Jül-4303

Jülich Supercomputing Centre (JSC)

Eclipse-based Client Support for Scientific Biological Applications in E-Science Infrastructures

Sonja Holl

Mitglied der Helmholtz-Gemeinschaft

JÜLICH
FORSCHUNGSZENTRUM

Eclipse-based Client Support for Scientific Biological Applications in E-Science Infrastructures

Sonja Holl

Berichte des Forschungszentrums Jülich; 4303
ISSN 0944-2952
Jülich Supercomputing Centre (JSC)
Jül-4303

Vollständig frei verfügbar im Internet auf dem Jülicher Open Access Server (JUWEL) unter
http://www.fz-juelich.de/zb/juwel

Zu beziehen durch: Forschungszentrum Jülich GmbH · Zentralbibliothek, Verlag
D-52425 Jülich · Bundesrepublik Deutschland
☎ 02461 61-5220 · Telefax: 02461 61-6103 · e-mail: zb-publikation@fz-juelich.de

Abstract

Today, most biological applications raise high demands in data resources and computational power. In order to meet these demands, powerful computer and storage systems are required used in conjunction with modern software technologies that allow for seamless execution of applications on these systems. Suitable to meet these requirements are e-Science infrastructures that provide access to data and computing resources for scientific domains, such as computational biology. They are commonly used via Grid middleware systems, which in turn are accessible by users via Grid clients. Grid clients allow biological scientists to seamlessly access e-Science infrastructures, without necessarily being very well versed in computer science. Furthermore, they provide powerful graphical user interfaces that hide the low level complexity of the underlying infrastructure, so that scientists can concentrate on their scientific challenges.

Since computational biology paradigms are used in a wide area of biological applications, this thesis aims to support biological applications, which typically consist of a set of small programs, such as molecular dynamic simulation tools. These programs require to be executed in a sequence, because program outputs must be piped to program inputs of a subsequent program. Such a sequence of programs must be executed one after another on a single Grid resource to avoid unnecessary transfers of large data sets between different, geographically distributed Grid resources. Since most of the available Grid workflow mechanisms execute each step of a workflow on a different resource,they do not provide a suitable solution. Thus, a specialised client software designed for sequences of programs is necessary. Apart from the support for program sequences, computational biology is data-driven and thus biological applications depend on input data of various kinds. Therefore, scientists need to seamlessly access biological data, in particular biological databases, and use conveniently chosen subsets of this data for biological applications. This data access needs to be integrated into Grid clients.

This thesis describes the design and implementation of client support for inherent sequential biological applications in e-Science infrastructures. The design enhances the existing Eclipse-based UNICORE Rich Client and enables the modeling and visualisation of sequences of programs as well as the access to databases. The use of these newly developed features enables researchers to manually design sequences of programs, modify program parameters, and submit the resulting sequence job to a UNICORE Grid middleware for the sequential execution on a single computational resource. Additionally, the user can browse the content of databases and use database items as inputs for computation. The extensibility of the developed framework enables developers of biological applications to easily add new programs via a so-called 'extension point', without having to implement graphical user interfaces for new programs. The thesis also provides an evaluation use case that takes requirements from the WISDOM workflow into account and validates the applicability of the developed approach by providing a molecular dynamic simulation tool in the UNICORE Rich Client. Therefore, the application package AMBER is plugged into the new framework with little effort and a simulation can be executed by creating a sequence of AMBER programs, using database files as inputs.

Acknowledgments

This master thesis has been written in collaboration between University of Düsseldorf and Jülich Supercomputing Center (JSC) of Forschungszentrum Jülich GmbH in Germany. I would like to deeply thank all those people, who supported me directly or indirectly to complete this thesis.

First and foremost I want to thank my supervisor Prof. Dr. Michael Leuschel from University of Düsseldorf for offering me this challenging opportunity. He guided my study of computer science and supported my master thesis with much dedication. Thanks to him for spending much time in discussing my ideas and suggestions. In addition, I want to thank Prof. Dr. Martin Lercher from University of Düsseldorf for agreeing to be my second reviewer and investing time in my master thesis.

Most notably I would like to thank Dr. Achim Streit, head of JSC division Distributed Systems and Grid Computing, for making it possible for me to write my master thesis in his division. Many thanks to Morris Riedel from JSC by providing me with very useful and helpful assistance. He supervised me in my time at Forschungszentrum Jülich and helped me in all aspects of research and writing of this thesis. I am also very thankful to Bastian Demuth, for supporting me and spending much time with me in problem solving and design discussions concerning the UNICORE Rich Client. I also thank Mathilde Romberg and others of the Distributed Systems and Grid Computing division, for providing much effort in supporting me in various aspects. I gratefully acknowledge the support of Dietmar Koschmieder, who made it possible to run AMBER and facilitate the use of other supercomputing resources for my demonstrating my prototype software.

Special thanks to Steve Brewer and Nishadi De Silva from University of Southampton, who made this collaboration and master thesis possible by establishing the contact to Forschungszentrum Jülich.

Especially, I want to thank the people I love, my whole family, my parents Gabriele and Hans as well as my brothers Markus and Andreas, and my amazing boyfriend Sebastian. They always supported me with a never exhausting encouragement in all my plans.

Sonja Holl, Februar 2009, Mönchengladbach, Germany

Contents

Chapter 1

Introduction

Today, enhanced Science (e-Science) [87] is performed in many scientific areas such as biology, chemistry, or physics. E-Science relies on 'next generation infrastructures' that enable seamless and secure collaborations over wide-area networks in key areas of science to support ongoing research. Grids [66] have emerged as an enabler for the production and management of such next generation infrastructures. They became the technology of choice for solving 'Grand Challenge Problems' of science and society today, like treating diseases, global weather prediction or determining the age of the earth. Solving such scientific problems often relies on collaborative usage of computational, storage or other physical resources, such as scientific instruments (i.e. Sensors, Telescopes) in combination with scientific computational-intensive applications.

Especially in computational biology, e-Science infrastructures evolved as a promising technology that brings insights in understanding biological phenomena. Often, biological experiments are too expensive, dangerous, time intensive or even impossible to realise. Hence, biological experiments are accomplished by biological applications in e-Science infrastructures these days in order to simulate behaviour instead of performing physical experiments. But biological applications, which aim to simulate biological experiments as well as to process and analyse biological data, are typically data and computing-intensive. E-Science infrastructures provide a seamless and scalable access for data and computing-intensive applications to geographically distributed and heterogeneous resources in general and large-scale computational resources and data resources in particular. Since user of biological applications are often not computer specialists, they use graphical-based Grid clients that provide scientific application abstractions, which allow for the access to e-Science infrastructures, and hide the complexity of underlying e-Science infrastructures.

At the time of writing, the Eclipse Rich Client Platform (RCP) [53] is a viable basis to design Grid clients. It is a very promising framework technology with a powerful plugin and extension mechanism for Eclipse-based standalone client developments. Over the years, the RCP technology became the base of many software developments that support e-Science, particularly Grid clients such as the UNICORE Rich Client (URC).

The motivation of this thesis is based on the approach to Grid clients with the support of biological applications that consist of a set of programs and thus require to be executed one after another on one Grid resource, such as molecular dynamic simulation tools. This restricted execution of programs is required to avoid large data transfers between geographically distributed resources, because program outputs must be piped to program inputs of a subsequent program execution. The well-known Grid workflows, are not a suitable solution for this, because they execute each workflow step on potentially different resources. This thesis offers a URC enhanced design and implementation of a technique that enables scientists to create and modify sequences of programs. Therefore a base mechanism for sequences is developed as well as an application extensible framework, which offers a powerful graphical user interface (GUI) for the modeling of sequences. Additionally, computational biology raises the demand of data inputs, typically taken from biological databases. Thus, this thesis also provides a design and implementation, based on the URC, which enables the access to biological databases. Within this scope, scientists can browse and evaluate database insides and select data, which can be taken then in turn as input for applications. The benefit of this technique is that scientists can easily access and evaluate data, which can be automatically delivered to the execution environment. To sum up, the URC enhancements, developed in the scope of this thesis, provide an extensible framework and database access, to allow scientists to modify, create and submit sequence application Grid jobs and use database inputs.

The remainder of this thesis is structured as follows. The first chapters provide an introduction to Grid technologies, the Eclipse Rich Client Platform and Eclipse-based tools. In more detail, Chapter 2 clarifies the concepts of e-Science infrastructures and Grids. This includes the use of e-Science applications in Grids and the Grid Programming Environment, which provides a high-level API for the design of e-Science applications in Grid clients. In Chapter 3 this thesis shortly introduces the Eclipse Rich Client Platform, its core elements and extension mechanism. Based on this, an overview of the UNICORE Rich Client, an Eclipse-based Grid Client, is given with a detailed view on its ServiceBrowser plugin and GPE4Eclipse plugin.
Based on these foundations, the requirements and a survey of related work are provided in Chapter 4 and the design is presented in Chapter 5. It gives insights into the developed design for the sequence support and database access in clients for biological applications in e-Science infrastructures. The design is based on technologies of today's Grid infrastructures and on the UNICORE Rich Client.
In Chapter 6, a precise view on the implementation and its evaluation is provided that take the particular requirements into account that are derived from the second step of the WISDOM [39] biological application-based workflow. To validate the concept of this thesis, this use case is supported by a reference implementation in context of the particular molecular dynamics package called AMBER. It is thus evaluated if the concept in general and the reference implementation in particular meet the requirements raised in Chapter 4. Finally, Chapter 7 summarises this thesis and highlights evaluation results. Furthermore, future work in the context of biological application client support is given.

Chapter 2

E-Science Infrastructures

E-Science evolved as a new research field and was defined by John Taylor, Director of the Research Councils in the UK, as the following:

> *'e-Science is about global collaboration in key areas of science and the next generation infrastructure that will enable it'* [89]

At the time of writing, the next generation infrastructures, referred by Taylor, is represented by Grids. They provide seamless and secure global collaborations over networks in key areas of science to realize the use of e-Science applications and thus extend scientific computing. E-Science infrastructures consist of large-scale, distributed, and heterogeneous resources. The seamless access to these geographically distributed and traditional heterogeneous resources, e.g. large-scale computational devices, data, applications, and scientific instruments, facilitates the solving of complex scientific problems. Grid and e-Science infrastructures offer a wide variety of services for the usage of resources by scientists. Examples of these infrastructures in Germany and Europe are D-Grid [13] (German Grid Initiative), DEISA [7] (Distributed European Infrastructure for Supercomputing Applications), or EGEE (Enabling Grids for e-Science) [10]. Other national and regional infrastructures are: TeraGrid [31], Open Science Grid (OSG) [27], SwissGrid [30] or NorduGrid [22], just to name a few. Additionally, there are many national e-Science initiatives, which propose to develop Grid applications in many scientific domains such as the United Kingdom e-Science initiative [35], or European projects such as the Chemomentum Project [6].
This chapter starts with an introduction to Grids and e-Science infrastructures today. In the Section 2.2 a detailed view of e-Science Applications is provided, which are often specifically supported by Grid clients. The Grid Programming Environment (GPE) [20], described in Section 2.3, offers a framework to Grid Clients. This framework provides interfaces for the graphical representation of e-Science applications and its introduction is essential to understand the thesis results.

2.1 Introduction to Grid Computing

Grid computing [66, 49] aims to facilitate the solving of scientific problems, which cannot be solved within a reasonable amount of time with conventional computers, like current top-end workstations or desktop computers. The heterogeneous resources, which are often geographically distributed, consist of different hardware and software components, but are collectively available in in Grid infrastructures. In order to tackle scientific challenges, Grid infrastructures provide dependable, consistent, secure, seamless and scalable access to different Grid resources. These resources are combined to one rather large virtual system and can be shared among different users, working in different institutions. Furthermore, resources in a Grid are well connected, mostly via common open standards and well-known protocols [45]. The term 'Computational Grid' is introduced as an analogon to the 'electric power grid' in Foster et al. [59]. The idea of the Computational Grid was to make computing resources as easy accessible as electrical power. Grid resources needed for scientific computing can be of different kinds to provide different kinds of services, like the electrical power comes from different sources (water, nuclear, fossils).

Among others, examples of Grid resources are supercomputer, computer cluster, data storages and scientific instruments, e.g. a 'nuclear magnetic resonance' (NMR) spectroscopy instrument or a space telescope. Grid resources might belong to multiple organisations and are typically not managed centrally to remain local control. However, Grid infrastructures share different resources among different *virtual organizations* (VOs) [60]. Virtual organisations are dynamic sets of organisations, where a group of individuals share a common goal that can be achieved using Grid resources. The group of people and available resources can vary over time, thus VOs are dynamic. This makes Grid infrastructures complex and complicated, because there is neither a fixed set of scientists nor a fixed amount of resources. Figure 2.1 shows a Grid infrastructure with two virtual organisations.

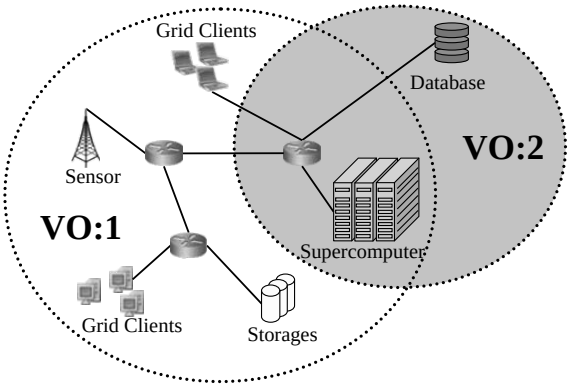

Figure 2.1: Grid resources such as supercomputers, storage systems, databases or other devices can be shared between virtual organizations. Here, the supercomputer is shared between two virtual organizations named as VO:1 and VO:2.

The software to manage a Grid is denoted as *middleware* [86]. Grid middleware systems such as UNICORE [36] (UNiform Interface to COmputing REsources), Globus Toolkit [15], and gLite [14], enable transparent, secure, efficient, and seamless access to distributed heterogeneous Grid resources. They hide the complexity and physical underlying infrastructures and ideally present the Grid as a single large virtual system. A Grid middleware fulfills the tasks of accessible discovering resources, negotiating the accessibility, mapping tasks to resources, managing applications, import/export data for processing and finally gathering results. Furthermore, it is responsible for monitoring the application execution progress and managing possible changes and resource failures after submission.

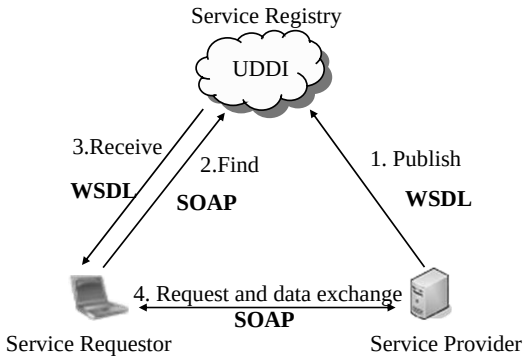

Figure 2.2: Web Services Architecture

These functionalities are mostly accessible via *Web services* [58] and many Grid middlewares are *Web Services Resource Framework* (WSRF) [47] compliant. Web services in general, and WSRF in particular, lay the foundation for Grid standards that are developed to homogenise different Grid resources. In analogy to the Internet that is open-standard based, standardisation bodies for Grid computing are OGF (Open Grid Forum) [26] and the OASIS (Organization for the Advancement of Structured Information Standards) [23]. W3C (World Wide Web Consortium) [38] and DMTF (Distributed Management Task Force) [8] were also engaged in some standardisations that are closely related to smoothly merge Internet and Grid technologies.
The open standard WSRF merges concepts from Grid computing and Web services. It defines a generic framework of specifications to model and access stateful resources. Additionally, it delivers functionality as loosely-coupled, interacting Grid services based on Web service technologies. Web services are defined in the Web Service Description Language (WSDL) [50], which characterises the abstract set of functionality that is provided by a service. Figure 2.2 shows the concept of Web services. The Universal Description, Discovery and Integration (UDDI) is a registry for Web services, in which each service provider can publish a WSDL, which describes an offered Web Service. A service requester can query a UDDI using SOAP (Simple Object Access Protocol) [74], a standard application layer protocol over HTTP, to get the corresponding WSDL of the requested service. Using the informations extracted from the WSDL document, the service requester (i.e. Grid client) can then request the service provider using SOAP.
Grid middleware systems, which are based on Web services, integrate also different security standards. The common standards typically cover authentication (examination of the identity) [51] and authorization (examination of use-able resources and services) [67] with IETF X.509 certificates [64] in common security models like Explicit Trust Delegation [84] or proxy certificates [88].

Grid technologies are often developed as a three tier architecture, such as UNICORE, which is shown in Figure 2.3. It consists of the client tier, server tier, and target system tier. The client tier consists of Grid clients, which seamlessly offer easy access to the server tier and through this the target system tier. Grid clients can be command line clients, Web portals or graphical desktop clients. The server tier contains a Grid middleware system, such as UNICORE [36], Globus Toolkit [15], g-Lite [14], as well as other software tools, necessary to access special Grid resources, e.g. Database Access Tools such as OGSA-DAI [24].

Execution of work descriptions created in the client tier and submitted via the server tier to the target system tier are called *jobs*. Jobs are described in abstract terms via the client tier by so called *job descriptions* that are interpreted at the server-side. In terms of UNICORE 6, the server tier is represented by the Gateway and the Network Job Supervisor (XNJS) of the UNICORE system. The XNJS represents one virtual Site, which represents in particular a set of resources and services. The user is checked up on before job execution by an authorization process via the UNICORE User Database (XUUDB). The job execution mechanism maps the abstract job description, send by the client, to a specific target system description and executable, using the Incarnation Database (IDB) and the Target System Interface (TSI). In addition, a File transfer service performs the input and output of files, which are defined in the job description. In context of UNICORE, the TSI located on the target system tier represents the interface to the underlying (super)computers as well to other Grid resources, e.g. databases or scientific instruments.

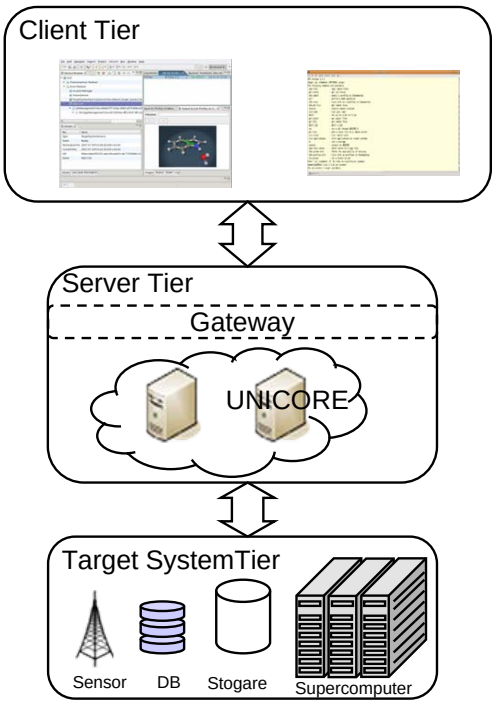

Figure 2.3: The Grid Three Tier Layer Architecture. As example, UNICORE provides technologies on each tier of its three-tier layered architecture.

