delete *unnecessary* runtime information from the *ODE model db*. All runtime information in the *ODE model db* are related to a an entry in the table `ODE.BPEL_INSTANCE`. Since only the scientist knows which instances are not needed anymore, only he or she can decide which runtime information are *unnecessary*. `[DVD]/Sources/SQL/ODE/Terminated_Instance_Cleanup.sql` contains a template for a database script that clears data of all terminated workflow instances from the *ODE model db*. The scientist can use the *DB2 Control Center*, a customized BPEL workflow, or an SQL tool of his choice to execute this script against the DB2 database *ODE*.

All databases are configured as *DataSources* in `tomcat-ode/conf/context.xml`. The *DataSources* define a JDBC connection URL, name and password for the user, and additional properties for the connection pool that Tomcat provides. We distinguish the following *DataSource*s by their purpose:

- *jdbc/OdeDb*: *ODE model db* on IBM DB2
- *jdbc/IDARES*: relational and XML *application data* on IBM DB2
- *jdbc/MemorySQL*: in-memory relational data on MonetDB SQL
- *jdbc/MemoryXQuery*: in-memory XML data on MonetDB XQuery

There is a downside of the connection pool technique that has to be mentioned at this point. As the connection pool caches JDBC connections for multiple reuse, these objects seem to be usable to the connection pool even if the related database process has been shutdown or is crashed. For this reason, a restart of such a database process is not propagated to the pooled connections. When the next Web Service attempts to use such an object, an `SQLException` is thrown, because the connection object points to a non-existent database process. Although the connection is immediately discarded from the connection pool, the other faulty connections reside until all of them are attempted to access. This issue is accepted, since the implemented environment will be replaced by the SIMPL framework in future.

Finally, we configure the databases and Apache Tomcat to be automatically started on system reboot. Apart from additional Web Services (cf. the following sections) the runtime environment is ready to execute simulation workflows.

6.3 IDARES Web Services

This section describes the realization of the *IDARES Web Services*, introduced as a component of the integrated database architecture (see Section 4.2 on page 72). These Web Services are deployed within the Axis2 web application service folder as an `AAR` file. The Web Services realize multiple functional requirements from the Chapter 3, mostly in the context of database related operations. In addition, the *IDARES Web Services* provide necessary methods for the database evaluation and the prototypes.

The definition of the XML schema and the WSDL↗ file can be found in Appendix A.4 on page 174. The XML namespace of the Web Services is `http://ahija.de/thesis/idares/webservices` (prefix `iws`). The XML namespace of the related XML schema is `http://ahija.de/thesis/idares/model` (prefix `im`). In this section, we discuss the operations and mention the realized requirements.

Before describing the *IDARES Web Services* one by one, we give a short overview first:

- *GenericDatabaseService* provides all database access operations required by IDARES.
- *DBApplicationSimulationPortType* is a port type that is implemented by three Web Services and is used for the evaluation of concurrent database activities.
- *WSHelperService* is a utility Web Service for debug information, which is exploited by the *IdaresDBEvaluation* workflow.
- *DBEvaluationService* is the Web Service definition that is implemented by the *IdaresDBEvaluation* workflow.

6.3.1 GenericDatabaseService

This Web Service is realized as a Java class, whose implementation can be found in `[DVD]/Sources/Projects/Idares/IdaresWS/src/de/ahija/thesis/idares/skeleton/GenericDatabaseServiceSkeleton.java`. The Axis2 `wsdl2java` generator is used to generate the Java server skeleton based on the WSDL port. The generated interface contains a Java method for each WSDL operation. The class `GenericDatabaseServiceSkeleton` implements this interface and hence all methods.

The Java methods declare Java exceptions that represent WSDL faults. Every Java method therefore ensures that all possible exceptions are caught and wrapped into the declared exceptions. Axis2 honors this approach by sending the exception bodies as SOAP↗ faults back to the client. Otherwise, Axis2 would complain about *undeclared exceptions* and return a generic `Axis2Fault` that may cause problems at Web Service clients.

All methods realize an access to a database and have to establish a connection. As already described in the prototype architecture (see Section 4.3 on page 76), we apply the connection pool of Apache Tomcat, which is accessible via *DataSource* objects. For this reason all database requests contain a name for a *DataSource*. One of the first activities of every method in the *GenericDatabaseService* is to resolve a *DataSource* for a given name. In order to minimize the effort for lookups, we cache the retrieved *DataSource* objects. Every method realizes a transactional scope around the queries to execute. Hence, we create JDBC connections with `setAutoCommit(false)`. Moreover, we force a rollback of the database transaction on any error that is caught. JDBC objects have to be closed in order to release database resources and add a connection back to the pool of currently idle connections. This is ensured by closing all `ResultSet`, `Statement`, and `Connection` objects in a Java `finally` block. The general structure of a method is depicted in the following source code template.

```
1 public Response executeViaDataSource(Request request) throws
     SqlErrorMessage {
2   Connection connection = null;
3   Statement statement = null;
4   ResultSet resultSet = null;
5   try {
6     connection =
          getJDBCConnectionPerDataSource(request.getDatasourceName(),
          request.getIsolation());
7     statement = connection.createStatement();
8     resultSet = statement.executeQuery("SELECT FROM WHERE");
9     // handle resultSet
10    connection.commit();
11    Response response = new Response();
12    // fill response
13    return response;
14  } catch (Throwable t) {
15    // force a rollback
16    try {
```

```
17        if (connection != null && !connection.isClosed()) {
              connection.rollback(); }
18      } catch (SQLException e) {}
19      throw new SqlErrorMessage("error in executeViaDataSource", e);
20    } finally {
21      try {
22        if (resultSet != null) { resultSet.close(); }
23      } catch (SQLException e) {}
24      try {
25        if (statement != null) { statement.close(); }
26      } catch (SQLException e) {}
27      try {
28        if (connection != null && !connection.isClosed()) {
              connection.close(); }
29      } catch (SQLException e) {}
30    }
31  }
```

All operations of this Web Service rely on a created type system that combines database types, Java types, and XML schema types. We use this type system to parse values from XML requests into Java objects and then set database values via JDBC driver objects. The core library of Hibernate[3] is utilized to choose the appropriate parametrization method of a JDBC statement for each type. The preamble of the XML schema file for the *IDARES Web Services* lists all supported types (see Section A.4.1 on page 174).

The request elements of the *GenericDatabaseService* are often containing at least on sub-element of the defined type `im:queryType`. This type stands for a database query and defines:

- the query language: `SQL`, `XQUERY`, or `XMLDB`
- the query type: `SELECT`, `UPDATE`, `INSERT`, `DELETE`, `DDL`, or `OTHER`
- the actual query with optional `?` signs as placeholders
- parameter bindings that define values for the placeholder; thereby, we can respect the actual type that is required for a placeholder

The methods of the class `GenericDatabaseServiceSkeleton` stop the time required to execute queries and return these information in their response. In the following paragraphs, we discuss the methods of the class `GenericDatabaseServiceSkeleton` individually.

executeQueryViaDataSource executes an SQL query via JDBC or an XQuery statement via JDBC or XML:DB. In addition, multiple preparation and cleanup statements are optional before and after a focused query. The focused query is expected to return a result set. For SQL this may be a `SELECT` statement. This relational based result is serialized into a well defined XML response.

[3] `https://www.hibernate.org/`

The response contains descriptions of the column names and the column data types. An example response element of the operation is shown in the following listing.

```
1 <im:executeQueryViaDataSourceResponse
      xmlns:im="http://ahija.de/thesis/idares/model"
      selectExecutionDuration="359" overallExecutionDuration="359"
      rowCount="1">
2   <im:headerRow>
3     <im:columnDescriptor columnName="SUITEID" dataTypeName="long" />
4     <im:columnDescriptor columnName="NAME" dataTypeName="string" />
5   </im:headerRow>
6   <im:row rowNumber="0">
7     <SUITEID>3</SUITEID>
8     <NAME>XMARK Suite</NAME>
9   </im:row>
10 </im:executeQueryViaDataSourceResponse>
```

If the result of the database retrieval is a sequence of XML elements, these elements are returned as sub-elements of `im:row`. The operation furthermore allows the specification of an optional request parameter `maxRows` that defines how many rows are returned at maximum. This operation was required by ℝ **7** on page 48.

executeViaDataSource is very similar to the operation before, but expects no result set of the focused query. Instead, this method is used for data manipulation or data definition statements. The operation returns the number of modified or deleted rows, if the JDBC driver supports this feature. This operation was required by ℝ **7** on page 48.

executeInsertViaDataSource is a specialized operation that is solely intended for SQL `INSERT` statements. It is utilized in the special case where the insert of a new relational row causes the creation of a primary key or another generated column value. The `executeInsertViaDataSource` operation returns the values of such generated columns in its response. This feature is especially used by the *IdaresDBEvaluation* workflow, which lets the database create an identifier for a new `RUN` and later uses the generated `RUNID` to associate `RESULT`s with this `RUN`.

measurePerformanceViaDataSource is an operation that is highly exploited by the *IdaresDB-Evaluation* workflow. In contrast to the other operations this method does not return database results in its response. Instead, it executes all requested queries and takes the time. If the query returns a result, the result is totally retrieved. Thereby database is forced to actually return the entire result and not only the first rows. Like the other operations, this method can process both SQL and XQuery requests.

copyContentViaDataSource is an operation that copies objects from one database to another. To achieve this, this operation requires a retrieval query for the source side and an insert query for the target side. The implementation consecutively retrieves a subset of source objects, e. g., 1,000 rows, into a buffer and then inserts the subset in the target database within the scope of a transaction. This approach to use intermediate commits for large insert activities is a commonly applied method when copying database objects. The important advantage is to avoid a long running transactions that block the target database and are therefore vulnerable to be rolled back. In order to fix a bug of MonetDB SQL, the insertion into the target database is retried when an insertion error occurs. Unfortunately, this was the only possible solution to insert large amounts of rows into MonetDB SQL.

The operation is required by ℝ **8** on page 48 and can also be used to synchronize data between a remote database and a local cache (cf. ℝ **38** on page 64).

importCSVToDataSource is used to import a CSV↗ based file into a relational table. The operation supports three options to insert the data. The first option is to specify a table name only. In this case, the operation expects the CSV file to have the same columns in same order like the table. The second option extends the first option by adding column names in the request. Hereby, columns can be reordered or even omitted. The third option is the most flexible one. The request contains an SQL `INSERT` script with `?` signs as placeholders. The operation uses the values parsed from the CSV file to set the values for placeholders.

To actual parse CSV files and split each line into tokens, we apply the open source library OpenCSV[4]. The operation `importCSVToDataSource` can utilize this library to support some options that address idiosyncrasies of CSV files. These options cover the definition of the separator character, a quotation character, an escape character, and a value that is interpreted as an SQL `NULL`. Another boolean option specifies that the first row contains a column header and is to be skipped. A sophisticated option lets the parser treat multiple subsequent separator chars as a single one. This was necessary due to the CSV files that are provided for the finite elements of the CO_2 example (cf. [CDR+07]), which contain multiple tabulators for visual alignment. The operation supports different ways to specify the CSV file: by an arbitrary URL or by attaching the content of the CSV file as text or binary. Like the operation `copyContentViaDataSource`, the insert of the rows in the target table are split into small subsets, which are each committed in a small transaction.

The CSV import feature was required by ℝ **9** on page 48.

exportFromDataSourceToCSV is the counterpart to the CSV import operation. It provides similar options and can export the relational result set of a database query into a CSV based file. The target of the file is specified by an URL and can therefore be a local file or for example an FTP↗ path. This operation is also required by ℝ **9** on page 48.

[4] `http://opencsv.sourceforge.net/`

exportFromDataSourceToFile is a utility operation to export a single cell from a result set of a relational or XML query into a file. The request message for this operation therefore requires a query that is expected to retrieve a result set. From the result set only the first column of the first row is taken. If the result set is empty, an empty target file is created. The path of the file is specified by an URL. This method is mainly intended to export BLOB↗ and CLOB↗ values into files, but can be used to export XML documents too.

6.3.2 DBApplicationSimulationPortType

This port type is realized by three Web Services that are implemented as Java classes. The sole operation `simulateApplications` is used to simulate concurrent access to a database. Beside a name of the *DataSource*, the request message specifies the number of concurrent threads and a so called *seed* value. The three Java classes are implemented dynamically in the way that they specify the manipulation queries statically but vary the predicates for the objects to manipulate. The following listing shows such a query.

```
1 UPDATE HOSPITAL.PATIENT p
2    SET p.P_ADDRESS = ?
3  WHERE p.p_PATIENTKEY BETWEEN ? AND ?
```

To actually set values for the predicates, a Java random generator is utilized. Since the Web Services are used for performance evaluation, the generated numbers have to be static for a given evaluation test. At this point, we apply the *seed*, which is used to initialize the random generator once. The random generator generates the same sequence of numbers, when initialized with the same *seed*. The three Web Services are:

- `TPCHCustomerReadSimulation` simulates concurrent reads on the table `HOSPITAL.CUSTOMER` by primary and foreign key and a non-indexed column. This Web Service is implemented by `[DVD]/Sources/Projects/Idares/IdaresWS/src/de/ahija/thesis/idares/skeleton/TPCHCustomerReadSimulationSkeleton.java`.
- `TPCHPatientWriteSimulation` simulates concurrent reads, insertions, modifications, and deletions on the table `HOSPITAL.PATIENT`. This Web Service is implemented by `[DVD]/Sources/Projects/Idares/IdaresWS/src/de/ahija/thesis/idares/skeleton/TPCHPatientWriteSimulationSkeleton.java`.
- `TPCHCallInsertSimulation` simulates the concurrent insertion of multiple rows into the table `HOSPITAL.EMERGENCY_CALL`. This Web Service is implemented by `[DVD]/Sources/Projects/Idares/IdaresWS/src/de/ahija/thesis/idares/skeleton/TPCHCallInsertSimulationSkeleton.java`.

6.3.3 WSHelperService

is a utility Web Service with the single operation `printXML`. The sole purpose of this operation is to send debug information to a `Logger` of the logging framework log4j. The operation

is used to print the actual state of ODE BPEL variables to the shell. The operation request a non empty sequence of XML elements and an optional argument that specifies the name of the log4j `Logger`. The latter argument is useful to direct debug information in a specific log file and ease the process of finding such log entries. As Apache ODE has trouble to process some XPath expressions, this operation is a pragmatic way to get to know what is actually stored in a BPEL variable. The implementation of this Web Service can be found in `[DVD]/Sources/Projects/Idares/IdaresWS/src/de/ahija/thesis/idares/skeleton/WSHelperServiceSkeleton.java`.

6.3.4 DBEvaluationService

is the Web Service that is implemented by the *IdaresDBEvaluation* workflow (cf. the description of the evaluation infrastructure in Section 5.2 on page 91).

6.4 Extension of the Web Service Interface

The Web Service Interface developed by [Rut09] is deployed and further extended in this thesis. The Web Service Interface is utilized by the prototypes to orchestrate the steps of a Dune or ChemShell simulation as a BPEL workflow. We extend the Web Service Interface with operations to handle typical data provisioning task in its context. As described for Scenario I and its two use cases (see Section 3.4.1 on page 56), we want to store input and output of such simulations in the database.

Before starting with the extension of the Web Service Interface, we import its source code into the same development environment used to develop the *IDARES Web Services*. This involves the transformation of the file encoding of the files to UTF-8 (8-bit UCS/Unicode Transformation Format) that supports any character of the Unicode standard and the definition of build scripts to automate the Axis2 code generation. Moreover, some inadequacies are fixed – e. g., all Web Service skeleton classes where named `Service`. This often leads to confusions and was tackled by naming the skeleton classes like their implemented Service, e. g., `WSIServiceSkeleton`. Furthermore, the logging statements were used to rely on slf4j too. The original version of the Web Service Interface also contained two utility Servlets for the web application *WSIConfigurer*. This web application is discarded, since its functionality is not needed anymore.

A first step of the extension is the definition of a table that keeps generic simulation *application data*, such as input and output files as CLOBs↗. A specification of a table for this purpose is presented in the following listing.

```
CREATE TABLE PROTOTYPE.SIMULATION_FILE (
  FILE_ID     BIGINT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
  PROJECT     VARCHAR(255) NOT NULL,
  FILENAME    VARCHAR(255) NOT NULL,
  FILETYPE    VARCHAR(32) NOT NULL DEFAULT 'CLOB',
  SUBID       INTEGER NOT NULL DEFAULT 1,
```

```
    VERSION     INTEGER NOT NULL DEFAULT 1,
    TS          TIMESTAMP NOT NULL DEFAULT CURRENT TIMESTAMP,
    ATTACHMENT XML,
    CONTENT     CLOB
  );
```

We store the table SIMULATION_FILE in the *DataSource IDARES*. The table contains a FILE_ID as primary key, which is automatically generated when inserting a new entry. Additionally an entry in this table can be assigned to a PROJECT (e. g., a simulation project), can have a FILENAME within this project, and has a FILETYPE. The FILETYPE specifies whether the content of this file is stored as a CLOB or is stored in specialized tables. The subsequent sections cover details about supported FILETYPEs for the prototype. Furthermore, a SIMULATION_FILE can specify a SUBID and a VERSION. Using these columns, it can be indicated that multiple SIMULATION_FILE entries with the same FILENAME are related. This feature is, for example, used to store multiple consecutive intermediate simulation results. An XML based ATTACHMENT can be used to store additional information or annotations as demanded by ℝ **32** on page 54.

In order to load data from a file into the database or store a content from the database into a file, two additional operations are added to the specification of the WebServiceInterface.wsdl of the Web Service Interface. The details of the adaptations in the XML schema and the WSDL file are shown in Appendix A.5 on page 197.

loadFileIntoDB requests the specification of a file relative to a simulation instance or an URL as input. Moreover, the FILETYPE has to be specified and an optional file encoding set can be stated. The operation retrieves the file and either store it as a CLOB (FILETYPE = CLOB) into the SIMULATION_FILE or use a FILETYPE specific handler to parse the content. The sole response of this operation is the generated FILE_ID of the loaded file. This FILE_ID can be used as a pointer in subsequent workflow activities.

storeFileFromDB requires information to address an entry in the table SIMULATION_FILE as its request message. This can either be its FILE_ID or a combination of PROJECT, FILENAME, etc. If multiple SIMULATION_FILE entries qualify for these predicates, the SIMULATION_FILE with the latest FILE_ID is used. The operation loads the SIMULATION_FILE from the CLOB value in the database or serializes its content from a relational representation. The target of this operation can either be a file path relative to a simulation instance or an URL. The operation has an empty response.

The two additional operations are implemented in the Web Service skeleton file [DVD]/Sources/Projects/WebServiceInterface/WebServiceInterface/src/de/simtech/wsi/ws/WSIServiceSkeleton.java. A part of the functionality relies on the already implemented GenericDatabaseService, while the core functionality is implemented independently. This includes the database access for the storage, or respectively retrieval, of a SIMULATION_FILE entry. This fact can be justified with the tight integration of FILETYPE specific handlers. Common functionality, such as the access of the columns PROJECT and FILENAME is done in the operations.

But the actual handling of the content is abstracted by an interface shown in the following listing.

```
1 public interface FileHandler {
2   public void loadFileToDatabase(long fileID, Reader fileReader,
        Connection databaseConnection) throws Exception;
3
4   public void storeFileFromDatabase(long fileID, Writer fileWriter,
        Connection databaseConnection) throws Exception;
5 }
```

An implementation of the method `FileHandler.loadFileToDatabase` reads the content of a file and stores a representation into the database. This can be either realized by using the CLOB column `CONTENT` of the table `SIMULATION_FILE` or by using a specialized table. The method `FileHandler.storeFileFromDatabase` has to write a file whose content is located in the database. Since a specific implementation knows how it has loaded a file into the database, it should find appropriate ways to serialize the content again and write it into a file. For both methods, we honor the specified file encoding and use the `Reader/Writer` API of Java rather than byte oriented `InputStream/OutputStream` API.

For the prototypes we implemented the following file handlers whose implementation can be found in `[DVD]/Sources/Projects/WebServiceInterface/WebServiceInterface/src/de/simtech/wsi/filehandler`:

- `ClobHandler` is a default implementation for the `FILETYPE CLOB` that simply stores the content of a file as a CLOB.
- `VTUHandler` is an implementation for the `FILETYPE VTU` that handles a finite element grid descriptions in the VTU format of the Visualization Toolkit. This handler is required by the prototype for the Dune Finite Volume.
- `PDBHandler` is an implementation for the `FILETYPE PDB` that handles Protein DataBank files. This handler is required by the prototype for the ChemShell example.
- `MoleculeHandler` is an implementation for the `FILETYPE MOLECULE` that handles molecule description files, like `dl-find.result`. This handler is required by the prototype for the ChemShell example.

6.5 Development notes

This section briefly covers some notes about the development environment. It is addressed to developers who want to use and/or extend the current implementation of this thesis.

Eclipse Java EE Galileo is used as integrated development environment. Hence, all sources are located in appropriate Eclipse projects. The projects can be found in the sub folders of `[DVD]/Sources/Projects`. They can be simply imported using the Eclipse project import wizard. All source code files use UTF-8↗ as file encoding to avoid conflicts with potential umlauts or other

international characters. The Eclipse projects are configured to automatically use UTF-8 as their encoding. The project *Libraries* contains all JAR↗ files of dependent libraries, each including a `version.txt` with information of the specific version of a library.

Apache Ant is used to manage automatic code generation and compilation. For example to create the Axis2 archive that contains all Web Services of this thesis, the `[DVD]/Sources/Projects/Idares/IdaresWSIArchive/build-manual.xml` can be called by Apache Ant. The result is the `IdaresWsiServices.aar` file in the sub folder `dist`.

A common configuration file for all Apache Ant scripts is located in `[DVD]/Sources/properties-main.xml`. It contains references to folders and common classpath definitions. The file `[DVD]/Sources/build-personal.properties` contains developer specific properties, such as an installation path of Apache Tomcat. When this path is specified, the Ant script in the project *IdaresWSIArchive* directly deploys the created archive into the Web Service hot deployment directory of Axis2. To achieve this, the build script has to be called with the following command line:

```
ant -f build-manual.xml clean-deploy
```

6.6 Tests and Prototypes

This section covers a brief overview of tests and prototypical applications of the developed integrated database architecture.

6.6.1 Testing

Functional tests are applied at two levels. For classes that contain complex implementations, we apply Unit Testing at class level (cf. [MS04]). We utilize the popular Unit Testing framework JUnit[5] to define and run the test cases. This approach is, for example, used to test the various `DataHandlers`.

In order to test the functionality of the developed Web Services, we deploy the archived compilation of the Web Services into the `services` folder of Apache Axis2. Furthermore, we prepare Web Service stubs with the Axis2 generation tools and design test cases with JUnit. This approach is more complicated than using a a specialized Web Service tool like soapUI[6], but we can manage all test cases in a common structure. Moreover, the test cases are implemented in Java and therefore are more flexible than their counterparts in soapUI would be.

All test cases are collected in the Eclipse project `[DVD]/Sources/Projects/Tests` and respect the Java package and name of their test classes. For example, the test cases for

[5] `http://www.junit.org/`
[6] `http://www.soapui.org/`

the class `de.ahija.thesis.idares.types.IdaresTypeHelper` are implemented in the class `de.ahija.thesis.idares.types.IdaresTypeHelperTest`. This is a conventional approach, since it simplifies the management of test cases.

In addition to the Unit Tests, the following prototypes can be seen as acceptance tests that prove the applicability of the integrated database architecture and its advantages.

6.6.2 Dune Finite Volume Prototype

The first use case of Scenario I (see Section 3.4.1 on page 59) is the Finite Volume example using the Dune framework. This first example is used to demonstrate the application of the integrated database architecture. Therefore, we implement a prototypical extensions to the BPEL workflow *finitevolume_interactive* of [Rut09]. We call the resulting workflow *DunePrototypeProcess*.

The first extension of the workflow concerns a data provisioning activity that supplies the initial grid file `unitcube.dgf` – the core input of the simulation. This file is kept as a CLOB in the table `SIMULATION_FILE`. The data provisioning activity in the workflow calls the operation `storeFileFromDB` of the Web Service Interface after the instance directory is created and before the actual simulation is initiated.

The second extension concerns a set of activities that load the results of the simulation into the database. These are 21 files containing finite element grids in the `VTU` format of the Visualization Toolkit. The operation `loadFileIntoDB` is called in a BPEL `forEach` loop with activated parallel option. A specific `VTUHandler` is responsible to parse `VTU` files and store the results in two specialized tables[7]. The table `GRID_POINTS` contains the points of the finite element grid and `GRID_CELLS` contains the finite elements as references for exactly eight points. Additionally, for each element the simulated concentration is stated.

The interested reader can find the workflow in `[DVD]/Sources/Projects/WebServiceInterface/WSIResources/DunePrototypeProcess.bpel`. Additionally, a preparatory workflow was implemented to load the initial grid file into the database: `PrepareDunePrototype.bpel`. The table definition statements can be found in `[DVD]/Sources/SQL/Prototype/Prototype.ddl.sql`.

6.6.3 ChemShell Example Prototype

The second use case of Scenario I (see Section 3.4.1 on page 60) is the example workflow *ChemshellPrototypeProcess* that orchestrates commands of the ChemShell workbench. It originates from the BPEL workflow *chemshell_interactive* of [Rut09].

As stated in the use case description, all five input files of the simulation are kept in the database architecture. They are provided by preparatory activities in the workflow – each activity utilizing

[7] [Mod09] describes the parts of the XML based `VTU` file and their purpose.

the operation `storeFileFromDB`. For the PDB↗ file, the specific table `PDB_FILE` is used to store the information of the PDB file. The `PDBHandler` manages the conversion from the file based to the relational format and back again[8]. At this point, it is crucial to mention that the `PDBHandler` is just build to demonstrate how the database architecture can be used to store an ATOM description in a relational table. It does not support other variation of ATOM records or other record types that are usually supported by the PDB file format.