2.2 E-Science Applications

Grid computing emerged as a paradigm to compute scientific applications on next generation infrastructures. E-Science today, particularly scientific experiments and studies, involve distributed and heterogeneous resources. The resources can be differentiated into three major categories:

- computational resources that are characterized by CPU, memory, network and bandwidth,
- data resources,which are represented through physical data devices or storage technologies, e.g. a hard disc, remote storage or database,
- large scientific instruments, e.g. a sensor or telescope.

E-Science applications effectively use resources of all three categories to enable different scientific methods, simulations, data analysis, modeling, or remote sensor data collection. Such applications tackle problems from very different scientific domains, e.g. Materials Science, Cosmology, Plasma Physics, or Life Sciences. The scientific applications use different approaches with similar characteristics in a wide variety of scientific domains and

typically require distributed, high-throughput, and data-intensive computing. As a result, there is an increasing number of applications available that require access to Grid resources in order to benefit from the wide variety of resources and services available on e-Science infrastructures. Grid middleware systems manage the accessibility and availability of the applications for scientists. The scientists interact with middleware via Grid Clients. These Clients provide easy to use GUIs to access Grid resources and services, by typically submitting application job descriptions to Grid middleware systems via Web services. Of course, researcher do not want to create job descriptions or Web Service requests manually, therefore Grid clients offer high-level GUIs to support the application configuration process. These high-level application GUIs are provided on the client side via uniform abstract programming interfaces (APIs). They offer interfaces for application Grid object abstraction, so called application models and their graphical user interfaces. The models represent a Grid application at the Grid client side. The application model is able to create a Grid job description containing proper data and parameters to correctly execute the application remotely on the Grid. This job description is delivered to a middleware system by using Web services. The middleware manages the execution of the job on an operating system and is thus responsible to initiate the execution of the application on the target systems.

2.3 The Grid Programming Environment

The Grid Programming Environment (GPE) [20] by Intel provides a framework with a uniform high-level API to develop Grid-enabled application models and GUIs in Grid Clients. In general, the framework offers Web Service interfaces and a GUI framework, which hides the complexity of the underlying Grid technology to the user. An example of GPE-based clients is the UNICORE Application Client [79], which is specifically designed to support scientists with one particular scientific application GUI. GPE offers components on three different levels: utility level, service level and application level. At the utility level, GPE provides a basic set of well-defined Grid services to interface with Grid systems. At the service level GPE adds higher-level services, including a dynamic resource and service registry that can be used in conjunction with Grid middleware systems that are compliant to WSRF. The GridBean Software Development Kit and a GUI framework is added at the level of applications.
This GridBean [79] concept is a plugin technology for clients that abstracts from applications. It is the fundamental concept of GPE and represents the foundation to link Grid technologies and e-Science applications. It provides methods for the creation of job descriptions and for job submission. In addition, it offers interfaces for user's interactions with Grid services and user interfaces for input/output data and parameter. GridBeans are divided in different modules to represent correspondent scientific application abstractions: One job description generation module and one or more user interface modules to enable the configuration of application behaviour. The job description module, named GridBean Model, holds input, output and parameter configurations. The interface modules, named panels, represent input, output and parameter as a graphical representative, e.g. a text box. In these graphical representatives scientists are able to modify and edit the input, output and parameters of an application. Additionally, a Plugin must be defined that offers a basic graphical interface type for the panels, e.g. Swing or Standard Widget Toolkit.

The Plugin in this case is called 'Plugin' because of historical causes, it is not of the well known type 'plugin'.

The implied data control mechanism in GridBeans manages the link of GridBean model and panel widgets. The base data control of GPE is windowing toolkit independent, thus for each GUI types might exist one specific Plugin.
An application description archive consists of the application GridBean model, one or several Plugins and one or several Panels, as shown in Figure 2.4. A developer of a new application GridBean would implement the application description archive parts by extending basic types. These basic types are offered by GPE and handle all generic aspects of job description setups. This mechanism decreases the implementation effort for developers. As a default implementation, GPE provides a Swing based user interface. An example of the use of GridBeans is presented by Morris Riedel et al. [78] in the context of scientific visualisations.

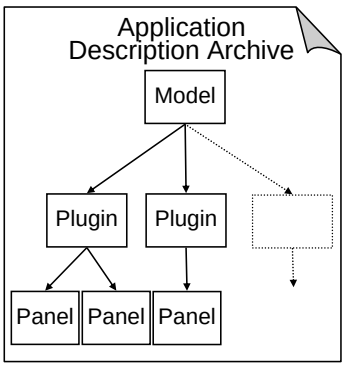

Figure 2.4: The application description archive architecture.

2.4 Summary and Conclusions

An overview of e-Science infrastructures and Grid computing was given in the first section of this chapter. It described that Grids have been established as the next generation infrastructure to enable e-Science. To conclude, e-Science infrastructures are very important to significantly enhance science in many areas today. E-Science applications, introduced in Section 2.2, are useful to enable Grids for scientific purposes. Often, these applications represent scientific experiments and studies. The last Section 2.3 motivated the use of frameworks for Grid clients, with a particular focus on the Grid Programming Environment, which provides useful interfaces to model and seamlessly execute e-Science applications in Grid infrastructures.

Chapter 3

Eclipse-based Architecture

The Eclipse Platform [9] is best known as a development environment for Java [33]. But it is also a platform independent software framework, which can be used to develop standalone so-called rich client applications. These applications can arise from different programming environments and scientific domains, e.g. Grid Clients, business administration, or health care. In this chapter,an introduction to the Eclipse Rich Client Platform is given. Section 3.2 provide an overview of the Eclipse-based UNICORE Rich Client.

3.1 The Eclipse Rich Client Platform

The Eclipse Platform was originally developed by IBM as a replacement for Visual Age for Java. Since 2001 it is available as open source software. In 2004 the Eclipse Foundation was created as a non-profit member-supported corporation of companies and individuals. It keeps track of the Eclipse open source project and community as well as the ecosystem of complementary products and services. In the beginning, the Eclipse Platform was an extensible integrated development environment (IDE) [41]. Since version 3.0, there was a shift from the IDE to the so called *Eclipse Rich Client Platform* (RCP) [9], which is build of sourced out core elements. This is the minimal set of plugins and core elements to build any kind of Rich Client Application. As a side-remark, the Eclipse Platform itself is based upon the Rich Client Platform.

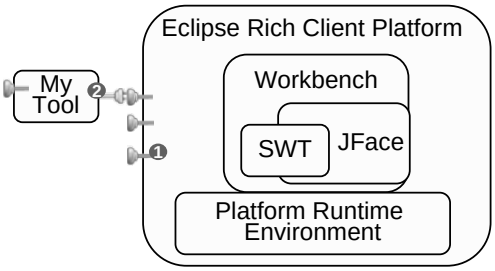

Figure 3.1: The Eclipse Rich Client Platform (RCP).

The Eclipse Platform Runtime Environment is the core engine of the RCP, as shown in Figure 3.1. It is based on the OSGi Alliance [28] specification, which is a standard for component oriented system environments. The platform runtime environment boots and loads the platform, manages the resources and is responsible for discovering, integrating, and running plugins. It keeps track of the plugins lifetimes, following the OSGi bundle concept.

Furthermore, it discovers which plugins are installed at start-up and uses 'lazy loading' to reduces the start-up time and resource usage. The mechanism called 'lazy loading' only loads and activates plugins, when their functionality is actually needed.
The minimal set of the core plugins are the Standard Widget Toolkit (SWT), JFace and the workbench, which are described in more detail in the following section. The plugin architecture of the Rich Client platform makes each Eclipse-based Rich Client application very extensible. The so-called extension points and plugins that implement the plugin architecture, are described in Section 3.1.2.

3.1.1 The Workbench, SWT, JFace

The basis elements of the Rich Client Platform are the platform runtime environment, the workbench, SWT and JFace [70]. In the following, views, editors and perspectives are firstly introduced, because they are important elements of the workbench graphical interface.

Views are resize-able and drag-able tab panels, and can also be closed and reopened. They containing widgets, e.g. buttons, text fields to provide any kind of information to users. Widgets are graphical controls in particular graphical user interface elements. User can change the state of a widget by mouse or keyboard actions. Editors show the content of files, which are supposed to be opened in Eclipse. Documents and input objects can be browsed or edited. The modifications made in an editor follow an 'open-save-close' life cycle model. A perspective stores the initial size and the set of open components, and controls them. This set of components is a well defined layout and contains a collection of editors and views. A new window layout can also be stored for later reuse.

The Standard Widget Toolkit (SWT) provides a common, platform independent API with standard widgets, e.g. button, text field, combo-box. Its approach is to have efficient, portable access to the native user interface facilities of the current running operating system. Thus the implemented applications look and response like a native system desktop application. It is a good alternative to Swing [68] and the Abstract Windowing Toolkit [32] and default for all Rich Client Platform GUI-based plugins.

JFace is a user interface toolkit with a platform independent API and implementation. It extends and inter-operates with SWT and provides higher-level application constructs. The provided toolkit components are wizards, dialogue components, text manipulation, image and font components, progress reporting for long running operations, actions and viewer. The action mechanism allows user commands to be defined independently from their exact whereabouts in the UI. Viewers are model-based adapters for certain SWT widgets, simplifying the presentation of application data structured as lists, tables or trees.

The user interface part of the Eclipse Platform is known as the generic workbench. It provides the user interface building blocks for Eclipse and defines a number of extension points for other plugins contributing to the UI. The workbench is built on top of the platform runtime environment and uses SWT and JFace. Thus, the user interface looks and feels like a native desktop application. The clear hierarchical structure of the workbench displays one or several workbench windows. Each window can host one workbench page. At any one time exactly one page per window is active and visible to the end user. The page displays the perspectives that consists of editors and views.

3.1.2 The Eclipse Plugin Mechanism

Eclipse's loosely coupled plugin mechanism [53] and the inherent extensibility mechanisms of the Rich Client Platform, as described below in Section 3.1.3, made it the popular development framework it is today. The RCP is built as a core runtime environment and a set of plugins. Any kind of application can be developed via building plugins that extend the core set. A *plugin* is the smallest unit of an application and can be developed and delivered separately from other plugins. All plugins are registered in a registry, which in turn is controlled by the runtime environment. At startup, it resolves dependencies and provides plugins with extensive context information. The life cycle of all plugins is managed by the overall framework, which automatically defers the loading of a plugin until its functionality is actually being requested by the user or by an already activated plugin. A plugin consists of an instance of a plugin runtime and an XML-based plugin manifest file. The plugin runtime provides methods for activating and deactivating the plugin. The manifest file holds information about the plugin, how the plugin can be activated by the runtime environment, and lists dependencies to other plugins.

3.1.3 The Eclipse Extension Mechanism

Plugins can also add functionality to the platform via extending predefined interfaces, so-called *extension points* [53], shown in Figure 3.1 (1). The extension points are the connection between the plugins. Eclipse offers a set of well-defined extension points. A developer can hook into the platform and contribute system behaviour by defining a so called *extension*, shown in Figure 3.1 (2). Like platform extension points, plugins can offer own extension points to allow other plugins to extend them with functionality via an extension.
Each extension point must provide a XML-based manifest file, and optional Java interfaces. This specification must be conform with each hooked in extension. In this way, the extended plugin can hold control of the extended functions. But the extension mechanism works seamlessly, making use of an one-side dependency. In other words, the extended plugin is never aware about plugged extensions. A plugin coordinates and controls all of its extension points with its own Java class loader. It enforces and checks the dependencies and visibility rules specified in the plugin manifest. This mechanism allows the RCP to 'lazy-load' a plugin, in detail a plugin offering an extension is only loaded, if its functionality is needed.

3.2 UNICORE Rich Client

The UNICORE Rich Client (URC) [37] is integrated in the client tier of the UNICORE three tier architecture (see Section 2.1). UNICORE offers at the client tier several other clients, like a command line client [34], an Application client [79] and a high level API. They exploit all features provided by the underlying infrastructure. The clients typically interact with the server tier that is connected via a single entry point: the Gateway, which offers an x509-certificate-based authentication mechanism that is used to enable a secured connection over the secure socket layer protocol [61]. The URC in comparison to the other clients offer many more bundled features and an easy to use GUI. Above all, its the best suitable client for definition of Grid workflows.
The URC was developed as a graphical client for the UNICORE 6 (UNiform Interface to COmputing REsources) middleware [80], but potentially can be used with other Web service-based Grid middlewares. It is based on the Eclipse Rich Client Platform to provide a flexible GUI and standard perspectives. The workspace concept of Eclipse decreases the effort for the URC, e.g. to export, import, share and reuse existing job workflows. The URC offers many functionalities to access Grid resources and services, while many of them are related to the submission of jobs and workflows. Additionally, the URC offers many extension points in order to be easily extended. The ServiceBrowser is one main graphical element of the URC and is not UNICORE specific. It offers a general view to the Grid that includes Grid resources, services, jobs and working directories. The workflow editor supports multiple workflow engines and furthermore, the workflow editor shows the whole workflow graph, while subgraphs can be hidden. Workflow graphs can also be zoomed and printed. Monitoring and tracing the workflow is also possible. In terms of Grid security, the URC offers a keystore view and a truststore view. The keystore view manages private keys, the truststore shows the certificates that the user trusts. Both allow for adding and deleting certificates. Furthermore, default credentials for particular security profiles and site specific credentials can be defined. To get an overview of the described functionality, a snapshot can be seen in Appendix A.
The main function of the URC is to create, submit, and manage Grid jobs and workflows. Workflows can include applications, scripts, variables, loops as well as file imports and exports. Additionally, applications within one workflow can be executed on different resources. The set of installed Grid application GUIs can be extended, thus each application GUI is separately loaded using the GridBean concept. The job description creation is realised via GPE4Eclipse, it allows for modification of application parameters, variables, files as well as resource selection and is described below.

3.2.1 URC Plugin Architecture

The UNICORE Rich Client is a standalone application and provides many plugins, which are build around the core of the Eclipse Platform. In the following, a short overview of the URC plugins is given.They are shown in Figure 3.2 together with their interrelations. The last term of each displayed class within the illustration denotes the plugins' name.

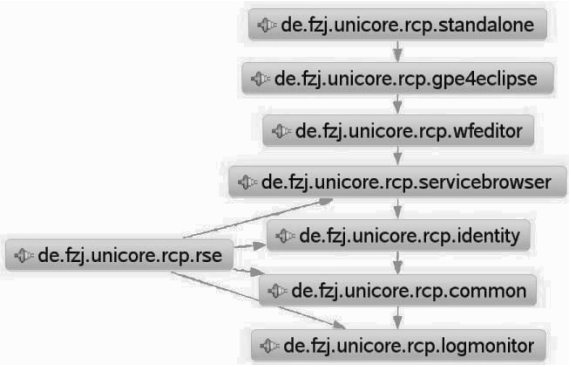

Figure 3.2: The plugin architecture of the UNICORE Rich Client.

The *logmonitor plugin* is the base of the architecture. All plugins can display and record their log messages via this plugin. The *common plugin* provides code base and client stubs to interact with middleware system Web services. The *identity plugin* is responsible for the security and the security view in the URC. The *rse plugin* extends the functionality to browse remote storages. The *servicebrowser plugin* is responsible for the graphical representation of the Grid, e.g. as a tree structure. The *wfeditor plugin* is responsible for the graphical representation of workflows. It supports multiple workflow languages. The *gpe4eclipse plugin* is responsible for the graphical representation of job descriptions as well as the linking between GPE (see Section 2.3) components and the URC. The *standalone plugin* creates the workbench and available perspectives.

In the context of this thesis, the most significant feature of the URC is its extensibility. In the following, we highlight two concrete descriptions of the plugins required in the scope of this thesis, namely the ServiceBrowser and GPE4Eclipse.

3.2.2 The ServiceBrowser Plugin

The ServiceBrowser plugin is mainly responsible for the graphical representation of the Grid (see snapshot Appendix A). It is used to monitor and perform actions on the Grid. It offers two views, a Grid browser and a details view, providing information about the services of Grid resources. The Grid browser is internally represented via a tree structure, which represents the hierarchic structure and hides the complexity of Grids. In this tree structure, each Grid service, Grid resource, submitted jobs and working directories (see Chapter2), is represented as one tree node at a different level. The nodes can have several actions, dependent on the node type and function, e.g. a registry and a job have different actions. Nodes can also have the same actions, e.g. the action 'refresh' that refreshes the known information of a Grid resource and thus checks the availability of the service. Node types and actions are extensible via extension points. The Grid tree is illustrated via different views, due to the fact that the tree view can be used in other contexts. The main Grid browser shows the Grid tree in its native form with all registries, which are expandable. Other views of the Grid tree can be filtered to only display special services. Additionally, the ServiceBrowser holds a registry that lists all applications installed on Grids.

3.2.3 The GPE4Eclipse Plugin

The GPE4Eclipse plugin is the link between GPE (see Section 2.3) and the UNICORE Rich Client. It is implemented using the GPE APIs and provide important Grid services to the user of the URC. The main offered actions are for job and workflow creation as well as fetching outcomes of a job.

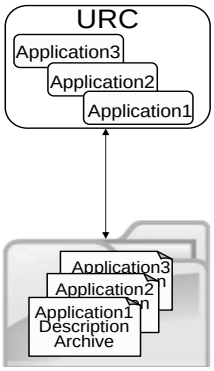

Figure 3.3: URC load mechanism for applications.

The usable applications may vary in each URC, because of the fact that they are integrated via an external load mechanism, shown in Figure 3.3. The untreated applications offer an application description archive (see Section 2.3), which is loaded by GPE4Eclipse into the URC. It consists of a GridBean, Plugins and Panels. The GridBean must declare all application environment parameters and file parameters. For simplicity, command line arguments are also modeled as environment variables. Their values are substituted with the help of the IDB in the UNICORE middleware system. The panels offer various information about the application but at least one panel must implement the GUI for the GridBean environment parameter. After loading the application description archives, user can create specific application jobs. Therefore, GPE4Eclipse provides a job view (please see Appendix B) to allow the modification of a job, and a responsible GridBean Model. The GridBean job view offers three generic panels plus the specific panels, defined in the application description archive. The three generic panels are a File panel, a Resource panel and a Variables panel. The File panel offers a view to the input and output file parameters of the application, defined in the GridBean Model. Input and output files can be added, modified or removed (fixed file parameters of the GridBean can not be removed). The user can set the following options for input files:

- a source type - the source location type of the file, e.g. local storage or database,
- a file, each source type offers a selection mechanism to select the file in the selected source location,
- a name for the file in the job directory,

and for output files:

- the name of the file,
- the destination type - the target location type of the file, e.g. a remote storage,
- a file, each target type offers a selection mechanism to set the file in the selected target location.

The Resource panel offers selectable job properties, e.g. the total number of CPUs or memory requirements for computational jobs. Additionally, the Resource panel offers a selection mechanism for the operating system. The Variables panel shows variable parameters of the GridBean.

The specific GridBean panel displays the GridBean environment parameters. All application panels are by default swing-based graphical interfaces. All parameter, which can be defined in any panel are displayed by panel widgets. Each panel widget is linked with its related parameter in the GridBean Model. This mechanism is shown in Figure 3.4 and called *data control*. For each widget of the graphical user interface, e.g. check box, combo box, and text field, exists one data control, which keeps user inputs and the GridBean parameter consistent. These data controls are special for each widget, due to the fact that user inputs must be translated into parameter values and and vice versa. It follows the basic concept of GPE data control (Section 2.3). GridBeans consist of different types of

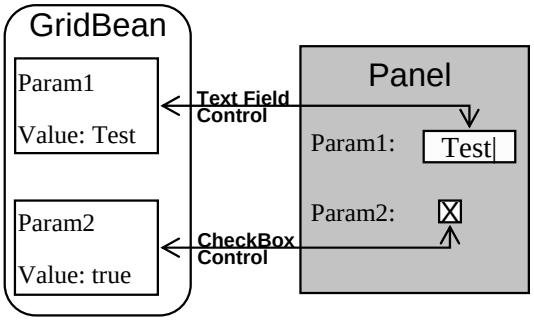

Figure 3.4: The data controls between a text field and an environment parameter as well as between a check box and another environment parameter.

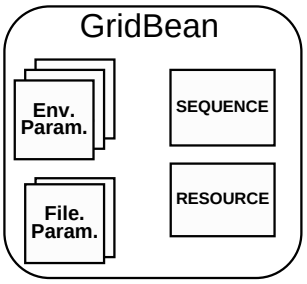

Figure 3.5: The parameter of a GridBean. Environment parameter, file parameter and two special environment parameter.

parameters: environment parameter and file parameter, as shown in Figure 3.5. Both, the environment parameter and the file parameter must be translated to be integrated into the job description for submission to a computational resources. The job description follows a Grid standard, thus environment parameter and file parameter have a special location in the complete job description. The translation of GridBean parameters is called *post-processing*. The post-processing step is realised via post-processors. The main post-processors are the environment parameter post-processors and the file input/output post-processors. Environment parameter post-processors identify their related parameters via the name of a parameter. Always, environment parameters are identified by their name. A non special environment parameter, which do not have an own post-processor is processed by a global-post-processor. The file input/output post-processors identify the type of the source or target of a file. For each source and target type, there exist a special post-processors. The post-processors have a fixed order. For sure, new post-processors can be included into the post-processing procedure, this is mainly needed, if a new parameter or a new source or a new target type is developed. We will introduce a new parameter within the scope of this thesis, named *'SEQUENCE'*, and the related post-processor. Additionally, we introduce a new source type of files: database, and the related post-processor.

3.3 Summary and Conclusions

This chapter introduced the Eclipse Rich Client Platform, which is a very promising framework, with interesting techniques to adopt and extend new features. Graphical user interfaces can be easily developed by building Eclipse plugins and by extensively use the concept of extension points. In e-Science environments, Eclipse is a viable basis for building standalone applications and GUIs for Grid clients. An example, which is reviewed in the Section 3.2, is the UNICORE Rich Client. It is an Eclipse-based Grid client to easy access Grid infrastructures in general and UNICORE-based Grids in particular. Also, the URC can be extended easily and provides a lot of base facilities for the support of scientific-specific applications, in our context applications to solve complex biological problems. The major benefit of URC is that it offers graphical representatives of applications, which allows for easy creation and modification of Grid jobs executing scientific applications.

Chapter 4

Requirements of Biological Applications in Grids

Biological applications are developed to solve and study biological problems. They are the bridge between life science and computational science. The major scientific techniques for biological applications arise from the areas of computer science, applied mathematics, statistics, chemistry and physics. Sub-areas of computational biology are molecular dynamics and protein modeling, genomic sequence and polymorphism analysis, neuronal networks, gene expression and regulation, evolution modeling, and biochemical pathway analysis. In Section 4.1, a biological background is given briefly. Section 4.2 outlines an overview of computational biology. Subareas of computational biological driven problems are provided with a particular focus on molecular docking and molecular dynamics in Section 4.3. These sections lay the foundation for our detailed statements of requirements of biological applications within a Grid environment that are provided in Section 4.4. The last section provides an overview to related work.

4.1 Biological Background

To lay a foundation for the requirement analysis process a biological background of the areas of molecular dynamics, genomic and genes are shortly introduced. Please see Figure 4.1 as an associated description.

The major difference in life on the earth is between eukaryoten and prokaryote. Humans, plants, or animals are developed organisms, called eukaryoten. Bacteria and archaea are lower forms of life, called prokaryote. Both forms of life exist as a cluster of cells. The most important components in a cell are the deoxyribonucleic acid (DNA), the ribonucleic acid (RNA) and proteins. DNA stores the genetic material and is packed in units called chromosomes. The chromosomes are hosted in the nucleolus for eukayote. DNA is hosted uncontrolled in the cell for procaryots. The DNA is not only genetic material, furthermore it stores important information to keep the cell and the organism alive. The DNA is double stranded and consists of two anti-parallel chains built upon four bases, adenine (A), thymine (T), guanine (G) and cytosine (C). Anti-parallel means that the two strands are hook up in a predictable manner: A can only link with T and G

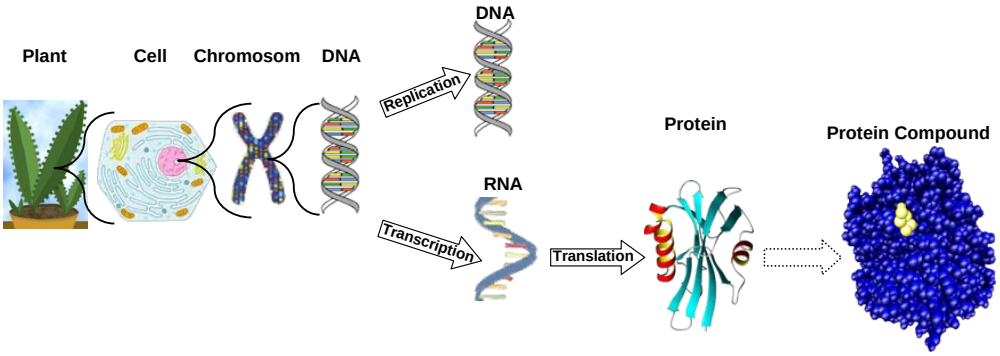

Figure 4.1: This Figure shows the biological elements of life. The proteins are compounded via the lock-key mechanism. Mostly, proteins are only chemical by active, if they are in a protein compound.

can only link with C. When a cell divides the DNA will be blue printed with a very small error rate. This mechanism is called 'replication'.
The proteins are considered as the building blocks of live. They catalyse almost all chemical reactions in a cell, regulate gene activity and provide much of the cellular structure. They are produced by the mechanisms of 'transcription' and 'translation'.
The DNA can not be used to give pieces of information directly to the cell. For this, the mechanism of 'transcription', where parts of the DNA are copied, takes place. The copies are called RNA. One RNA strand is a copy of a DNA region. That means, the RNA holds special information for a special region, which codes for e.g. a special protein. The RNA is single stranded and consists of a chain build upon four bases, adenine (A), uracil (U), guanine (G) and cytosine (C), where U replaces the T of the DNA. The RNA is responsible for a lot of things in the cell that are not fully detected until today. One purpose of the RNA is to be translated into proteins. This process is called 'translation'. A protein mainly consists of a combination of 20 amino acids. Amino acids for their part consist of chemical atoms, like carbon or hydrogen. In the translation mechanism of RNA three bases (one codon) are translated into one amino acid. That means there are 64 possible combinations of three bases to correspond to amino acids. For the fact that there are just 20 amino acids, different base combinations can code the same amino acid. The resulted proteins will fold during the translation process into an unique 3D shape. This complex folding process is not fully reproduce-able until today. Proteins are molecules in the cell and each has its own function. Often the function only arise, if a protein is in a compound with other proteins. The binding of two proteins agrees with the lock-and-key principle. The place, where a protein can bind to another molecule is called binding-site or active-site. The active site is highly specially in structure and bonds. Next to the three-dimensional structure (topology), the binding also depends on the interactions and bonds between the two molecules. The stronger both molecules bind the better they can work. In Figure 4.1, on the far right, a protein compound is shown. The yellow, very small protein binds to the binding site of the blue, big protein. The special binding process for example makes the research of new drugs very difficult, because it is the key to a functioning protein.

4.2 Computational biology

Computational Biology refers generally to biological applications, which are based on mathematical modeling, computer simulations, data integration, and algorithm development. Biological applications are meant for the generation of testable hypotheses about biological entities and processes. But computational biology spans a wide variety of biological areas, as examples molecular mechanics, protein modeling, genomic sequence, polymorphism analysis, neuronal networks, gene expression, gene regulation, evolution modeling and biochemical pathway analysis, to name a few. These areas are divided in several subareas, which in turn provide different research approaches with a wide variety of proposed biological applications. Therefore, this thesis restricts itself to one area and focuses on molecular dynamics.

Todays computational biological problems are based on knowledge that arose from the scientific progress made around the 1960s. The first important step was the development of Watson and Crick of a double-helix model of the DNA [91]. Next, the genetic code were cracked. In 1965 Margarete Dayhoff published the Atlas of Protein Sequences [55]. The Needleman-Wunsch Algorithm [75] and the Smith-Waterman [83] algorithm are methods for comparing two sequences (DNA or Protein) for similarity and were developed in 1970 and 1981. The first revolutionary biological application was developed in 1990 and provided a fast sequence similarity searching tool, BLAST (Basic Local Alignment Search Tool) [43]. It is an important tool for the evolutionary research and the best known biological application until today. The human genome project [19], which started in 1990 changed the understanding of computational biology in basic. The aim of the project was to determine the sequences of the chemical base pairs that make up the human DNA and the identification of the genome in the human DNA, which contains approximately 20,000-25,000 genes. The project was successfully finished in the year 2003. A side effect of the human genome project was the insight that computational biology is an information-based science and the next computational hurdle is to store and analyse these biological data.
Thus in the last years of the human genome project, the broad development of high-throughput measurement tools and high-throughput computing measurement strategies started. In this development the demand of computational tools to store and analyse recovered biological data has grown. Until today, this need is not slaked, because the data knowledge and analysis of biological data is still rising on an ever increasing rate. Thus, one of the major goals of modern bioinformatics is the storing and studying of bimolecular data to provide a fundamental insight into life.

Biological science is very complex, time intensive and expensive, because the encompassing study of all living organisms following a broad range of approaches, from the level of molecules to the level of ecosystems. Computational biology can help to understand and predict the behaviour of biological systems through the use of mathematical models and simulation. Computational biology lays the foundation for *in silico* experiments [62]. *In silico* is the analogy to *in vitro* and *in vivo* and refers to the computerised simulation of biological processes, with the help of computational tools. *In vitro* means to perform biological processes in an environment outside of a living organism, e.g. in a test tube.

While *in vivo* is the experiment in the living organism.
Computational biological tools analyse large biological data sets of complex processes to predict or reveal important biological behaviours. In fact, most problems, especially those involving combinatorial optimisation, are very complex and cannot be solved in a reasonable amount of time with ordinary computers, and thus raise the demand for e-Science infrastructures.

4.3 Biological Sequences and Data Challenges in Grids

E-Science infrastructures, which mainly consist of resources and middleware that are well interconnected, provide the computing power and data resource for complex biological problem domains. Solutions to many important problems in biology, such as molecular dynamic simulations, protein folding or gene expression profiling, have been effectively enabled on the Grid by computer intensive application tools [46]. The two main Grid infrastructures for computational science that include biological applications in Europe are DEISA [7] (Distributed European Infrastructure for Supercomputing Applications) and EGEE [10] (Enabling Grids for E-sciencE). Both offer a set of middleware services to make Grid resources available to scientists 24 hours a day. Many projects, e.g. XXLBIOMD [40] or OMII-Europe [25], use the services and resources provided by these Grid infrastructures. Another example is the PUMA2 Project [69], which uses Grid infrastructures to perform computationally intensive tasks for high-throughput genetic sequence analysis and metabolic reconstructions from sequence data.

The large-scale in silico experiments, used in Grid infrastructures, are also seen to have great potential in fields such as drug development. In this context, computational biology may reduce the amount of animal or human testing necessary, because researcher can predict with the help of e-science applications, a small enclosing area of promising drugs. One example for this is the WISDOM project [39]. It uses a Grid based molecular modeling workflow, named as virtual screening to discover new drugs, in particular against malaria. It uses the EGEE infrastructure for molecular docking and the DEISA infrastructure for molecular dynamics simulation in silico experiments. Molecular docking [85] is a method that predicts the binding (docking) of two molecules to one molecule complex. This binding prediction is very difficult and computational intensive, because it depends on many factors. The interaction of molecules is modeled by a scoring function, which includes terms that describes the inter- and intra molecular energy. Molecular docking is computed via tools such as FlexX [11] or AutoDock [4].

The docking methods have been significantly improved in the last decade, but a general opinion is, that docking results need to be post-processed by more accurate molecular modeling tools, such as molecular dynamics. Started with 1.000.000 molecular dockings, the best 10% hits are taken to be post-processed by a molecular dynamic (MD) [76, 42] tool. MD simulates the motion of a molecule compound to analyse and validate the complete docking process. In short, a MD application simulates the behaviour of the interaction of two molecules over a specific time in a specific system. The essence of

molecular dynamics simulations is to treat all atoms of a system under a classical mechanic consideration. The equation of the motion and the interaction between all atoms are numerically integrated in the simulation. The interactions are divided in non-bounded and bounded interactions. A potential function can describe these forces on all atoms and integrate in time to calculate Newton's equations of motion for all atoms in the system. The main result of such a calculation is a trajectory of all atoms in time: coordinates and velocities of all atoms at any simulation step.

The molecular dynamic simulation processes is realised via highly scalable MD simulation packages, e.g. AMBER [1], NAMD [21] or GROMACS [18]. These packages offer different kinds of small programs, summarised in a software package. The AMBER molecular dynamic simulation package is used as an example in the scope of this thesis. In Figure 4.2

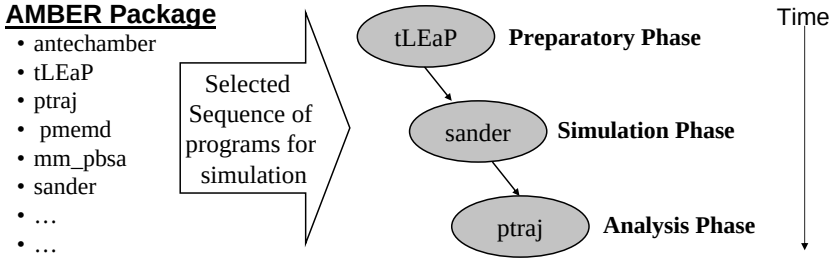

Figure 4.2: The left side shows a selection of programs of the AMBER package. The right side shows a minimal sequence of programs and its phases.

the AMBER simulation package is shown on the left side. The list shows only a subset of AMBER programs, because in total it consist of about 50 programs. The right side of the Figure, shows a minimal sequence of a molecular dynamic simulation. For the execution of a molecular dynamic simulation three main linear phases are essential. First, a preparation or creation of the input data is done, e.g. converting their formats or adding missing atoms. In the minimal sequence this step is represented via the program tLEaP. The next phase is the simulation, in which the molecules are simulated via a numerical normalisation method. This method uses well-known laws, theories and algorithms from mathematics, computer science, physics and chemistry, foremost thermodynamics and kinetics. In the minimal sequence, the simulation program is sander. Finally, several results of the simulation are analysed and evaluated. This analysis is done in the minimal sequence by the program ptraj.

It is very important to understand that the execution of the series of programs in Grid infrastructures must be strictly split into two different kinds. One is the well known method of coarse-grained Grid workflows that refer to a series of distributed executions performed on geographically dispersed Grid resources, because each execution step is represented by one single job. Thus the executions are executed on several CPUs and different job working directories typically in parallel on the Grid. Furthermore, the execution environments must not be connected, and thus the executions are independent. In contrast, the second kind is the fine-grained sequence. This is the execution of programs on a single Grid resource

(i.e. Supercomputer) and one job working environment. It enables executions to run one after another in the same environment to avoid unnecessary transfers of large data sets between different, geographically distributed Grid resources. This is beneficiary since the output of one job is the input of a subsequent job, as in the applications described above. Like evaluated in Section 2.2, e-Science applications, like molecular dynamic simulations raise the demand of being supported in Grid clients. Thus, Grid Clients raise the demand of being supported to enable the sequence method for e-Science applications in Grid clients.

The first demands in bioinformatics were only based on data storage and management. The human genome project, mentioned above, created a huge amount of data, namely the human genetic material, all sequences of 23 chromosomes (as example, the human chromosome 10 consists of 131,666,441 base pairs [56]). Consequentially, this sequences had to be analysed by biological applications to gather feasible information. Biological applications require data inputs, to be executed, biological data require biological applications to become evaluable. As example, for a MD simulation a force field that describes potential energy in a system, Cartesian coordinates for each atom of each molecule and topologies of the interacting bonds is required.

Since biological applications require biological data and researchers are organised in geographically dispersed teams, they need to extract information from large collections of data and would like to share and reuse data results of distributed resources to conduct computational applications in different locations around the world. In order to tackle the data resource challenges, Grid infrastructures provide services for data intensive computing applications [46] to access seamless and scalable different distributed resources, especially data resources. In order to obtain biological significant data for biological applications, it is desirable to use Grid environments to access data resources and perform application executions, e.g molecular dynamics, on the well-connected computational resources. In this context, one fundamental challenge arises due to the fact how to store biological data. Because of the large amount of public databases and their quality limitations, researchers use private databases. Private databases have the benefits of quality meeting own standards and that experiment results as well as different kinds of data can be stored together. Another aspect is, that private databases can be easy shared in e-Science infrastructures worldwide, by today's technologies, e.g. database access middlewares. Private databases can also vary in their types, e.g. relational databases [54] or XML databases [52]. There is an approach to store biological data in different kinds of storage methods, because biological data can be very different, e.g. in size and type. Thus, biological data is stored as plain text or numbers in a database. But large files are typically stored in a remote storage and its logical file name is saved in the database URL combined with metadata. Following, biological applications in e-Science environments require well-connections to databases and data storages. As described in Section 2.2, scientists use Grid clients to get access to Grid resources, which thus implies the demand of supporting access to biological databases.