After the workflow is finished, the main result – the final state of the molecule in file `dl-find.result` – is loaded into the database by an invoke activity that calls the operation `loadFileIntoDB`. The `MoleculeHandler` handles the parsing of the molecule description file and stores the information about the atoms of the molecule in the table `MOLECULE_FILE` and the references between the atoms in the table `MOLECULE_REFERENCES`. Foreign keys are used to define the connections between atoms pairs.

The interested reader can find the workflow in `[DVD]/Sources/Projects/WebServiceInterface/WSIResources/ChemshellPrototypeProcess.bpel`. Additionally, a preparatory workflow was implemented to load the five initial files into the database: `PrepareChemshellPrototype.bpel`. The table definition statements can be found in `[DVD]/Sources/SQL/Prototype/Prototype.ddl.sql`.

6.6.4 Example Workflow – Genetic Algorithm

The example workflow, called *GeneticProcess*, illustrates how the various features of the integrated database architecture can be utilized (cf. ℝ **24** on page 50). It is specially implemented for this purpose and has currently no special use case. The workflow is based on BPEL↗ and various specialized Web Services. The *GeneticProcess* can be classified as a scientific workflow.

The universe of discourse behind the example workflow is the *Lottery problem*. This problem tries to answer the following question: when knowing all numbers of all tickets for a lottery drawing, which numbers have to be selected to minimize the total payout. More precise, we have 49 numbers $n \in N = \{1, \ldots, 49\}$ in the lottery, and each customer c chooses exactly 6 numbers per ticket $t \in T \subset 2^N$, where T is the set of all potential tickets and defined as $T = \{t : t \subset N \wedge |t| = 6\}$. Before the drawing, we have a multiset of bought tickets $B \subseteq T$ and a payout function $p : \{0, \ldots, 6\} \to \mathbb{R}^+$. The payout function p defines the monetary value of a ticket, based on the quantity of correctly predicted numbers. Now, we want exactly 6 winning numbers $d \in T$ to be drawn so that $\sum_{t \in B} p\left(|t \cap d|\right) \stackrel{!}{=} min$.

A naive approach would try all 13,983,816 possible variants of d, which causes a huge effort. Our solution is based on a genetic algorithm[9]. Genetic algorithms typically address maximization or minimization problems by applying techniques known from the theory of evolution. The problems are often NP-complete or otherwise not analytically solvable. Genetic algorithms do not

[8] [Pro09] describes the format of ATOM records of a PDB file.
[9] For details on genetic algorithms, the reader is referred to [Mit96]

necessarily find the best solution, but their techniques converge to the optimum sufficiently. In order to develop an genetic algorithm, individuals have to be identified that relate to possible solutions of the given problem. In most cases the individuals are vectors with numbers or just binary vectors. In our example, the individuals are the winning numbers $d_i \in T$. Each individual has a so called fitness, which represents its degree of how good it solves a problem as a decimal number. In our example, we can apply the total payout for a fixed set of bought tickets $\sum_{t \in B} p\left(|t \cap d|\right)$. The idea is now: when we find an individual d_i that has a very low fitness (minimization problem), we have found a good solution for our *Lottery problem*.

Our approach is based on the hill climbing technique, a special kind of genetic algorithms. Hill climbing randomizes an initial individual and explores the *neighborhood* of this individual until a better neighbor is found. A neighbor of an individual is typically related in some way. We consider individuals with exactly 5 identical numbers: $neighbor : T \rightarrow 2^T, neighbor(t) = \{s : s \in T \wedge |s \cap t| = 5\}$. When the hill climbing approach finds a better neighbor, the exploration goes on with this neighbor. Several exit conditions for the repetition of the neighborhood exploration are imaginably: iterate a fixed number of times (implemented), in the last x iterations no better individual is found, the fitness falls below a given threshold.

An important issue at this point is that we can not simply calculate the fitness of an individual as a scalar function. To calculate the total payout, the 6 numbers of an individual have to be compared with all numbers of all tickets of a drawing (e. g., 3,000,000 numbers for 500,000 tickets). For our example workflow, we expect the tickets to be stored in a remote relational database in a table of the following form.

```
CREATE TABLE PROTOTYPE.LOTTERY_TICKET (
  DRAWING_ID INTEGER  NOT NULL,
  TICKET_ID  INTEGER  NOT NULL,
  NUMBER1    SMALLINT NOT NULL,
  NUMBER2    SMALLINT NOT NULL,
  NUMBER3    SMALLINT NOT NULL,
  NUMBER4    SMALLINT NOT NULL,
  NUMBER5    SMALLINT NOT NULL,
  NUMBER6    SMALLINT NOT NULL
);
```

Since we consider to access these tickets a huge number of times, we apply the application field of the integrated database architecture as a local cache and provide the information of the tickets locally. We just consider one drawing and can hence use a subset of the remote database table and further adapt the information for our purposes. Taking these characteristics into account, the *GeneticProcess* represents a workflow of Scenario II (cf. Section 3.4.2 on page 61).

This preparatory step concerns the first activities of our example workflow, which utilizes the *IDARES Web Services* to provide the data locally. The logic of the hill climbing approach is realized with BPEL `forEach` loops and special Web Services that realize the neighborhood exploration or encapsulate the actual fitness calculation. The workflow exploits the `parallel` option of a `forEach` loop to calculate the fitness of competing neighbors concurrently. In order to keep track of the best individuals, they are stored into the integrated database architecture. To preserve their

structure, we store the individuals as XML elements, like they are specified. After the workflow is finished, we can investigate the individuals using simple or sophisticated XQuery statements.

The implementation of the *GeneticProcess* can be found in `[DVD]/Sources/Projects/Lottery/GeneticProcess/GeneticProcess.bpel`. Related table definitions are specified in `[DVD]/Sources/SQL/Lottery`. A set of 500,000 lottery tickets was loaded into the local database and is ready to be used for runs of the *GeneticProcess*. The workflow *PrepareGeneticProcess* can be used to load the tickets into the local cache once. We transfer this data provisioning step into a separate workflow so that multiple runs of the *GeneticProcess* are possible without the requirement to copy 500,000 records each time. A simple SQL `DELETE` statement is sufficient to clear the local cache totally in order to fulfill $\mathbb{R}$ **37** on page 64.

A special variant of the example workflow is applied for the evaluation of the *ODE model db* (cf. Criterion 11 on page 89). This variant does not actually calculate the fitness, but rather randomizes the values. The reason therefore lies in the purpose of the evaluation. The criterion investigates the performance of the *ODE model db*, and not of the database of the local cache. Hence, it is important to avoid potential interference of the database for the evaluation setup.

6.7 Runtime Environment and its Virtualization

Section 6.2 on page 132 already introduced the runtime environment and the installed components. This section covers background information and facts about the virtual machine where the runtime environment is installed on. These information are useful when an interested reader actually wants the runtime environment to get going.

6.7.1 Remarks about the Virtualization Software

The runtime environment for simulation workflows is developed on a virtual machine. At the first attempt, the virtualization software Sun VirtualBox[10] was used. Virtual Box is a straightforward, easy to install, and yet powerful virtualization environment for a vast number of operating systems. Virtual Box has an open source license and a large community. Nevertheless, Virtual Box caused a number of critical problems. On the one hand, Virtual Box was installed on a 32 bit host system and hence only supported 1.5 GB RAM for guest systems. As we intended to evaluate in-memory databases, this limitation was not acceptable. On the other hand, Virtual Box seemed to have problems with timekeeping in the virtual system. When the guest system was highly utilized because of the performance tests, the clock of the guest lost its synchronization. Performance evaluation results with negative durations or much to long durations were the result.

[10] `http://www.virtualbox.org/`

For these reasons, the virtual machine was ported to VMWare Server 2.0.2[11]. The VMWare company provides virtualization software for small and large scale environments. Besides multiple commercial products, VMWare offers the free virtualization platform VMWare Server. VMWare Server supports multiple host and guest operating systems, as well as various system architectures. VMWare Server did not show any signs of the issues described for VirtualBox above. [VMW08] describe aspects that help a VMWare guest system with timekeeping. A software installed on the guest system – VMWare tools – can directly communicate with the virtualization environment and synchronizes the virtual clock of the guest system. To achieve that, the virtual machine was modified via the *VMware Infrastructure Web Access* and the property `tools.syncTime` was set to `true`.

The export mechanism of VirtualBox and the VMWare vCenter Converter made it possible to migrate the already created virtual machine and the virtual hard drives from the VirtualBox format `VDI` to the VMWare format `VMDK`. Nevertheless, the migration did not succeed at the first attempt and required much patience, as the conversion of the virtual drives took hours.

6.7.2 The Virtual Machine

The name of the virtual machine is *Diadumenian 2*[12]. It has the following virtual components for the evaluation and the distribution version:

Processor	Intel Core 2 Quad Q9300 @ 2.5 GHz, two virtual cores
Memory	2.7 GB RAM
Operating System	Debian 5.0.3, kernel: 2.6.26-2-686, window manager KDE
Hard Disk 1 (IDE 0:0)	30.00 GB without preallocation; contains the root file node
Hard Disk 2 (IDE 0:1)	30.00 GB without preallocation; mounted to `/home`
Hard Disk 3 (IDE 1:1)	2.00 GB for swap space
Hard Disk 4 (SCSI 0:0)	30.00 GB without preallocation; mounted to `/media/idares-evaluation`
Hard Disk 5 (SCSI 0:1)	30.00 GB without preallocation; mounted to `/media/idares-prototype`

The main memory setting of 2.7 GB was a compromise: it was estimated as the highest value the host system of the evaluation would tolerate without having to swap virtual memory. The virtual processor and the amount of virtual memory directly depend on the host system and can be adjusted when used on another computer than the one used for the evaluation.

Debian Linux is installed on the virtual guest machine as operating system. Instead of a slim version, we choose to install all default packages of Debian, including KDE[13] as window manager,

[11] `http://www.vmware.com/products/server/`
[12] Marcus Opellius Antoninus Diadumenianus (208–218 A. D.) was a Caesar in the Roman Empire.
[13] `http://www.kde.org/`

which provides a graphical user interface. Hence, administration and installation tasks are more simple to achieve and the graphical *DB2 Control Center* is usable. We choose a 32 bit version of Debian, because this way both 32 bit and 64 bit host systems can use the virtual machine.

Apart from the local login, the operating system provides a remote shell via the Secure Shell (SSH) protocol on port 22. SSH is especially useful to forward ports of databases for remote management. The logins and passwords for the virtual machine are stated in `[DVD]/readme.html`. The workflow runtime environment and all databases are configured to automatically start on system boot.

6.7.3 Export and Import

After the virtual machine has been used on the computer provided for the evaluation, the virtual hard disks had a total size of about 38.8 GB. This is due to the fact that VMWare does not know which sections of the virtual hard disks are currently in use and hence can not compact the files. [Tim08] suggests an approach that we applied. The guest system is instructed to fill all free space of all drives with zeros. Afterwards, the VMWare *vdiskmanager* is used to compact the files representing the virtual hard disks. This is now possible, since the *vdiskmanager* simply ignores large sections with zeros. In addition to that, the resulting hard disk files are additionally compressed with the archive format 7z[14]. 7z provides an extremely high compression ratio and hence the whole virtual machine easily fits on a 4.3 GB DVD.

To import the virtual machine on a hosting system, the archive `[DVD]/VirtualMachine/Diadumenian 2.7z` has to be extracted first. The archive is saved with a password, in order to restrict access to privileged persons only. The used virtual hards disk type is compatible to both VMWare Server 2.x and VirtualBox 3.x (accepting the limitations stated above). In VMWare Server, the following steps have to be performed for import:

1. All necessary components of VMWare have to be started. Usually, VMWare has defined its services to automatically start on host system startup.
2. The virtual machine has to be extracted or moved to the parent folder of all virtual machines of VMWare (on Windows systems `C:\Virtual Machines` per default).
3. The *VMWare Infrastructure Web Access* has to be opened (usually available by the url `https://localhost:8333/`).
4. The task *Add Virtual Machine to Inventory* of the *Commands* panel has to be selected.
5. The VMWare configuration file `Diadumenian 2.vmx` has to be located and selected for import.
6. The *Run* button can be used to startup the virtual machine *Diadumenian 2*.

[14] `http://www.7-zip.org/`

7. The tab *Console* directly opens the graphical console of the virtual machine for visual interaction.

For VirtualBox, the virtual machine can not be imported directly. Instead, a new VirtualBox version of virtual machine has to be created, which reuses the virtual hard disks. The following steps have to be performed:

1. VirtualBox has to be started.
2. Via the *Virtual File Manager* all five `VMDK` files have to be imported.
3. The *New* virtual machine wizard has to be opened.
   a) A *name*, the operating system *Linux*, and the version *Debian* have to be stated in the second step.
   b) A memory of at least 1,024 MB has to be given in the third step.
   c) The existing disk `hda - root.vmdk` has to be selected as primary master in the fourth step.
4. The other virtual disks have to be added using the *Settings* dialog:
   - `hdb - home.vmdk` as primary slave
   - `hdd - swap.vmdk` as secondary slave
   - `sda - eval.vmdk` as SATA Port 0
   - `sdb - prot.vmdk` as SATA Port 1
5. Via the *Start* button, the virtual machine can be started – the graphical console will open automatically.

7 Summary and Future Work

In this chapter we give a résumé, discuss possible extensions to the database architecture, and look ahead to future work.

Summary

In this thesis, we have considered how an integrated database architecture can enrich a runtime environment for control flow-oriented simulation workflows. We have taken a look at typical requirements of scientific and simulation workflows towards their data integration and have derived requirements for the integrated database architecture. We have developed three scenarios that put data provisioning aspects of traditional simulation software and simulation workflows in concrete terms. We have then discussed possible optimizations for simulations that can be classified to one of the scenarios. Furthermore, we have defined functional requirements to the database architecture.

A possible integration of the database architecture into a workflow runtime environment based on BPEL↗ as the control-flow oriented workflow language and Apache ODE as the workflow engine has been proposed. Therefore, we have presented how the integrated database architecture can exploit the potential of the ongoing student project SIMPL. Since the student project is yet to be finished, an alternative architecture for the sake of evaluation and prototyping within this thesis has been suggested.

A major part of this thesis has considered the evaluation of different database systems for their application in the integrated database architecture. We have chosen candidates in order to support both the relational and the XML data model. Besides disk based database systems, we have considered in-memory database systems, too. Based on numerous requirements, an evaluation schema with eleven criteria has been developed to test the applicability of the database candidates. Besides criteria that inspect functional characteristics of the candidates, the databases have actually been installed and integrated into the evaluation environment. The experience gained during these integration attempts has been an important factor for the outcome of the evaluation. In addition to that, benchmark tests and scientific oriented database queries have evaluated the performance of the databases. The *IdaresDBEvaluation* workflow has been developed to automate and orchestrate this performance evaluation. Finally, three databases have been suggested for the integrated database architecture.

Subsequently, the conceptualized runtime environment has been implemented and prepared to execute BPEL based simulation workflows. The runtime environment has been developed

on a virtual machine and is therefore easily portable to other computers. We have developed prototypical workflows for two use cases of Scenario I (calculations and equation solvers). In this context, the integrated database architecture has proven its application as a storage for input data and intermediate and final result data. The added data management activities in the prototypical workflows have demonstrated how file based input and output can be simply loaded into and restored from the database architecture. The *IDARES Web Services* implement operations to load and store CSV based files into and from relational database tables. Additionally, the `DataHandler` concept facilitates a file type specific representation of arbitrary files in the database. Thereby, input and output files can be kept in a specialized relational or XML based format within the database architecture. When the database architecture is used for such data provisioning, it is possible to embed these simulation workflows in superordinate simulation workflows.

An example workflow has further shown the application of the integrated database architecture as a local cache. The principle to keep frequently accessed data objects locally to the workflow environment increases the throughput of data-intensive queries. Therefore, the *IDARES Web Services* has provided an operation to exchange data between a remote data store and the integrated database architecture. For the example workflow, we have demonstrated the in-memory storage capability of the database architecture, which made it possible to run the example workflow in an acceptable time frame. Moreover, we have shown how the integrated database architecture can be used to store XML based simulation results and query them using sophisticated XQuery statements.

All in all, the developed integrated database architecture has proven to be an integral component for data management use cases in the runtime environment for simulation workflows.

Future Work

The database evaluation of this thesis offers potential for future work. Although the evaluation has found satisfying database systems for all application fields of the database architecture, the evaluation of additional database systems is recommended. PostgreSQL shows a promising set of features and would be a good open source alternative or even competitor for IBM DB2 as relational database. Unfortunately, the candidate eXist-db does not support a transactional API. Oracle Berkeley DB XML, which can also be recommended according to [HSW08], would be a replacement candidate for eXist-db. On top of this, MonetDB SQL has shown severe instability during the tests. A further assessment might investigate reasons for this issue or consider alternative main memory databases, such as IBM solidDB.

Especially for the XML evaluation we had difficulties in finding appropriate reference data and feature requests from the simulation domain. Additional tasks in evaluating XML databases, e. g., regarding the storage of XML based provenance data, would be a useful extension of the evaluation and would increase the connection of the related criteria to the simulation domain.

The *IdaresDBEvaluation* workflow has been a useful artifact during the automated database performance evaluation. Up to now, the workflow only supports two port types: one for the

execution of a database query and one for the simulation of concurrent access to a database. Future projects may increase the value of the *IdaresDBEvaluation* workflow by adding support for arbitrary port types in performance tests. Moreover, the *IdaresDBEvaluation* workflow can be adapted to exploit the enhanced features of the SIMPL framework rather than the utility Web Services developed in this thesis. Finally, an idea for a useful extension is the transformation of the synchronous Web Service `invokes` into asynchronous ones. This is extremely beneficial, since many database performance tests tend to long durations.

The IBM DB2 was used as *ODE model db* through the Hibernate DAO layer of Apache ODE. The performance evaluation has attested a poor performance of this approach. A further investigation might suggest and implement optimizations. At the same time, the thesis suggests the evaluation of the performance of the *SIMPL DAO* layer. In the context of SIMPL it is crucial to remind the reader that, although the integrated database architecture was conceptualized to honor the design of the SIMPL framework, its compatibility could not yet be proven. Moreover, the suggested architecture for a runtime environment for simulation workflows based on SIMPL and the database architecture (cf. Section 4.2 on page 72) is yet to be implemented.

For the local cache, the database architecture currently only supports a static caching mechanism that has to be initiated with a respective set of workflow activities. Future research may investigate how data synchronization between the local cache and its remote data source can be realized. Hereby, consistency aspects have to be considered when the local cache or the remote data source is manipulated.

The concept of the integrated database architecture carries great potential itself. It has already been announced that an upcoming diploma thesis by Dormien will utilize the integrated database architecture to store a relational representation of an FEM grid. The goal of this work is to provide a common data base for FEM grids in order to share them between different simulation tools. Based on these results, a simulation workflow can then, for example, advise the PANDAS[1] tool to build up an FEM grid and afterwards utilize a DUNE application to solve equations for the grid.

Another potential future work investigates concepts how the integrated database architecture can be further coupled to the workflow engine Apache ODE in order to assist in the execution of typical workflow engine tasks. An example for this approach are variable assignments, which might be performed by database manipulation queries directly. Thereby, the overall performance of the execution of simulation workflow instances is aspired to be improved.

[1] `http://www.mechbau.uni-stuttgart.de/pandas/`

A Appendix

A.1 Abbreviations

In every problem domain a lot of technical terms arise. Since many of them consist of more than one word, abbreviations are created. Everyone in the problem domain knows them by heart, while an outsider is often unable to distinguish between them. In order to manage the issue within this document, the current section provides all relevant abbreviations used in this thesis. When a new abbreviation is introduced at any other part, it is referenced to this chapter, e. g. WS↗. In addition to that, it is expected that the reader has a comprehension of the basic terminology and the related abbreviations of information technology.

2PC: Two Phase Commit

ACID: atomicity, consistency, isolation, and durability; the state of the art aspects for reliable transaction processing

ANSI: American National Standards Institute

ASCII: American Standard Code for Information Interchange

BLOB: Binary Large Object: a database type for very large binary streams

BPEL: Business Process Execution Language; also WS-BPEL or BPEL4WS

CAD: Computer-aided design

CLOB: Character Large Object: a database type for very large character streams

CSV: Comma-separated values

DAO: Data Access Objects

DAS: Data Access Service

DBMS: Database Management System

DBS: Database System

DDL: Data Definition Language: a part of SQL↗ that involves database schema creations and modifications

DUNE: Distributed and Unified Numerics Environment: `http://www.dune-project.org/`

EPL: Eclipse Public License

ESB: Enterprise Service Bus

ETL: Extraction Transformation Load of data

FEM: finite element method

FLWOR: For Let Where Order Return; an expression type in XQuery ([BCF+07])

FTP: File Transfer Protocol

GPL: Gnu Public License

HBM: Hibernate Mapping file

HPC: High Performance Computing

HTTP: Hypertext Transfer Protocol

IDARES: Integrated Database Architecture for a Runtime Environment for Simulation Workflows = the title of this thesis; IDARES is used as a prefix for components to state that the components belong to this thesis

IMDB: In-memory database

JBI: Java Business Integration

JDBC: Java Database Connectivity

JMS: Java Message Service

JRE: Java Runtime Environment

JVM: Java Virtual Machine: the execution environment for Java based applications

LGPL: GNU Lesser General Public License

MEP: Message Exchange Patterns

MPL: Mozilla Public License

MMDB: Main memory database (cf. MMDBMS↗)

MMDBMS: Main memory database management system

MVCC: Multi-Version Concurrency Control

NDB: Network DataBase: a storage technique in the MySQL Cluster

OASIS: Organization for the Advancement of Structured Information Standards: `http://www.oasis-open.org/`

OLAP: Online analytical processing

QM/MM: Quantum Mechanics/Molecular Mechanics

PDE: partial differential equation

RDBMS: Relational database management system

SCA: Service Component Architecture

SCUFL: Simple Conceptual Unified Flow Language

SDO: Service Data Objects; see [GPP09]

SIMPL: SimTech: Information Management, Processes and Languages: `http://www.ipvs.uni-stuttgart.de/abteilungen/as/lehre/lehrveranstaltungen/studienprojekte/SS09/StuPro.SIMPL`

SimTech: Cluster of Excellence "Simulation Technology": `http://www.simtech.uni-stuttgart.de/`

SMTP: Simple Mail Transfer Protocol

SOA: Service Oriented Architecture

SOAP: Simple Object Access Protocol

SQL: Structured Query Language

SSL: Secure Sockets Layer

TCL: Tool Command Language: `http://www.tcl.tk/`

TCP: Transmission Control Protocol

UDDI: Universal Description Discovery and Integration

UTF-8: 8-bit UCS/Unicode Transformation Format

UUID: Universally Unique Identifier

WS: used as prefix for Web Service, e. g. WS-Security

W3C: World Wide Web Consortium: `http://www.w3.org/`

WSDL: Web Services Description Language; former: Web Services Definition Language

WSRF: Web Services Resource Framework

XML: Extensible Markup Language

XQJ: XQuery API for Java; see [OC09]

XSD: XML↗ Schema Definition

A.2 List of Requirements

This sections lists all requirements for the integrated database architecture, which are spread through the textual descriptions of Chapter 3 on page 47 and marked with ℝ. Every requirement is applied to a scope, which is noted in square brackets *[]*.

A.3 Performance Queries

A.3.1 Relational Performance Queries

This appendix lists tests and queries used in the relational benchmark. The definition of all queries can also be found in `[DVD]/Sources/SQL/Evaluation/evaluation-data-*.sql`. The relational benchmark respects different SQL dialects, while the following statements are specified to work with Apache Derby.

[TPCH] Schema Creation The following tests create the required SQL schemata.

```
-- The TPCH schema contains the eight original tables of the TPC-H benchmark
CREATE SCHEMA TPCH
CREATE TABLE TPCH.NATION (
   N_NATIONKEY  INTEGER NOT NULL,
   N_NAME       CHAR(25) NOT NULL,
   N_REGIONKEY  INTEGER NOT NULL,
   N_COMMENT    VARCHAR(152)
)
CREATE TABLE TPCH.PART (
   P_PARTKEY     INTEGER NOT NULL,
   P_NAME        VARCHAR(55) NOT NULL,
   P_MFGR        CHAR(25) NOT NULL,
   P_BRAND       CHAR(10) NOT NULL,
   P_TYPE        VARCHAR(25) NOT NULL,
   P_SIZE        INTEGER NOT NULL,
   P_CONTAINER   CHAR(10) NOT NULL,
   P_RETAILPRICE DECIMAL(15,2) NOT NULL,
   P_COMMENT     VARCHAR(23) NOT NULL
)
CREATE TABLE TPCH.SUPPLIER (
   S_SUPPKEY     INTEGER NOT NULL,
   S_NAME        CHAR(25) NOT NULL,
   S_ADDRESS     VARCHAR(40) NOT NULL,
   S_NATIONKEY   INTEGER NOT NULL,
   S_PHONE       CHAR(15) NOT NULL,
   S_ACCTBAL     DECIMAL(15,2) NOT NULL,
   S_COMMENT     VARCHAR(101) NOT NULL
)
CREATE TABLE TPCH.PARTSUPP (
   PS_PARTKEY     INTEGER NOT NULL,
   PS_SUPPKEY     INTEGER NOT NULL,
   PS_AVAILQTY    INTEGER NOT NULL,
   PS_SUPPLYCOST  DECIMAL(15,2)  NOT NULL,
   PS_COMMENT     VARCHAR(199) NOT NULL
)
CREATE TABLE TPCH.CUSTOMER (
   C_CUSTKEY     INTEGER NOT NULL,
   C_NAME        VARCHAR(25) NOT NULL,
   C_ADDRESS     VARCHAR(40) NOT NULL,
   C_NATIONKEY   INTEGER NOT NULL,
   C_PHONE       CHAR(15) NOT NULL,
   C_ACCTBAL     DECIMAL(15,2)   NOT NULL,
   C_MKTSEGMENT  CHAR(10) NOT NULL,
   C_COMMENT     VARCHAR(117) NOT NULL
)
CREATE TABLE TPCH.ORDERS (
```

```
47    O_ORDERKEY        INTEGER NOT NULL,
48    O_CUSTKEY         INTEGER NOT NULL,
49    O_ORDERSTATUS     CHAR(1) NOT NULL,
50    O_TOTALPRICE      DECIMAL(15,2) NOT NULL,
51    O_ORDERDATE       DATE NOT NULL,
52    O_ORDERPRIORITY   CHAR(15) NOT NULL,
53    O_CLERK           CHAR(15) NOT NULL,
54    O_SHIPPRIORITY    INTEGER NOT NULL,
55    O_COMMENT         VARCHAR(79) NOT NULL
56  )
57 CREATE TABLE TPCH.LINEITEM (
58    L_ORDERKEY        INTEGER NOT NULL,
59    L_PARTKEY         INTEGER NOT NULL,
60    L_SUPPKEY         INTEGER NOT NULL,
61    L_LINENUMBER      INTEGER NOT NULL,
62    L_QUANTITY        DECIMAL(15,2) NOT NULL,
63    L_EXTENDEDPRICE   DECIMAL(15,2) NOT NULL,
64    L_DISCOUNT        DECIMAL(15,2) NOT NULL,
65    L_TAX             DECIMAL(15,2) NOT NULL,
66    L_RETURNFLAG      CHAR(1) NOT NULL,
67    L_LINESTATUS      CHAR(1) NOT NULL,
68    L_SHIPDATE        DATE NOT NULL,
69    L_COMMITDATE      DATE NOT NULL,
70    L_RECEIPTDATE     DATE NOT NULL,
71    L_SHIPINSTRUCT    CHAR(25) NOT NULL,
72    L_SHIPMODE        CHAR(10) NOT NULL,
73    L_COMMENT         VARCHAR(44) NOT NULL
74  )
```

[TPCH] Import CSV * These tests import data from CSV files into the eight TPC-H tables. The CSV files can be found in `[DVD]/Frameworks/TPCH/TPCH data 1GB/*.tbl`. The Web Service operation `importCSVTODataSource` is performing the import. Its definition can be found in Appendix A.4.2 on page 188. An example message for the CSV import shows the following listing:

```
1 <importCSVToDataSourceRequest xmlns="http://ahija.de/thesis/idares/model"
      commitInterval="1000" datePattern="yyyy-MM-dd" escapeChar="\" hasHeader="false"
      nullString="" quoteChar="&quot;" separatorChar="|" sourceMaxRows="-1"
      timePattern="HH:mm:ss" timestampPattern="yyyy-MM-dd HH:mm:ss.SSS">
2   <csvSource>
3     <sourceURL charset="ISO-8859-1">
4       file:///home/our/benchmark/TPCH\%20data\%201GB/region.tbl
5     </sourceURL>
6   </csvSource>
7   <targetDatasource>jdbc/IdaresEvaluation/DerbyEmbedded</targetDatasource>
8   <targetProcessNamePattern>
9     .*org\.apache\.catalina\.startup\.Bootstrap.*
10  </targetProcessNamePattern>
11  <targetTable>
12    <tableName>TPCH.REGION</tableName>
13  </targetTable>
14 </importCSVToDataSourceRequest>
```

[TPCH] Primary Key * The following tests create primary keys for the TPC-H tables.

```
1 ALTER TABLE TPCH.REGION ADD PRIMARY KEY (R_REGIONKEY)
2 ALTER TABLE TPCH.NATION ADD PRIMARY KEY (N_NATIONKEY)
3 ALTER TABLE TPCH.PART ADD PRIMARY KEY (P_PARTKEY)
4 ALTER TABLE TPCH.SUPPLIER ADD PRIMARY KEY (S_SUPPKEY)
5 ALTER TABLE TPCH.PARTSUPP ADD PRIMARY KEY (PS_PARTKEY, PS_SUPPKEY)
6 ALTER TABLE TPCH.CUSTOMER ADD PRIMARY KEY (C_CUSTKEY)
7 ALTER TABLE TPCH.ORDERS ADD PRIMARY KEY (O_ORDERKEY)
8 ALTER TABLE TPCH.LINEITEM ADD PRIMARY KEY (L_ORDERKEY, L_LINENUMBER)
```

[TPCH] Foreign key * The following tests create foreign keys for the TPC-H tables as specified in the TPC-H benchmark.

```
1 ALTER TABLE TPCH.NATION ADD CONSTRAINT FK_NATION_TO_REGION FOREIGN KEY (N_REGIONKEY)
      REFERENCES TPCH.REGION (R_REGIONKEY)
2 ALTER TABLE TPCH.SUPPLIER ADD CONSTRAINT FK_SUPPLIER_TO_NATION FOREIGN KEY
      (S_NATIONKEY) REFERENCES TPCH.NATION (N_NATIONKEY)
3 ALTER TABLE TPCH.PARTSUPP ADD CONSTRAINT FK_PARTSUPP_TO_PART FOREIGN KEY
      (PS_PARTKEY)  REFERENCES TPCH.PART (P_PARTKEY)
4 ALTER TABLE TPCH.PARTSUPP ADD CONSTRAINT FK_PARTSUPP_TO_SUPPLIER FOREIGN KEY
      (PS_SUPPKEY) REFERENCES TPCH.SUPPLIER (S_SUPPKEY)
5 ALTER TABLE TPCH.CUSTOMER ADD CONSTRAINT FK_CUSTOMER_TO_NATION FOREIGN KEY
      (C_NATIONKEY) REFERENCES TPCH.NATION (N_NATIONKEY)
6 ALTER TABLE TPCH.ORDERS ADD CONSTRAINT FK_ORDERS_TO_CUSTOMER FOREIGN KEY (O_CUSTKEY)
        REFERENCES TPCH.CUSTOMER (C_CUSTKEY)
7 ALTER TABLE TPCH.LINEITEM ADD CONSTRAINT FK_LINEITEM_TO_PART FOREIGN KEY (L_PARTKEY)
        REFERENCES TPCH.PART (P_PARTKEY)
8 ALTER TABLE TPCH.LINEITEM ADD CONSTRAINT FK_LINEITEM_TO_SUPPLIER FOREIGN KEY
      (L_SUPPKEY) REFERENCES TPCH.SUPPLIER (S_SUPPKEY)
9 ALTER TABLE TPCH.LINEITEM ADD CONSTRAINT FK_LINEITEM_TO_PARTSUPP FOREIGN KEY
      (L_PARTKEY, L_SUPPKEY) REFERENCES TPCH.PARTSUPP (PS_PARTKEY, PS_SUPPKEY)
10 ALTER TABLE TPCH.LINEITEM ADD CONSTRAINT FK_LINEITEM_TO_ORDERS FOREIGN KEY
      (L_ORDERKEY) REFERENCES TPCH.ORDERS (O_ORDERKEY)
```

[TPCH] Performance Queries Simple The following tests implement simple queries to cover selection, projection, joins, aggregation, and sorting.

```
1 -- [TPCH] Performance Select Count CUSTOMER
2 SELECT COUNT(*) FROM TPCH.CUSTOMER
3 -- [TPCH] Performance Select Count LINEITEM
4 SELECT COUNT(*) FROM TPCH.LINEITEM
5 -- [TPCH] Performance Select Count Distinct ORDERS
6 SELECT COUNT(DISTINCT o.O_ORDERKEY) FROM TPCH.ORDERS o
7 -- [TPCH] Performance Select * CUSTOMER
8 SELECT * FROM TPCH.CUSTOMER
9 -- [TPCH] Performance Select Column CUSTOMER
10 SELECT c.C_ACCTBAL
11   FROM TPCH.CUSTOMER c
12 -- [TPCH] Performance Select * Where ORDERS
13 SELECT *
14   FROM TPCH.ORDERS o
15  WHERE o.O_ORDERSTATUS = 'O'
16 -- [TPCH] Performance Select Column Where ORDERS
17 SELECT o.O_TOTALPRICE
18   FROM TPCH.ORDERS o
19  WHERE o.O_ORDERSTATUS = 'O'
20 -- [TPCH] Intermediate Index IDX_ORDERS_ORDERSTATUS
21 CREATE INDEX TPCH.IDX_ORDERS_ORDERSTATUS ON TPCH.ORDERS(O_ORDERSTATUS)
```

```
22 -- [TPCH] Intermediate Index IDX_ORDERS_ORDERSTATUS
23 SELECT o.O_TOTALPRICE
24   FROM TPCH.ORDERS o
25  WHERE o.O_ORDERSTATUS = 'O'
26 -- [TPCH] Performance Select Join without Index
27 SELECT *
28   FROM TPCH.CUSTOMER c,
29        TPCH.NATION n
30  WHERE c.C_NAME = n.N_NAME
31 -- [TPCH] Performance Select Join with Index
32 SELECT c.C_NAME, c.C_ADDRESS, n.N_NAME
33   FROM TPCH.CUSTOMER c,
34        TPCH.NATION n
35  WHERE c.C_NATIONKEY = n.N_NATIONKEY
36 -- [TPCH] Performance Select Aggregation
37 SELECT c.C_NAME, COUNT(*), MIN(o.O_TOTALPRICE), MAX(o.O_TOTALPRICE),
       AVG(o.O_TOTALPRICE)
38   FROM TPCH.ORDERS o
39  INNER JOIN TPCH.CUSTOMER c ON c.C_CUSTKEY = o.O_CUSTKEY
40  GROUP BY c.C_CUSTKEY, c.C_NAME
41 -- [TPCH] Performance Select Aggregation Join Order
42 SELECT c.C_NAME AS CUSTOMER, COUNT(*) AS COUNT, SUM(o.O_TOTALPRICE) AS TOTAL,
       MIN(o.O_TOTALPRICE) AS MINIMUM, MAX(o.O_TOTALPRICE) AS MAXIMUM,
       AVG(o.O_TOTALPRICE) AS AVERAGE
43   FROM TPCH.ORDERS o
44  INNER JOIN TPCH.CUSTOMER c ON c.C_CUSTKEY = o.O_CUSTKEY
45  INNER JOIN TPCH.NATION n ON c.C_NATIONKEY = n.N_NATIONKEY
46  WHERE n.N_NAME = 'GERMANY'
47  GROUP BY c.C_CUSTKEY, c.C_NAME
48  ORDER BY TOTAL DESC
```

[TPCH] Intermediate Index * We create additional indexes at this point because we wanted to inspect the database without these indexes before. For the next set of queries, these indexes are very important since they are excellent access paths.

```
1 -- Performance Index IDX_ORDERS_ORDERDATE
2 CREATE INDEX TPCH.IDX_ORDERS_ORDERDATE ON TPCH.ORDERS(O_ORDERDATE)
3 -- Performance Index IDX_LINEITEM_SHIPDATE
4 CREATE INDEX TPCH.IDX_LINEITEM_SHIPDATE ON TPCH.LINEITEM(L_SHIPDATE)
5 -- Performance Index IDX_PART_TYPE
6 CREATE INDEX TPCH.IDX_PART_TYPE ON TPCH.PART(P_TYPE)
```

[TPCH] Performance TPCH * These queries originate from the TPC-H benchmark. The corresponding TPC-H query number is noted for each query.

```
1 -- [TPCH] Performance TPCH Query 2
2 SELECT S_ACCTBAL, S_NAME, N_NAME, P_PARTKEY, P_MFGR, S_ADDRESS, S_PHONE, S_COMMENT
3   FROM TPCH.PART,
4        TPCH.SUPPLIER,
5        TPCH.PARTSUPP,
6        TPCH.NATION,
7        TPCH.REGION
8  WHERE P_PARTKEY = PS_PARTKEY
9    AND S_SUPPKEY = PS_SUPPKEY
10   AND P_SIZE = 13
11   AND P_TYPE LIKE '%NICKEL'
12   AND S_NATIONKEY = N_NATIONKEY
```

```
13    AND N_REGIONKEY = R_REGIONKEY
14    AND R_NAME = 'ASIA'
15    AND PS_SUPPLYCOST = (SELECT MIN(PS_SUPPLYCOST)
16                           FROM TPCH.PARTSUPP, TPCH.SUPPLIER, TPCH.NATION, TPCH.REGION
17                          WHERE P_PARTKEY = PS_PARTKEY
18                            AND S_SUPPKEY = PS_SUPPKEY
19                            AND S_NATIONKEY = N_NATIONKEY
20                            AND N_REGIONKEY = R_REGIONKEY
21                            AND R_NAME = 'ASIA')
22  ORDER BY S_ACCTBAL DESC, N_NAME, S_NAME, P_PARTKEY
23
24 -- [TPCH] Performance TPCH Query 4 modified
25 SELECT O_ORDERPRIORITY, COUNT(*) AS ORDER_COUNT
26   FROM TPCH.ORDERS
27  WHERE O_ORDERDATE BETWEEN DATE('1995-10-01') AND DATE('1996-08-01')
28  GROUP BY O_ORDERPRIORITY
29  ORDER BY O_ORDERPRIORITY
30
31 -- [TPCH] Performance TPCH Query 6
32 SELECT SUM(L_EXTENDEDPRICE * L_DISCOUNT) AS REVENUE
33   FROM TPCH.LINEITEM
34  WHERE L_SHIPDATE >= DATE('1993-01-01')
35    AND L_SHIPDATE < DATE('1993-06-01')
36    AND L_DISCOUNT BETWEEN 0.09 - 0.01 AND 0.09 + 0.01
37    AND L_QUANTITY < 25
38
39 -- [TPCH] Performance TPCH Query 16
40 SELECT P_BRAND, P_TYPE, P_SIZE, COUNT(DISTINCT PS_SUPPKEY) AS SUPPLIER_CNT
41   FROM TPCH.PARTSUPP,
42        TPCH.PART
43  WHERE P_PARTKEY = PS_PARTKEY
44    AND P_BRAND <> 'BRAND#31'
45    AND P_TYPE NOT LIKE 'ECONOMY POLISHED%'
46    AND P_SIZE IN (3, 19, 34, 6, 49, 28, 39, 44)
47    AND PS_SUPPKEY NOT IN (SELECT S_SUPPKEY
48                             FROM TPCH.SUPPLIER
49                            WHERE S_COMMENT LIKE '%CUSTOMER%COMPLAINTS%')
50  GROUP BY P_BRAND, P_TYPE, P_SIZE
51  ORDER BY SUPPLIER_CNT DESC, P_BRAND, P_TYPE, P_SIZE
52
53 -- [TPCH] Performance TPCH Query 17
54 SELECT SUM(L_EXTENDEDPRICE) / 7.0 AS AVG_YEARLY
55   FROM TPCH.LINEITEM,
56        TPCH.PART
57  WHERE P_PARTKEY = L_PARTKEY
58    AND P_BRAND = 'Brand#14'
59    AND P_CONTAINER = 'JUMBO JAR'
60    AND L_QUANTITY < (SELECT 0.2 * AVG(L_QUANTITY)
61                        FROM TPCH.LINEITEM
62                       WHERE L_PARTKEY = P_PARTKEY)
63
64 -- [TPCH] Performance TPCH Query 22
65 SELECT CNTRYCODE, COUNT(*) AS NUMCUST, SUM(C_ACCTBAL) AS TOTACCTBAL
66   FROM (SELECT SUBSTRING(C_PHONE FROM 1 FOR 2) AS CNTRYCODE, C_ACCTBAL
67           FROM TPCH.CUSTOMER
68          WHERE SUBSTRING(C_PHONE FROM 1 FOR 2) IN (''33'', ''16'', ''23'', ''11'',
                ''13'', ''32'', ''18'')
69            AND C_ACCTBAL > (SELECT AVG(C_ACCTBAL)
70                               FROM TPCH.CUSTOMER
71                              WHERE C_ACCTBAL > 0.00
72                                AND SUBSTRING(C_PHONE FROM 1 FOR 2) IN (''33'',
                                      ''16'', ''23'', ''11'', ''13'', ''32'', ''18''))
```

```
73            AND NOT EXISTS (SELECT *
74                              FROM TPCH.ORDERS
75                             WHERE O_CUSTKEY = C_CUSTKEY)) AS CUSTSALE
76  GROUP BY CNTRYCODE
77  ORDER BY CNTRYCODE
```

[TPCH] Schema Dropping This test realizes the cleanup by dropping the data, the tables, and the schemata. The droppings have to occur in an order that respects referential constraints. Hence, if a column of table *A* refers to a column of table *B*, table *A* has to be cleared or dropped before table *B*. Otherwise the database complains about violated referential constraints. Some databases allow the deletion of the whole schema at once. Others – as shown in the following listing – require that a schema is cleared of all content before it is dropped with the keyword `RESTRICT`.

```
1 DROP TABLE TPCH.LINEITEM
2 DROP TABLE TPCH.ORDERS
3 DROP TABLE TPCH.CUSTOMER
4 DROP TABLE TPCH.PARTSUPP
5 DROP TABLE TPCH.PART
6 DROP TABLE TPCH.SUPPLIER
7 DROP TABLE TPCH.NATION
8 DROP TABLE TPCH.REGION
9 DROP SCHEMA TPCH RESTRICT
```

[CONCURRENT] * These tests simulate concurrent access to the database.

```
1 -- [CONCURRENT] Schema Creation
2 CREATE SCHEMA HOSPITAL
3 CREATE TABLE HOSPITAL.CUSTOMER (
4    C_CUSTKEY     INTEGER NOT NULL,
5    C_NAME        VARCHAR(25) NOT NULL,
6    C_ADDRESS     VARCHAR(40) NOT NULL,
7    C_NATIONKEY   INTEGER NOT NULL,
8    C_PHONE       CHAR(15) NOT NULL,
9    C_ACCTBAL     DECIMAL(15,2) NOT NULL,
10   C_MKTSEGMENT  CHAR(10) NOT NULL,
11   C_COMMENT     VARCHAR(117) NOT NULL
12 )
13 CREATE TABLE HOSPITAL.PATIENT (
14   P_PATIENTKEY  INTEGER NOT NULL PRIMARY KEY,
15   P_NAME        VARCHAR(25) NOT NULL,
16   P_ADDRESS     VARCHAR(40) NOT NULL,
17   P_PHONE       CHAR(15) NOT NULL
18 )
19 CREATE TABLE HOSPITAL.EMERGENCY_CALL (
20   C_CALLID     BIGINT NOT NULL PRIMARY KEY GENERATED ALWAYS AS IDENTITY (START WITH
         1, INCREMENT BY 1),
21   C_PHONE      CHAR(15) NOT NULL,
22   C_TIMESTAMP  TIMESTAMP NOT NULL,
23   C_NOTE       VARCHAR(32672) NOT NULL
24 )
```

With two more queries, the content of the table `customer` is filled with the same data as for the TPC-H test. A primary key and an index follow:

```
1 ALTER TABLE HOSPITAL.CUSTOMER ADD PRIMARY KEY (C_CUSTKEY)
2 CREATE INDEX HOSPITAL.IDX_CUSTOMER_SHIPDATE ON HOSPITAL.CUSTOMER(C_NATIONKEY)
```

The Web Service operation `simulateApplications` is realizing the concurrency simulation. Its definition can be found in Appendix A.4.2 on page 188. An example message for the request shows the following listing:

```
1 <simulateApplicationsRequest xmlns="http://ahija.de/thesis/idares/model">
2   <datasourceName>jdbc/IdaresEvaluation/DerbyEmbedded</datasourceName>
3   <processNamePattern>
4     .*org\.apache\.catalina\.startup\.Bootstrap.*
5   </processNamePattern>
6   <appCount>10</appCount>
7   <seed>314</seed>
8   <isolation>READ_COMMITTED</isolation>
9 </simulateApplicationsRequest>
```

Web Services providing the WSDL↗ operation `simulateApplications` are described in detail in Section 6.3.2 on page 141.

Finally the schema for the concurrent test is dropped with a test.

```
1 -- [CONCURRENT] Schema Dropping
2 DROP TABLE HOSPITAL.CUSTOMER
3 DROP TABLE HOSPITAL.PATIENT
4 DROP TABLE HOSPITAL.EMERGENCY_CALL
5 DROP SCHEMA HOSPITAL RESTRICT
```

[CO2] * These tests use data from a finite element simulation for performance benchmarking.

```
1 -- [CO2] Schema Creation
2 CREATE TABLE CO2.VERTICES (
3   V_ID            INTEGER NOT NULL,
4   V_X             DOUBLE NOT NULL,
5   V_Y             DOUBLE NOT NULL,
6   V_Z             DOUBLE NOT NULL,
7   V_POROSITY      DOUBLE,
8   V_PERMEABILITY DOUBLE
9 )
10 CREATE TABLE CO2.IMPORT_PROPERTIES (
11   P_DUMMY         VARCHAR(1),
12   P_X             DOUBLE NOT NULL,
13   P_Y             DOUBLE NOT NULL,
14   P_Z             DOUBLE NOT NULL,
15   P_POROSITY      DOUBLE NOT NULL,
16   P_PERMEABILITY DOUBLE NOT NULL
17 )
18 CREATE TABLE CO2.ELEMENTS (
19   E_ID            INTEGER NOT NULL,
20   E_V1            INTEGER NOT NULL,
21   E_V2            INTEGER NOT NULL,
22   E_V3            INTEGER NOT NULL,
23   E_V4            INTEGER NOT NULL,
24   E_V5            INTEGER NOT NULL,
25   E_V6            INTEGER NOT NULL,
26   E_V7            INTEGER NOT NULL,
27   E_V8            INTEGER NOT NULL
28 )
```

In the following three CSV files are imported using the already described Web Service operation `importCSVViaDataSource`. The files are:

- `[DVD]/Frameworks/CO2 Data/vertices_johansen.dat`
- `[DVD]/Frameworks/CO2 Data/properties_johansen.dat`
- `[DVD]/Frameworks/CO2 Data/elements_johansen.dat`