4.4 Requirement Analysis for Client Support

Today's Grid infrastructures offer a large variety of services and distributed resources to provide among others large amounts of storage and compute power. In fact, Grids appear as a very promising technology to improve the virtual screening process by providing the connectivity between compute and storage resources to efficiently perform molecular dynamic simulations. As identified in previous sections, scientists raise the demand of biological applications and data access support in clients. From a high-level perspective, the identified requirements for support of biological applications in clients are the possibility to define sequences of programs and database access. Therefore, the aim of this thesis should be to satisfy these requirements by the provisioning of a design enabled in a suitable Grid client technology. Naturally, these requirements demand that the development implies as few as possible changes at the server tier. As a following benefit, the development will be independent of the underlying infrastructure.

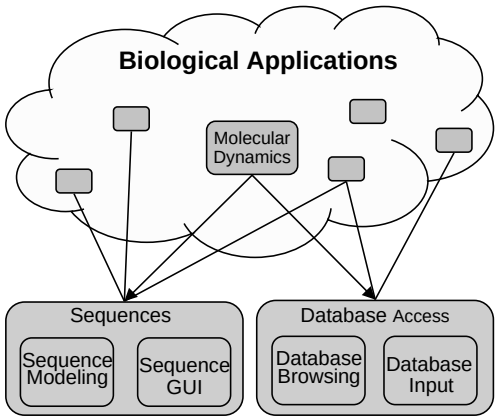

Figure 4.3: The four requirements for Client support of biological applications in e-Science infrastructures, with a particular focus on molecular dynamics

In the scope of this thesis, molecular dynamic simulation packages represent the biological application, because computational biology spans a broad variety of biological applications, too broad to mind them all. We do this without imitating the generality, as other biological applications (and from any science area) refer to the same properties, demands, behaviours and consequently to the same requirements as molecular dynamic simulation tools. This is illustrated in Figure 4.3 via a cloud that represents biological applications and in particular molecular dynamics that are built on two major requirements, fine-grained sequences and database access. In more detail, we thus identify the requirements as Sequence Modeling, Sequence GUI, Database Browsing and Data Input.

Molecular dynamic simulation packages consist of many small programs, and a simulation process consists of a sequence of programs. Additionally, the programs are depend on the data, preceding them in the sequence executions. That means, an output file from one program can be the input file of a subsequent one. An example is the preparation phase of

a simulation, where the input files are prepared for the next phase that is the simulation phase. In many Grids however, the method of sequences is not yet supported. Nevertheless, these sequences of programs are required in clients for the efficient support of molecular dynamic simulation packages in Grid infrastructures, and can be further clarified as follows:

- Sequence Modeling — The support of sequences of programs that are part of molecular dynamic simulation packages requires a basic data model. This model must describe the sequence of programs as well as the parameter of each program.

- Sequence GUI — The sequence also requires a GUI to be effectively used by scientists, since they are not computer specialists and only have a benefit of the sequence method, if they can conveniently use it in a GUI. This interface should offer possibilities to define individual sequences and parameters for each program as well as defining the sequence order without being to complicated to be used. The existing job submission service should be usable in this context.

Another major requirement is the database support since most biological applications are based on biological data (see Section 4.2). Calculations, simulations and experiments are dependent of the input data that is modified and analysed during execution. Biological data is stored in databases and thus should be accessible via a Grid Client.

Since private databases became widely known and researchers share databases for different types of data, it is very important for researchers to have access to databases that also is further clarified as follows:.

- Database Browsing — Before using any kind of data as input for biological applications, data needs to be analysed and examined. Following the prerequisite for data input from databases is the possibility of browsing databases and tables. This must be easily possible with GUIs.

- Database Inputs — Biological applications raise the requirement to fetch data from databases as input. Thus it is required to allow convenient database access and easy-to-use database file import. As a further requirement, data can only be used as input, if it is available on the target system, where the application runs. Therefore, data must be transfered between data resources and computational resources within the Grid environment transparently to the scientists. Additionally, the selected data should then be transfered with a high performance to the computational resource, without being performed manually by the scientists.

4.5 Related Work

The UNICORE Rich Client, introduced in Section 3.2 is only one option to access Grid resources in a graphical client. Many other tools can also be used to access computing resources. Additionally, there are already a lot of tools enabling the use of biological applications in Grids. In the following section we review related work and give some examples of other graphical user interfaces to access Grid infrastructures and software tools that support biological applications.

4.5.1 UNICORE Clients

The UNICORE middleware can be accessed via several different client tools. UNICORE specific ones are a command line based client named UCC [34], a Swing based GPE application client [79], and A high-level API, named HiLA [73]. The UCC is very low-level and allows for job submission, fetch output, different file transfers and administrative use, via a command line terminal. It can also be extended in various ways by using Java-based Groovy scripts. Example extensions are a Chemomentum project [6] workflow extension and a Common Information Service [72] extension.
The GPE application client, on the other hand, is a graphical client that offers easy single job submission mechanisms for applications based on GridBeans. It is only extensible via GridBeans, which extends the client functionalities with a particular focus on one particular application represented by the corresponding GridBean. For this client, several GridBeans for biological applications exist, for example a BLAST GridBean [43]. The HiLA client offers a single interface with multiple providers to easy develop clients and higher level services for accessing different back-end Grid environments. In the context of UNICORE, the HiLA API provides two back end implementations at the moment, for UNICORE 5 and a Web service based UNICORE 6 API.

To sum up, all UNICORE Clients offer seamless mechanisms to access Grid infrastructures via the UNICORE middleware. They provide mechanisms for submitting and managing jobs, use file transfer and administrate certain configuration options. But neither of them is Eclipse-based nor offers a convenient access to database resources. Only the GPE application client has a graphical support for applications and already integrates some biological applications via extended GridBeans. But this client do not support any type of sequence representation for biological applications.

4.5.2 The g-Eclipse Project

The European g-Eclipse project [12], developed an integrated workbench framework to access existing Grid infrastructures. The extensible framework is based on the Eclipse Rich Client Platform and provides a Grid model to seamlessly integrate Grid resources. The access is independent of the underlying Grid middleware, and thus the project supports a set of connectors for different middlewares, in particular to gLite [14], GRIA [17] and Amazon Web Services [5].

The central access points in g-Eclipse are virtual organisations, which are often represented with different concepts in different Grid middlewares. Thus, each user must be a member in at least one virtual organisation to get authorised and access resources and services within Grid infrastructures. One VO access to a Grid infrastructure is realised via a so-called Grid project. For example, a Grid project manages computing resources, storage resources and services. A connection in a project can be established to a local or remote file system. User can manage the files stored on local or remote file system via the file management. The Grid job management allows for the creation and management of Grid jobs.
In order to submit jobs to Grids, g-Eclipse offers a Grid Job wizard to create job descriptions, which can then be send to a target system. Workflows, consisting of single sub-jobs and can be created via a light-weight graphical editor. The workflows are translated to middleware specific workflow descriptions, but currently only for the middleware gLite. Another interesting approach is that Grid Applications can be developed within g-Eclipse development tools, debugger and monitor. These tools allow for the development of various applications, e.g. from computational biology domains. They can be deployed on single Grid nodes within the users home directory or can be uploaded to a Grid application repository for later usage or automatic installation on remote Grid nodes. A graphical support in g-Eclipse for these applications is not provided.

To sum up, the g-Eclipse Grid Client allows for the access of existing Grid infrastructures via various middleware systems, and VOs authorization mechanisms. The integrated middleware systems are GRIA, gLite and Amazon Web Services, while g-Eclipse still lacks the support of UNICORE. Additionally, users can submit and manage jobs, or access file resources, but they can not access database resources. All in all, users can develop various applications, but they can not use them within a graphical interface in g-Eclipse. Additionally, g-Eclipse do not offer any support for applications used in a sequence manner.

4.5.3 Parallel Tools Platform

The Parallel Tools Platform (PTP) project [29] is an official Eclipse project that was established in 2005 by Los Alamos National Laboratories. It aims to develop an open-source industry-strength platform for the development of parallel applications, with the properties of being robust, portable, and scalable parallel. The platform integrates a wide variety of tools for parallel programming development, currently scalable parallel debugging, parallel performance analysis, and runtime tools as well as parallel IDEs for several parallel architectures and runtime systems. More recently, the developers also work on tools to access Grid middlewares such as UNICORE. For now, the target system connection is established via a proxy protocol that communicates with a light-weight agent running in the remote system. Additionally, PTP provides a remote services abstraction layer that allow for a uniform access of local and remote computing and storage resources, but except from databases.

To sum up, PTP provides a rich platform for parallel application developers and is specifically optimized for parallel programs in general that use the Message Passing Interface (MPI) [63] for communication, but lacks sufficient support for particular biological applications currently deployed in Grids. Additionally, PTP do not provide a mechanism of using developed parallel applications in a graphical user interface.

4.5.4 Other Tools in Bioinformatics

Beside the above described Eclipse-based and/or Grid-able client platforms, there are a wide variety of tools in bioinformatics, supporting biological applications. This section highlights a few examples starting with the BioWMS [48] (Web-based Workflow Management System for bioinformatics) that supports the execution and result management of biological workflows. This system is implemented with an agent-based mobile computing middleware, which allows for a loosely-coupled intelligent execution of activity-based applications in distributed infrastructures. BioWMS offers a Web-based GUI to create workflows and enable users to add application domain features via embedding domain-specific component libraries into the agents. But BioWMS neither offers access to biological databases nor offers data inputs from databases during job execution.

Another software tool that provides the dynamically creation of workflows of sequence analyses is Pegasys [82]. It provides a GUI based workflow mechanism with a unified data model to store results. Other sequence analysis tools can be added to the Pegasys system via small parameter overheads. File inputs can not be taken from databases and the execution of workflows are executed in parallel on a computer cluster if they are independent.

Another approach to support biological applications is the discovery net system [81]. It is a middleware system, which allows developers to integrate biological analysis tools, which can be accessed by clients via Web services. Workflows can be created in clients by connecting resources and services, based on the XML-based language Discovery Process Mark-up Language. Databases are integrated resources for literature analysis and file type definition.

To sum up, typically tools in bioinformatics use Grids or computational cluster as execution environments, but are mostly specialised for biological domains, e.g. sequence analysis. Discovery net system is the only tool that integrates databases, but job inputs taken from databases are not supported. Pegasys offers an own database model which is integrated to store results. The BioWMS tool explicitly supports sequences of programs in a Grid environment, but lacks required data support features.

4.6 Summary and Conclusions

This chapter clarified that the area of computational biology is very wide and biological applications can differ from each other. Nevertheless, a wide variety of these applications can significantly benefit from Grid infrastructures, which are a suitable technology to perform large scale computing and run data intensive biological applications.
The biological background in the area of genetics was given in Section 4.1 to better understand the function of biological applications and thus the identified requirements. Therefore, Section 4.2 reviewed computational biology in many aspects. Section 4.3 outlined the use of computational biology in the Grid. This section described in detail the biological application example identified as molecular dynamics (MD), which is used as a complex example throughout in this thesis. Furthermore, this section introduced the demand of biological applications for the support of sequences of programs. These sequences of programs are necessary for applications, which consist of a set of small programs. These programs are typically executed one after another in a sequence, because output data from programs must be piped to input data of the subsequent programs. This support is necessary in Grid clients to enable biological scientists an abstract use of biological applications and Grid resources. Additionally, this section outlined, that biological applications are typically data dependent, therefore access to biological databases is required.
Section 4.5 reviewed other UNICORE-based, Eclipse-based or biological application enabling software tools. The result taken from these analyses is that non of these tools supports the combination of a graphical application support in clients, the support of sequences of programs, database access and non domain-specific client application extensibility. Based on this facts, the aim of this thesis is to develop a powerful graphical client support that enables the use of sequences of programs and database access, as well as being extensible to support various different applications.

To conclude the requirement analysis, the two major requirements are sequence support and database access in clients to support biological applications in e-Science environments. These requirements have to be satisfied with approaches that not break usual design principles of Grid infrastructures nor significantly change the technologies that are already deployed on Grid infrastructures. In more detail with respect to the clarification of the both major requirements, four concrete requirements are: sequence modeling, sequence graphical user interface, database browsing and database inputs. The sequence modeling aspect requires a method to model sequences of programs, collected in an application package. These models require a graphical user interface to enable users to build and execute sequence jobs. The sequence method enables the execution of programs in a sequence to avoid transfer of data between different geographically distributed Grid resources. The programs also require data from databases, thus a database data input method is required. While databases store large amounts of data, it must be possible to browse databases and tables by scientists ideally using the same client that is used to submit computational jobs later.

Chapter 5

Design of the Client Support for Biological Applications

The design presented in this Chapter has been developed with a focus on an Eclipse-based client support in order to facilitate the use of biological applications in e-Science infrastructures. The requirement analysis identified four main requirements, which must be met by the developed framework to satisfy the support of such applications, in particular molecular dynamic simulation packages. The requirements can be differentiated into two main categories, sequence requirements and database access requirements. The sequence requirement main aspects are sequence modeling and sequence GUI support, in particular the possibility to define and create sequences of programs of molecular dynamic application packages. In contrast, the database access main aspects are database browsing and database inputs, in particular the integration of databases into Grids to enable database access and data services in Grid clients. The design in this chapter fulfills the four identified requirements. In Section 5.1 we describe how the solution presented in this thesis is embedded in the greater architectural framework of UNICORE-based Grids. The detailed design is then described in Section 5.2 and Section 5.3.

5.1 Basic Architecture

The client support of biological applications in e-Science infrastructures in this thesis is based on the environment of the UNICORE middleware. The UNICORE 6 [36] middleware offers a ready-to-run, seamless, and secure Grid system and also provides a powerful Eclipse Rich Client Platform-based client, the UNICORE Rich Client (URC) (see Chapter 3.2). Since the users of scientific applications are often not computer specialists, they need a user-friendly GUI in order to effectively use Grids in general and the functionalities of UNICORE in particular. The URC provides many possibilities to use conveniently a wide variety of Grid resources and services. With its plugin mechanism, it also provides the extensibility to support the access to various Grid resources in general and numerous scientific applications in particular. Due to these capabilities of the URC, the approach and design principles rely on this technology and are thus embedded on the client-level of the greater Grid architecture.

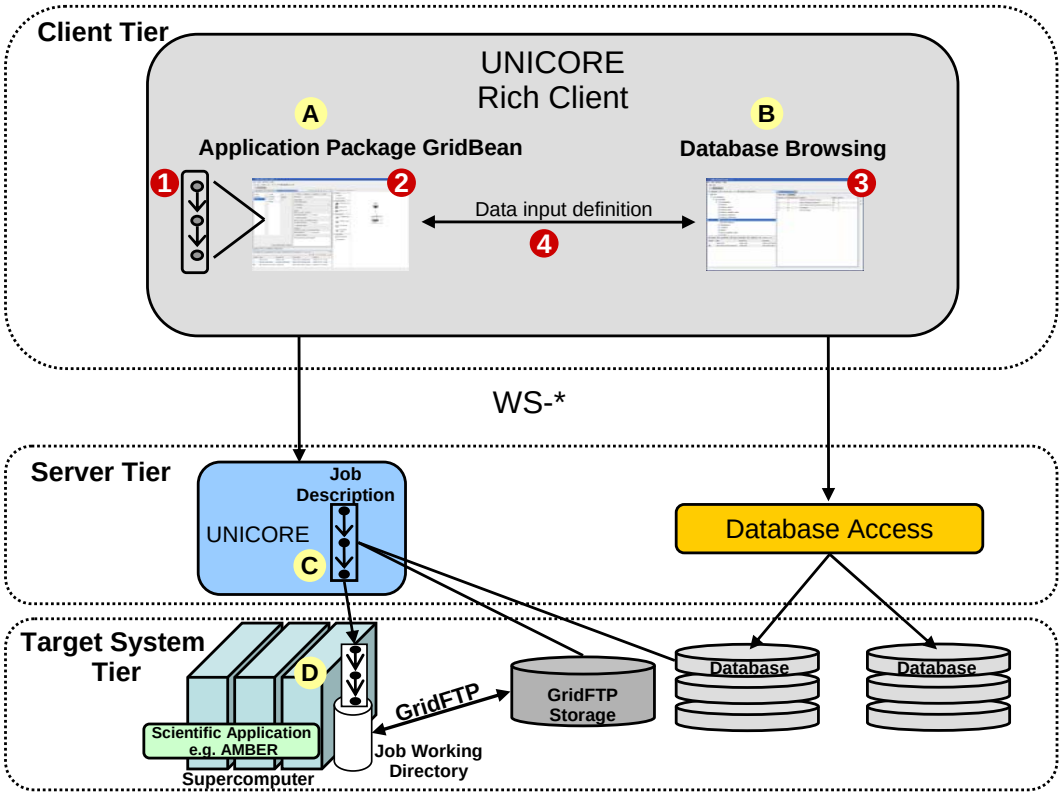

Figure 5.1: The design of the URC enhancements within the UNICORE architecture

The client tier in the UNICORE three tier architecture, provides the URC, as shown in Figure 5.1. The URC offers a ServiceBrowser that provides a view to all Grid resources and services, as well as a job view to create and modify application jobs. In this context, one goal of the design is to allow the use of molecular dynamic application packages, e.g. AMBER [1]. This design also contains a mechanisms to create sequences of programs from a set of supported programs by such respective packages. An example of a minimal sequence was illustrated in Figure 4.2 in Section 4.2. In this context, our design provides a graphical interface in the job view. The sequence creation and correspondent GUI are represented in Figure 5.1 by (A). Additionally, the design offers the possibility to assign input files from databases for each program by browsing databases and selecting required files. This selection is represented in Figure 5.1 by (B). The created sequence in turn is send as a job description (see Section 2.1 and Section 2.2) to the server tier, a middleware system (i.e. UNICORE). In Figure 5.1, this important step is described by (C). The job is processed by the middleware, which includes the substitution of environment parameters by the help of the UNICORE Incarnation Data Base (IDB) (see Section 2.1), and the files are transferred from databases or GridFTP-enabled Storage to the job execution environment at the target system tier. Finally, the sequence of programs is send to one specific target system and each program is executed in the execution environment, one after another, or more precisely: in a sequential order. This is illustrated via (D).