The tests go on with ordinary SQL statements.

```
-- [CO2] Primary Key VERTICES
ALTER TABLE CO2.VERTICES ADD PRIMARY KEY (V_ID)
ALTER TABLE CO2.ELEMENTS ADD PRIMARY KEY (E_ID)
-- [CO2] Foreign Keys ELEMENTS
ALTER TABLE CO2.ELEMENTS ADD CONSTRAINT FK_ELEMENTS_TO_VERTICES1 FOREIGN KEY (E_V1)
    REFERENCES CO2.VERTICES (V_ID)
ALTER TABLE CO2.ELEMENTS ADD CONSTRAINT FK_ELEMENTS_TO_VERTICES2 FOREIGN KEY (E_V1)
    REFERENCES CO2.VERTICES (V_ID)
ALTER TABLE CO2.ELEMENTS ADD CONSTRAINT FK_ELEMENTS_TO_VERTICES3 FOREIGN KEY (E_V1)
    REFERENCES CO2.VERTICES (V_ID)
ALTER TABLE CO2.ELEMENTS ADD CONSTRAINT FK_ELEMENTS_TO_VERTICES4 FOREIGN KEY (E_V1)
    REFERENCES CO2.VERTICES (V_ID)
ALTER TABLE CO2.ELEMENTS ADD CONSTRAINT FK_ELEMENTS_TO_VERTICES5 FOREIGN KEY (E_V1)
    REFERENCES CO2.VERTICES (V_ID)
ALTER TABLE CO2.ELEMENTS ADD CONSTRAINT FK_ELEMENTS_TO_VERTICES6 FOREIGN KEY (E_V1)
    REFERENCES CO2.VERTICES (V_ID)
ALTER TABLE CO2.ELEMENTS ADD CONSTRAINT FK_ELEMENTS_TO_VERTICES7 FOREIGN KEY (E_V1)
    REFERENCES CO2.VERTICES (V_ID)
ALTER TABLE CO2.ELEMENTS ADD CONSTRAINT FK_ELEMENTS_TO_VERTICES8 FOREIGN KEY (E_V1)
    REFERENCES CO2.VERTICES (V_ID)
-- [CO2] Indexes
CREATE UNIQUE INDEX CO2.IDX_VERTICES_XYZ ON CO2.VERTICES(V_X ASC, V_Y ASC, V_Z ASC)
CREATE UNIQUE INDEX CO2.IMPORT_PROPERTIES_XYZ ON CO2.IMPORT_PROPERTIES(P_X, P_Y, P_Z)
-- [CO2] Performance Adopt Properties
-- We copy the properties POROSITY and PERMEABILITY to the table CO2.VERTICES
UPDATE CO2.VERTICES v
   SET v.V_POROSITY = (SELECT p.P_POROSITY FROM CO2.IMPORT_PROPERTIES p WHERE v.V_X
       = p.P_X AND v.V_Y = p.P_Y AND v.V_Z = p.P_Z),
       v.V_PERMEABILITY = (SELECT p.P_PERMEABILITY FROM CO2.IMPORT_PROPERTIES p
           WHERE v.V_X = p.P_X AND v.V_Y = p.P_Y AND v.V_Z = p.P_Z)
-- [CO2] Performance Select All
SELECT *
  FROM CO2.VERTICES v
 ORDER BY v.V_X,  v.V_Y,  v.V_Z
-- [CO2] Performance Join 1
-- We calculate the average porosity for each element (hexahedron)
SELECT e.E_ID AS ELEMENT, ((v1.V_POROSITY + v2.V_POROSITY + v3.V_POROSITY +
    v4.V_POROSITY + v5.V_POROSITY + v6.V_POROSITY + v7.V_POROSITY + v8.V_POROSITY) /
    8) AS AVG_POROSITY
  FROM CO2.ELEMENTS e,
       CO2.VERTICES v1, CO2.VERTICES v2, CO2.VERTICES v3, CO2.VERTICES v4,
           CO2.VERTICES v5, CO2.VERTICES v6, CO2.VERTICES v7, CO2.VERTICES v8
 WHERE e.E_V1 = v1.V_ID AND e.E_V2 = v2.V_ID AND e.E_V3 = v3.V_ID AND e.E_V4 =
     v4.V_ID AND e.E_V5 = v5.V_ID AND e.E_V6 = v6.V_ID AND e.E_V7 = v7.V_ID AND
     e.E_V8 = v8.V_ID
   AND (v1.V_POROSITY + v2.V_POROSITY + v3.V_POROSITY + v4.V_POROSITY +
       v5.V_POROSITY + v6.V_POROSITY + v7.V_POROSITY + v8.V_POROSITY) IS NOT NULL
 ORDER BY ((v1.V_POROSITY + v2.V_POROSITY + v3.V_POROSITY + v4.V_POROSITY +
     v5.V_POROSITY + v6.V_POROSITY + v7.V_POROSITY + v8.V_POROSITY) / 8) DESC
-- [CO2] Performance Join 2
-- We split the overall shape in slices and calculate how many elements each contains
```

```
35 SELECT g.GROUP_NO, (SELECT MIN(v.V_X) FROM CO2.VERTICES v WHERE (ROUND(v.V_X /
       481.0, 0) = g.GROUP_NO)) AS MIN_X,
36                    (SELECT MAX(v.V_X) FROM CO2.VERTICES v WHERE (ROUND(v.V_X /
                          481.0, 0) = g.GROUP_NO)) AS MAX_X,
37         g.VERTICES_COUNT, COUNT(DISTINCT e.E_ID) AS ELEMENT_COUNT,
38         AVG(v1.V_POROSITY + v2.V_POROSITY + v3.V_POROSITY + v4.V_POROSITY +
               v5.V_POROSITY + v6.V_POROSITY + v7.V_POROSITY + v8.V_POROSITY) / 8 AS
               AVG_POROSITY
39   FROM (SELECT COUNT(*) AS VERTICES_COUNT, ROUND(v.V_X / 481.0, 0) AS GROUP_NO
40            FROM CO2.VERTICES v
41           GROUP BY ROUND(v.V_X / 481.0, 0)) g,
42         CO2.ELEMENTS e,
43         CO2.VERTICES v1, CO2.VERTICES v2, CO2.VERTICES v3, CO2.VERTICES v4,
               CO2.VERTICES v5, CO2.VERTICES v6, CO2.VERTICES v7, CO2.VERTICES v8
44  WHERE e.E_V1 = v1.V_ID AND e.E_V2 = v2.V_ID AND e.E_V3 = v3.V_ID AND e.E_V4 =
        v4.V_ID AND e.E_V5 = v5.V_ID AND e.E_V6 = v6.V_ID AND e.E_V7 = v7.V_ID AND
        e.E_V8 = v8.V_ID
45    AND (ROUND(v1.V_X / 481.0, 0) = g.GROUP_NO) AND (ROUND(v2.V_X / 481.0, 0) =
          g.GROUP_NO) AND (ROUND(v3.V_X / 481.0, 0) = g.GROUP_NO) AND (ROUND(v4.V_X /
          481.0, 0) = g.GROUP_NO)
46    AND (ROUND(v5.V_X / 481.0, 0) = g.GROUP_NO) AND (ROUND(v6.V_X / 481.0, 0) =
          g.GROUP_NO) AND (ROUND(v7.V_X / 481.0, 0) = g.GROUP_NO) AND (ROUND(v8.V_X /
          481.0, 0) = g.GROUP_NO)
47  GROUP BY g.GROUP_NO, g.VERTICES_COUNT
48 -- [CO2] Performance Join 3
49 -- We calculate which elements (hexahedrons) are neighbors
50 SELECT e1.E_ID AS E1, e2.E_ID AS E2
51   FROM ((SELECT e.E_ID, e.E_V1 AS V FROM CO2.ELEMENTS e) UNION
52         (SELECT e.E_ID, e.E_V2 AS V FROM CO2.ELEMENTS e) UNION
53          (SELECT e.E_ID, e.E_V3 AS V FROM CO2.ELEMENTS e) UNION
54          (SELECT e.E_ID, e.E_V4 AS V FROM CO2.ELEMENTS e) UNION
55          (SELECT e.E_ID, e.E_V5 AS V FROM CO2.ELEMENTS e) UNION
56          (SELECT e.E_ID, e.E_V6 AS V FROM CO2.ELEMENTS e) UNION
57          (SELECT e.E_ID, e.E_V7 AS V FROM CO2.ELEMENTS e) UNION
58          (SELECT e.E_ID, e.E_V8 AS V FROM CO2.ELEMENTS e)) e1,
59         ((SELECT e.E_ID, e.E_V1 AS V FROM CO2.ELEMENTS e) UNION
60          (SELECT e.E_ID, e.E_V2 AS V FROM CO2.ELEMENTS e) UNION
61          (SELECT e.E_ID, e.E_V3 AS V FROM CO2.ELEMENTS e) UNION
62          (SELECT e.E_ID, e.E_V4 AS V FROM CO2.ELEMENTS e) UNION
63          (SELECT e.E_ID, e.E_V5 AS V FROM CO2.ELEMENTS e) UNION
64          (SELECT e.E_ID, e.E_V6 AS V FROM CO2.ELEMENTS e) UNION
65          (SELECT e.E_ID, e.E_V7 AS V FROM CO2.ELEMENTS e) UNION
66          (SELECT e.E_ID, e.E_V8 AS V FROM CO2.ELEMENTS e)) e2
67  WHERE e1.V = e2.V AND e1.E_ID < e2.E_ID
68  GROUP BY e1.E_ID, e2.E_ID
69  HAVING COUNT(*) = 4
70 -- [CO2] Drop Schema
71 DROP TABLE CO2.ELEMENTS
72 DROP TABLE CO2.VERTICES
73 DROP TABLE CO2.IMPORT_PROPERTIES
74 DROP SCHEMA CO2 RESTRICT
```

A.3.2 XML Performance Queries

This appendix lists tests and queries used in the XML benchmark. The definition of all queries can also be found in `[DVD]/Sources/SQL/Evaluation/evaluation-data-XMark.sql`. The XML benchmark respects different XQuery dialects and idiosyncrasies, while the following statements are specified to work with eXist-db.

The purpose of primitive XQuery statements is often stated by the name of the test in the following listing. XMark queries have the original number of the published benchmark query in their title and are explained by [SWK+01].

```
1 -- [XMARK] Load 0.02 no split
2 xmldb:store("idares-xml-collection", "content.xml", xs:anyURI("file:///media/
      idares-evaluation/benchmark/XMark/scale-0.02_no-split/content.xml"))
3
4 -- [XMARK] Query Collections
5 xmldb:get-child-collections("/")
6
7 -- [XMARK] Query Documents
8 for $doc in collection("idares-xml-collection")
9  return util:document-name($doc/child::*[1])
10
11 -- [XMARK] Query All Element Count
12 count(fn:collection("idares-xml-collection")/descendant::node())
13
14 -- [XMARK] Query Person Element Count
15 count(fn:collection("idares-xml-collection")//child::person)
16
17 -- [XMARK] Query Retrieve Regions
18 fn:collection("idares-xml-collection")/child::site/child::regions/child::europe
19
20 -- [XMARK] Query Text Analysis
21 for $text in fn:collection("idares-xml-collection")//child::*/child::text()
22 let $subtext := fn:replace(fn:replace($text,"\s+$",""),"^\s+","")
23   where fn:string-length($subtext) ge 10 and fn:contains($subtext, "sees")
24   order by fn:string-length($subtext) descending, $subtext ascending
25   return <text lengt="{fn:string-length($subtext)}">{fn:concat("sees",
         fn:substring(fn:substring-after($subtext, "sees"), 0, 50))}</text>
26
27 -- [XMARK] Query XMark 1
28 let $auction := fn:collection("idares-xml-collection")
29 for $p in $auction/site/people/person[@id = "person0"]
30   return $p/name/text()
31
32 -- [XMARK] Query XMark 2
33 let $auction := fn:collection("idares-xml-collection")
34 for $b in $auction/site/open_auctions/open_auction
35   return <increase>{$b/bidder[1]/increase/text()}</increase>
36
37 -- [XMARK] Query XMark 3
38 let $auction := fn:collection("idares-xml-collection")
39 for $b in $auction/site/open_auctions/open_auction
40   where zero-or-one($b/bidder[1]/increase/text()) * 2 <=
         $b/bidder[last()]/increase/text()
41   return <increase first="{$b/bidder[1]/increase/text()}"
42                    last="{$b/bidder[last()]/increase/text()}"/>
43
44 -- [XMARK] Query XMark 6
45 let $auction := fn:collection("idares-xml-collection") return
46 for $b in $auction/site/regions return count($b//item)
47
48 -- [XMARK] Query XMark 8
49 let $auction := fn:collection("idares-xml-collection")
50 for $p in $auction/site/people/person
51 let $a := count(for $t in $auction/site/closed_auctions/closed_auction
52                   where $t/buyer/@person eq $p/@id
53                   return $t)
54   where $a > 0
55   order by $a descending
```

```
56   return <item person="{$p/name/text()}">{$a}</item>
57
58 -- [XMARK] Query XMark 9
59 let $auction := fn:collection("idares-xml-collection")
60 let $ca := $auction/site/closed_auctions/closed_auction
61 let $ei := $auction/site/regions/child::europe/item
62 for $p in $auction/site/people/person
63 let $p_name := exactly-one($p/name/text())
64 let $a :=
65   for $t in $ca
66     where $p/@id = $t/buyer/@person
67     return let $n := for $t2 in $ei
68                      where $t/itemref/@item = $t2/@id
69                      return $t2
70              where count($n) > 0
71              return <item>{$n/name/text()}</item>
72   where count($a) > 0
73   order by count($a) descending, $p_name ascending
74   return <person name="{$p_name}">{$a}</person>
75
76 -- [XMARK] Query XMark 10
77 let $auction := fn:collection("idares-xml-collection")
78 for $i in distinct-values($auction/site/people/person/profile/interest/@category)
79 let $p :=
80   for $t in $auction/site/people/person
81   where $t/profile/interest/@category = $i
82   return
83     <personne>
84       <statistiques>
85         <sexe>{$t/profile/gender/text()}</sexe>
86         <age>{$t/profile/age/text()}</age>
87         <education>{$t/profile/education/text()}</education>
88      <revenu>{fn:data($t/profile/@income)}</revenu>
89       </statistiques>
90       <coordonnees>
91         <nom>{$t/name/text()}</nom>
92         <rue>{$t/address/street/text()}</rue>
93         <ville>{$t/address/city/text()}</ville>
94         <pays>{$t/address/country/text()}</pays>
95         <reseau>
96           <courrier>{$t/emailaddress/text()}</courrier>
97           <pagePerso>{$t/homepage/text()}</pagePerso>
98         </reseau>
99       </coordonnees>
100      <cartePaiement>{$t/creditcard/text()}</cartePaiement>
101    </personne>
102 return <categorie>{<id>{$i}</id>, $p}</categorie>
103
104 -- [XMARK] Query XMark 11
105 let $auction := fn:collection("idares-xml-collection") return
106 for $p in $auction/site/people/person
107 let $l :=
108   for $i in $auction/site/open_auctions/open_auction/initial
109   where $p/profile/@income > 5000 * exactly-one($i/text())
110   return $i
111 return <items name="{$p/name/text()}">{count($l)}</items>
112
113 -- [XMARK] Query XMark 14
114 let $auction := fn:collection("idares-xml-collection") return
115 for $i in $auction/site//item
116 where contains(string(exactly-one($i/description)), "gold")
117 return $i/name/text()
```

```
118
119 -- [XMARK] Query XMark 16
120 let $auction := fn:collection("idares-xml-collection") return
121 for $a in $auction/site/closed_auctions/closed_auction
122 where
123   not(
124     empty(
125       $a/annotation/description/parlist/listitem/parlist/listitem/text/emph/
126        keyword/
127        text()
128     )
129   )
130 return <person id="{$a/seller/@person}"/>
131
132 -- [XMARK] Query XMark 18
133 declare namespace local = "http://www.foobar.org";
134 declare function local:convert($v as xs:decimal?) as xs:decimal?
135 {
136   2.20371 * $v (: convert Dfl to Euro :)
137 };
138 let $auction := fn:collection("idares-xml-collection")
139 for $i in $auction/site/open_auctions/open_auction
140   return local:convert(zero-or-one($i/reserve))
141
142 -- [XMARK] Query XMark 19
143 let $auction := fn:collection("idares-xml-collection")
144 for $b in $auction/site/regions//item
145 let $k := $b/name/text()
146  order by zero-or-one($b/location) ascending empty greatest
147  return <item name="{$k}">{$b/location/text()}</item>
148
149 -- [XMARK] Query XMark 20
150 let $auction := fn:collection("idares-xml-collection") return
151 <result>
152   <preferred>
153     {count($auction/site/people/person/profile[@income >= 100000])}
154   </preferred>
155   <standard>
156     {count($auction/site/people/person/profile[@income < 100000 and @income >=
          30000])}
157   </standard>
158   <challenge>
159     {count($auction/site/people/person/profile[@income < 30000])}
160   </challenge>
161   <na>
162     {count(for $p in $auction/site/people/person
163               where empty($p/profile/@income)
164               return $p)}
165  </na>
166 </result>
167
168 -- [XMARK] Drop Collection
169 for $doc in collection("idares-xml-collection")/child::*
170  return xmldb:remove("idares-xml-collection", util:document-name($doc))
```

A.4 IDARES Web Services

In the following sections, the XML schema and the WSDL↗ file specifying the *IDARES Web Service* are attached.

A.4.1 XML Schema for the Messages

The following listing shows the content of `[DVD]/Sources/Projects/Idares/IdaresEvalProcess/idares-model.xsd`.

```
 1 <?xml version="1.0" encoding="UTF-8"?>
 2 <!-- $Id: idares-model.xsd 3413 2010-01-27 09:27:47Z ahija $ -->
 3 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 4            xmlns:im="http://ahija.de/thesis/idares/model"
 5            elementFormDefault="qualified"
 6            targetNamespace="http://ahija.de/thesis/idares/model">
 7
 8   <xs:annotation>
 9     <xs:documentation>
10   This XML schema describes the required XML elements and type for the IDARES
11   Web Services.
12
13   IDARES stands for "Integrated Database Architecture for a Runtime Environment
14   for Simulation Workflows" and was developed during the diploma thesis of
15   Christoph Mueller in 2009 at the University of Stuttgart.
16
17   Throughout the following elements often so called type names are used. The
18   following table shows an overview
19
20   |  Type Name |           Java Class |  SQL Type |      Sample XML Instance |
21   |    boolean |    java.lang.Boolean |   BOOLEAN |                     true |
22   |       byte |       java.lang.Byte |   TINYINT |                      -84 |
23   |      short |      java.lang.Short |  SMALLINT |                       14 |
24   |    integer |    java.lang.Integer |   INTEGER |                       17 |
25   |       long |       java.lang.Long |    BIGINT |                       18 |
26   |      float |      java.lang.Float |     FLOAT |                     3.14 |
27   |     double |     java.lang.Double |    DOUBLE |                    2.718 |
28   |       real |     java.lang.Double |      REAL |                     1.51 |
29   |    decimal | java.math.BigDecimal |   DECIMAL |                     8.25 |
30   | bigdecimal | java.math.BigDecimal |   NUMERIC |                  1288.75 |
31   |    numeric | java.math.BigInteger |   NUMERIC |                      128 |
32   |     string |     java.lang.String |   VARCHAR |                    Hello |
33   |      chars |     java.lang.String |      CHAR |                      Bye |
34   |       char |  java.lang.Character |      CHAR |                        Z |
35   |       clob |        java.sql.Clob |      CLOB |                    Hello |
36   |     binary |               byte[] | VARBINARY | base64      w7bDpM08w58= |
37   |            |                      |           | for CSV use hex encoding |
38   |       blob |        java.sql.Blob |      BLOB | base64      w7bDpM08w58= |
39   |            |                      |           | for CSV use hex encoding |
40   |  timestamp |       java.util.Date | TIMESTAMP | 2009-02-15T19:43:15.897Z |
41   |   datetime |       java.util.Date | TIMESTAMP | 2009-02-15T19:43:15.897Z |
42   |       date |       java.util.Date |      DATE |               2009-02-15 |
43   |       time |       java.util.Date |      TIME |             19:43:15.897 |
44     </xs:documentation>
45   </xs:annotation>
46
47   <!--
```

```
48    ! GenericDatabasePortType#executeQueryViaDataSource
49    !-->
50
51    <xs:element name="executeQueryViaDataSourceRequest">
52      <xs:complexType>
53        <xs:sequence>
54          <xs:element name="datasourceName" type="xs:string" />
55          <xs:element name="maxRows" type="xs:unsignedInt" minOccurs="0"/>
56          <xs:element name="isolation" type="im:transactionIsolationSType"
                minOccurs="0"/>
57          <xs:element name="queries" type="im:queriesType" />
58        </xs:sequence>
59      </xs:complexType>
60    </xs:element>
61
62    <xs:complexType name="queriesType">
63      <xs:sequence>
64        <xs:element name="preparationQuery" type="im:queryType" minOccurs="0"
              maxOccurs="unbounded" />
65        <xs:element name="focusedQuery" type="im:queryWithResultType" />
66        <xs:element name="cleanupQuery" type="im:queryType" minOccurs="0"
              maxOccurs="unbounded" />
67      </xs:sequence>
68    </xs:complexType>
69
70    <xs:complexType name="queryType">
71      <xs:sequence>
72        <xs:element name="queryLanguage" type="im:queryLanguageSType" />
73        <xs:element name="queryType" type="im:queryTypeSType" />
74        <xs:element name="query" type="xs:string" />
75        <xs:element name="parameterBind" type="im:parameterBindType" minOccurs="0"
              maxOccurs="unbounded"/>
76      </xs:sequence>
77    </xs:complexType>
78
79    <xs:complexType name="queryWithResultType">
80      <xs:complexContent>
81        <xs:extension base="im:queryType">
82          <xs:sequence>
83            <xs:element name="resultColumnName" type="xs:string" minOccurs="0"
                  maxOccurs="unbounded" />
84          </xs:sequence>
85        </xs:extension>
86      </xs:complexContent>
87    </xs:complexType>
88
89    <xs:complexType name="parameterBindType">
90      <xs:annotation><xs:documentation>
91        This type is used to set a ? parameter within a JDBC statement.
92        See http://java.sun.com/j2se/1.5.0/docs/api/java/sql/PreparedStatement.html
93        For large binary and textual content (dataType blob or clob) the binaryContent
              or charContent
94        element HAVE to be used instead of simpleContent.
95      </xs:documentation></xs:annotation>
96      <xs:choice>
97        <xs:element name="charContent" type="im:characterSourceType">
98          <xs:annotation><xs:documentation>
99            This element is used to refer to character content. dataType has to be
                  clob!
100         </xs:documentation></xs:annotation>
101       </xs:element>
102       <xs:element name="binaryContent" type="im:binarySourceType">
```

```
103         <xs:annotation><xs:documentation>
104           This element is used to refer to binary content. dataType has to be blob!
105         </xs:documentation></xs:annotation>
106       </xs:element>
107       <xs:element name="simpleContent" type="xs:string" />
108     </xs:choice>
109     <xs:attribute name="dataType" type="xs:string" use="required" />
110     <xs:attribute name="isNull" type="xs:boolean" use="optional" />
111   </xs:complexType>
112
113   <xs:simpleType name="queryLanguageSType">
114     <xs:restriction base="xs:string">
115       <xs:enumeration value="SQL" />
116       <xs:enumeration value="XQUERY" />
117       <xs:enumeration value="XMLDB" />
118     </xs:restriction>
119   </xs:simpleType>
120
121   <xs:simpleType name="queryTypeSType">
122     <xs:restriction base="xs:string">
123       <xs:enumeration value="SELECT" />
124       <xs:enumeration value="UPDATE" />
125       <xs:enumeration value="INSERT" />
126       <xs:enumeration value="DELETE" />
127       <xs:enumeration value="DDL" />
128       <xs:enumeration value="OTHER" />
129     </xs:restriction>
130   </xs:simpleType>
131
132   <xs:element name="executeQueryViaDataSourceResponse">
133     <xs:complexType>
134       <xs:sequence>
135         <xs:element name="headerRow" type="im:headerRowType" />
136         <xs:element name="row" type="im:rowType" minOccurs="0"
              maxOccurs="unbounded"/>
137       </xs:sequence>
138       <xs:attribute name="selectExecutionDuration" type="xs:unsignedLong"
            use="required"/>
139       <xs:attribute name="overallExecutionDuration" type="xs:unsignedLong"
            use="required"/>
140       <xs:attribute name="rowCount" type="xs:unsignedInt" use="required"/>
141     </xs:complexType>
142   </xs:element>
143
144   <xs:complexType name="headerRowType">
145     <xs:sequence>
146       <xs:element name="columnDescriptor" type="im:columnDescriptorType"
            minOccurs="1" maxOccurs="unbounded" />
147     </xs:sequence>
148   </xs:complexType>
149
150   <xs:complexType name="columnDescriptorType">
151     <xs:attribute name="columnName" type="xs:string" use="required"/>
152     <xs:attribute name="sqlDataType" type="xs:string" use="required"/>
153     <xs:attribute name="dataTypeName" type="xs:string" use="required"/>
154     <xs:attribute name="precision" type="xs:int"/>
155     <xs:attribute name="scale" type="xs:int"/>
156   </xs:complexType>
157
158   <xs:complexType name="rowType">
159     <xs:sequence>
160       <xs:any minOccurs="1" maxOccurs="unbounded"/>
```

```
161     </xs:sequence>
162     <xs:attribute name="rowNumber" type="xs:unsignedInt" use="required" />
163   </xs:complexType>
164
165   <!--
166    ! GenericDatabasePortType#executeViaDataSource
167    !-->
168   <xs:element name="executeViaDataSourceRequest">
169     <xs:complexType>
170       <xs:sequence>
171         <xs:element name="datasourceName" type="xs:string" />
172         <xs:element name="isolation" type="im:transactionIsolationSType"
                minOccurs="0"/>
173         <xs:element name="query" type="im:queryType" minOccurs="1"
                maxOccurs="unbounded" />
174       </xs:sequence>
175     </xs:complexType>
176   </xs:element>
177
178   <xs:element name="executeViaDataSourceResponse">
179     <xs:complexType>
180       <xs:attribute name="overallExecutionDuration" type="xs:unsignedLong"
            use="required"/>
181       <xs:attribute name="affectedRowCount" type="xs:unsignedInt" use="required"/>
182     </xs:complexType>
183   </xs:element>
184
185   <!--
186    ! GenericDatabasePortType#executeInsertViaDataSource
187    !-->
188   <xs:element name="executeInsertViaDataSourceRequest">
189     <xs:complexType>
190       <xs:sequence>
191         <xs:element name="datasourceName" type="xs:string" />
192         <xs:element name="processNamePattern" type="xs:string" minOccurs="0"
                default=""/>
193         <xs:element name="isolation" type="im:transactionIsolationSType"
                minOccurs="0"/>
194         <xs:element name="insert" type="im:queryType" />
195         <xs:element name="generatedColumnName" type="xs:string" minOccurs="0"
                maxOccurs="unbounded" />
196       </xs:sequence>
197     </xs:complexType>
198   </xs:element>
199
200   <xs:element name="executeInsertViaDataSourceResponse">
201     <xs:complexType>
202       <xs:sequence>
203         <xs:element name="memoryDelta" type="im:memoryDeltaType" />
204         <xs:element name="generatedKey" type="im:generatedKeyType" minOccurs="0"
                maxOccurs="unbounded" />
205       </xs:sequence>
206       <xs:attribute name="overallExecutionDuration" type="xs:unsignedLong"/>
207     </xs:complexType>
208   </xs:element>
209
210   <xs:complexType name="generatedKeyType">
211     <xs:simpleContent>
212       <xs:extension base="xs:string">
213         <xs:attribute name="columnName" type="xs:string" use="required"/>
214         <xs:attribute name="sqlDataType" type="xs:int" use="required"/>
215         <xs:attribute name="dataTypeName" type="xs:string" use="required"/>
```

```
216         <xs:attribute name="precision" type="xs:int"/>
217         <xs:attribute name="scale" type="xs:int"/>
218         <xs:attribute name="isNull" type="xs:boolean"/>
219       </xs:extension>
220     </xs:simpleContent>
221   </xs:complexType>
222
223   <!--
224    ! GenericDatabasePortType#measurePerformanceViaDataSource
225    !-->
226   <xs:element name="measurePerformanceViaDataSourceRequest">
227     <xs:complexType>
228       <xs:sequence>
229         <xs:element name="datasourceName" type="xs:string" />
230         <xs:element name="processNamePattern" type="xs:string" minOccurs="0"
              default=""/>
231         <xs:element name="isolation" type="im:transactionIsolationSType"
              minOccurs="0"/>
232         <xs:element name="queries" type="im:queriesType" />
233       </xs:sequence>
234       <xs:attribute name="logErrors" type="xs:boolean" use="optional"
            default="true"/>
235     </xs:complexType>
236   </xs:element>
237
238   <xs:element name="measurePerformanceViaDataSourceResponse">
239     <xs:complexType>
240       <xs:sequence>
241         <xs:element name="memoryDelta" type="im:memoryDeltaType" />
242       </xs:sequence>
243       <xs:attribute name="beginTS" type="xs:dateTime" use="required"/>
244       <xs:attribute name="endTS" type="xs:dateTime" use="required"/>
245       <xs:attribute name="focusedExecutionDuration" type="xs:unsignedLong"
              use="required"/>
246       <xs:attribute name="otherExecutionDuration" type="xs:unsignedLong"
              use="required"/>
247       <xs:attribute name="overallExecutionDuration" type="xs:unsignedLong"
              use="required"/>
248       <xs:attribute name="commitExecutionDuration" type="xs:unsignedLong"
              use="required"/>
249     </xs:complexType>
250   </xs:element>
251
252   <xs:complexType name="memoryDeltaType">
253     <xs:attribute name="residentSetSizeBefore" type="xs:unsignedLong" use="required">
254       <xs:annotation>
255         <xs:documentation>Resident set size: the non-swapped physical memory that a
              task has used in KiB</xs:documentation>
256       </xs:annotation>
257     </xs:attribute>
258     <xs:attribute name="residentSetSizeAfter" type="xs:unsignedLong" use="required"
          />
259     <xs:attribute name="virtualMemorySizeBefore" type="xs:unsignedLong"
          use="required">
260       <xs:annotation>
261         <xs:documentation>Virtual memory size of the process in
              KiB</xs:documentation>
262       </xs:annotation>
263     </xs:attribute>
264     <xs:attribute name="virtualMemorySizeAfter" type="xs:unsignedLong"
          use="required" />
265   </xs:complexType>
```

```
266
267  <!--
268   ! GenericDatabasePortType#copyContentViaDataSource
269   !-->
270  <xs:element name="copyContentViaDataSourceRequest">
271    <xs:complexType>
272      <xs:sequence>
273        <xs:element name="sourceDatasource" type="xs:string" />
274        <xs:element name="targetDatasource" type="xs:string" />
275        <xs:element name="targetProcessNamePattern" type="xs:string" minOccurs="0"
               default=""/>
276        <xs:element name="targetBeforeQuery" type="im:queryType" minOccurs="0"
               maxOccurs="unbounded"/>
277        <xs:element name="sourceRetrieveQuery" type="im:queryType" />
278        <xs:element name="targetInsertQuery" type="im:queryType" />
279        <xs:element name="targetAfterQuery" type="im:queryType" minOccurs="0"
               maxOccurs="unbounded"/>
280      </xs:sequence>
281      <xs:attribute name="sourceMaxRows" type="xs:unsignedInt" use="optional" />
282      <xs:attribute name="commitInterval" type="xs:unsignedInt" use="optional"
             default="1000" />
283    </xs:complexType>
284  </xs:element>
285
286  <xs:element name="copyContentViaDataSourceResponse">
287    <xs:complexType>
288      <xs:sequence>
289        <xs:element name="memoryDelta" type="im:memoryDeltaType" />
290        <xs:element name="warnings" type="im:failureContainerType" minOccurs="0"
               maxOccurs="unbounded" />
291      </xs:sequence>
292      <xs:attribute name="beginTS" type="xs:dateTime" use="required"/>
293      <xs:attribute name="endTS" type="xs:dateTime" use="required"/>
294      <xs:attribute name="retrieveExecutionDuration" type="xs:unsignedLong"
             use="required"/>
295      <xs:attribute name="insertExecutionDuration" type="xs:unsignedLong"
             use="required"/>
296      <xs:attribute name="overallExecutionDuration" type="xs:unsignedLong"
             use="required"/>
297      <xs:attribute name="copiedRows" type="xs:unsignedLong" use="required"/>
298    </xs:complexType>
299  </xs:element>
300
301  <!--
302   ! GenericDatabasePortType#importCSVToDataSource
303   !-->
304  <xs:element name="importCSVToDataSourceRequest">
305    <xs:complexType>
306      <xs:sequence>
307        <xs:element name="csvSource" type="im:characterSourceType" />
308        <xs:element name="targetDatasource" type="xs:string" />
309        <xs:element name="targetProcessNamePattern" type="xs:string" minOccurs="0"
               default=""/>
310        <xs:element name="targetBeforeQuery" type="im:queryType" minOccurs="0"
               maxOccurs="unbounded"/>
311        <xs:choice>
312          <xs:element name="targetInsertQuery" type="im:queryType">
313            <xs:annotation>
314              <xs:documentation>
315                An insert query of the form
316                INSERT INTO SCHEMA.TABE (NAME, VALUE) VALUES(?, ?)
317                The JDBC driver has to support parameter meta data in order to
```

```
318                 get column types.
319               </xs:documentation>
320             </xs:annotation>
321           </xs:element>
322           <xs:element name="targetTable">
323             <xs:complexType>
324               <xs:sequence>
325                 <xs:element name="tableName" type="xs:string" />
326                 <xs:element name="column" type="xs:string" minOccurs="0"
                        maxOccurs="unbounded">
327                   <xs:annotation>
328                     <xs:documentation>
329                       List the target column names here, to control the order in
330                       which the CSV columns are stored into the table columns.
331                       If NO column is present, all columns of the table are used
332                       in the order they are declared.
333                     </xs:documentation>
334                   </xs:annotation>
335                 </xs:element>
336               </xs:sequence>
337             </xs:complexType>
338           </xs:element>
339         </xs:choice>
340         <xs:element name="targetAfterQuery" type="im:queryType" minOccurs="0"
              maxOccurs="unbounded"/>
341       </xs:sequence>
342       <xs:attribute name="sourceMaxRows" type="xs:int" use="optional">
343        <xs:annotation>
344          <xs:documentation>How many rows should be copied at maximum? If &lt; 0,
               this parameter is ignored.</xs:documentation>
345        </xs:annotation>
346       </xs:attribute>
347       <xs:attribute name="commitInterval" type="xs:unsignedInt" use="optional"
            default="1000" />
348       <xs:attribute name="hasHeader" type="xs:boolean" use="optional"
            default="false">
349         <xs:annotation>
350           <xs:documentation>If the CSV file has a header row, it is skipped for
                import.</xs:documentation>
351         </xs:annotation>
352       </xs:attribute>
353       <xs:attribute name="separatorChar" type="im:characterSType" use="optional"
            default=";">
354         <xs:annotation>
355           <xs:documentation>Which separator should be used to split the
                columns?</xs:documentation>
356         </xs:annotation>
357       </xs:attribute>
358       <xs:attribute name="quoteChar" type="im:characterSType" use="optional"
            default="&quot;">
359         <xs:annotation>
360           <xs:documentation>Which separator is used for text quotation start and
                end?</xs:documentation>
361         </xs:annotation>
362       </xs:attribute>
363       <xs:attribute name="escapeChar" type="im:characterSType" use="optional"
            default="\">
364         <xs:annotation>
365           <xs:documentation>Which escape character is used to escape the separator
                or quote character?</xs:documentation>
366         </xs:annotation>
367       </xs:attribute>
```

```
368       <xs:attribute name="nullString" type="xs:string" use="optional" default="">
369         <xs:annotation>
370           <xs:documentation>If a cell has this value it is interpreted as a NULL
                  value.</xs:documentation>
371         </xs:annotation>
372       </xs:attribute>
373       <xs:attribute name="skipMultipleSeparators" type="xs:boolean" use="optional"
              default="false">
374         <xs:annotation>
375           <xs:documentation>If two or multiple separators follow each other, they
                  are ignored and treated as one.</xs:documentation>
376         </xs:annotation>
377       </xs:attribute>
378       <xs:attribute name="datePattern" type="xs:string" use="optional"
              default="yyyy-MM-dd">
379         <xs:annotation>
380           <xs:documentation>See java.text.SimpleDateFormat for pattern
                  description!</xs:documentation>
381         </xs:annotation>
382       </xs:attribute>
383       <xs:attribute name="timePattern" type="xs:string" use="optional"
              default="HH:mm:ss">
384         <xs:annotation>
385           <xs:documentation>See java.text.SimpleDateFormat for pattern
                  description!</xs:documentation>
386         </xs:annotation>
387       </xs:attribute>
388       <xs:attribute name="timestampPattern" type="xs:string" use="optional"
              default="yyyy-MM-dd HH:mm:ss.SSS">
389         <xs:annotation>
390           <xs:documentation>See java.text.SimpleDateFormat for pattern
                  description!</xs:documentation>
391         </xs:annotation>
392       </xs:attribute>
393     </xs:complexType>
394   </xs:element>
395
396   <xs:complexType name="characterSourceType">
397     <xs:annotation>
398       <xs:documentation>This XSD type is used to refer to a large text, either by
              its character content,
399                         by its binary content and a character set, or via an URL and
                              a character set.
400       </xs:documentation>
401     </xs:annotation>
402     <xs:choice>
403       <xs:element name="sourceURL">
404         <xs:complexType>
405           <xs:simpleContent>
406             <xs:extension base="xs:anyURI">
407               <xs:attribute name="charset" type="xs:string" use="optional"
                      default="ISO-8859-1" />
408             </xs:extension>
409           </xs:simpleContent>
410         </xs:complexType>
411       </xs:element>
412       <xs:element name="hexBinary">
413         <xs:complexType>
414           <xs:simpleContent>
415             <xs:extension base="xs:hexBinary">
416               <xs:attribute name="charset" type="xs:string" use="optional"
                      default="ISO-8859-1" />
```

```
417             </xs:extension>
418           </xs:simpleContent>
419         </xs:complexType>
420       </xs:element>
421       <xs:element name="base64Binary">
422         <xs:complexType>
423           <xs:simpleContent>
424             <xs:extension base="xs:base64Binary">
425               <xs:attribute name="charset" type="xs:string" use="optional"
                      default="ISO-8859-1" />
426             </xs:extension>
427           </xs:simpleContent>
428         </xs:complexType>
429       </xs:element>
430       <xs:element name="cdataContent" type="xs:string" />
431     </xs:choice>
432   </xs:complexType>
433
434   <xs:complexType name="binarySourceType">
435     <xs:annotation>
436       <xs:documentation>This XSD type is used to refer to binary data, either by its
              content or via an URL.
437       </xs:documentation>
438     </xs:annotation>
439     <xs:choice>
440       <xs:element name="sourceURL" type="xs:anyURI" />
441       <xs:element name="hexBinary" type="xs:hexBinary" />
442       <xs:element name="base64Binary" type="xs:base64Binary" />
443     </xs:choice>
444   </xs:complexType>
445
446   <xs:simpleType name="characterSType">
447     <xs:restriction base="xs:string">
448       <xs:length value="1" />
449     </xs:restriction>
450   </xs:simpleType>
451
452   <xs:element name="importCSVToDataSourceResponse">
453     <xs:complexType>
454       <xs:sequence>
455         <xs:element name="memoryDelta" type="im:memoryDeltaType" />
456         <xs:element name="warnings" type="im:failureContainerType" minOccurs="0"
              maxOccurs="unbounded" />
457       </xs:sequence>
458       <xs:attribute name="beginTS" type="xs:dateTime" use="required"/>
459       <xs:attribute name="endTS" type="xs:dateTime" use="required"/>
460       <xs:attribute name="readingExecutionDuration" type="xs:unsignedLong"
            use="required"/>
461       <xs:attribute name="insertExecutionDuration" type="xs:unsignedLong"
            use="required"/>
462       <xs:attribute name="overallExecutionDuration" type="xs:unsignedLong"
            use="required"/>
463       <xs:attribute name="copiedRows" type="xs:unsignedLong" use="required"/>
464     </xs:complexType>
465   </xs:element>
466
467   <!--
468    ! GenericDatabasePortType#importCSVToDataSource
469    !-->
470   <xs:element name="exportFromDataSourceToCSVRequest">
471     <xs:complexType>
472       <xs:sequence>
```

```
473         <xs:element name="datasourceName" type="xs:string" />
474         <xs:element name="processNamePattern" type="xs:string" minOccurs="0"
                default=""/>
475         <xs:element name="beforeQuery" type="im:queryType" minOccurs="0"
                maxOccurs="unbounded"/>
476         <xs:element name="retrieveQuery" type="im:queryType">
477           <xs:annotation>
478             <xs:documentation>A SELECT query with a ResultSet.</xs:documentation>
479           </xs:annotation>
480         </xs:element>
481         <xs:element name="afterQuery" type="im:queryType" minOccurs="0"
                maxOccurs="unbounded"/>
482         <xs:element name="csvTargetURL" type="xs:anyURI">
483           <xs:annotation>
484             <xs:documentation>
485               This can be a "file://path", where the user role of the web server
486               is able to write to. It can be a "ftp://user:password@server/path"
487               or an arbitrary "http://server/path", that accepts files as post.
488             </xs:documentation>
489           </xs:annotation>
490         </xs:element>
491         <xs:element  name="charset" type="xs:string" minOccurs="0"
                default="ISO-8859-1" />
492       </xs:sequence>
493       <xs:attribute name="sourceMaxRows" type="xs:int" use="optional">
494        <xs:annotation>
495          <xs:documentation>How many rows should be retrieved at maximum? If &lt; 0,
                this parameter is ignored.</xs:documentation>
496        </xs:annotation>
497       </xs:attribute>
498       <xs:attribute name="produceHeader" type="xs:boolean" use="optional"
            default="false">
499         <xs:annotation>
500           <xs:documentation>If the true, the CSV file has a header row with the
                column names.</xs:documentation>
501         </xs:annotation>
502       </xs:attribute>
503       <xs:attribute name="separatorChar" type="im:characterSType" use="optional"
            default=";">
504         <xs:annotation>
505           <xs:documentation>Which separator should be used to split the
                columns?</xs:documentation>
506         </xs:annotation>
507       </xs:attribute>
508       <xs:attribute name="quoteChar" type="im:characterSType" use="optional"
            default="&quot;">
509         <xs:annotation>
510           <xs:documentation>Which separator is used for text quotation start and
                end?</xs:documentation>
511         </xs:annotation>
512       </xs:attribute>
513       <xs:attribute name="escapeChar" type="im:characterSType" use="optional"
            default="\">
514         <xs:annotation>
515           <xs:documentation>Which escape character is used to escape the separator
                or quote character?</xs:documentation>
516         </xs:annotation>
517       </xs:attribute>
518       <xs:attribute name="nullString" type="xs:string" use="optional" default="">
519         <xs:annotation>
520           <xs:documentation>If a cell has is NULL, this value it is written into the
                CSV file.</xs:documentation>
```

```
521         </xs:annotation>
522       </xs:attribute>
523       <xs:attribute name="datePattern" type="xs:string" use="optional"
              default="yyyy-MM-dd">
524         <xs:annotation>
525           <xs:documentation>See java.text.SimpleDateFormat for pattern
                  description!</xs:documentation>
526         </xs:annotation>
527       </xs:attribute>
528       <xs:attribute name="timePattern" type="xs:string" use="optional"
              default="HH:mm:ss">
529         <xs:annotation>
530           <xs:documentation>See java.text.SimpleDateFormat for pattern
                  description!</xs:documentation>
531         </xs:annotation>
532       </xs:attribute>
533       <xs:attribute name="timestampPattern" type="xs:string" use="optional"
              default="yyyy-MM-dd HH:mm:ss.SSS">
534         <xs:annotation>
535           <xs:documentation>See java.text.SimpleDateFormat for pattern
                  description!</xs:documentation>
536         </xs:annotation>
537       </xs:attribute>
538     </xs:complexType>
539   </xs:element>
540
541   <xs:element name="exportFromDataSourceToCSVResponse">
542     <xs:complexType>
543       <xs:sequence>
544         <xs:element name="memoryDelta" type="im:memoryDeltaType" />
545         <xs:element name="warnings" type="im:failureContainerType" minOccurs="0"
              maxOccurs="unbounded" />
546       </xs:sequence>
547       <xs:attribute name="beginTS" type="xs:dateTime" use="required"/>
548       <xs:attribute name="endTS" type="xs:dateTime" use="required"/>
549       <xs:attribute name="retrievalExecutionDuration" type="xs:unsignedLong"
              use="required"/>
550       <xs:attribute name="writingExecutionDuration" type="xs:unsignedLong"
              use="required"/>
551       <xs:attribute name="overallExecutionDuration" type="xs:unsignedLong"
              use="required"/>
552       <xs:attribute name="retrievedRows" type="xs:unsignedLong" use="required"/>
553     </xs:complexType>
554   </xs:element>
555
556   <!--
557    ! GenericDatabasePortType#exportFromDataSourceToFile
558    !-->
559   <xs:element name="exportFromDataSourceToFileRequest">
560     <xs:annotation>
561       <xs:documentation>Exports the content of a table cell into a file. Can be used
              with language = XQUERY
562                         too. If the result is empty, an empty file is created. The
                              target file is overwritten
563                         in any case.
564                         Any data type can be retrieved and stored into the file. It
                              will be transformed into
565                         a String and then written using the charset
566       </xs:documentation>
567     </xs:annotation>
568     <xs:complexType>
569       <xs:sequence>
```

```
570         <xs:element name="sourceDatasource" type="xs:string" />
571         <xs:element name="sourceProcessNamePattern" type="xs:string" minOccurs="0"
                default=""/>
572         <xs:element name="retreiveQuery" type="im:queryType">
573           <xs:annotation>
574             <xs:documentation>
575               A SELECT query that retrieves exactly one cell (one column, one row).
                      More columns or rows
576               are ignored.
577             </xs:documentation>
578           </xs:annotation>
579         </xs:element>
580         <xs:element name="targetURL" type="xs:anyURI">
581           <xs:annotation>
582             <xs:documentation>
583               This can be a "file://path", where the user role of the web server
584               is able to write to. It can be a "ftp://user:password@server/path"
585               or an arbitrary "http://server/path", that accepts files as post.
586             </xs:documentation>
587           </xs:annotation>
588         </xs:element>
589         <xs:element name="charset" type="xs:string" minOccurs="0"
                default="ISO-8859-1">
590           <xs:annotation>
591             <xs:documentation>The charset is used, when non-binary data is retrieved
                    and has to be written
592                               as binary data.</xs:documentation>
593           </xs:annotation>
594         </xs:element>
595       </xs:sequence>
596     </xs:complexType>
597   </xs:element>
598
599   <xs:element name="exportFromDataSourceToFileResponse">
600     <xs:complexType>
601       <xs:sequence>
602         <xs:element name="memoryDelta" type="im:memoryDeltaType" />
603         <xs:element name="warnings" type="im:failureContainerType" minOccurs="0"
                maxOccurs="unbounded" />
604       </xs:sequence>
605       <xs:attribute name="beginTS" type="xs:dateTime" use="required"/>
606       <xs:attribute name="endTS" type="xs:dateTime" use="required"/>
607       <xs:attribute name="overallExecutionDuration" type="xs:unsignedLong"
              use="required"/>
608       <xs:attribute name="writtenBytes" type="xs:unsignedLong" use="required">
609         <xs:annotation>
610           <xs:documentation>
611             The number of written bytes or characters, which depends on
612             the data type that is written.
613           </xs:documentation>
614         </xs:annotation>
615       </xs:attribute>
616     </xs:complexType>
617   </xs:element>
618
619   <!--
620    ! WSHelperPortType#printXML
621    !-->
622   <xs:element name="printRequest">
623     <xs:complexType mixed="true">
624       <xs:sequence>
625         <xs:any minOccurs="0" maxOccurs="unbounded"/>
```

```
626       </xs:sequence>
627       <xs:attribute name="loggerName" type="xs:string" use="optional"/>
628       <xs:attribute name="level" type="im:loggerLevelSType" use="optional"
              default="INFO"/>
629       <xs:attribute name="prefix" type="xs:string" use="optional" default=""/>
630     </xs:complexType>
631   </xs:element>
632
633   <xs:simpleType name="loggerLevelSType">
634     <xs:restriction base="xs:string">
635       <xs:enumeration value="TRACE" />
636       <xs:enumeration value="DEBUG" />
637       <xs:enumeration value="INFO" />
638       <xs:enumeration value="WARN" />
639       <xs:enumeration value="ERROR" />
640     </xs:restriction>
641   </xs:simpleType>
642
643   <xs:element name="printResponse">
644     <xs:complexType mixed="true">
645       <xs:sequence>
646         <xs:any minOccurs="0" maxOccurs="unbounded"/>
647       </xs:sequence>
648     </xs:complexType>
649   </xs:element>
650
651   <!--
652    ! DBEvaluationPortType#executeEvaluationRun
653    !-->
654   <xs:element name="executeEvaluationRunRequest">
655     <xs:complexType>
656       <xs:sequence>
657         <xs:element name="datasourceName" type="xs:string" />
658         <xs:element name="suiteid" type="xs:long" />
659       </xs:sequence>
660     </xs:complexType>
661   </xs:element>
662
663   <xs:element name="executeEvaluationRunResponse">
664     <xs:complexType>
665       <xs:sequence>
666         <xs:element name="runid" type="xs:long" />
667       </xs:sequence>
668     </xs:complexType>
669   </xs:element>
670
671   <!--
672    ! DBApplicationSimulationPortType#simulateApplications
673    !-->
674   <xs:element name="simulateApplicationsRequest">
675     <xs:complexType>
676       <xs:sequence>
677         <xs:element name="datasourceName" type="xs:string" />
678         <xs:element name="processNamePattern" type="xs:string" minOccurs="0"
              default=""/>
679         <xs:element name="appCount" type="xs:unsignedShort">
680           <xs:annotation>
681             <xs:documentation>
682   How many threads should concurrently executed. Only limited by the
683   maximum number of connections to the database in the connection
684   pool.
```

```
685   See "apache-tomcat-6.0.20
          Ode/conf/context.xml/Context/Resource/attribute::maxActive"
686             </xs:documentation>
687           </xs:annotation>
688         </xs:element>
689         <xs:element name="seed" type="xs:long">
690           <xs:annotation>
691             <xs:documentation>
692               The seed is used for the random numbers needed to decide which
693               rows to retrieve or manipulate.
694             </xs:documentation>
695           </xs:annotation>
696         </xs:element>
697         <xs:element name="isolation" type="im:transactionIsolationSType"
              minOccurs="0"/>
698       </xs:sequence>
699     </xs:complexType>
700   </xs:element>
701
702   <!-- See java.sql.Connection -->
703   <xs:simpleType name="transactionIsolationSType">
704     <xs:restriction base="xs:string">
705       <xs:enumeration value="NONE" />
706       <xs:enumeration value="READ_UNCOMMITTED" />
707       <xs:enumeration value="READ_COMMITTED" />
708       <xs:enumeration value="REPEATABLE_READ" />
709       <xs:enumeration value="SERIALIZABLE" />
710     </xs:restriction>
711   </xs:simpleType>
712
713   <xs:element name="simulateApplicationsResponse">
714     <xs:complexType>
715       <xs:sequence>
716         <xs:element name="memoryDelta" type="im:memoryDeltaType" />
717       </xs:sequence>
718       <xs:attribute name="beginTS" type="xs:dateTime" use="required"/>
719       <xs:attribute name="endTS" type="xs:dateTime" use="required"/>
720       <xs:attribute name="overallExecutionDuration" type="xs:unsignedLong"
            use="required">
721         <xs:annotation>
722           <xs:documentation>How long took the parallel execution in
                ms</xs:documentation>
723         </xs:annotation>
724       </xs:attribute>
725       <xs:attribute name="sumExecutionDuration" type="xs:unsignedLong"
            use="required">
726         <xs:annotation>
727           <xs:documentation>How long ran the application threads all
                together</xs:documentation>
728         </xs:annotation>
729       </xs:attribute>
730     </xs:complexType>
731   </xs:element>
732
733   <!-- We need different elements for failure messages, that Axis2 can distinguish
          them -->
734   <xs:element name="executeQueryViaDataSourceSqlErrorFailureContainer"
          type="im:failureContainerType" />
735   <xs:element name="executeQueryViaDataSourceInternalErrorFailureContainer"
          type="im:failureContainerType" />
736   <xs:element name="executeViaDataSourceSqlErrorFailureContainer"
          type="im:failureContainerType" />
```

```
737   <xs:element name="executeViaDataSourceInternalErrorFailureContainer"
          type="im:failureContainerType" />
738   <xs:element name="executeInsertViaDataSourceSqlErrorFailureContainer"
          type="im:failureContainerType" />
739   <xs:element name="executeInsertViaDataSourceInternalErrorFailureContainer"
          type="im:failureContainerType" />
740   <xs:element name="measurePerformanceViaDataSourceSqlErrorFailureContainer"
          type="im:failureContainerType" />
741   <xs:element name="measurePerformanceViaDataSourceInternalErrorFailureContainer"
          type="im:failureContainerType" />
742   <xs:element name="copyContentViaDataSourceSqlErrorFailureContainer"
          type="im:failureContainerType" />
743   <xs:element name="copyContentViaDataSourceInternalErrorFailureContainer"
          type="im:failureContainerType" />
744   <xs:element name="importCSVToDataSourceSqlErrorFailureContainer"
          type="im:failureContainerType" />
745   <xs:element name="importCSVToDataSourceInternalErrorFailureContainer"
          type="im:failureContainerType" />
746   <xs:element name="exportFromDataSourceToCSVSqlErrorFailureContainer"
          type="im:failureContainerType" />
747   <xs:element name="exportFromDataSourceToCSVInternalErrorFailureContainer"
          type="im:failureContainerType" />
748   <xs:element name="exportFromDataSourceToFileSqlErrorFailureContainer"
          type="im:failureContainerType" />
749   <xs:element name="exportFromDataSourceToFileInternalErrorFailureContainer"
          type="im:failureContainerType" />
750   <xs:element name="executeEvaluationRunInternalErrorFailureContainer"
          type="im:failureContainerType" />
751   <xs:element name="executeEvaluationRunUserErrorFailureContainer"
          type="im:failureContainerType" />
752   <xs:element name="simulateApplicationsSqlErrorFailureContainer"
          type="im:failureContainerType" />
753   <xs:element name="simulateApplicationsInternalErrorFailureContainer"
          type="im:failureContainerType" />
754   <xs:complexType name="failureContainerType">
755     <xs:sequence>
756       <xs:element name="reason" type="xs:string" />
757       <xs:element name="attachment" minOccurs="0">
758         <xs:complexType mixed="true">
759           <xs:sequence>
760             <xs:any minOccurs="0" maxOccurs="unbounded"/>
761           </xs:sequence>
762         </xs:complexType>
763       </xs:element>
764     </xs:sequence>
765   </xs:complexType>
766 </xs:schema>
```