The design satisfies an important aspect of the requirements in terms of changing the overall environment or technologies. Hence, these steps only require very small configuration setups on the server tier, in particular an IDB entry for the molecular dynamic simulation package in UNICORE. Another requirement is the database connection at the server tier that must be established and can be realised via different database access tools, e.g. AMGA [2], OGSA-DAI [24] or GRelC [16]. These tools enable databases in Grids and can be accessed via common Grid protocols, e.g. Web services.

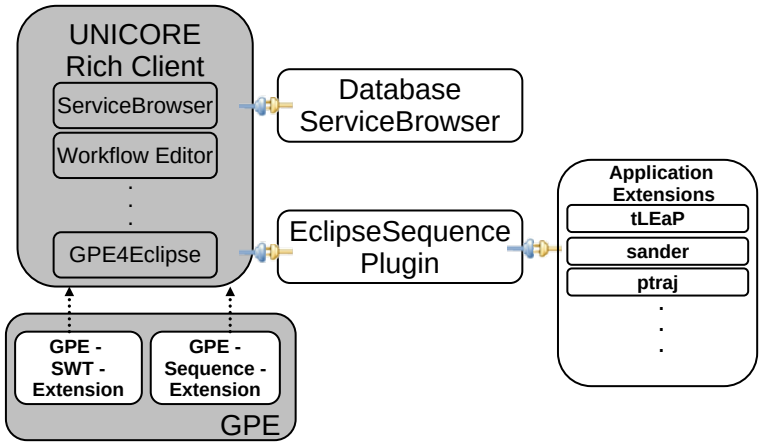

Figure 5.2: The plugin architecture of the developed enhancements in URC and GPE.

Figure 5.2 illustrates the extensible design of the new client site enhancements of the URC. These consist of a GPE-Sequence-Extension, EclipseSequencePlugin, DatabaseServiceBrowser and GPE-SWT-Extension. The GPE-Sequence-Extension and the GPE-SWT-Extension are GPE API extensions, responsible for the basic methods of sequence support and to allow for purely SWT-based user interfaces. The EclipseSequencePlugin and DatabaseServiceBrowser are URC plugins. They extend the URC to enable the possibility of using sequences as well as having seamless database access.
To satisfy the Eclipse-based client requirement, the designed GPE-SWT-Extension is necessary, because the existing Swing solution (see Section 2.3) can not accomplish this requirement. Hence, the GPE-SWT-Extension provides data controls for the native Eclipse-based Standard Widget Toolkit (see Section 3.1.1), particularly for all widgets of SWT, e.g. text field, combo box. They provide SWT interfaces for the GridBean plugins and GridBean panels.
The other extensions and plugins are described in detail in the following sections. In this context, we grouped the GPE-Sequence-Plugin and EclipseSequencePlugin to the area of Sequence Enhancements in Section 5.2. The DatabaseServiceBrowser is described in the Section 5.3 with respect to the two different aspects of database browsing and database file import capabilities.

5.2 Sequence Enhancements

One requirement, identified in our requirement analysis, is the possibility to define fine-grained sequences of programs in e-Science clients. Such programs are collected in application packages, e.g. for molecular dynamics the AMBER [1] or NAMD [21] software. The enhancements required for the design of sequence support are based on a URC plugin and a GPE extension. In this context, a new design for sequence modeling as well as for the graphical illustration of sequences within in the URC was developed. Since all application models in the URC are realised via GridBeans (see Section 2.3), it is required that the design of sequence models is also realised via GridBeans. The design features of the sequence modeling are: linear execution of the sequences, possibility to a freely selected sequence of programs as well as choose-able parameter and input files for each program in each step of the sequence. In the following a design specification of our solution for sequence support is given while the aspect of sequence modeling in Section 5.2.1 and the GUI client extensions is described in Section 5.2.2.

5.2.1 Sequence Modeling via the GPE-Sequence-Extension

The basic idea of the design of fine-grained sequence modeling is to develop an approach that fulfils sequence support demands by still matching the existing design of GPE and the URC.

The developed design concept of the GPE-Sequence-Extension extends the GPE API, and provides base types as well as special techniques to model sequences of programs and a new post-processor method to create the responsible Grid job description parts, shown in Figure 5.1 by (1). Basically, all GridBeans extend a basic GridBean model, provided by the API of the GPE. This is required to be correctly loaded into the URC. Thus, the key factor of the newly developed technique of sequence modeling is the new introduced GridBean model for a specific application package. Applying this, the package GridBean can describe a sequence of programs and is correctly loaded into the URC. Additionally, the developed technique includes that each program is also represented as a GridBean model. These program GridBean models can store typical parameter and some additional information, e.g. command line parameters of the program. All in all, the design follows a parent-child model and is shown in Figure 5.3. The parent GridBean model represents the application package, for instance a molecular dynamic simulation package. Child GridBean models represent programs in the sequence. In Figure 5.3 a minimal model for a simulation is designed by using the sequence of programs: program1 → program2 → program3.
The developed sequence modeling technique also requires a new method to produce job descriptions. This new post-processing method produces a job description for a sequence of programs, particularly for the parent GridBean and its child GridBeans.

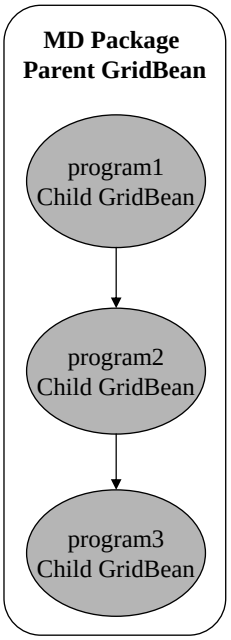

Figure 5.3: The parent GridBean stores a sequence of programs, represented as children GridBeans.

5.2.2 Sequence Graphical Interface via the EclipseSequencePlugin

The EclipseSequencePlugin provides the GUI to create sequences of programs. Because of the fact that the sequence design is independent of a special application, the EclipseSequencePlugin provides a new extension point for the addition of programs. The plugin manages extended GridBean implementations and provides generic GUIs. Parent GridBean models, like the scientific molecular dynamic simulation package, are integrated into the URC via the known load mechanism (see Section 3.2.3). This application packages are presented via a typical job view in the URC. This view shows the three generic panels (see Section 3.2.3) to provide basic services and options for job submission. The application package generic panel offers a selection mechanism for the sequence of programs to the user, as shown in Figure 5.1 by (2). It creates panels for each program to let the user modify parameter and input/output files.

5.3 Enabling Database Access

As stated in the requirements, database access is essential for the here proposed design. The previous described parts support biological applications, while this section involves databases as Grid resources into the three tier architecture by making use of an existing database access software, such as OGSA-DAI [24], AMGA [2] or GRelC [16]. In fact, database resources are mainly very heterogeneous on the physical and logical level, and data management. Due to this, database access tools typically provide uniform interfaces to access database resources (i.e. WS-DAIS OGF standard [44]). To enable database access in the URC, both data browsing and database file input have been considered in the design. These two URC extensions are incorporated in the DatabaseServiceBrowser plugin. In Section 5.3.1 we describe the design of browsing data and in Section 5.3.2 we show the design of data import.

5.3.1 Database Browsing via the DatabaseServiceBrowser

Data and metadata for biological applications are stored typically in private databases. To let the user select the right data or analyse application results, the developed DatabaseServiceBrowser provides services to establish a direct access from the client to databases. This part is illustrated in Figure 5.1 by (3). These new client functionalities enable to browse data that is stored in database tables. In addition, the DatabaseServiceBrowser provides functionalities to design and make SQL[1] queries to database tables. In the URC, Databases are represented via a new extension as Grid resource in the Grid Browser. The design of the new database Grid Browser resource offers several well defined actions for database resources. Among typical actions, it is possibility to view contents of databases and table schema's. The developed design also provides a view of table datasets and results of small or complex queries. Database resources as well as database tables and queries are shown in the Grid browser. Of course, they act on different levels of the Grid tree in the Grid Browser(see Section 3.2.2).

[1]SQL (Structured Query Language) is a standard database language to manage relational databases, as well as to define, query or update data, stored in relational databases.

5.3.2 Database Inputs via the DatabaseServiceBrowser

Another design feature of database access is the possibility to obtain data inputs from databases. The DatabaseServiceBrowser provides databases as a new source type for files in the URC. As shown in Figure 5.1 (4), data stored in databases can be selected as input for Grid application jobs. The developed design of this feature extends the URC file input mechanism. One important extension provides a selection mechanism for database items. This selection mechanism implies a manual selection that is support with a wizard feature. This wizard guides the user through a selection, in which a database resource, a table, a column and a row conveniently can be selected. Internally, it makes use of the Grid browser and a filter mechanism for database resources. As a side-remark, this wizard should be extensible in order to meet special needs of the scientists. Another extension is a special database input file post-processor (see Section 3.2.3). It is responsible for the files, whose sources are denoted as database. The post-processor translates database metadata into a file input command that is integrated into the Grid job description. Based on this precise description, the file transfer is then handled by the Grid middleware system (i.e. UNICORE).

5.4 Summary and Conclusions

In this chapter, an overview of the main architectural design is shown. In particular, it offers a detailed description of the four main developed parts, which are sequence modeling, sequence graphical user interface, database browsing and database data input. In the context of sequence support, the design of the sequence model and graphical illustration for sequences are described in detail. Here, the key features of the sequence model is the representation of an application package as parent GridBean and the sequence of programs as child GridBeans. As a side effect, the design concept leads to an GPE-Sequence-Extension. The key feature dealing with the graphical illustration for sequences is developed in the EclipseSequencePlugin and offers additionally an extension point to add other programs and their panels later. This is especially useful once a new application package version includes a new program to be supported. Database access is realised via a database access tool at the server tier and the DatabaseServiceBrowser as a URC plugin at the client tier. The DatabaseServiceBrowser plugin provides database browsing and database file input mechanisms.
To conclude, the design of sequence support and database access are the features to provide scientists with a strong support for biological applications. The sequence design makes use of the database access to enable the scientists to load data typically stored in databases. These data is used as inputs for programs of the sequence that in turn are sequentially executed on a single particular Grid resource.

Chapter 6

Design Implementation and Evaluation

This chapter describes the implementation of the proposed design of the client support for biological applications. In order to proof that this design meet the requirements of real world applications, an evaluation of a use case taken from the WISDOM workflow is also provided in this Chapter. The four basic design elements that have been implemented are the GPE-Sequence-Extension and EclipseSequencePlugin, whose implementations are described in detail in Section 6.1, the DatabaseServiceBrowser that is described in Section 6.2 and the GPE-SWT-Extension, which is described in Section 6.3.

6.1 Implementations of Sequences

The design in Chapter 5 introduced the core building blocks of sequence support for biological applications. Based on this, the intention of this section is to describe how the sequence modeling implementation and sequence GUI implementation fit into the UNICORE Rich Client, how both work and how they complement each other.
The implementation is fundamentally based on the load mechanism for application description archives of the URC. This raises the demand for a basic model that represents a sequence of programs, and a mechanism to dynamically add and remove programs. Needless to say, the application package, as well as the sequence and all programs require a GUI. In order to satisfy these requirements, the GPE-Extension provides basic types and models for sequences, the EclipseSequencePlugin provides an extensible framework,to add further program implementations, and GUI elements for programs. These implementations are described in detail in Section 6.1.1 and Section 6.1.2, respectively.

6.1.1 GPE-Sequence-Extension

As part of GPE, the application description archives (see Chapter 2.3) consist of three parts, namely the Model, Plugins and Panels. The implementation reuses the application

description archive loading, by separating the application package from programs. Such an application package description archive thus only holds information about the application package. In this context, a new mechanism to define application package GridBean Models, namely a abstract Sequence GridBean Model, was developed. Because of these developments, the load mechanism has to be extended to enable the loading of the application packages and their responsible programs as well as GUIs.

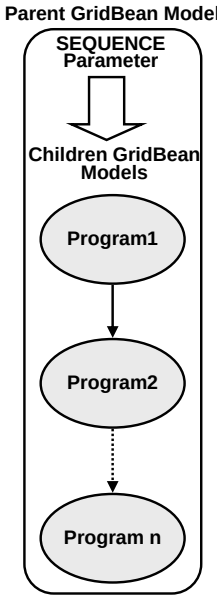

Figure 6.1: The *'SEQUENCE'* Parameter of the parent GridBean stores an ordered list of children GridBeans.

The GPE-Sequence-Extension implements the idea of the design of sequence modeling, which was described in Section 5.2.1. The new introduced method for sequence modeling manages application packages as parent and maps programs as child GridBean children. Therefore a new GridBean Model type for application packages was developed. In this parent GridBean Model, a new GridBean parameter named *'SEQUENCE'* was developed. This parameter stores a list with GridBean Models of the children, which in turn represent the user selected sequence of programs. Figure 6.1 shows the parent GridBean Model and the ordered list of child GridBean Models, saved in the new *'SEQUENCE'* parameter.
As described in Chapter 3.2.3, each special environment parameter requires a special post-processor. Therefore a new post-processor specialised for the parameter name *'SEQUENCE'* was also developed. This post-processor translates the sequence of programs by creating the commands and values for each program. The commands and values in turn stored in each child GridBean as parameter. Each parameter stores a command and the value, modified via the GUI elements by the user. The commands and values are then integrated sequentially into one script. This emerging script then represents the command line for the sequence of programs. This script is an essential element to be used in conjunction with the Grid job description for submission of the job to the Grid middleware.

The newly developed Sequence GridBean Model of the *application package description archive* is loaded into the URC via the standard load-mechanism. The Model links the load-mechanism to a newly developed *SequencePlugin*, which is a redesign of the standard GPE plugin type. This new SequencePlugin type represents the interface for the newly introduced parent GridBean model for application packages. As a side remark, the SequencePlugin naming is because of historical causes and has nothing in common with the EclipseSequencePlugin. The new load mechanism, of the application package description archives, shown in Figure 6.2, extends the standard load mechanism of the GPE4Eclipse plugin (see 3.2.3). During the load process, the new SequencePlugin forwards the parent GridBean Model via a *ModelFactoryConfigurator* to the graphical user interface of the job view of the URC. The ModelFactoryConfigurator is the link between the application package description archive and the EclipseSequencePlugin. The parent GridBean Model defines a program-list of possible programs that is evaluated by the EclipseSequencePlugin.

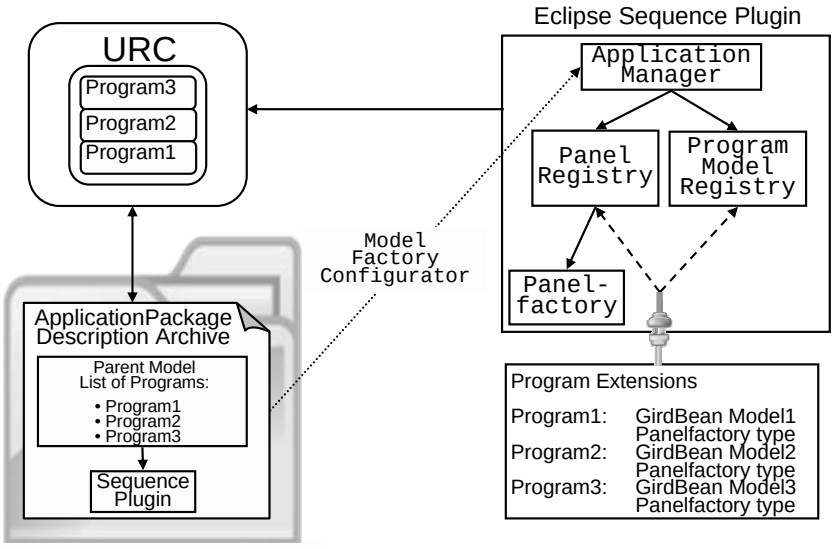

Figure 6.2: The extended load mechanism: the URC loads the Plugin of the application package description archive. The Plugin forwarded to the ModelFactoryConfigurator that links to the EclipseSequencePlugin.

To sum up, we reused the basic method of application description archive loading of the URC. The extended load mechanism links the SequencePlugin to the EclipseSequencePlugin, which in turn then creates the job view responsible for the application package GridBean Model. The new Model type for parent GridBean stores available programs and offers a new parameter named *'SEQUENCE'*, which stores the sequence of program GridBeans selected by the user. Finally, all developments can be used in any kind of application and are thus not application-specific.

6.1.2 The EclipseSequencePlugin

The EclipseSequencePlugin is an extensible framework that aims to deploy new application packages and provide GUIs for application packages and their programs. The fundamental idea is to support any application package, because many of them raise the demand of a sequential execution of their programs. Thus, we designed a dynamic framework that is extensible via plugins to add application packages and their programs. Another aspect is that for each program a GUI is required. Therefore the framework implements a dynamical build of GUI panels for programs. The EclipseSequencePlugin is extensible via a new developed extension point to plug in new program definitions. The extension point attributes consist of a fully qualified name of the program, a GridBean Model instance of the program, and a fully qualified name of a Panel Factory. All registered extensions, in this extension point are handled by a so called *ApplicationManager* (compare to Figure 6.2). The ApplicationManager is a singleton and the core of this developed framework. It sorts the extended GridBean Models into a *ProgramModelRegistry*, that maps the

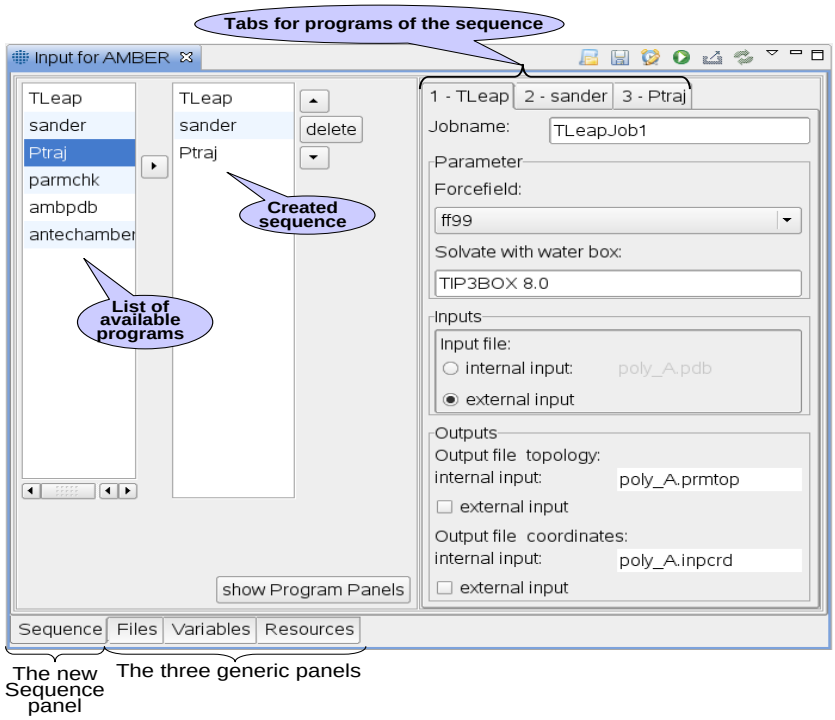

Figure 6.3: The left side shows the selection mechanism to define a sequence, the right side shows tab-panels for each program to modify certain program parameter.

program name to a program Model. The extended program GridBean Model must extend an *AbstractProgramModel*. This base model provides several parameter and methods, which program models have in common, e.g. a parameter that stores the command line. Additionally, the Panel Factory is sorted into a *PanelRegsitry*, by mapping the name of the program to the Panel Factory. We have implemented one universal *PanelFactory*, but Panel Factories can also be extended to fit special needs. The developed PanelFactory provides an implementation, which creates a graphical user interface for each program. The programs GUIs offer graphical widgets for the parameters that are defined in the GridBean model. The ApplicationManager also manages the loading of the job view for an application package. Figure 6.3 shows a screen-shot of the application package job view. The application package job view offers four panels. The three generic panels, Resource panel, File panel and Variables panel (described in 3.2.3), and our newly developed *Sequence panel*. This panel shows at the left side, a selection mechanism to create a sequence of programs. On the right side the panel shows a tab-panel-view that provides a tab for each program in the sequence in order a tab is shown to modify the program parameter.

The GUI of a job is created via selecting an application package, e.g. AMBER [1]. The URC loads the application package description archive, which delegates the parent GridBean via the ModelConfigurator to the ApplicationManager. The ApplicationManager creates the newly developed Sequence panel by selecting available programs in the corresponding application package. The ApplicationManager creates the program list by

evaluating the program-list, given in the application description archive, in particular in the application package GridBean Model, and combine these with the available programs in the EclipseSequencePlugin via extension points. The user can now create in the Sequence panel an ordered sequence of programs while multi-selection of one program is possible. The button 'show Program Panels' creates the model and GUI for each program that is defined in the sequence. The GridBean models are saved in the *'SEQUENCE'* parameter of the parent GridBean. The panel for each program is automatically build via the corresponding Panel Factory, via representing each parameter of the GridBean model by a corresponding graphical widget. The program panel state and program model state require to be consistent. Therefore, the existing consistency mechanism from GPE (see Section 3.2.3) is is applied to the models. Each child model parameter is linked by a special SWT data control with its widget. The newly developed SWT data controls are introduced in Section 6.3.

GridBean models offer the possibility to define file parameter. These file parameters are post-processed by special source or target type file post-processors. Even in child GridBean Models of programs of a sequence, we offer the possibility to define input/output files. These files can be loaded from internal or external. Internal location in this context means that the file is created by a previous program during execution that implies that the file is automatically located in the job working directory without any load or transfer mechanism. In contrast, external located files are all others that are stored, loaded, or transferred from any remote Grid resource into the job working directory for execution. To enable the facility of loading internal and external files, a *ModelLinker* in the GPE-Sequence-Extension (see Section 6.1.1) was developed. This ModelLinker is described in the following, because required design features are already introduced.

Typically, all GridBean Models are post-processed before job submission to create for all parameter the respective part in the job description. The design of sequences, in particular the storage of child GridBean Models in a parent GridBean Model bears a problem, because child GridBeans are not included in the typical post-process procedure of the URC. The newly introduced parent GridBean's post-processor for *'SEQUENCE'* parameter, post-processes each child GridBean. Thus, all parameter of the child GridBeans are post-processed by this post-processor. Nevertheless, file parameters of child GridBeans are not post-processed by their specific file type post-processors. But file inputs loaded from external, raise the demand of being post-processed by their special post-processor. The file post-processors are necessary, because they translate the file protocol, source and target into the job description, so that the Grid middleware system can load these files into the job working directory. Hence, without the file post-processors, the files would not be loaded into the job working directory. Therefore, we developed a method for child GridBean file parameters marked as external, so that they are already post-processed by their special file post-processor. This mechanism is implemented in the ModelLinker. It links a child and its parent GridBean, but only if the child GridBean uses external file inputs or outputs. The file parameters marked as external are copied and added to the parent GridBean parameters by the ModelLinker, as shown in Figure 6.4. The value of the file parameters are editable in the parent GridBean via the File panel. The ModelLinker synchronises the values of these linked file parameters of the parent and the child GridBean. All in all,

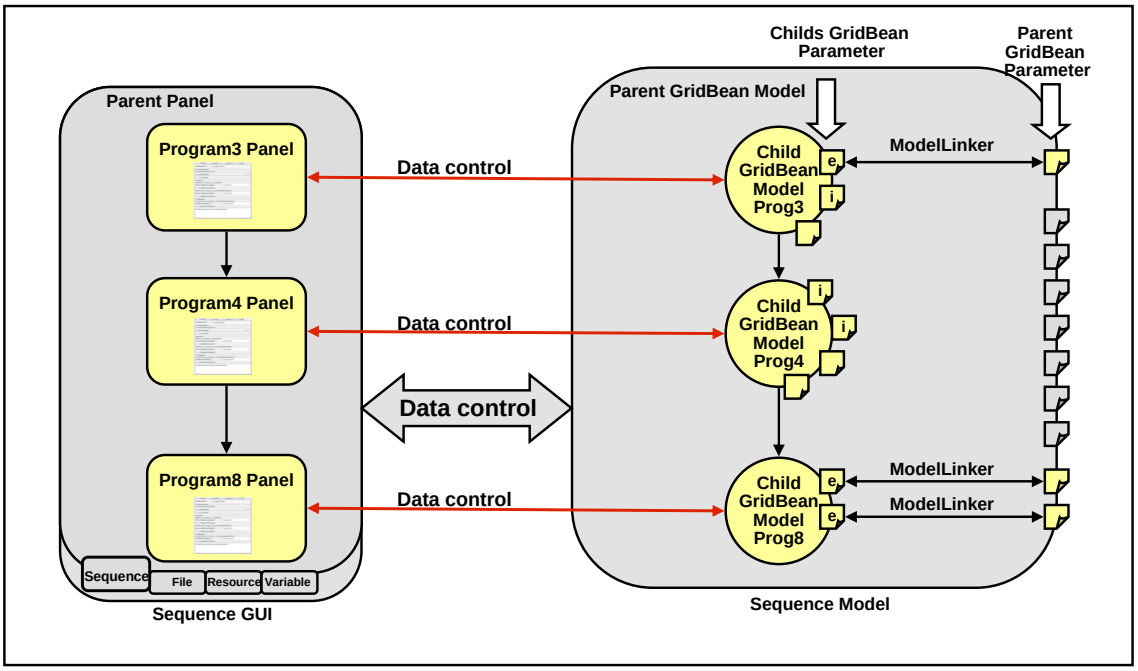

Figure 6.4: The ModelLinker links child GridBeans and parent GridBeans, if files of the child GridBean are marked as external (e). The external files are copied and added to the parent GridBean file parameter. In the Figure, three file parameter of sequence children are marked as external (e) and three are marked as internal (i). The external file parameter are also file parameter of the parent GridBean.

the external inputs and outputs of each program in the sequence are parameters of the child GridBean and the parent GridBean. The parent GridBean is thus post-processed like a normal GridBean. The specific file post-processors will post-process all file parameters including those from the child GridBeans, because they are file parameters of the parent GridBean. The implementation achieved that all external file parameters of the child GridBeans are post-processed by the specific file post-processors. Furthermore, they are translated in the job description and loaded by the Grid middleware system into the job working directory for execution.

6.2 Database Access Support

As the result of a review of the state of the art Grid data-resource access tools, the database access implementation is realized with OGSA-DAI [24] at the server tier. OGSA-DAI provides access to heterogeneous data resources in a Grid environment, e.g. relational databases or XML databases. The access is realised by using an uniform interface for database access and integration. This interface hides the different properties of databases, like database driver, data formatting and delivery mechanisms. One installation of OGSA-DAI can be used to access and manage several different databases. With OGSA-DAI, the data can be transformed, delivered, updated or queried via Web services. The provided

Web services are compliant with the Web Services Resource Framework (WSRF) [47] and thus fits nicely with the URC that also makes intensive use of this standard. It also offers large data transfers via common data transfer mechanisms, such as GridFTP [90]. According to the design, a mechanism to access databases via the UNICORE Rich Client have been implemented to enable database browsing and to import data from databases into job working directories. This development is an Eclipse-based plugin named *DatabaseServiceBrowser*. In Section 6.2.1 the integration of an OGSA-DAI connection into the URC is described. In Section 6.2.2 the developed mechanism to use database files as job inputs is presented that can be conveniently used in conjunction with the above mentioned sequence plugin.

6.2.1 OGSA-DAI Integration via the DatabaseServiceBrowser

Many scientific applications rely on data, stored in relational or other types of databases. So the aim of the DatabaseServiceBrowser is to enable the browsing and querying of databases for the users of Grid clients with the particular focus on requirements from biological applications. This implementation provides a view to databases in the Grid Browser as Grid resources. Database browsing and querying enable to see the inside of result tables. Some elements were implemented by extending and improving the OGSA-DAI client toolkit [65].

The goal of OGSA-DAI is to provide seamless access to heterogeneous data sources and resources in a Grid environment. It provides an uniform interface for data access and integration with hiding the differences of resources. In many use cases, the OGSA-DAI software is deployed in an Apache Tomcat server [3] at the server tier. The OGSA-DAI services are then accessible through the address of the tomcat server and the name of the OGSA-DAI Web services folder. This URL is responsible to access all databases, registered on one particular OGSA-DAI installation. OGSA-DAI also offers a Client toolkit [65], which provides a high-level set of Java APIs and thus protects the developer from future changes to special data service interfaces or Grid services. Basically, the client toolkit implements a set of Client stubs to access OGSA-DAI Web services at the server side. We extended the Client toolkit to reuse the base functions and build a database URC integration in the DatabaseServiceBrowser, according to the design in Figure 5.1 (3).

The *SQLClient* is the base of the developed DatabaseServiceBrowser. It offers many methods to set up a server and an OGSA-DAI specific DataRequestExecutionResource, which supports OGSA-DAI activities and executes OGSA-DAI workflow requests at the server side. These are the central accessing points in the OGSA-DAI software to make standard requests to the databases. Like described in Section 3.2.2, the ServiceBrowser of the URC provides a view to all Grid resources, in this context, the developments integrated databases as Grid resources into this view. In order to achieve that new node types for the GridBrowser to represent database resources and services have been implemented. In more detail, a *Database WSRF Node*, a *Database Registry Node*, a *Database Node*, a *Table Node* and a *Query Node* are newly integrated.
A Database WSRF Node represents a super-class of all database nodes. It provides basic

methods and facilities, which are essential for all database nodes, e.g. the 'refresh' action that refreshes all information about databases. The Database Registry Node represents an OGSA-DAI installation instance that is deployed on a Tomcat server and thus can be considered to be a root node. Typically such an installation provides access to many databases, hence each database is represented by a Database Node. A database node can have several Table and Query nodes. A Table Node represents a table of the database, while a Query Node stores and represents an executed query. Queries can address database tables or whole databases. Even, complex join queries are possible. All nodes also have several associated developed actions: an SQL query can be modified or an end-user can choose to view the table schema of a particular database table, et cetera. The contexts of tables and queries can be shown in a newly developed *DatabaseTableView*, which can be reused in several contexts.

Summarised, the implementation efforts includes extensions and improvements of the client toolkit by the above mentioned functions and offer several new nodes, actions and views in the ServiceBrowser to enable end-users with database browsing and querying in the URC. As a side remark, the DatabaseServiceBrowser is generic enough in order to reuse the extensions with other database access software tools, e.g. AMGA or GRelC. Appendix C shows some snapshots of the developed extensions.

6.2.2 Database File inputs via the DatabaseServiceBrowser

The database URC integration not only enables database browsing. Another aspect of using databases within Grids is the import of data to be used while processing jobs. The DatabaseServiceBrowser enables the possibility to select as a source type 'Database File' in the file import panel of the URC. The database file selection mechanism lets the user browse tables and select data, which can be inserted as external files into the job, which then must be transferred by the Grid middleware system data transfer mechanisms.
The development of database file input is realised via several extensions of the URC GPE4Eclipse (see Section 3.2.3). One element is the development of a new import file type, particularly a new source type. This development extends the GPE4Eclipse file input extension point. The newly database file input type must implement several methods. The file type must declare special protocol constants, a cell editor, which describes the selection mechanism in the File panel (see Section 3.2.3), as well as a description of its Grid file address. Additionally, the new file input type must define a protocol post-processor. Like mentioned above and described in Section 3.2.3, each file parameter is post-processed by its special file type post-processor to insert the actual file address for database files into the job description. The database file input post-processor is described in detail later in this section. The other main element is the selection mechanism of a database file. Biological researchers mostly use the mechanism of storing Grid file addresses in the database. That means, they store the real file in e.g. a remote storage and the file address together with metadata in the database. To take this into account, a special selection mechanism for database files was developed. It offers a wizard, where step by step a file can be conveniently selected without having any knowledge of SQL or such like. This selection mechanism begins with the selection of a table, continues with the selection of a column, where the file address is stored and ends with the selection of a row

that points to one file. This selection mechanism allows users to select a file, based on the required metadata and thus represents an important implementation feature. The wizard also reuses the Grid Browser view with a filter and the above mentioned developed table view to show the data of a table. This selection is evaluated by the database file input post-processor for job preparation. It translates the file address into a file description. This description is inserted into the job script. The middleware system then can interpret the file description and load the specified file into the working directory for execution.

To sum up, the base aspects of database file inputs were developed within the scope of the DatabaseServiceBrowser. This file import has been realised with OGSA-DAI and can also be conveniently used in conjunction with the previously described sequence plugin functionality. The implementation of the database selection method and file type post-processor is specialised to the need of biological researchers and may be extended for other scientific areas. Appendix C shows some snapshots of the developed database file import.

6.3 GPE-SWT-Extension

Intel's Grid Programming Environment (GPE), as described in Section 2.3, does not offer any support for a GUI within Eclipse. The already existing solution described in Section 3.2.3 is Swing-based [68]. However, Swing is not a solution for the developed client support, because the client requires an Eclipse based technology. Due to this fact, an extension of GPE with the Standard Widget Toolkit (see Section 3.1.1) has been implemented. The extension contains a SWT-Plugin, SWT-Panel and SWT-Data-Controls. The SWT-Plugin and SWT Panel implement conditional methods that are adapted to SWT objects and methods. SWT Button and SWT Combo (see Section 3.1.1) are examples for this. Data Controls (see Section The EclipseSequencePlugin 3.2.3) are the link between GridBean models and user interface SWT widgets, such as button or text areas. They keep the GridBean models and user inputs consistent. Therefore we developed some data controls for SWT widgets, e.g. check box, combo box and text. Thereby, the existing design of consistency maintenance was reused. The Application interfaces, derived from these newly implemented SWT GPE API extensions, are based on native Eclipse SWT objects and methods.

6.4 Evaluation Scenario

In this section, a real world use case is described that uses the developed enhancements of the UNICORE Rich Client (URC). It demonstrates feasibility of the proposed design and its implementation of database access and the developed application sequence support within a realistic e-Science environment. Additionally, it evaluates the extensibility of the URC enhancements. The WISDOM [39] use case is a scientific two step workflow of docking using Flexx or AutoDock, and molecular dynamic simulations using AMBER. It aims to detect molecules, which are potential drugs against malaria. In Chapter 6.4.1 an overview is given of the WISDOM use case in the context of the contribution of this thesis. In Chapter 6.4.2, the implementation of the AMBER GridBean is described that

represents our reference implementation of the rather abstract sequence GridBean concept. In the last chapter we evaluate the benefit of the proposed design and its implementation for database access in context of the WISDOM use case.

6.4.1 WISDOM Project Use Case

Large-scale in silico experiments are one of the most promising approaches to reduce the cost, and speed-up the development of new drugs to treat diseases like malaria. In this context, virtual screening [71] is the most computing intensive step. Virtual screening is about finding the interesting compounds of molecules, consequently a promising drug. The screening step can be done in vitro, by using real chemical molecules. But this process is very expensive and time consuming due to the fact that in turn lead to millions of molecules discovered today, following millions of possible compounds. In silico experiments, on the other hand, can not replace the in vitro and in vivo steps but they can reduce the number of in vitro experiments. This saves money and time for new drug discovery.
In silico experiments can effectively leverage the computational power provided in large scale e-Science infrastructures. This is the goal of the WISDOM (World-wide In Silico Docking On Malaria) project [39]. It aims at the development of new drugs via in silico experiments in such Grid infrastructures. The WISDOM workflow is a two step virtual screening to give the opportunity of predictions whether one molecule will bind to a particular target protein. The first part of the WISDOM workflow, is performed via molecular docking. It is computed via docking tools, like FlexX [11] or AutoDock [4]. In WISDOM, molecular docking is performed on the EGEE [10] Grid infrastructure, which is EGEE is one of the largest Grid infrastructures in Europe. The second part of the workflow are the molecular dynamic (MD) simulations [76] (compare to Section 4.2). The best 10% bindings of the molecular docking results are evaluated by a complex MD simulation process. This molecular dynamic simulation process is realised in WISDOM via a highly scalable MD application package AMBER [1], while other MD tools such as NAMD [21] or GROMACS [18] can be also used in principle.
In more detail, a MD application simulates the behaviour of a molecule compound over a specific time in a specific system. For the simulation of this second part, the WISDOM project would like to use the large-scale supercomputing facilities available on DEISA [7], which is the European high performance computing-driven Grid infrastructure. This infrastructure is suitable for massively parallel scientific jobs. In order to achieve MD on DEISA, the second WISDOM workflow step was already improved by automating the execution of a sequence of several AMBER programs [57] while its transformation to use massively parallel executions is still a work in progress.
Molecular dockings and molecular dynamic simulations are an essential preparation for in vitro testing. In this context, the OMII-Europe project [25] effort has been to deploy docking and MD on Grid infrastructures. The main goal of this project was to improve the interoperability of EGEE and DEISA to provide a seamless access to both Grids and thus accelerate drug discovery by simplifying the daily work of an e-Scientist [77]. The combination of the before mentioned workflow steps is the most promising approach to reduce the cost and to speed-up the development of new drugs to treat diseases like malaria. The developed URC enhancements in this thesis fundamentally support scientists in taking advantage of the described interoperability of both Grid infrastructures.