A.4.2 WSDL of the Web Services

The following listing shows the content of [DVD]/Sources/Projects/Idares/IdaresEvalProcess/IdaresWS.wsdl.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!-- $Id: IdaresWS.wsdl 3347 2010-01-23 21:15:57Z ahija $ -->
3 <wsdl:definitions name="IdaresWebServices"
4                   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
5                   xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
6                   xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

```
 7                  xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
 8                  xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
 9                  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
10                  xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
11                  xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
12                  xmlns:im="http://ahija.de/thesis/idares/model"
13                  xmlns:iws="http://ahija.de/thesis/idares/webservices"
14
15                  targetNamespace="http://ahija.de/thesis/idares/webservices">
16
17  <wsdl:documentation>IDARES Web Services</wsdl:documentation>
18
19  <wsdl:import namespace="http://ahija.de/thesis/idares/model"
        location="idares-model.xsd" />
20
21  <!--////////////////////////////////////////////////////////////////////
22   ! Partner Link
23   !////////////////////////////////////////////////////////////////////-->
24
25  <plnk:partnerLinkType name="GenericDatabaseExecutor">
26    <plnk:role name="GenericDatabaseExecutorProvider"
          portType="iws:GenericDatabasePortType"/>
27  </plnk:partnerLinkType>
28
29  <plnk:partnerLinkType name="WSHelper">
30    <plnk:role name="WSHelperProvider" portType="iws:WSHelperPortType"/>
31  </plnk:partnerLinkType>
32
33  <plnk:partnerLinkType name="DBEvaluation">
34    <plnk:role name="DBEvaluationProvider" portType="iws:DBEvaluationPortType"/>
35  </plnk:partnerLinkType>
36
37  <plnk:partnerLinkType name="DBApplicationSimulation">
38    <plnk:role name="DBApplicationSimulationProvider"
          portType="iws:DBApplicationSimulationPortType"/>
39  </plnk:partnerLinkType>
40
41  <!--////////////////////////////////////////////////////////////////////
42   ! Messages
43   !////////////////////////////////////////////////////////////////////-->
44  <wsdl:message name="executeQueryViaDataSourceRequestMessage">
45    <wsdl:part name="parameters" element="im:executeQueryViaDataSourceRequest" />
46  </wsdl:message>
47  <wsdl:message name="executeQueryViaDataSourceResponseMessage">
48    <wsdl:part name="response" element="im:executeQueryViaDataSourceResponse" />
49  </wsdl:message>
50  <wsdl:message name="executeQueryViaDataSourceSqlErrorMessage">
51    <wsdl:part name="fault"
          element="im:executeQueryViaDataSourceSqlErrorFailureContainer" />
52  </wsdl:message>
53  <wsdl:message name="executeQueryViaDataSourceInternalErrorMessage">
54    <wsdl:part name="fault"
          element="im:executeQueryViaDataSourceInternalErrorFailureContainer" />
55  </wsdl:message>
56
57  <wsdl:message name="executeViaDataSourceRequestMessage">
58    <wsdl:part name="parameters" element="im:executeViaDataSourceRequest" />
59  </wsdl:message>
60  <wsdl:message name="executeViaDataSourceResponseMessage">
61    <wsdl:part name="response" element="im:executeViaDataSourceResponse" />
62  </wsdl:message>
63  <wsdl:message name="executeViaDataSourceSqlErrorMessage">
```

```
64      <wsdl:part name="fault"
            element="im:executeViaDataSourceSqlErrorFailureContainer" />
65    </wsdl:message>
66    <wsdl:message name="executeViaDataSourceInternalErrorMessage">
67      <wsdl:part name="fault"
            element="im:executeViaDataSourceInternalErrorFailureContainer" />
68    </wsdl:message>
69
70    <wsdl:message name="executeInsertViaDataSourceRequestMessage">
71      <wsdl:part name="parameters" element="im:executeInsertViaDataSourceRequest" />
72    </wsdl:message>
73    <wsdl:message name="executeInsertViaDataSourceResponseMessage">
74      <wsdl:part name="response" element="im:executeInsertViaDataSourceResponse" />
75    </wsdl:message>
76    <wsdl:message name="executeInsertViaDataSourceSqlErrorMessage">
77      <wsdl:part name="fault"
            element="im:executeInsertViaDataSourceSqlErrorFailureContainer" />
78    </wsdl:message>
79    <wsdl:message name="executeInsertViaDataSourceInternalErrorMessage">
80      <wsdl:part name="fault"
            element="im:executeInsertViaDataSourceInternalErrorFailureContainer" />
81    </wsdl:message>
82
83    <wsdl:message name="measurePerformanceViaDataSourceRequestMessage">
84      <wsdl:part name="parameters" element="im:measurePerformanceViaDataSourceRequest"
            />
85    </wsdl:message>
86    <wsdl:message name="measurePerformanceViaDataSourceResponseMessage">
87      <wsdl:part name="response" element="im:measurePerformanceViaDataSourceResponse"
            />
88    </wsdl:message>
89    <wsdl:message name="measurePerformanceViaDataSourceSqlErrorMessage">
90      <wsdl:part name="fault"
            element="im:measurePerformanceViaDataSourceSqlErrorFailureContainer" />
91    </wsdl:message>
92    <wsdl:message name="measurePerformanceViaDataSourceInternalErrorMessage">
93      <wsdl:part name="fault"
            element="im:measurePerformanceViaDataSourceInternalErrorFailureContainer" />
94    </wsdl:message>
95
96    <wsdl:message name="copyContentViaDataSourceRequestMessage">
97      <wsdl:part name="parameters" element="im:copyContentViaDataSourceRequest" />
98    </wsdl:message>
99    <wsdl:message name="copyContentViaDataSourceResponseMessage">
100     <wsdl:part name="response" element="im:copyContentViaDataSourceResponse" />
101   </wsdl:message>
102   <wsdl:message name="copyContentViaDataSourceSqlErrorMessage">
103     <wsdl:part name="fault"
            element="im:copyContentViaDataSourceSqlErrorFailureContainer" />
104   </wsdl:message>
105   <wsdl:message name="copyContentViaDataSourceInternalErrorMessage">
106     <wsdl:part name="fault"
            element="im:copyContentViaDataSourceInternalErrorFailureContainer" />
107   </wsdl:message>
108
109   <wsdl:message name="importCSVToDataSourceRequestMessage">
110     <wsdl:part name="parameters" element="im:importCSVToDataSourceRequest" />
111   </wsdl:message>
112   <wsdl:message name="importCSVToDataSourceResponseMessage">
113     <wsdl:part name="response" element="im:importCSVToDataSourceResponse" />
114   </wsdl:message>
115   <wsdl:message name="importCSVToDataSourceSqlErrorMessage">
```

```
116     <wsdl:part name="fault"
            element="im:importCSVToDataSourceSqlErrorFailureContainer" />
117   </wsdl:message>
118   <wsdl:message name="importCSVToDataSourceInternalErrorMessage">
119     <wsdl:part name="fault"
            element="im:importCSVToDataSourceInternalErrorFailureContainer" />
120   </wsdl:message>
121
122   <wsdl:message name="exportFromDataSourceToCSVRequestMessage">
123     <wsdl:part name="parameters" element="im:exportFromDataSourceToCSVRequest" />
124   </wsdl:message>
125   <wsdl:message name="exportFromDataSourceToCSVResponseMessage">
126     <wsdl:part name="response" element="im:exportFromDataSourceToCSVResponse" />
127   </wsdl:message>
128   <wsdl:message name="exportFromDataSourceToCSVSqlErrorMessage">
129     <wsdl:part name="fault"
            element="im:exportFromDataSourceToCSVSqlErrorFailureContainer" />
130   </wsdl:message>
131   <wsdl:message name="exportFromDataSourceToCSVInternalErrorMessage">
132     <wsdl:part name="fault"
            element="im:exportFromDataSourceToCSVInternalErrorFailureContainer" />
133   </wsdl:message>
134
135   <wsdl:message name="exportFromDataSourceToFileRequestMessage">
136     <wsdl:part name="parameters" element="im:exportFromDataSourceToFileRequest" />
137   </wsdl:message>
138   <wsdl:message name="exportFromDataSourceToFileResponseMessage">
139     <wsdl:part name="response" element="im:exportFromDataSourceToFileResponse" />
140   </wsdl:message>
141   <wsdl:message name="exportFromDataSourceToFileSqlErrorMessage">
142     <wsdl:part name="fault"
            element="im:exportFromDataSourceToFileSqlErrorFailureContainer" />
143   </wsdl:message>
144   <wsdl:message name="exportFromDataSourceToFileInternalErrorMessage">
145     <wsdl:part name="fault"
            element="im:exportFromDataSourceToFileInternalErrorFailureContainer" />
146   </wsdl:message>
147
148   <wsdl:message name="printXMLRequestMessage">
149     <wsdl:part name="parameters" element="im:printRequest" />
150   </wsdl:message>
151
152   <wsdl:message name="executeEvaluationRunRequestMessage">
153     <wsdl:part name="parameters" element="im:executeEvaluationRunRequest" />
154   </wsdl:message>
155   <wsdl:message name="executeEvaluationRunResponseMessage">
156     <wsdl:part name="response" element="im:executeEvaluationRunResponse" />
157   </wsdl:message>
158   <wsdl:message name="executeEvaluationRunInternalErrorMessage">
159     <wsdl:part name="fault"
            element="im:executeEvaluationRunInternalErrorFailureContainer" />
160   </wsdl:message>
161   <wsdl:message name="executeEvaluationRunUserErrorMessage">
162     <wsdl:part name="fault"
            element="im:executeEvaluationRunUserErrorFailureContainer" />
163   </wsdl:message>
164
165   <wsdl:message name="simulateApplicationsRequestMessage">
166     <wsdl:part name="parameters" element="im:simulateApplicationsRequest" />
167   </wsdl:message>
168   <wsdl:message name="simulateApplicationsResponseMessage">
169     <wsdl:part name="response" element="im:simulateApplicationsResponse" />
```

```
170   </wsdl:message>
171   <wsdl:message name="simulateApplicationsSqlErrorMessage">
172     <wsdl:part name="fault"
          element="im:simulateApplicationsSqlErrorFailureContainer" />
173   </wsdl:message>
174   <wsdl:message name="simulateApplicationsInternalErrorMessage">
175     <wsdl:part name="fault"
          element="im:simulateApplicationsInternalErrorFailureContainer" />
176   </wsdl:message>
177
178   <!--////////////////////////////////////////////////////////////////////
179    ! PortTypes
180    !////////////////////////////////////////////////////////////////////-->
181
182   <wsdl:portType name="GenericDatabasePortType">
183     <wsdl:operation name="executeQueryViaDataSource">
184       <wsdl:input message="iws:executeQueryViaDataSourceRequestMessage" />
185       <wsdl:output message="iws:executeQueryViaDataSourceResponseMessage" />
186       <wsdl:fault message="iws:executeQueryViaDataSourceSqlErrorMessage"
            name="sqlError" />
187       <wsdl:fault message="iws:executeQueryViaDataSourceInternalErrorMessage"
            name="internalError" />
188     </wsdl:operation>
189
190     <wsdl:operation name="executeViaDataSource">
191       <wsdl:input message="iws:executeViaDataSourceRequestMessage" />
192       <wsdl:output message="iws:executeViaDataSourceResponseMessage" />
193       <wsdl:fault message="iws:executeViaDataSourceSqlErrorMessage" name="sqlError"
            />
194       <wsdl:fault message="iws:executeViaDataSourceInternalErrorMessage"
            name="internalError" />
195     </wsdl:operation>
196
197     <wsdl:operation name="executeInsertViaDataSource">
198       <wsdl:input message="iws:executeInsertViaDataSourceRequestMessage" />
199       <wsdl:output message="iws:executeInsertViaDataSourceResponseMessage" />
200       <wsdl:fault message="iws:executeInsertViaDataSourceSqlErrorMessage"
            name="sqlError" />
201       <wsdl:fault message="iws:executeInsertViaDataSourceInternalErrorMessage"
            name="internalError" />
202     </wsdl:operation>
203
204     <wsdl:operation name="measurePerformanceViaDataSource">
205       <wsdl:input message="iws:measurePerformanceViaDataSourceRequestMessage" />
206       <wsdl:output message="iws:measurePerformanceViaDataSourceResponseMessage" />
207       <wsdl:fault message="iws:measurePerformanceViaDataSourceSqlErrorMessage"
            name="sqlError" />
208       <wsdl:fault message="iws:measurePerformanceViaDataSourceInternalErrorMessage"
            name="internalError" />
209     </wsdl:operation>
210
211     <wsdl:operation name="copyContentViaDataSource">
212       <wsdl:input message="iws:copyContentViaDataSourceRequestMessage" />
213       <wsdl:output message="iws:copyContentViaDataSourceResponseMessage" />
214       <wsdl:fault message="iws:copyContentViaDataSourceSqlErrorMessage"
            name="sqlError" />
215       <wsdl:fault message="iws:copyContentViaDataSourceInternalErrorMessage"
            name="internalError" />
216     </wsdl:operation>
217
218     <wsdl:operation name="importCSVToDataSource">
219       <wsdl:input message="iws:importCSVToDataSourceRequestMessage" />
```

```
220       <wsdl:output message="iws:importCSVToDataSourceResponseMessage" />
221       <wsdl:fault message="iws:importCSVToDataSourceSqlErrorMessage" name="sqlError"
              />
222       <wsdl:fault message="iws:importCSVToDataSourceInternalErrorMessage"
              name="internalError" />
223     </wsdl:operation>
224
225     <wsdl:operation name="exportFromDataSourceToCSV">
226       <wsdl:input message="iws:exportFromDataSourceToCSVRequestMessage" />
227       <wsdl:output message="iws:exportFromDataSourceToCSVResponseMessage" />
228       <wsdl:fault message="iws:exportFromDataSourceToCSVSqlErrorMessage"
              name="sqlError" />
229       <wsdl:fault message="iws:exportFromDataSourceToCSVInternalErrorMessage"
              name="internalError" />
230     </wsdl:operation>
231
232     <wsdl:operation name="exportFromDataSourceToFile">
233       <wsdl:input message="iws:exportFromDataSourceToFileRequestMessage" />
234       <wsdl:output message="iws:exportFromDataSourceToFileResponseMessage" />
235       <wsdl:fault message="iws:exportFromDataSourceToFileSqlErrorMessage"
              name="sqlError" />
236       <wsdl:fault message="iws:exportFromDataSourceToFileInternalErrorMessage"
              name="internalError" />
237     </wsdl:operation>
238   </wsdl:portType>
239
240   <wsdl:portType name="WSHelperPortType">
241     <wsdl:operation name="printXML">
242       <wsdl:input message="iws:printXMLRequestMessage" />
243     </wsdl:operation>
244   </wsdl:portType>
245
246   <wsdl:portType name="DBEvaluationPortType">
247     <wsdl:operation name="executeEvaluationRun">
248       <wsdl:input message="iws:executeEvaluationRunRequestMessage" />
249       <wsdl:output message="iws:executeEvaluationRunResponseMessage" />
250       <wsdl:fault message="iws:executeEvaluationRunInternalErrorMessage"
              name="internalError" />
251       <wsdl:fault message="iws:executeEvaluationRunUserErrorMessage"
              name="userError" />
252     </wsdl:operation>
253   </wsdl:portType>
254
255   <wsdl:portType name="DBApplicationSimulationPortType">
256     <wsdl:operation name="simulateApplications">
257       <wsdl:input message="iws:simulateApplicationsRequestMessage" />
258       <wsdl:output message="iws:simulateApplicationsResponseMessage" />
259       <wsdl:fault message="iws:simulateApplicationsSqlErrorMessage" name="sqlError"
              />
260       <wsdl:fault message="iws:simulateApplicationsInternalErrorMessage"
              name="internalError" />
261     </wsdl:operation>
262   </wsdl:portType>
263
264   <!--////////////////////////////////////////////////////////////////////
265    ! Bindings
266    !////////////////////////////////////////////////////////////////////-->
267
268   <wsdl:binding name="GenericDatabaseSOAP11Binding"
          type="iws:GenericDatabasePortType">
269     <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"
          />
```

```
270     <wsdl:operation name="executeQueryViaDataSource">
271       <soap:operation soapAction="urn:executeQueryViaDataSource" />
272       <wsdl:input>
273         <soap:body use="literal" />
274       </wsdl:input>
275       <wsdl:output>
276         <soap:body use="literal" />
277       </wsdl:output>
278       <wsdl:fault name="sqlError">
279         <soap:fault name="sqlError" use="literal"/>
280       </wsdl:fault>
281       <wsdl:fault name="internalError">
282         <soap:fault name="internalError" use="literal" />
283       </wsdl:fault>
284     </wsdl:operation>
285     <wsdl:operation name="executeViaDataSource">
286       <soap:operation soapAction="urn:executeViaDataSource" />
287       <wsdl:input>
288         <soap:body use="literal" />
289       </wsdl:input>
290       <wsdl:output>
291         <soap:body use="literal" />
292       </wsdl:output>
293       <wsdl:fault name="sqlError">
294         <soap:fault name="sqlError" use="literal"/>
295       </wsdl:fault>
296       <wsdl:fault name="internalError">
297         <soap:fault name="internalError" use="literal" />
298       </wsdl:fault>
299     </wsdl:operation>
300     <wsdl:operation name="executeInsertViaDataSource">
301       <soap:operation soapAction="urn:executeInsertViaDataSource" />
302       <wsdl:input>
303         <soap:body use="literal" />
304       </wsdl:input>
305       <wsdl:output>
306         <soap:body use="literal" />
307       </wsdl:output>
308       <wsdl:fault name="sqlError">
309         <soap:fault name="sqlError" use="literal"/>
310       </wsdl:fault>
311       <wsdl:fault name="internalError">
312         <soap:fault name="internalError" use="literal" />
313       </wsdl:fault>
314     </wsdl:operation>
315     <wsdl:operation name="measurePerformanceViaDataSource">
316       <soap:operation soapAction="urn:measurePerformanceViaDataSource" />
317       <wsdl:input>
318         <soap:body use="literal" />
319       </wsdl:input>
320       <wsdl:output>
321         <soap:body use="literal" />
322       </wsdl:output>
323       <wsdl:fault name="sqlError">
324         <soap:fault name="sqlError" use="literal"/>
325       </wsdl:fault>
326       <wsdl:fault name="internalError">
327         <soap:fault name="internalError" use="literal" />
328       </wsdl:fault>
329     </wsdl:operation>
330     <wsdl:operation name="copyContentViaDataSource">
331       <soap:operation soapAction="urn:copyContentViaDataSource" />
```

```
332       <wsdl:input>
333         <soap:body use="literal" />
334       </wsdl:input>
335       <wsdl:output>
336         <soap:body use="literal" />
337       </wsdl:output>
338       <wsdl:fault name="sqlError">
339         <soap:fault name="sqlError" use="literal"/>
340       </wsdl:fault>
341       <wsdl:fault name="internalError">
342         <soap:fault name="internalError" use="literal" />
343       </wsdl:fault>
344     </wsdl:operation>
345     <wsdl:operation name="importCSVToDataSource">
346       <soap:operation soapAction="urn:importCSVToDataSource" />
347       <wsdl:input>
348         <soap:body use="literal" />
349       </wsdl:input>
350       <wsdl:output>
351         <soap:body use="literal" />
352       </wsdl:output>
353       <wsdl:fault name="sqlError">
354         <soap:fault name="sqlError" use="literal"/>
355       </wsdl:fault>
356       <wsdl:fault name="internalError">
357         <soap:fault name="internalError" use="literal" />
358       </wsdl:fault>
359     </wsdl:operation>
360     <wsdl:operation name="exportFromDataSourceToCSV">
361       <soap:operation soapAction="urn:exportFromDataSourceToCSV" />
362       <wsdl:input>
363         <soap:body use="literal" />
364       </wsdl:input>
365       <wsdl:output>
366         <soap:body use="literal" />
367       </wsdl:output>
368       <wsdl:fault name="sqlError">
369         <soap:fault name="sqlError" use="literal"/>
370       </wsdl:fault>
371       <wsdl:fault name="internalError">
372         <soap:fault name="internalError" use="literal" />
373       </wsdl:fault>
374     </wsdl:operation>
375     <wsdl:operation name="exportFromDataSourceToFile">
376       <soap:operation soapAction="urn:exportFromDataSourceToFile" />
377       <wsdl:input>
378         <soap:body use="literal" />
379       </wsdl:input>
380       <wsdl:output>
381         <soap:body use="literal" />
382       </wsdl:output>
383       <wsdl:fault name="sqlError">
384         <soap:fault name="sqlError" use="literal"/>
385       </wsdl:fault>
386       <wsdl:fault name="internalError">
387         <soap:fault name="internalError" use="literal" />
388       </wsdl:fault>
389     </wsdl:operation>
390   </wsdl:binding>
391
392   <wsdl:binding name="WSHelperSOAP11Binding" type="iws:WSHelperPortType">
```

```
393     <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"
            />
394     <wsdl:operation name="printXML">
395       <soap:operation soapAction="urn:printXML" />
396       <wsdl:input>
397         <soap:body use="literal" />
398       </wsdl:input>
399     </wsdl:operation>
400   </wsdl:binding>
401
402   <wsdl:binding name="DBEvaluationSOAP11Binding" type="iws:DBEvaluationPortType">
403     <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"
            />
404     <wsdl:operation name="executeEvaluationRun">
405       <soap:operation soapAction="urn:executeEvaluationRun" />
406       <wsdl:input>
407         <soap:body use="literal" />
408       </wsdl:input>
409       <wsdl:output>
410         <soap:body use="literal" />
411       </wsdl:output>
412       <wsdl:fault name="internalError">
413         <soap:fault name="internalError" use="literal" />
414       </wsdl:fault>
415       <wsdl:fault name="userError">
416         <soap:fault name="userError" use="literal" />
417       </wsdl:fault>
418     </wsdl:operation>
419   </wsdl:binding>
420
421   <wsdl:binding name="DBApplicationSimulationSOAP11Binding"
        type="iws:DBApplicationSimulationPortType">
422     <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"
            />
423     <wsdl:operation name="simulateApplications">
424       <soap:operation soapAction="urn:simulateApplications" />
425       <wsdl:input>
426         <soap:body use="literal" />
427       </wsdl:input>
428       <wsdl:output>
429         <soap:body use="literal" />
430       </wsdl:output>
431       <wsdl:fault name="sqlError">
432         <soap:fault name="sqlError" use="literal"/>
433       </wsdl:fault>
434       <wsdl:fault name="internalError">
435         <soap:fault name="internalError" use="literal" />
436       </wsdl:fault>
437     </wsdl:operation>
438   </wsdl:binding>
439
440   <!--////////////////////////////////////////////////////////////////////
441    ! Services
442    !////////////////////////////////////////////////////////////////////-->
443
444   <wsdl:service name="GenericDatabaseService">
445     <wsdl:port name="GenericDatabaseSOAP11Port"
            binding="iws:GenericDatabaseSOAP11Binding">
446       <soap:address
              location="http://localhost:8080/axis2/services/GenericDatabaseService" />
447     </wsdl:port>
448   </wsdl:service>
```

```
449
450   <wsdl:service name="WSHelperService">
451     <wsdl:port name="WSHelperSOAP11Port" binding="iws:WSHelperSOAP11Binding">
452       <soap:address location="http://localhost:8080/axis2/services/WSHelperService"
              />
453     </wsdl:port>
454   </wsdl:service>
455
456   <wsdl:service name="DBEvaluationService">
457     <wsdl:port name="DBEvaluationPort" binding="iws:DBEvaluationSOAP11Binding">
458       <soap:address
              location="http://localhost:8080/ode/processes/DBEvaluationService" />
459     </wsdl:port>
460   </wsdl:service>
461
462   <wsdl:service name="TPCHCustomerReadSimulation">
463     <wsdl:port name="TPCHCustomerReadSimulationPort"
          binding="iws:DBApplicationSimulationSOAP11Binding">
464       <soap:address
              location="http://localhost:8080/axis2/services/TPCHCustomerReadSimulation"
              />
465     </wsdl:port>
466   </wsdl:service>
467
468   <wsdl:service name="TPCHPatientWriteSimulation">
469     <wsdl:port name="TPCHPatientWriteSimulationPort"
          binding="iws:DBApplicationSimulationSOAP11Binding">
470       <soap:address
              location="http://localhost:8080/axis2/services/TPCHPatientWriteSimulation"
              />
471     </wsdl:port>
472   </wsdl:service>
473
474   <wsdl:service name="TPCHCallInsertSimulation">
475     <wsdl:port name="TPCHCallInsertSimulationPort"
          binding="iws:DBApplicationSimulationSOAP11Binding">
476       <soap:address
              location="http://localhost:8080/axis2/services/TPCHCallInsertSimulation" />
477     </wsdl:port>
478   </wsdl:service>
479 </wsdl:definitions>
```

A.5 Extensions to the Web Service Interface

In the following sections, parts of the XML schema and the WSDL↗ file of the Web Service Interface are attached that contain the extensions made by this thesis.