6.4.2 The AMBER Sequence

The second step in the WISDOM workflow is the molecular dynamic simulation that is the main driver of the developments in this thesis. The simulations are only done for the 10% of compounds with the best docking score that have been computed during the first workflow step. The MD simulation is realised by the molecular dynamic application package AMBER that was chosen by the WISDOM scientists. AMBER provides a set of programs that consists of approximately 50 small programs. One simulation of a chemical compound corresponds to a sequence of selected programs from this set. This sequence is typically represented by the three basic molecular dynamic simulation steps, described in Section 4.2. The AMBER programs must be executed one after another in a sequence, because program outputs must be piped for program inputs of a subsequent program. These input and output files, as well as input parameters have to be defined by command line parameters before execution.
The classical strategy of biological researchers is to modify an existing concrete program sequence [57] for an automated simulation run. It consists of a sequence of AMBER programs. Other researchers may use this sequence to simulate their compounds in other use cases. The input files and input parameters, as well as the workflow must be manually downloaded from external sources and uploaded to a target system and the sequence must be called from the command line. The sequence can simulate several molecule compounds in one execution.
The classical strategy and the simulation sequence can be fundamentally simplified by using the URC in conjunction with the developments of this thesis. The developed GUI for sequences enables the researchers to conveniently configure the classical sequence of programs. Furthermore, sequences can be modified and designed individually. Another benefit is, that researchers must not act directly on the operating system, which requires low-level access and accounts on a many different resources with different hardware architectures. The manual download and upload steps are omitted, because the selected data in the URC can be automatically loaded into the job execution environment by a middleware system.
The supported biological application package in the WISDOM use case is AMBER [1]. The developed framework can be extended with application packages and their programs. Therefore, we extended the framework with just a few steps by the AMBER package. These steps included: creation of an AMBER application package description archive with an AMBER parent GridBean model and AMBER SequencePlugin. Additionally, one step includes the creation of extensions for supported programs, with the name of the program, a child GridBean model and the name of a Panel Factory. Other application packages can be added analogue.
After these steps, the AMBER application package can be used in the URC. This package is based among other things on the abstract Sequence GridBean, as shown in Figure 6.5 (1). The graphical user interface (see Appendix D) is built automatically and each user can conveniently choose an individual sequence of AMBER programs. In Figure 6.5 (2) the program GridBean GUIs are shown. They are created automatically by the Panel Factory of the framework described in Section 6.1.2. The user is able to modify parameter and input files for each program much more simpler as before.

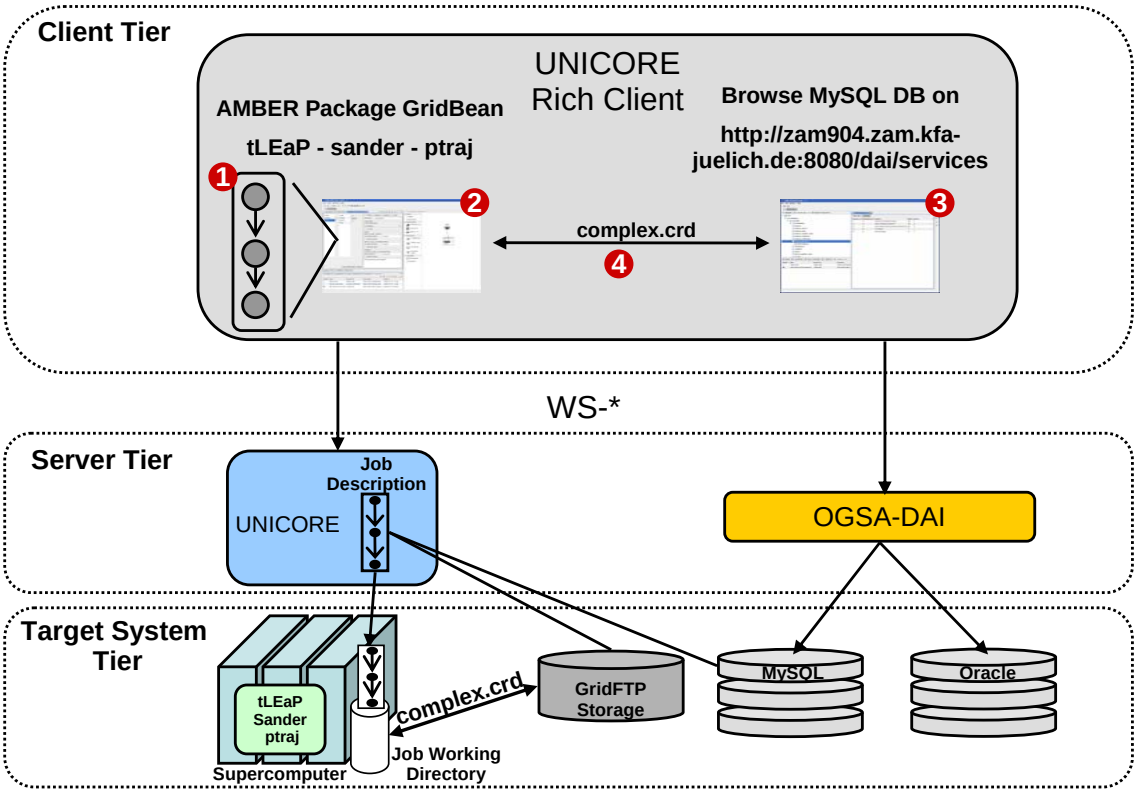

Figure 6.5: The overview of the developed AMBER GridBean and database access. As example, a file named complex.crd is used as external input for the sequence of AMBER programs.

By submitting the job, all these parameters are post-processed and a corresponding job script is send to the Grid middleware system. It forwards the job for execution to a batch-system that schedules the job on the correspondent Grid resource. The results are received by the typical URC functionalities.

6.4.3 Simplified Database Access

Other important implemented features next to the above described sequence of programs, are the URC database browsing and database file input functionalities. This database access methods were developed and integrated into the URC within this thesis. The use case, described in 6.4.1 raises the demand for this feature. The biological programs of AMBER, typically need biological input data to calculate a simulation. To avoid long running download and upload processes, researchers can conveniently use the newly developed database browsing (see Figure 6.5(3)) and database file imports (see Figure 6.5(4)) in the URC for workflows or jobs, as one significant result of this thesis.
To provide an example, in Figure 6.5 we use as file input a 'complex.crd' file. The file is stored on a GridFTP server. The Grid file address is stored with some metadata in a MySQL database. This scenario is very typical in a biological environment and influenced

the design and implementation of the data import mechanism, as described in Section 6.2.1. The needed file address for 'complex.crd' is saved in a database in the database registry 'http://localhost:8080/dai/services'. The researcher is able to browse through the database to find the right complex.crd file, using the available meta information (compare to Figure 6.5 (3)). The next step is to include this file via the database file import mechanism (see Section 6.2.2). Therefore, the researcher selects the database file input parameter and mark it as external file in the program panel (see Section 6.1.2). In the File panel (see Section 3.2.3) of the parent GridBean this parameter can now be modified by the selection method, described in Section 6.2.2. The researcher can select the complex.crd file. The database file post-processor loads required pieces of information from the database and includes the address of the selected complex.crd file into the Job description, please compare this to Figure 6.5 (4). Based on this description, the UNICORE middleware can load this file via GridFTP [90] into the working directory of the job. The job can then be executed taking the browsed files as input.

6.5 Summary and Conclusions

This chapter presents a new way of supporting clients of biological applications within an e-Science environment based on the design provided in Chapter 5. Mainly these implementations are focused on the development of sequences and database access within the UNICORE Rich Client, but many concepts can be applied to other Grid middleware clients as well. The four essential parts of the implementation in this thesis are: sequence modeling, sequence graphical user interface, database browsing, and database data input. In this context, the building blocks of the implementations are the advancement of the graphical representation of GridBeans, with the possibility to use the Eclipse Standard Widget Toolkit. Furthermore the development of the model for application packages and their programs is another building block. All in all, MD application packages can now be used conveniently as a sequence of programs within the UNICORE Rich Client. These sequences are modeled as a GridBean, storing several program GridBeans as a sorted list. The GUI for GridBeans was also extended to support the visualisation of the new model of sequences and thus to ease the use of sequence GridBeans.
The database access was realised at the server tier by using OGSA-DAI as one possible accessing tool and at the client tier by developing the DatabaseServiceBrowser. It extends and enhances the OGSA-DAI Client toolkit. Our database browsing mechanism presents databases, tables and queries as Grid resources. As a result, table and query results can be displayed in a newly introduced Eclipse view in the URC. Data input can be used for every application, it is therefore developed as a new input mechanism type of the URC. The layered wizard selection mechanism for databases was specialised for biological domains but this concept can be also applied to other application domains.

Finally, we evaluated the developed URC-based enhancements and the proposed design on a concrete use case. The WISDOM use case is a scientific workflow of docking and MD simulations to detect molecules, which are potential drugs against malaria. We adopted the MD simulation step on our development and keep close contacts to the researchers in order to fulfil their needs. The MD simulation package used in WISDOM is AMBER. In

the scope of the evaluation, we integrated several AMBER programs in the URC. This was possible, because of the extensibility of the developed URC enhancements that allow for the development of specialized sequence-based GridBeans like the AMBER GridBean. Mainly we integrated those programs, which are essential to build a program sequence for a fully functioning MD simulation. As a consequence, the program sequence and simulation is identical with the manually developed workflow [57], but much simpler to be defined and used by e-scientists. This workflow can be rebuild and then re-run by the URC, with the benefit of getting access to computational resources and to data resources. The needed data can be via the newly developed database input mechanism in the URC.

Chapter 7

Summary and Conclusion

This thesis described the Eclipse-based client support for biological applications within e-Science infrastructures. The undertaken requirement analysis and its evaluation identified that biological applications usually require database access and the possibility to define sequences of programs. These requirements are satisfied by the usage of Grid infrastructures and by provisioning of an extensible framework, which is implemented as a proof of concept within the Eclipse-based UNICORE Rich Client (URC). All in all, several core building blocks have been developed to realise database access and sequence design features in the URC, particularly the support of sequence modeling, sequence GUIs, database browsing and data inputs obtained from databases. To sum up, the implementation contains a mechanism to define sequences of programs and a framework for conveniently add other program models, providing an automated graphical visualisation technique for sequences of programs. Additionally, the implementation contains a plugin that enables the user to browse databases and include data of databases into job executions. A gained result of the developed abstraction of applications is that, in contrast to manually created UNIX scripts, the creation of program sequences has become faster and fail-safe. Hence, scientists do not need to be aware of application dependent languages and program specific parameters.

This thesis started with a short introduction of e-Science infrastructures and Grid technology in Chapter 2. An overview of e-Science Applications and an introduction to the Grid Programming Environment that provides a higher-level API for the development of e-Science applications in Grid clients is also given. As a basic foundation, Chapter 3 reviews the core concepts of the Eclipse Rich Client Platform and outlines the architecture of the URC, an Eclipse-based technology. The URC is described in detail with attention to the ServiceBrowser and the GPE4Eclipse plugin. Based on the URC and conversations with scientists that apply molecular dynamic simulations on supercomputers, Chapter 4 lists the identified requirements. For better understanding, a short background of theory about genetics and describes computational biology is given. Additionally, an extensive analysis of required features for client support of biological applications in e-Science infrastructures is provided, along with a survey of related work. The requirements have been taken into account to create several URC enhancements described in Chapter 5 and thus gives insights to the core building blocks of the design of sequence support as well as

database access. Finally, Chapter 6 describes a proof-of-concept implementation of the design within the URC. In more detail, this implementation includes enhancements of the GPE API to support the sequence design of programs. Furthermore, in the scope of this thesis, a framework has been developed that provides graphical visualisation for sequences of programs, and an extension point to easy plug in other programs for sequential use into the URC. Additionally, the UNICORE Rich Client File import was extended by a DatabaseServiceBrowser and data input types for database data. The proposed design of enhancements of the UNICORE Rich Client offer extension points and plugin mechanisms to add additional functionality.

In order to prove whether our approach is feasible in the context of real applications, an evaluation use case of the developments was provided. The evaluation, which adopted the second step of the WISDOM workflow, has shown that the developed enhancements, which are sequence modeling, sequence graphical user interface, database browsing and database data input of the URC, are suitable for biological applications within an Grid environment. This use case reveals, by extending the developed framework with several AMBER programs that the developed enhancements of the URC meet our requirements for the support of program sequences within an e-Science environment, particularly within a Grid infrastructure that uses UNICORE 6. The developed UNICORE Rich Client enhancements provide the following benefits for inherent sequential biological applications as well as for the daily work of an scientist by using e-Science infrastructures.

Advantages of the developed framework

- New application packages and their programs can be easily integrated by using the extension point mechanism.
- The GUIs for sequences is offered by the framework, GUI implementations are unnecessary.
- GUI implementations for the programs are unnecessary but possible, GUIs for programs are created automatically.

Advantages for scientists using the UNICORE Rich Client

- The sequence order of applications can be manually defined by using a nice graphical user interface. Programs may be selected multiple times in one sequence definition.
- Parameter and input files can be declared and selected via a convenient graphical user interface. The command line parameters are created automatically.
- The sequence is executed linearly in the defined order to guarantee file passing between the sequence steps.
- Files can be used as input for applications, these files may be produced within the sequence or imported from external sources.

- Data from biological databases can be conveniently browsed, selected and used as inputs for applications. The associated file transfers that are defined on the client-level are handled automatically by the server middleware without manual client interaction.
- Scientists can submit computational jobs with sequences of programs via the UNICORE Rich Client and must not act directly on a target system.
- Sequences might be embedded into a for-each-loop, each iteration may then be executed independently, even in parallel.

One interesting future work option is to offer a mechanism to reuse the sequences, thus biologists can share their developed sequences with other researchers. Additionally, the integration of public biological databases as Grid data resources is a logical next step, as soon as they offer Web service interfaces or become accessible via a database accessing tool. Furthermore, within our development, parameter sweeps for biological applications, in particular molecular dynamic simulations, are another interesting use case. Our discussions also lead to ideas that store execution outputs in databases with meta-data and the file address. Furthermore, other application packages may be developed as an URC extension using the results of this thesis as a foundation. One particular example in this context might be a RemoteCompilerGridBean that takes source code (e.g. C++) and provides a compilation step as initial part of the sequence while its execution is the latter part of the necessarily ordered sequence.

Bibliography

[1] AMBER - Assisted Model Building with Energy Refinement. http://ambermd.org/.

[2] AMGA - ARDA Metadata Catalogue Project. http://amga.web.cern.ch/amga/.

[3] Apache Tomcat. http://tomcat.apache.org/.

[4] AutoDock. http://autodock.scripps.edu/.

[5] AWS - Amazon Web Services. http://aws.amazon.com/.

[6] Chemomentum project. http://www.chemomentum.org/c9m.

[7] DEISA - Distributed European Infrastructure for Supercomputing Applications. http://www.deisa.org.

[8] Distributed Management Task Force, inc. http://www.dmtf.org/home.

[9] Eclipse Rich Client Platform. http://www.eclipse.org/home/categories/rcp.php.

[10] EGEE - Enabling Grids for E-sciencE. http://www.eu-egee.org/.

[11] FlexX. http://www.biosolveit.de/FlexX/.

[12] g-Eclipse - Access the power of the Grid. http://www.geclipse.org.

[13] German National Grid: D-Grid. http://www.d-grid.de.

[14] gLite. http://glite.web.cern.ch/glite/.

[15] Globus Toolkit. http://www.globus.org/toolkit/.

[16] GRelC - Grid Relational Catalog Project. http://grelc.unile.it/home.php.

[17] GRIA - Service Oriented Collaborations for Industry and Commerce. http://www.gria.org.

[18] GROMACS. http://www.gromacs.org/.

[19] Human Genome Project Information. www.ornl.gov/hgmis/home.html.

[20] Intel´s Grid Programming Environment. http://gpe4gtk.sourceforge.net/GPE-Whitepaper.pdf.

[21] NAMD - Scaleable Molecular Dynmaics. http://www.ks.uiuc.edu/Research/namd/.

[22] Nordu Grid. http://www.nordugrid.org/.

[23] OASIS - Advancing Open Standards for the global Information Society. http://www.oasis-open.org/home/index.php.

[24] OGSA-DAI. http://www.ogsadai.org.uk/.

[25] OMII - Europe. http://omii-europe.org/.

[26] Open Grid Forum. http://www.ogf.org/.

[27] Open Science Grid. http://www.opensciencegrid.org/.

[28] OSGi Alliance - The Dynamic Module System for Java. http://www.osgi.org/Main/HomePage.

[29] PTP - Parallel Tools Platform. http://www.eclipse.org/ptp/.

[30] Swiss Grid. http://www.swissgrid.ch/.

[31] TeraGrid. http://www.teragrid.org.

[32] The AWT - Abstract Window Toolkit. http://java.sun.com/products/jdk/awt/.

[33] The Source for Java Developers. http://java.sun.com/.

[34] UCC - UNICORE command line client. http://www.unicore.eu/documentation/manuals/unicore6/ucc/index.html.

[35] UK e-Science Initiative. http://www.rcuk.ac.uk/escience/.

[36] UNICORE - UNiform Interface to COmputing REsources. http://www.unicore.eu/.

[37] UNICORE Rich Client. http://www.unicore.eu/documentation/manuals/unicore6/files/RichClient.pdf.

[38] W3C - The World Wide Web Consortium. http://www.w3.org/.

[39] WISDOM - Wide In Silico Docking On Malaria. http://wisdom.eu-egee.fr/.

[40] XXLBIOMD - Large Scale Protein Interactions. http://www.deisa.eu/science/deci/projects2007-2008/XXLBIOMD.

[41] *Getting Started with an Integrated Development Environment (IDE)*, 2005. http://java.sun.com/developer/technicalArticles/tools/intro.html.

[42] M. P. Allen. Introduction to Molecular Dynamics Simulation. *Computational Soft Matter: From Synthetic Polymers to Proteins*, Vol. 23:1–28, 2004.

[43] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410, Oct 1990.

[44] M. Antonioletti et al. Web Services Data Access and Integration - The Relational Realisation (WS-DAIR) Specification, 2006. http://www.ogf.org/documents/GFD.76.pdf.

[45] M. Baker, A. W. Apon, C. Ferner, and J. Brown. Emerging Grid Standards. *IEEE Computer*, 38(4):43–50, 2005.

[46] K. Baldridge and P. E. Bourne. *The New Biology and the Grid.* John Wiley and Sons, April 2003.

[47] T. Banks. Web Services Resource Framework (WSRF), Primer v1.2, 2006. http://docs.oasis-open.org/wsrf/wsrf-primer-1.2-primer-cd-01.pdf.