A.5.1 XML Schema for the Messages

The following listing shows the extensions to `[DVD]/Sources/Projects/WebServiceInterface/WSIResources/wsi-types.xsd` made by this thesis.

```
<xs:element name="FileID" type="xs:long" />

<xs:element name="storeFileFromDBRequest">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="fileInDBSource">
        <xs:complexType>
          <xs:choice>
            <xs:annotation><xs:documentation>
              How to pin down the file? FILE_ID or PROJECT, FILENAME, etc. If
              the latter is used and multiple results are possible, the file
              with the latest FILE_ID is used.
            </xs:documentation></xs:annotation>
            <xs:element name="fileID" type="xs:long" />
            <xs:element name="projectAndFile">
              <xs:complexType>
                <xs:attribute name="projectName" type="xs:string" use="required"/>
                <xs:attribute name="fileName" type="xs:string" use="required"/>
                <xs:attribute name="subID" type="xs:unsignedInt" use="optional"
                    default="1"/>
                <xs:attribute name="version" type="xs:unsignedInt" use="optional"
                    default="1"/>
              </xs:complexType>
            </xs:element>
          </xs:choice>
        </xs:complexType>
      </xs:element>
      <xs:element name="targetFile">
        <xs:annotation>
          <xs:documentation>Where to store the file? Relative to the simulation
              instance folder or to an URL.
          </xs:documentation>
        </xs:annotation>
        <xs:complexType>
          <xs:choice>
            <xs:element name="simulationInstanceTarget">
              <xs:complexType>
                <xs:attribute name="simID" type="xs:long" use="required"/>
                <xs:attribute name="file" type="xs:string" use="required" />
              </xs:complexType>
            </xs:element>
            <xs:element name="urlTarget">
              <xs:complexType>
                <xs:attribute name="url" type="xs:anyURI" use="required"/>
              </xs:complexType>
            </xs:element>
          </xs:choice>
        <xs:attribute name="charset" type="xs:string" use="optional"
            default="ISO-8859-1"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="loadFileIntoDBRequest">
  <xs:complexType>
    <xs:sequence>
       <xs:element name="sourceFile">
        <xs:annotation>
          <xs:documentation>Where to load the file from? Relative to the simulation
              instance folder or to an URL.
```

```
58          </xs:documentation>
59        </xs:annotation>
60        <xs:complexType>
61          <xs:choice>
62            <xs:element name="simulationInstanceSource">
63              <xs:complexType>
64                <xs:attribute name="simID" type="xs:long" use="required"/>
65                <xs:attribute name="file" type="xs:string" use="required" />
66              </xs:complexType>
67            </xs:element>
68            <xs:element name="urlSource">
69              <xs:complexType>
70                <xs:attribute name="url" type="xs:anyURI" use="required"/>
71              </xs:complexType>
72            </xs:element>
73          </xs:choice>
74          <xs:attribute name="fileType" type="types:simulationFileSType"
                use="required"/>
75          <xs:attribute name="charset" type="xs:string" use="optional"
                default="ISO-8859-1"/>
76        </xs:complexType>
77      </xs:element>
78      <xs:element name="projectAndFile">
79        <xs:annotation>
80          <xs:documentation>Describes the column of the target
                table.</xs:documentation>
81        </xs:annotation>
82        <xs:complexType>
83          <xs:sequence>
84            <xs:element name="attachment">
85              <xs:complexType>
86                <xs:sequence>
87                  <xs:any />
88                </xs:sequence>
89              </xs:complexType>
90            </xs:element>
91          </xs:sequence>
92          <xs:attribute name="projectName" type="xs:string" use="required"/>
93          <xs:attribute name="fileName" type="xs:string" use="required"/>
94          <xs:attribute name="subID" type="xs:unsignedInt" use="optional"
                default="1"/>
95          <xs:attribute name="version" type="xs:unsignedInt" use="optional"
                default="1"/>
96        </xs:complexType>
97      </xs:element>
98    </xs:sequence>
99  </xs:complexType>
100 </xs:element>
```

A.5.2 WSDL of the Web Service

The following listing shows the extensions to `[DVD]/Sources/Projects/WebServiceInterface/WSIResources/WebServiceInterface.wsdl` made by this thesis.

```
1 [..]
2 <wsdl:message name="storeFileFromDBRequestMessage">
3   <wsdl:part name="parameters" element="types:storeFileFromDBRequest" />
4 </wsdl:message>
5
```

```
 6 <wsdl:message name="loadFileIntoDBRequestMessage">
 7   <wsdl:part name="parameters" element="types:loadFileIntoDBRequest" />
 8 </wsdl:message>
 9 [..]
10 <wsdl:portType name="WebServiceInterface">
11   [..]
12   <wsdl:operation name="storeFileFromDB">
13     <wsdl:input message="wsiws:storeFileFromDBRequestMessage"/>
14     <wsdl:output message="wsiws:EmptyMessage"/>
15     <wsdl:fault name="FileOperationFault" message="wsiws:FileOperationException"/>
16     <wsdl:fault name="InvalidStateFault" message="wsiws:InvalidStateException"/>
17   </wsdl:operation>
18   <wsdl:operation name="loadFileIntoDB">
19     <wsdl:input message="wsiws:loadFileIntoDBRequestMessage"/>
20     <wsdl:output message="wsiws:FileIDMessage"/>
21     <wsdl:fault name="FileOperationFault" message="wsiws:FileOperationException"/>
22     <wsdl:fault name="InvalidStateFault" message="wsiws:InvalidStateException"/>
23   </wsdl:operation>
24   [..]
25 </wsdl:portType>
26
27 <wsdl:binding name="WSIBinding" type="wsiws:WebServiceInterface">
28   [..]
29   <wsdl:operation name="storeFileFromDB">
30     <soap:operation soapAction="http://wsi.simtech.de/ws/storeFileFromDB" />
31     <wsdl:input>
32       <soap:body use="literal" />
33     </wsdl:input>
34     <wsdl:output>
35       <soap:body use="literal" />
36     </wsdl:output>
37     <wsdl:fault name="FileOperationFault">
38       <soap:fault use="literal" name="FileOperationFault" />
39     </wsdl:fault>
40     <wsdl:fault name="InvalidStateFault">
41       <soap:fault use="literal" name="InvalidStateFault" />
42     </wsdl:fault>
43   </wsdl:operation>
44   <wsdl:operation name="loadFileIntoDB">
45     <soap:operation soapAction="http://wsi.simtech.de/ws/loadFileIntoDB" />
46     <wsdl:input>
47       <soap:body use="literal" />
48     </wsdl:input>
49     <wsdl:output>
50       <soap:body use="literal" />
51     </wsdl:output>
52     <wsdl:fault name="FileOperationFault">
53       <soap:fault use="literal" name="FileOperationFault" />
54     </wsdl:fault>
55     <wsdl:fault name="InvalidStateFault">
56       <soap:fault use="literal" name="InvalidStateFault" />
57     </wsdl:fault>
58   </wsdl:operation>
59   [..]
60 </wsdl:binding>
61 [..]
```

A.6 Performance Results

A.6.1 Performance Results of Relational Databases

This performance evaluation is made in the context of Criterion 9 on page 85. At this point, only the aggregated performance results are presented. The interested reader is referred to `[DVD]/Documents/Stuff/Relational Results.xlsx` for all individual results. The following table shows the durations for all seven databases. For retrieval tests that where repeated a number of times, the table contains the arithmetic mean of the durations. If a cell is empty, an error prevented the execution of a test with the given database. The colors of the cells indicate the rank of the database for a given test: green refers to the best value (shortest duration) and red indicates the worst value (longest duration).

Test Group	DB2	MonetDB	MySQL Clust.	Derby Client	Derby Emb.	H2 Client	H2 Emb.	MIN	MEAN	MEDIAN	MAX
[TPCH] Import	00:02:08.313	00:09:00.076	00:21:08.439	00:06:50.254	00:09:03.751	00:07:16.830	00:02:45.391	00:02:08.313	00:08:19.008	00:07:16.830	00:21:08.439
[TPCH] Keys, Indizes	00:03:18.948	00:00:27.252	00:09:50.307	00:05:49.289	00:08:23.469	00:05:14.086	00:05:46.116	00:00:27.252	00:05:32.781	00:05:46.116	00:09:50.307
[TPCH] Select Count CUSTOMER	00:00:00.033	00:00:00.005	00:00:00.028	00:00:00.123	00:00:00.424	00:00:00.001	00:00:00.002	00:00:00.001	00:00:00.088	00:00:00.028	00:00:00.424
[TPCH] Select Count LINEITEM	00:00:00.153	00:00:00.003	00:00:00.005	00:00:00.587	00:00:00.933	00:00:00.002	00:00:00.002	00:00:00.002	00:00:00.241	00:00:00.005	00:00:00.933
[TPCH] Select Count Distinct ORDERS	00:00:00.341	00:00:00.093	00:00:08.785	00:00:08.741	00:00:12.299	00:00:12.990	00:00:15.968	00:00:00.093	00:00:08.459	00:00:08.785	00:00:15.968
[TPCH] Select * CUSTOMER	00:00:01.164	00:00:03.444	00:00:02.354	00:00:02.116	00:00:00.947	00:00:09.223	00:00:03.679	00:00:00.947	00:00:03.275	00:00:02.354	00:00:09.223
[TPCH] Select Column CUSTOMER	00:00:00.187	00:00:00.299	00:00:00.975	00:00:00.412	00:00:00.304	00:00:02.876	00:00:02.233	00:00:00.187	00:00:01.041	00:00:00.412	00:00:02.876
[TPCH] Select * Where ORDERS	00:00:09.711	00:00:19.989	00:00:16.106	00:00:18.954	00:00:08.276	00:00:54.531	00:00:30.060	00:00:08.276	00:00:22.518	00:00:18.954	00:00:54.531
[TPCH] Select Column Where ORDERS	00:00:08.219	00:00:02.237	00:00:06.984	00:00:05.310	00:00:04.958	00:00:26.396	00:00:25.787	00:00:02.237	00:00:11.413	00:00:06.984	00:00:26.396
[TPCH] Select Column Where ORDERS with Index	00:00:08.504	00:00:02.182	00:00:05.255	00:00:05.348	00:00:05.917	00:00:22.126	00:00:22.372	00:00:02.182	00:00:10.243	00:00:05.917	00:00:22.372
[TPCH] Select Join without Index	00:00:01.197	00:00:00.034	00:00:01.777	00:00:00.361	00:00:00.509	00:00:02.550	00:00:02.936	00:00:00.034	00:00:01.338	00:00:01.197	00:00:02.936
[TPCH] Select Join with Index	00:00:00.628	00:00:01.439	00:00:01.361	00:00:01.072	00:00:00.583	00:00:04.852	00:00:02.865	00:00:00.583	00:00:01.829	00:00:01.361	00:00:04.852
[TPCH] Select Aggregation	00:00:20.362	00:00:13.741	00:00:29.827	00:00:45.403	00:00:45.844	00:00:51.828	00:00:53.290	00:00:13.741	00:00:37.185	00:00:45.403	00:00:53.290
[TPCH] Select Aggregation Join Order	00:00:08.974	00:00:00.197	00:00:01.213	00:00:02.403	00:00:02.981	00:00:02.317	00:00:03.515	00:00:00.197	00:00:03.086	00:00:02.403	00:00:08.974
[TPCH] Query 2	00:00:01.261	00:00:01.222	00:10:16.643	00:00:05.952	00:00:07.646	00:00:23.748	00:00:32.518	00:00:01.222	00:01:38.427	00:00:07.646	00:10:16.643
[TPCH] Query 4 mod	00:00:09.044	00:00:00.132	00:00:01.434	00:00:08.113	00:00:08.965	00:00:04.400	00:00:07.318	00:00:00.132	00:00:05.629	00:00:07.318	00:00:09.044
[TPCH] Query 6	00:00:01.310	00:00:00.177	00:00:00.461	00:00:01.521	00:00:01.900	00:00:00.331	00:00:00.462	00:00:00.177	00:00:00.880	00:00:00.462	00:00:01.900
[TPCH] Query 16	00:00:01.169	00:00:01.251	00:03:23.675	00:00:07.084	00:00:08.634	00:00:01.354	00:00:01.598	00:00:01.169	00:00:32.109	00:00:01.598	00:03:23.675
[TPCH] Query 17	00:18:32.992	00:00:00.625	00:02:05.812	00:00:00.206	00:00:00.377	00:00:00.113	00:00:00.203	00:00:00.113	00:02:57.190	00:00:00.377	00:18:32.992
[TPCH] Query 22	00:00:13.464	00:00:00.612	00:00:05.392			00:00:07.270	00:00:09.375	00:00:00.612	00:00:07.223	00:00:07.270	00:00:13.464
[CONCURRENT] Read Customer	00:00:05.773		00:00:12.359	00:00:07.126	00:00:12.526	00:00:12.476	00:00:08.073	00:00:05.773	00:00:09.722	00:00:10.216	00:00:12.526
[CONCURRENT] Modify Patient (Read Committed)	00:00:58.599			00:01:52.037		00:01:03.791	00:01:40.910	00:00:58.599	00:01:23.834	00:01:22.351	00:01:52.037
[CONCURRENT] Modify Patient (Repeatable Read)	00:01:06.533			00:01:16.595		00:01:01.535	00:01:41.951	00:01:01.535	00:01:16.653	00:01:11.564	00:01:41.951
[CONCURRENT] Insert Emergency Call (Read Committed)	00:10:34.808	00:08:24.996	00:10:47.513	00:03:36.529	00:05:01.558	00:03:02.118	00:04:28.882	00:03:02.118	00:06:33.772	00:05:01.558	00:10:47.513
[CONCURRENT] Insert Emergency Call (Repeatable Read)	00:09:51.093		00:10:52.210	00:03:37.465	00:04:55.469	00:03:02.178	00:04:32.623	00:03:02.178	00:06:08.506	00:04:44.046	00:10:52.210
[CO2] Import	00:00:12.921	00:00:09.373	00:00:53.419	00:00:16.945	00:00:13.208	00:00:57.311	00:00:15.402	00:00:09.373	00:00:25.511	00:00:15.402	00:00:57.311
[CO2] Keys	00:00:11.111	00:00:00.910	00:00:44.208	00:00:20.655	00:00:21.888	00:00:24.990	00:00:07.875	00:00:00.910	00:00:18.805	00:00:20.655	00:00:44.208
[CO2] Performance Adopt Properties	00:00:00.679	00:00:00.244		00:00:06.853	00:00:13.199	00:00:19.488	00:00:08.499	00:00:00.244	00:00:08.160	00:00:07.676	00:00:19.488
[CO2] Performance Select All	00:00:00.281	00:00:01.209	00:00:01.166	00:00:00.803	00:00:00.703	00:00:02.125	00:00:00.935	00:00:00.281	00:00:01.032	00:00:00.935	00:00:02.125
[CO2] Performance Join 1	00:00:00.469	00:00:00.111	00:00:51.393	00:00:36.602	00:00:34.695	00:00:00.991	00:00:01.272	00:00:00.111	00:00:17.933	00:00:01.272	00:00:51.393
[CO2] Performance Join 2	00:00:01.639		05:31:45.111	00:00:29.075	00:00:31.648	00:00:08.983	00:00:09.316	00:00:01.639	00:55:30.962	00:00:19.195	05:31:45.111
[CO2] Performance Join 3	00:00:09.630	00:00:01.432		00:00:45.048	00:00:45.496	00:00:24.588	00:00:22.707	00:00:01.432	00:00:24.817	00:00:23.647	00:00:45.496
Schema Dropping	00:00:03.829		00:00:09.954	00:00:03.459	00:00:03.536	00:00:05.556	00:00:47.427	00:00:03.459	00:00:12.294	00:00:04.693	00:00:47.427
RSS Memory Peak	786,444	1,645,604	1,131,460	706,320	1,337,052	676,700	1,386,524	676,700	1,095,729	1,131,460	1,645,604

Figure A-i: Performance Results of Relational Databases

A.6.2 Performance Results of XML Databases

This performance evaluation is made in the context of Criterion 10 on page 87. At this point, only the aggregated performance results are presented. The interested reader is referred to `[DVD]/Documents/Stuff/XML Results.xlsx` for all individual results. The following table shows the durations for all three databases. For tests that are repeated a number of times, the table contains the arithmetic mean of the durations. If a cell is empty, an error prevented the execution of a test with the given database. The colors of the cells indicate the rank of the database for a given test: green refers to the best value (shortest duration) and red indicates the worst value (longest duration).

Test	DB2	MonetDB XQ	eXist-db	MIN	MEAN	MEDIAN	MAX
[XMARK] Load 0.02 no split	00:00:06.739	00:00:01.464	00:00:04.353	00:00:01.464	00:00:04.185	00:00:04.353	00:00:06.739
[XMARK] Query All Element Count	00:00:00.542	00:00:00.143	00:00:00.305	00:00:00.143	00:00:00.330	00:00:00.305	00:00:00.542
[XMARK] Query Person Element Count	00:00:00.056	00:00:00.067	00:00:00.019	00:00:00.019	00:00:00.048	00:00:00.056	00:00:00.067
[XMARK] Query Retrieve Regions	00:00:00.023	00:00:00.202	00:00:00.015	00:00:00.015	00:00:00.080	00:00:00.023	00:00:00.202
[XMARK] Query Text Analysis	00:00:02.835	00:00:03.946	00:00:01.559	00:00:01.559	00:00:02.780	00:00:02.835	00:00:03.946
[XMARK] Query XMark 1	00:00:00.011	00:00:00.308	00:00:00.017	00:00:00.011	00:00:00.112	00:00:00.017	00:00:00.308
[XMARK] Query XMark 2	00:00:00.080	00:00:00.120	00:00:00.072	00:00:00.072	00:00:00.091	00:00:00.080	00:00:00.120
[XMARK] Query XMark 3	00:00:00.078	00:00:00.580	00:00:00.060	00:00:00.060	00:00:00.239	00:00:00.078	00:00:00.580
[XMARK] Query XMark 6	00:00:00.024	00:00:00.039	00:00:00.008	00:00:00.008	00:00:00.024	00:00:00.024	00:00:00.039
[XMARK] Query XMark 8	00:00:02.666	00:00:00.239	00:00:02.096	00:00:00.239	00:00:01.667	00:00:02.096	00:00:02.666
[XMARK] Query XMark 9	00:07:05.237	00:00:00.699	00:00:02.474	00:00:00.699	00:02:22.803	00:00:02.474	00:07:05.237
[XMARK] Query XMark 10	00:00:00.434	00:00:01.098	00:00:00.278	00:00:00.278	00:00:00.603	00:00:00.434	00:00:01.098
[XMARK] Query XMark 11	00:00:05.302	00:00:00.223	00:00:03.464	00:00:00.223	00:00:02.996	00:00:03.464	00:00:05.302
[XMARK] Query XMark 14	00:00:00.080	00:00:00.073	00:00:00.045	00:00:00.045	00:00:00.066	00:00:00.073	00:00:00.080
[XMARK] Query XMark 16	00:00:00.007	00:00:00.135	00:00:00.024	00:00:00.007	00:00:00.055	00:00:00.024	00:00:00.135
[XMARK] Query XMark 18		00:00:00.059	00:00:00.024	00:00:00.024	00:00:00.042	00:00:00.042	00:00:00.059
[XMARK] Query XMark 19	00:00:00.129	00:00:00.189	00:00:00.058	00:00:00.058	00:00:00.125	00:00:00.129	00:00:00.189
[XMARK] Query XMark 20	00:00:00.036	00:00:00.578	00:00:00.041	00:00:00.036	00:00:00.218	00:00:00.041	00:00:00.578
[XMARK] Drop Collection	00:00:00.058	00:00:03.500	00:00:00.764	00:00:00.058	00:00:01.441	00:00:00.764	00:00:03.500
[XMARK] Load 0.1 no split	00:00:04.147	00:00:06.779	00:00:18.704	00:00:04.147	00:00:09.877	00:00:06.779	00:00:18.704
[XMARK] Query All Element Count	00:00:09.371	00:00:00.162	00:00:01.082	00:00:00.162	00:00:03.539	00:00:01.082	00:00:09.371
[XMARK] Query Person Element Count	00:00:00.414	00:00:00.023	00:00:00.011	00:00:00.011	00:00:00.149	00:00:00.023	00:00:00.414
[XMARK] Query Retrieve Regions	00:00:00.054	00:00:00.206	00:00:00.008	00:00:00.008	00:00:00.089	00:00:00.054	00:00:00.206
[XMARK] Query Text Analysis	00:00:14.409	00:00:44.487	00:00:06.485	00:00:06.485	00:00:21.794	00:00:14.409	00:00:44.487
[XMARK] Query XMark 1	00:00:00.036	00:00:00.181	00:00:00.035	00:00:00.035	00:00:00.084	00:00:00.036	00:00:00.181
[XMARK] Query XMark 2	00:00:00.247	00:00:00.142	00:00:00.109	00:00:00.109	00:00:00.166	00:00:00.142	00:00:00.247
[XMARK] Query XMark 3	00:00:00.403	00:00:00.528	00:00:00.185	00:00:00.185	00:00:00.372	00:00:00.403	00:00:00.528
[XMARK] Query XMark 6	00:00:00.105	00:00:00.041	00:00:00.012	00:00:00.012	00:00:00.053	00:00:00.041	00:00:00.105
[XMARK] Query XMark 8	00:01:07.950	00:00:00.245	00:01:41.718	00:00:00.245	00:00:56.637	00:01:07.950	00:01:41.718
[XMARK] Query XMark 9		00:00:00.949	00:02:02.082	00:00:00.949	00:01:01.515	00:01:01.515	00:02:02.082
[XMARK] Query XMark 10	00:00:09.079	00:00:01.283	00:00:04.379	00:00:01.283	00:00:04.914	00:00:04.379	00:00:09.079
[XMARK] Query XMark 11	00:02:19.122	00:00:05.932	00:02:48.675	00:00:05.932	00:01:44.576	00:02:19.122	00:02:48.675
[XMARK] Query XMark 14	00:00:00.486	00:00:00.474	00:00:00.189	00:00:00.189	00:00:00.383	00:00:00.474	00:00:00.486
[XMARK] Query XMark 16	00:00:00.028	00:00:00.137	00:00:00.074	00:00:00.028	00:00:00.080	00:00:00.074	00:00:00.137
[XMARK] Query XMark 18		00:00:00.064	00:00:00.084	00:00:00.064	00:00:00.074	00:00:00.074	00:00:00.084
[XMARK] Query XMark 19	00:00:00.569	00:00:00.135	00:00:00.229	00:00:00.135	00:00:00.311	00:00:00.229	00:00:00.569
[XMARK] Query XMark 20	00:00:00.180	00:00:00.224	00:00:00.152	00:00:00.152	00:00:00.185	00:00:00.180	00:00:00.224
[XMARK] Drop Collection	00:00:00.527	00:00:00.800	00:00:02.587	00:00:00.527	00:00:01.305	00:00:00.800	00:00:02.587
RSS Memory Peak	624,972	1,234,032	1,229,484	624,972	1,029,496	1,229,484	1,234,032

Figure A-ii: Performance Results of XML Databases

The following figure shows the slowdown of the databases for between the two differently sized XML documents. Green cell backgrounds refer to the best value (smallest slowdown) and red indicates the worst value (highest slowdown). The values can be understood the following way:

a value of 2.0 indicates that the database needed twice as long to perform a test with the larger scale XML file as for the smaller XML file

Test	DB2	MonetDB XQ	eXist-db
[XMARK] Load	0.62	4.63	4.30
[XMARK] Query All Element Count	17.31	1.13	3.55
[XMARK] Query Person Element Count	7.39	0.34	0.58
[XMARK] Query Retrieve Regions	2.35	1.02	0.51
[XMARK] Query Text Analysis	5.08	11.27	4.16
[XMARK] Query XMark 1	3.44	0.59	2.08
[XMARK] Query XMark 2	3.09	1.18	1.52
[XMARK] Query XMark 3	5.18	0.91	3.07
[XMARK] Query XMark 6	4.43	1.04	1.45
[XMARK] Query XMark 8	25.48	1.03	48.52
[XMARK] Query XMark 9		1.36	49.34
[XMARK] Query XMark 10	20.92	1.17	15.76
[XMARK] Query XMark 11	26.24	26.62	48.70
[XMARK] Query XMark 14	6.12	6.49	4.23
[XMARK] Query XMark 16	4.01	1.01	3.09
[XMARK] Query XMark 18		1.08	3.50
[XMARK] Query XMark 19	4.40	0.72	3.95
[XMARK] Query XMark 20	4.96	0.39	3.68
[XMARK] Drop Collection	9.09	0.23	3.39
Average Slowdown	8.83	3.27	10.81

Figure A-iii: Slowdown of the XML Databases between the small (2.3 MB) and the large scale (11.2 MB) XML document

A.6.3 Performance Results of the Candidates for the ODE Model DB

This performance evaluation is made in the context of Criterion 11 on page 89. At this point, only the aggregated performance results are presented. The interested reader is referred to `[DVD]/Documents/Stuff/ODE Model DB Results.xlsx` for all individual results. The following table shows the durations for all three databases. All tests were applied twice and the arithmetic mean is applied. The tests with the prefix *Concurrent* involved the execution of multiple concurrent workflows – the arithmetic mean is applied here too. The colors of the cells indicate the rank of the database for a given test: green refers to the best value (shortest duration) and red indicates the worst value (longest duration).

Test	Derby Embedded	DB2	H2 Client	MIN	MEAN	MEDIAN	MAX
Startup	00:00:26.340	00:00:37.090	00:00:25.435	00:25.435	00:29.622	00:26.340	00:37.090
Non-concurrent first	00:00:52.971	00:05:07.385	00:01:15.767	00:52.971	02:25.374	01:15.767	05:07.385
Non-concurrent second	00:00:29.972	00:05:44.693	00:01:23.551	00:29.972	02:32.739	01:23.551	05:44.693
Non-concurrent third	00:03:15.933	00:41:23.487	00:10:31.620	03:15.933	18:23.680	10:31.620	41:23.487
Concurrent first	00:01:58.326	00:06:38.781	00:06:03.564	01:58.326	04:53.557	06:03.564	06:38.781
Concurrent second	00:05:37.170	00:16:06.403	00:15:51.679	05:37.170	12:31.751	15:51.679	16:06.403
Shutdown	00:00:03.890	00:00:05.915	00:00:03.300	00:03.300	00:04.368	00:03.890	00:05.915
Restart	00:00:25.785	00:00:40.795	00:00:24.260	00:24.260	00:30.280	00:25.785	00:40.795

Figure A-iv: Performance Results of the Candidates for the *ODE model db*

List of Figures

List of Tables

Bibliography

[AAD+07] A. Agrawal, M. Amend, M. Das, M. Ford, C. Keller, M. Kloppmann, D. König, F. Leymann, R. Müller, G. Pfau, K. Plösser, R. Rangaswamy, A. Rickayzen, M. Rowley, P. Schmidt, I. Trickovic, A. Yiu, M. Zeller. WS-BPEL Extension for People (BPEL4People), Version 1.0, 2007. URL `http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel4people/BPEL4People_v1.pdf`. (Cited on page 13)

[AM06] L. Afanasiev, M. Marx. An analysis of the current XQuery benchmarks. In *In proceedings of the First International Workshop on Performance and Evaluation of Data Management Systems (ExpDB 2006)*. ACM Press, ACM Press, Chicago, Illinois, USA, 2006. (Cited on page 87)

[AMA06] A. Akram, D. Meredith, R. Allan. Evaluation of BPEL to Scientific Workflows. In *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on*, volume 1, pp. 269–274. 2006. (Cited on pages 1, 18 and 50)

[Atk01] P. Atkins. *Kurzlehrbuch Physikalische Chemie*. WILEY-VCH Verlag GmbH Weinheim, 2001. (Cited on page 32)

[BBC+07] A. Berglund, S. Boag, D. Chamberlin, M. F. Fernández, M. Kay, J. Robie, J. Siméon. XML Path Language (XPath) 2.0. Recommendation by the W3C XSL Working Group and the W3C XML Query Working Group, 2007. URL `http://www.w3.org/TR/2007/REC-xpath20-20070123/`. (Cited on pages 43 and 98)

[BBD+09] P. Bastian, M. Blatt, A. Dedner, C. Engwer, R. Klöfkorn, M. Nolte, M. Ohlberger, O. Sander. *The Distributed and Unified Numerics Environment (DUNE) Grid Interface HOWTO*. Abteilung 'Simulation großer Systeme', Universität Stuttgart; Abteilung für Angewandte Mathematik, Universität Freiburg; Institut für Numerische und Angewandte Mathematik, Universität Münster; Institut f¨ur Mathematik II, Freie Universität Berlin, version 1.3svn edition, 2009. URL `http://www.dune-project.org/doc/grid-howto/grid-howto.pdf`. (Cited on pages 25, 30 and 31)

[BCF+07] S. Boag, D. Chamberlin, M. F. Fernández, D. Florescu, J. Robie, J. Siméon. XQuery 1.0: An XML Query Language. Recommendation by the W3C XML Query Working Group, 2007. URL `http://www.w3.org/TR/2007/REC-xquery-20070123/`. (Cited on pages 37, 43, 48, 83, 87, 98, 119 and 158)

[BFA08] S. S. Bokhari, A. Ferworn, A. Abhari. Architectural model for grid resources discovery. In *SpringSim '08: Proceedings of the 2008 Spring simulation multiconference*, pp. 1–5. Society for Computer Simulation International, San Diego, CA, USA, 2008. (Cited on page 18)

[BG06] R. Barga, D. Gannon. *Scientific versus Business Workflows*, chapter 2, pp. 9–16. In Taylor et al. [TDGS06], 2006. (Cited on pages 1, 17 and 18)

[BHM+04] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, D. Orchard. Web Services Architecture. Working Group Note by the W3C Web Services Architecture Working Group, 2004. URL `http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/`. (Cited on page 7)

[BKML+09] D. A. Benson, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, E. W. Sayers. GenBank. *Nucleic Acids Research*, 37(Database-Issue):26–31, 2009. (Cited on page 61)

[BKT00] P. Buneman, S. Khanna, W.-C. Tan. Data Provenance: Some Basic Issues, 2000. (Cited on page 61)

[Blu] The Blue Brain Project. URL `http://bluebrain.epfl.ch/`. (Cited on page 17)

[BMR09] D. Beck, J. Müller, F. Ruthardt. *Evaluation von Datenintegrationssystemen im e-Science-Bereich*. Fachstudie, Universität Stuttgart, Institut für Parallele und Verteilte Systeme, 2009. Fachstudie Nr. 107. (Cited on page 53)

[Bon02] P. A. Boncz. *Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications*. Ph.d. thesis, Universiteit van Amsterdam, Amsterdam, The Netherlands, 2002. (Cited on page 36)

[BSP+08] N. B. Bangera, D. L. Schomer, S. Papavasiliou, D. Hagler, N. Dehghani, I. Ulbert, E. Halgren, A. M. Dale. Experimental Validation of the Influence of White Matter Anisotropy on EEG Forward Solution, 2008. URL `http://scv.bu.edu/visualization/gallery/anisotropy/`. (Cited on page 23)

[But05] R. Butek. Which style of WSDL should I use?, 2005. URL `http://www.ibm.com/developerworks/webservices/library/ws-whichwsdl/`. (Cited on page 10)

[CCbD06] S. Cohen, S. Cohen-boulakia, S. Davidson. Towards a model of provenance and user views in scientific workflows. In *In Data Integration in the Life Sciences*, pp. 264–279. 2006. (Cited on page 61)

[CCMW01] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana. Web Services Description Language (WSDL) 1.1. Submission to W3C, 2001. URL `http://www.w3.org/TR/2001/NOTE-wsdl-20010315`. (Cited on pages 7 and 39)

[CD99] J. Clark, S. DeRose. XML Path Language (XPath) 1.0. Endorsed as a W3C Recommendation, 1999. URL `http://www.w3.org/TR/1999/REC-xpath-19991116`. (Cited on pages 14 and 37)

[CDR+07] H. Class, H. Dahle, F. Riis, A. Ebigbo1, G. Eigestad. Estimation of the CO_2 Storage Capacity of a Geological Formation, 2007. URL http://www.iws.uni-stuttgart.de/co2-workshop/correctedProblem3.pdf. (Cited on pages 84, 85, 122 and 140)

[CMRW07] R. Chinnici, J.-J. Moreau, A. Ryman, S. Weerawarana. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. W3C Recommendation produced by the Web Services Description Working Group, 2007. URL http://www.w3.org/TR/2007/REC-wsdl20-20070626/. (Cited on pages 8 and 39)

[Cod71] E. F. Codd. Further Normalization of the Data Base Relational Model. *IBM Research Report, San Jose, California*, RJ909, 1971. (Cited on page 87)

[CRZ03] A. B. Chaudhri, A. Rashid, R. Zicari. *XML Data Management: Native XML and XML-Enabled Database Systems*. Addison-Wesley Professional, 2003. (Cited on page 37)

[DCBE+07] S. B. Davidson, S. Cohen-Boulakia, A. Eyal, B. Ludäscher, T. M. McPhillips, S. Bowers, M. K. Anand, J. Freire. Provenance in Scientific Workflow Systems. *IEEE Bulletin of the Technical Committee on Data Engineering*, 30(4):44–50, 2007. (Cited on page 51)

[DG06] E. Deelman, Y. Gil. Workshop on the Challenges of Scientific Workflows, 2006. URL http://vtcpc.isi.edu/wiki/images/b/bf/NSFWorkflow-Final.pdf. (Cited on pages 1, 17, 18 and 53)

[DMS+06] E. Deelman, G. Mehta, G. Singh, M.-H. Su, K. Vahi. *Pegasus: Mapping Large-Scale Workflows to Distributed Resources*, chapter 23, pp. 376–394. In Taylor et al. [TDGS06], 2006. (Cited on pages 53 and 69)

[FCL97] M. J. Franklin, M. J. Carey, M. Livny. Transactional Client-Server Cache Consistency: Alternatives and Performance. *ACM Trans. Database Syst.*, 22(3):315–363, 1997. (Cited on pages 35, 52 and 63)

[Feh09] J. Fehr. Model reduction in flexible multibody dynamics – Project Description. Unpublished, 2009. URL http://www.itm.uni-stuttgart.de/research/model_reduction/model_reduction_en.php. (Cited on page 67)

[Feh10] J. Fehr. Personal meeting with Jörg Fehr, 2010. Meeting took place on January 22, 2010; Jörg Fehr is research assistent at the Institute of Engineering and Computational Mechanics at the University of Stuttgart and member of multiple scientific projects in the context of simulations on multi body systems. (Cited on pages 2, 67 and 68)

[FFHE09] J. Fehr, M. Fischer, B. Haasdonk, P. Eberhard. Snapshot-based Approximation of Frequency-weighted Gramian Matrices for Model Reduction in Multibody Dynamics, 2009. Posted for "Model Reduction of Parametrized Systems" (MoRePaS 09): http://www.uni-muenster.de/CeNoS/ocs/index.php/MRP/MRP09/. (Cited on page 67)

[FKT01] I. Foster, C. Kesselman, S. Tuecke. The Anatomy of the Grid – Enabling Scalable Virtual Organizations. *International Journal of Supercomputer Applications*, 15:2001, 2001. (Cited on page 18)

[GDE+07] Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, J. Myers. Examining the Challenges of Scientific Workflows. *Computer*, 40(12):24–32, 2007. (Cited on page 1)

[GHR06] M. Gudgin, M. Hadley, T. Rogers. Web Services Addressing 1.0 - Core. Recommendation produced by the Web Services Addressing Working Group, 2006. URL `http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/`. (Cited on pages 13 and 39)

[GKC+09] S. Glotzer, S. Kim, P. Cummings, A. Deshmukh, M. Head-Gordon, G. Karniadakis, L. Petzold, C. Sagui, M. Shinozuka. International Assessment of Research and Development in Simulation-Based Engineering and Science. Technical report, World Technology Evaluation Center, World Technology Evaluation Center, Inc.; 4800 Roland Avenue, Baltimore, Maryland 21210, 2009. URL `http://www.wtec.org/sbes/SBES-InitialFullDraftReport-15April2009.pdf`. (Cited on pages 1 and 17)

[GLNS+05] J. Gray, D. T. Liu, M. Nieto-Santisteban, A. Szalay, D. J. Dewitt, G. Heber. Scientific data management in the coming decade. *SIGMOD Rec.*, 34(4):34–41, 2005. (Cited on pages 17, 51 and 53)

[GMS92] H. Garcia-Molina, K. Salem. Main Memory Database Systems: An Overview. *IEEE Transactions on Knowledge and Data Engineering*, 4(6):509–516, 1992. (Cited on page 36)

[GPP09] K. Goodson, R. Preotiuc-Pietro. Service Data Objects. Java Specification Requests 235; final release; proposed by BEA Systems Inc. and IBM Corporation, 2009. URL `http://jcp.org/en/jsr/detail?id=235`. (Cited on page 159)

[Gra02] S. Graves. Examining Main Memory Databases, 2002. URL `http://www.iapplianceweb.com/story/OEG20020104S0070.htm`. (Cited on pages 36 and 37)

[GS93] D. Garlan, M. Shaw. An Introduction to Software Architecture. In V. Ambriola, G. Tortora, editors, *Advances in Software Engineering and Knowledge Engineering*, pp. 1–39. World Scientific Publishing Company, Singapore, 1993. (Cited on page 55)

[H209] H2. Performance Comparison, 2009. URL `http://www.h2database.com/html/performance.html#performance_comparison`. (Cited on page 38)

[Har05] S. Hartmann. The world as a process: Simulations in the natural and social sciences, 2005. URL `http://philsci-archive.pitt.edu/archive/00002412/01/Simulations.pdf`. (Cited on pages 1 and 21)

[HB04] H. Haas, A. Brown. Web Services Glossary. Working Group Note by the W3C Web Services Architecture Working Group, 2004. URL http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/. (Cited on page 8)

[HG07] G. Heber, J. Gray. Supporting Finite Element Analysis with a Relational Database Backend, Part I: There is Life beyond Files. *CoRR*, abs/cs/0701159, 2007. (Cited on pages 2, 52, 53, 54 and 59)

[HR01] T. Härder, E. Rahm. *Datenbanksysteme: Konzepte und Techniken der Implementierung, 2. Auflage*. Springer, 2001. (Cited on pages 35 and 36)

[HSW08] A. Haufler, A. Staudenecker, A. Wobsers. *Vergleich von XML-Datenbanken*. Fachstudie, Universität Stuttgart, Institut für Architektur von Anwendungssystemen, 2008. Fachstudie Nr. 87. (Cited on pages 85, 97, 98, 101, 103, 104, 128 and 154)

[Hum90] P. Humphreys. Computer Simulations. In *Proceedings of the Biennial Meeting of the Philosophy of Science Association*, volume 1990 – 2, pp. 497–506. The University of Chicago Press on behalf of the Philosophy of Science Association, 1990. (Cited on page 20)

[HVG+07] H. Hallez, B. Vanrumste, R. Grech, J. Muscat, W. Declercq, A. Vergult, Y. D'Asseler, K. P. Camilleri, S. G. Fabri, S. Van Huffel, I. Lemahieu. Review on solving the forward problem in EEG source analysis. *Journal of NeuroEngineering and Rehabilitation*, 4:46+, 2007. (Cited on page 23)

[IEE90] IEEE standard glossary of software engineering terminology, 1990. URL http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=159342&isnumber=4148. IEEE 610.12-1990. (Cited on page 19)

[JE07] D. Jordan, J. Evdemon. Web Services Business Process Execution Language Version 2.0, 2007. URL http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html. (Cited on pages 10, 12 and 16)

[JXW08] C. Jiang, X. Xu, J. Wan. Grid computing based large scale Distributed Cooperative Virtual Environment Simulation. In *12th International Conference on Computer Supported Cooperative Work in Design, 2008.*, pp. 507–512. 2008. doi:10.1109/CSCWD.2008.4537030. (Cited on page 18)

[KE04] A. Kemper, A. Eickler. *Datenbanksysteme - Eine Einführung, 5. Auflage*. Oldenbourg, 2004. (Cited on page 35)

[KKS+06] D. Karastoyanova, R. Khalaf, R. Schroth, M. Paluszek, F. Leymann. BPEL Event Model. Technischer Bericht Informatik 2006/10, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany, Universität Stuttgart, Institut für Architektur von Anwendungssystemen, 2006. (Cited on page 13)

[LAB+06] B. Ludäscher, I. Altintas, S. Bowers, J. Cummings, T. Critchlow, E. Deelman, D. D. Roure, J. Freire, C. Goble, M. Jones, S. Klasky, T. McPhillips, N. Podhorszki, C. Silva, I. Taylor, M. Vouk. *Scientific Process Automation and Workflow Management*, chapter 13. Scientific Data Management: Challenges, Technology, and Deployment. Chapman & Hall/CRC Computational Science, 1 edition, 2006. URL `http://daks.ucdavis.edu/%7Eludaesch/Paper/ch13-preprint.pdf`. (Cited on pages 16, 17, 18, 19, 22, 24 and 56)

[Law05] J. Lawrence. Performance Benchmarking of Embedded Databases. Blog, 2005. URL `http://jamie.ideasasylum.com/2005/03/performance-benchmarking-of-embedded-databases/`. (Cited on page 85)

[Les09] T. van Lessen. Personal meeting with Tammo van Lessen, 2009. The meeting took place on November 25, 2009; Tammo van Lessen is Apache Committer / PMC Member for Apache ODE. (Cited on pages 42, 90, 129 and 133)

[LL07] J. Ludewig, H. Lichter. *Software Engineering – Grundlagen, Menschen, Prozesse, Techniken*. dpunkt.verlag GmbH, Heidelberg, 1 edition, 2007. (Cited on pages 21 and 47)

[LLF+09] C. Lin, S. Lu, X. Fei, A. Chebotko, D. Pai, Z. Lai, F. Fotouhi, J. Hua. A Reference Architecture for Scientific Workflow Management Systems and the VIEW SOA Solution. *Services Computing, IEEE Transactions on*, 2(1):79–92, 2009. (Cited on page 71)

[LM00] A. Laux, L. Martin. XML:DB XUpdate. Working Draft, 2000. URL `http://xmldb-org.sourceforge.net/xupdate/xupdate-wd.html`. (Cited on page 98)

[LR00] F. Leymann, D. Roller. *Production workflow: concepts and techniques*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000. (Cited on pages 1, 10, 11, 12 and 205)

[McC09] D. McCreary. XQuery – Synchronizing Remote Collections, 2009. URL `http://en.wikibooks.org/w/index.php?title=XQuery/Synchronizing_Remote_Collections&oldid=1666322`. (Cited on page 121)

[MH06] C. Mathis, T. Härder. Twig Query Processing Under Concurrent Updates. In *Data Engineering Workshops, 2006. Proceedings. 22nd International Conference on*, pp. x141–x141. IEEE Computer Society, 2006. (Cited on page 37)

[MI06] L. Moreau, J. Ibbotson. The EU Provenance Project: Enabling and Supporting Provenance in Grids for Complex Problems (Final Report). Technical report, The EU Provenance Consortium, 2006. (Cited on page 51)

[Mic08] Microsoft External Research. Trident Workbench: A Scientific Workflow System, 2008. URL `http://research.microsoft.com/en-us/collaboration/tools/trident_workbench.doc`. (Cited on page 71)

[Mit95] B. Mitschang. *Anfrageverarbeitung in Datenbanksystemen – Entwurfs- und Implementierungskonzepte*. Vieweg, 1995. (Cited on page 36)

[Mit96] M. Mitchell. *An introduction to genetic algorithms*. MIT Press, Cambridge, MA, USA, 1996. (Cited on page 147)

[MMLD05] A. Maier, B. Mitschang, F. Leymann, W. Dan. On Combining Business Process Integration and ETL Technologies. In G. für Informatik, editor, *Datenbanksysteme in Business, Technologie und Web (BTW'05)*, pp. 533–546. Köllen, 2005. (Cited on pages 45 and 53)

[Mod09] C. Moder. VTK File Formats, 2009. URL `http://www.geophysik.uni-muenchen.de/intranet/it-service/applications/paraview/vtk-file-formats/`. (Cited on page 146)

[MP07] A. Morken, P. O. R. Pahr. *Apache Derby SMP scalability*. Masterthesis, Norwegian University of Science and Technology, Department of Computer and Information Science, 2007. (Cited on page 38)

[MS04] G. J. Myers, C. Sandler. *The Art of Software Testing*. John Wiley & Sons, 2004. (Cited on page 145)

[MSK+95] J. Miller, A. Sheth, K. Kochut, X. Wang, A. Murugan. Simulation modeling within workflow technology. In *Simulation Conference Proceedings, 1995. Winter*, pp. 612–619. 1995. (Cited on pages 1 and 17)

[Nar06] R. Narang. *Database Management Systems*. Prentice-Hall of India Pvt.Ltd, 2006. (Cited on page 35)

[NSGS+05] M. A. Nieto-Santisteban, J. Gray, A. S. Szalay, J. Annis, A. R. Thakar, W. O'Mullane. When Database Systems Meet the Grid. In *In CIDR*, pp. 154–161. 2005. (Cited on page 52)

[OAS06] OASIS. Web Services Resource Framework (WSRF) – Primer v1.2, 2006. URL `http://docs.oasis-open.org/wsrf/wsrf-primer-1.2-primer-cd-02.pdf`. (Cited on page 18)

[OC09] M. Orgiyan, M. V. Cappellen. XQuery API for Java (XQJ). Java Specification Requests 225, final release; proposed by IBM and Oracle Corporation, 2009. URL `http://jcp.org/en/jsr/detail?id=225`. Submitted by IBM & Oracle Corporation; Last checked October 8, 2009. (Cited on pages 73 and 159)

[OMG09] Business Process Model and Notation (BPMN) Version 1.2. online, 2009. URL `http://www.omg.org/spec/BPMN/1.2/PDF`. OMG Document Number: formal/2009-01-03. (Cited on page 94)

[Pel03] C. Peltz. Web services orchestration and choreography. *Computer*, 36(10):46–52, 2003. (Cited on page 11)

[PHP+06] D. Pennington, D. Higgins, A. Peterson, M. Jones, B. Ludaescher, S. Bowers. *Ecological Niche Modeling Using the Kepler Workflow System*, chapter 7, pp. 91–108. In Taylor et al. [TDGS06], 2006. (Cited on pages 2, 18, 34, 50, 51, 53 and 66)

[Pol07] PolePosition Project. PolePosition – the open source database benchmark, 2007. URL http://polepos.sourceforge.net/results/PolePosition.pdf. (Cited on pages 38 and 85)

[Pro09] Protein Data Bank. Atomic Coordinate Entry Format Version 3.2 – ATOM records, 2009. URL http://www.wwpdb.org/documentation/format32/sect9.html#ATOM. (Cited on page 147)

[Rei08] P. Reimann. *Optimization of BPEL/SQL Flows in Federated Database Systems*. Diplomarbeit, Universität Stuttgart, 2008. Diplomarbeit Nr. 2744. (Cited on pages 3, 52, 63 and 64)

[Roh90] F. Rohrlich. Computer Simulation in the Physical Sciences. In *Proceedings of the Biennial Meeting of the Philosophy of Science Association*, volume 1990 – 2, pp. 507–518. The University of Chicago Press on behalf of the Philosophy of Science Association, 1990. (Cited on page 1)

[Rut09] J. Rutschmann. *Generisches Web Service Interface um Simulationsanwendungen in BPEL-Prozesse einzubinden*. Dimplomarbeit, Universität Stuttgart, 2009. Dimplomarbeit Nr. 2895. (Cited on pages 3, 29, 30, 57, 58, 59, 60, 67, 71, 72, 74, 78, 134, 142, 146 and 161)

[SKDN05] S. Shankar, A. Kini, D. J. DeWitt, J. Naughton. Integrating databases and workflow systems. *SIGMOD Rec.*, 34(3):5–11, 2005. (Cited on pages 50 and 51)

[Slo06] A. Slominski. *Adapting BPEL to Scientific Workflows*, chapter 14, pp. 208–226. In Taylor et al. [TDGS06], 2006. (Cited on pages 18 and 51)

[Sta73] H. Stachowiak. *Allgemeine Modelltheorie*. Springer, 1973. (Cited on page 20)

[Sta01] K. Staken. XML:DB Database API. Working Draft, 2001. URL http://xmldb-org.sourceforge.net/xapi/xapi-draft.html. (Cited on pages 98, 109 and 113)

[Sta03] K. Staken. Frequently Asked Questions About XML:DB:, 2003. URL http://xmldb-org.sourceforge.net/faqs.html. (Cited on page 37)

[STH06] O. Sandstå, D. Tjeldvoll, K. A. Hatlen. Apache Derby Performance, 2006. URL http://docs.huihoo.com/apache/apachecon/us2005/ApacheCon05usDerbyPerformance.pdf. (Cited on page 38)

[Stu09] Studienprojekt SIMPL. SIMPL – Grobentwurf, 2009. Version 1.5 from December 7, 2009. (Cited on pages 73 and 74)

[SVG+03] P. Sherwood, A. H. de Vries, M. F. Guest, G. Schreckenbach, C. R. A. Catlow, S. A. French, A. A. Sokol, S. T. Bromley, W. Thiel, A. J. Turner, S. Billeter, F. Terstegen, S. Thielc, J. Kendrick, S. C. Rogers, J. Casci, M. Watson, F. Kinge, E. Karlsen, M. Sjøvoll, A. Fahmi, A. Schäfer, C. Lennartz. QUASI: A general purpose implementation of the QM/MM approach and its application to problems in catalysis. *JOURNAL OF MOLECULAR STRUCTURE-THEOCHEM*, 632:1–28, 2003. (Cited on page 33)

[SWK+01] A. R. Schmidt, F. Waas, M. L. Kersten, D. Florescu, I. Manolescu, M. J. Carey, R. Busse. The XML Benchmark Project. Technical Report INS-R0103, CWI, Amsterdam, The Netherlands, 2001. (Cited on pages 84, 87 and 171)

[Tan07] W.-C. Tan. Provenance in Databases: Past, Current, and Future. *IEEE Bulletin of the Technical Committee on Data Engineering*, 30(4):3–12, 2007. (Cited on page 51)

[TDGS06] I. J. Taylor, E. Deelman, D. B. Gannon, M. Shields, editors. *Workflows for e-Science: Scientific Workflows for Grids*. Springer, 2006. (Cited on pages 1, 7, 17, 18, 66, 208, 209, 213, 214 and 215)

[Tim08] F. Timme. How To Shrink VMware Virtual Disk Files, 2008. URL `http://www.howtoforge.com/how-to-shrink-vmware-virtual-disk-files-vmdk`. (Cited on page 151)

[TMMF09] W. Tan, P. Missier, R. Madduri, I. Foster. Building Scientific Workflow with Taverna and BPEL: A Comparative Study in caGrid. pp. 118–129, 2009. (Cited on page 24)

[Tra08] Transaction Processing Performance Council (TPC). TPC Benchmark H Standard Specification Revision 2.8.0, 2008. URL `http://www.tpc.org/tpch/spec/tpch2.8.0.pdf`. (Cited on pages 84 and 85)

[VMW08] VMWare. *Timekeeping in VMware Virtual Machines*, 2008. URL `http://www.vmware.com/pdf/vmware_timekeeping.pdf`. (Cited on page 150)

[VSRM08] M. Vrhovnik, H. Schwarz, S. Radeschütz, B. Mitschang. An Overview of SQL Support in Workflow Products. In *ICDE 2008. Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, pp. 1287–1296. 2008. (Cited on page 44)

[VSS+07] M. Vrhovnik, H. Schwarz, O. Suhre, B. Mitschang, V. Markl, A. Maier, T. Kraft. An approach to optimize data processing in business processes. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pp. 615–626. VLDB Endowment, 2007. (Cited on pages 44 and 63)

[W3C07] W3C Web Services Policy Working Group. Web Services Policy 1.5 - Framework, 2007. URL `http://www.w3.org/TR/2007/REC-ws-policy-20070904/`. (Cited on page 10)

[WCL+05] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, D. F. Ferguson. *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005. (Cited on pages 1, 7, 9 and 10)

[WEB+06] B. Wassermann, W. Emmerich, B. Butchart, N. Cameron, L. Chen, J. Patel. *Sedna: A BPEL-Based Environment for Visual Scientific Workflow Modeling*, chapter 26, pp. 428–449. In Taylor et al. [TDGS06], 2006. (Cited on page 18)

[Win03] D. Winer. XML-RPC Specification, 2003. URL `http://www.xmlrpc.com/spec`. (Cited on page 98)

[WK07] R. Wrembel, C. Koncilia. *Data warehouses and OLAP: concepts, architectures, and solutions*. Idea Group Inc., 2007. (Cited on page 68)

[Wor99] Workflow Management Coalition. Workflow Management Coalition – Terminology & Glossary, 1999. URL `http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf`. Document Number WFMC-TC-1011; Last checked September 24, 2009. (Cited on pages 10, 11 and 16)

[Wya00] N. Wyatt. Pure Java databases for deployed applications. In *Data Engineering, 2000. Proceedings. 16th International Conference on*, pp. 477–485. 2000. (Cited on page 38)

[ZB07] Y. Zhang, P. Boncz. XRPC: interoperable and efficient distributed XQuery. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pp. 99–110. VLDB Endowment, 2007. (Cited on page 113)

[ZTZ05] O. C. Zienkiewicz, R. L. Taylor, J. Z. Zhu. *The Finite Element Method – Its Basis & Fundamentals*. Butterworth-Heinemann, 6 edition, 2005. (Cited on page 24)

All links were last followed on February 1, 2010.

Declaration

All the work contained within this thesis, except where otherwise acknowledged, was solely the effort of the author. At no stage was any collaboration entered into with any other party.

(Place, Date) (Christoph Marian Müller)