[48] E. Bartocci, F. Corradini, E. Merelli, and L. Scortichini. BioWMS: a web-based Workflow Management System for bioinformatics. *BMC Bioinformatics*, 8 Suppl 1, 2007.

[49] F. Berman, G. C. Fox, and A. J. G. Hey, editors. *Grid Computing: Making The Global Infrastructure a Reality.* Wiley and Sons, April 2003.

[50] D. Booth and C. K. Liu. Web Services Description Language (WSDL) Version 2.0. Candidate recommendation, W3C, March 2006. http://www.w3.org/TR/wsdl.

[51] J. Brainard, A. Juels, R. L. Rivest, M. Szydlo, and M. Yung. Fourth-factor Authentication: somebody you know. In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, pages 168–178, New York, NY, USA, 2006. ACM.

[52] A. Chaudhri, R. Zicari, and A. Rashid. *XML Data Management: Native XML and XML Enabled DataBase Systems.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[53] E. Clayberg and D. Rubel. *Eclipse: building commercial-quality plug-ins, third edition.* Addison Wesley Professional, 2008.

[54] E. F. Codd. *The relational model for database management: Version 2.* Addison-Wesley Longman Publishing Co., Boston, USA, 1990.

[55] M. O. Dayhoff. *Atlas of protein sequence and structure.* National Biomed Research Foundation, 1965.

[56] P. Deloukas et al. The DNA sequence and comparative analysis of human chromosome 10. *Nature*, 429:375–381, 2004.

[57] A. M. Ferrari, G. Degliesposto, M. Sgobba, and G. Rastelli. Validation of an automated procedure for the prediction of relative free energies of binding on a set of aldose reductase inhibitors. *Bioorganic and medicinal chemistry*, 15(24):7865–7877, 2007.

[58] I. Foster, J. Frey, S. Graham, and S. Tuecke. Modelling Stateful Resources with Web Services Version1.1, 2004.

[59] I. Foster and C. Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure (2nd Edition).* Morgan Kaufmann, November 2003.

[60] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications*, 15(3):200–222, August 2001.

[61] A. O. Freier, P. Karlton, and P. C. Kocher. The SSL Protocol Version 3.0. January 1996.

[62] C. Goble, C. Greenhalgh, S. Pettifer, and R. Stevens. Knowledge Integration: In Silicio Experiments in Bioinformatics. In *The Grid 2: Blueprint for a New Computing Infrastructure (2nd Edition)*, pages 121–134. Morgan Kaufmann, 2004.

[63] W. Gropp et al. *MPI – The Complete Reference: Volume 2, the MPI-2 Extensions.* MIT Press, Cambridge, MA, EUA, 1998.

[64] R. Housley, W. Ford, W. Polk, and D. Solo. Internet X.509 Public Key Infrastructure Certificate and CRL Profile. RFC 2459, Internet Engineering Task Force, jan 1999.

[65] A. Hume, M. Jackson, et al. Protecting Application Developers - A Client Toolkit for OGSA-DAI. In *UK e-Science All Hands Meeting*, 2004.

[66] B. Jacob, M. Brown, et al. *Introduction to Grid Computing.* IBM Corp., Riverton, NJ, USA, 2005.

[67] M. Lorch, D. Kafura, et al. Authorisation and identity mapping services for the Open Science Grid. *Int. J. High Perform. Comput. Netw.*, 5(3):144–155, 2008.

[68] M. Loy, B. Cole, J. Elliot, R. Eckstein, and D. Wood. *Java Swing.* O'Reilly Media, 2 edition, 2002.

[69] N. Maltsev, E. Glass, et al. PUMA2, Grid-based high-throughput analysis of genomes and metabolic pathways. *Nucleic Acids Research*, 34:369–372, 2006.

[70] J. McAffer and J.-M. Lemieux. *Eclipse Rich Client Platform: Designing, Coding, and Packaging Java(TM) Applications.* Addison-Wesley Professional, 2005.

[71] C. McInnes. Virtual screening strategies in drug discovery. *Current Opinion in Chemical Biology*, 2007.

[72] A. S. Memon, M. S. Memon, P. Wieder, and B. Schuller. CIS: An Information Service Based on the Common Information Model. In *E-SCIENCE '07: Proceedings of the Third IEEE International Conference on e-Science and Grid Computing*, pages 465–473, Washington, DC, USA, 2007. IEEE Computer Society.

[73] R. Menday and B. Hagemeier. HiLA 1.0. http://www.unicore.eu/community/development/hila-reference.pdf.

[74] N. Mitra. SOAP Version 1.2. Technical report, W3C, June 2003. http://www.w3.org/TR/soap/.

[75] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, March 1970.

[76] D. C. Rapaport. *The Art of Molecular Dynamics Simulation - 2nd Edition.* Cambridge University Press, New York, NY, USA, 2004.

[77] M. Riedel et al. Improving e-Science with Interoperability of the e-Infrastructures EGEE and DEISA. In *Proceedings of the 31st International Convention MIPRO, Conference on Grid and Visualization Systems (GVS)*, pages 225–231, Opatija, Croatia, 2008.

[78] M. Riedel, W. Frings, et al. Requirements and Design of a Collaborative Online Visualization and Steering Framework for Grid and e-Science infrastructures, 2007.

[79] M. Riedel, D. Mallmann, et al. GridBeans: Supporting e-Science and Grid Applications. In *e-Science*, page 45. IEEE Computer Society, 2006.

[80] M. Riedel, B. Schuller, et al. Web Services Interfaces and Open Standards Integration into the European UNICORE 6 Grid Middleware. In *Proceedings of 2007 Middleware for Web Services Workshop at 11th International IEEE EDOC Conference "The Enterprise Computing Conference"*, pages 57–60, 2007.

[81] A. Rowe, D. Kalaitzopoulos, M. Osmond, M. Ghanem, and Y. Guo. The discovery net system for high throughput bioinformatics. In *ISMB (Supplement of Bioinformatics)*, pages 225–231, 2003.

[82] S. P. Shah et al. Pegasys: software for executing and integrating analyses of biological sequences. *BMC Bioinformatics*, 5, April 2004.

[83] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, March 1981.

[84] D. F. Snelling, S. van den Berghe, and V. Qian. Explicit trust delegation: Security for dynamic grids. *FUJITSU Sci.Tech.Journal*, 40:282–294, 2004.

[85] L. T and R. M. Computational methods for biomolecular docking. *Current Opinion in Structural Biology*, 6:402–406, 1996.

[86] D. Talia, R. Yahyapour, and W. Ziegler. *Grid Middleware and Services: Challenges and Solutions.* Springer Publishing Company, Incorporated, 2008.

[87] I. J. Taylor, E. Deelman, D. B. Gannon, and M. Shields. *Workflows for e-Science: Scientific Workflows for Grids.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[88] S. Tuecke et al. Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile. RFC 3820, Internet Engineering Task Force, jun 2004.

[89] J. Tyler. e-Science Definition. http://www.e-science.clrc.ac.uk.

[90] W.Allcock et al. *GridFTP: Protocol Extensions to FTP for the Grid.* Open Grid Forum Draft - No. 20, 2004. http://www.ggf.org/documents/.

[91] J. D. Watson and F. H. C. Crick. Molecular Structure of Nucleic Acids: A Structure for Deoxyribose Nucleic Acid. *Nature*, 171:737–738, April 1953.

List of Figures

List of Acronyms

AMBER - Assisted Model Building with Energy Refinement
API - Application Programming Interface
BioWMS - Web-based Workflow Management System for bioinformatics
CPU - Central Processing Unit
DEISA - Distributed European Infrastructure for Supercomputing Applications
DNA - Deoxyribonucleic Acid
e-Science - Enhanced Science
EGEE - Enabling Grids for E-SciencE
GPE - Grid Programming Environment
GUI - Graphical User Interface
HiLA - High Level API
HTTP - Hypertext Transfer Protocol
IDB - Incarnation Database
IDE - Integrated Development Environment
IETF - Internet Engineering TaskForce
MD - Molecular Dynamics
OGF - Open Grid Forum
OSGi - Open Services Gateway initiative
PTP - Parallel Tools Platform
RCP - Rich Client Platform
RNA - Ribonucleic Acid
SOAP - Simple Object Access Protocol
SQL - Structured Query Language
SWT - Standard Widget Toolkit
TSI - Target System Interface
UCC - UNICORE Command line Client
UDDI - Universal Description, Discovery and Integration
UNICORE - UNiform Interface to COmputing REsources
URC - UNICORE Rich Client
VO - Virtual Organizations
WISDOM - World-wide In Silico Docking On Malaria
WS - Web Service
WSDL - Web Service Description Language
WSRF - Web Services Resource Framework
XNJS - Network Job Supervisor
XUUDB - UNICORE User Database

Appendix A - The UNICORE Rich Client

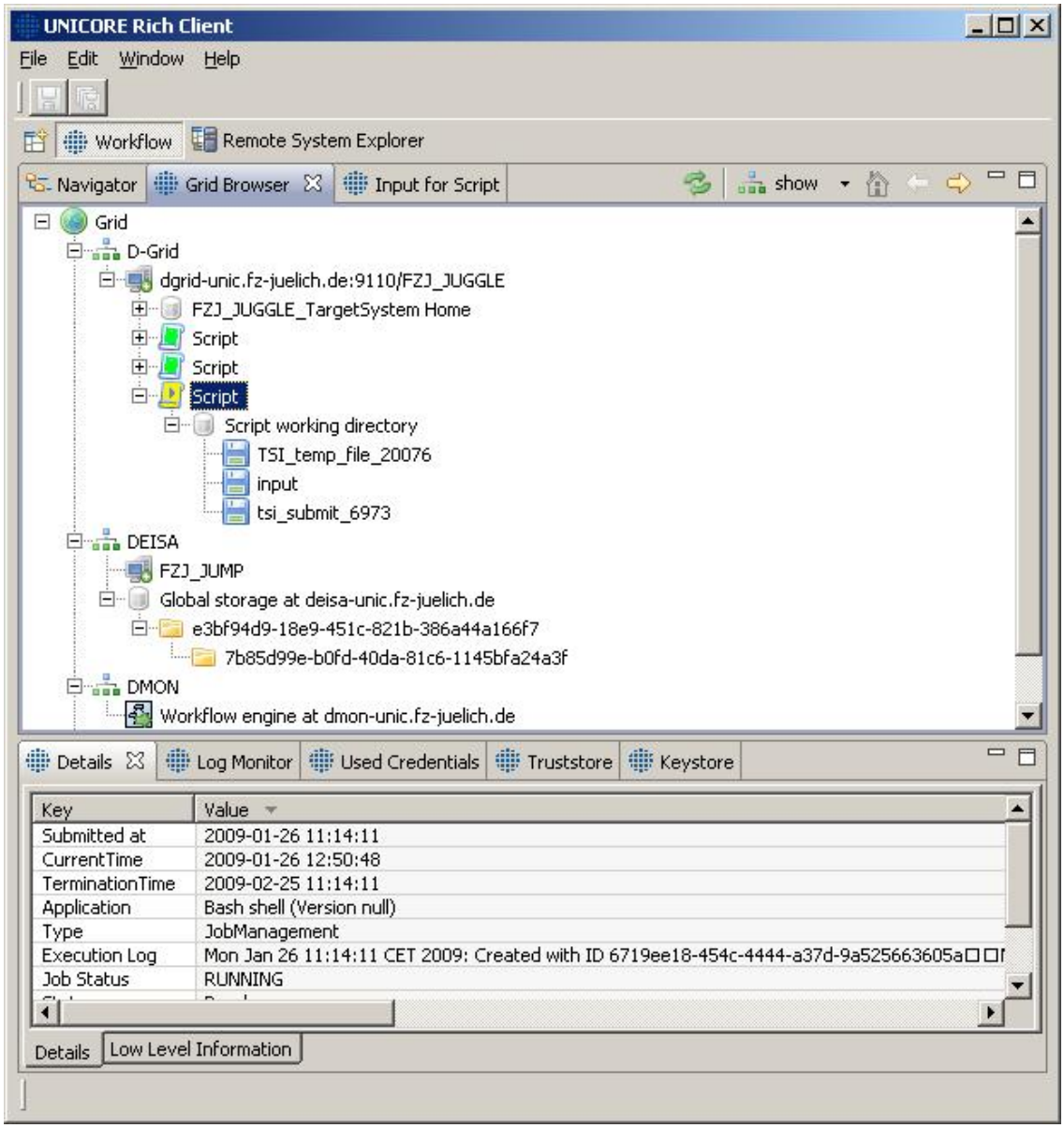

Figure 1: The screenshot shows the Grid Browser and its hierarchical tree structure. The green marked scripts are successful Grid jobs, the yellow marked script is a currently running Grid job. In the bottom different views are shown with detailed information about the resources, the log monitor, the keystore view, the truststore view, and the used credentials.

Appendix B - The UNICORE Rich Client Job Creation View

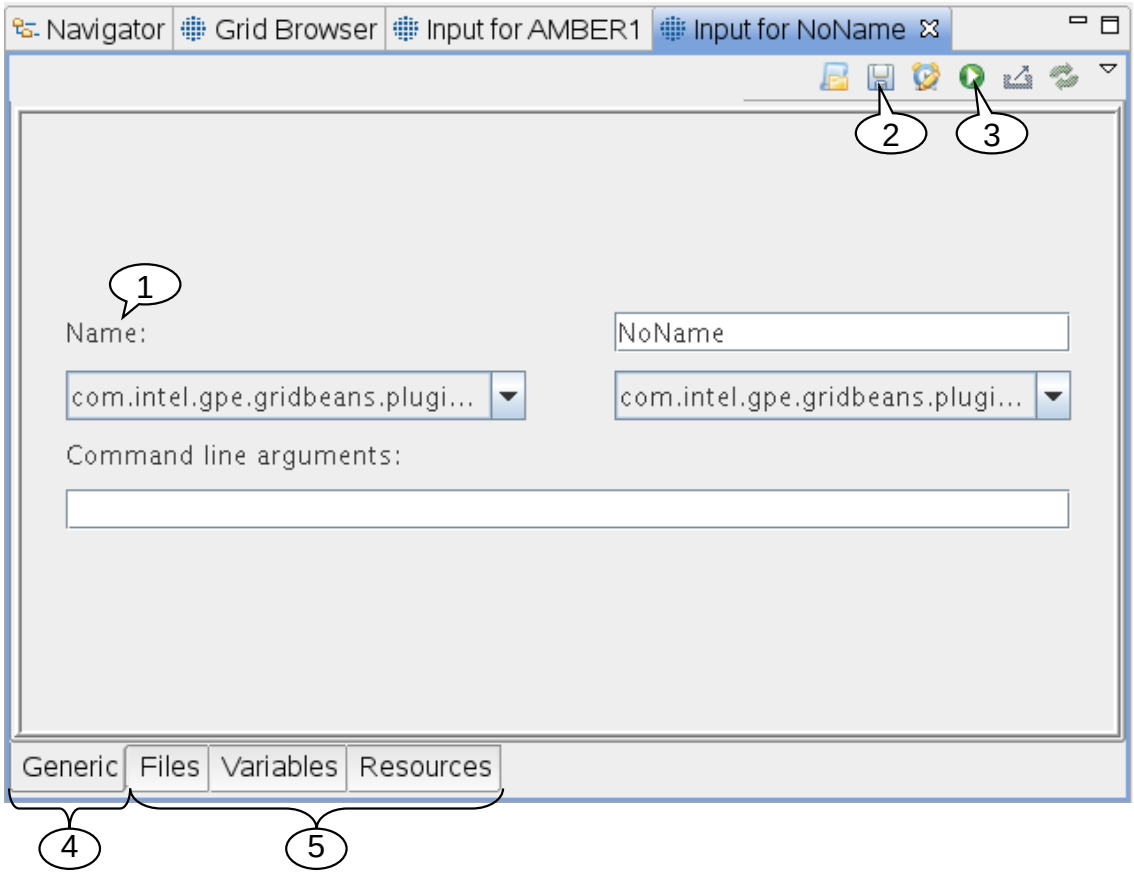

Figure 2: The screenshot shows a typical view of a job. This job view provided by the Generic GridBean. (1) shows the name, which is essential for all jobs. (2) is the button to store the Grid job, (3) is the button to submit the Grid job. (4) is the activated tab panel for this GridBean, (5) shows the three generic tab panels, which provide key features for each job view.

Appendix C - The UNICORE Rich Client Database Access(1)

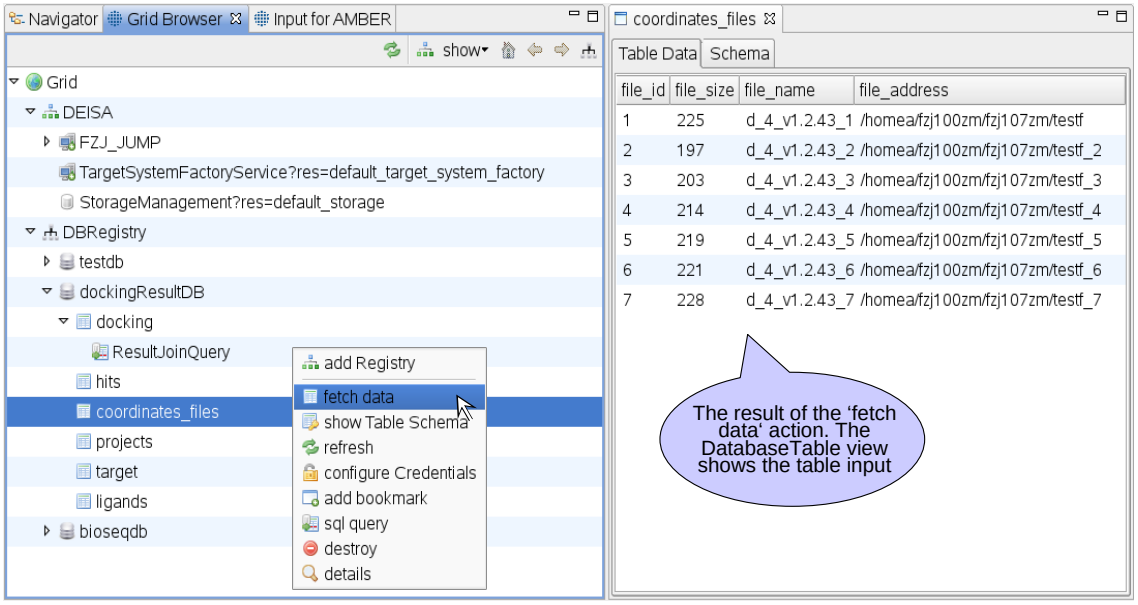

Figure 3: The screenshot shows a connection to a database and the result of a 'fetch data' request in the DatabaseTable view.

Appendix C - The UNICORE Rich Client Database Access(2)

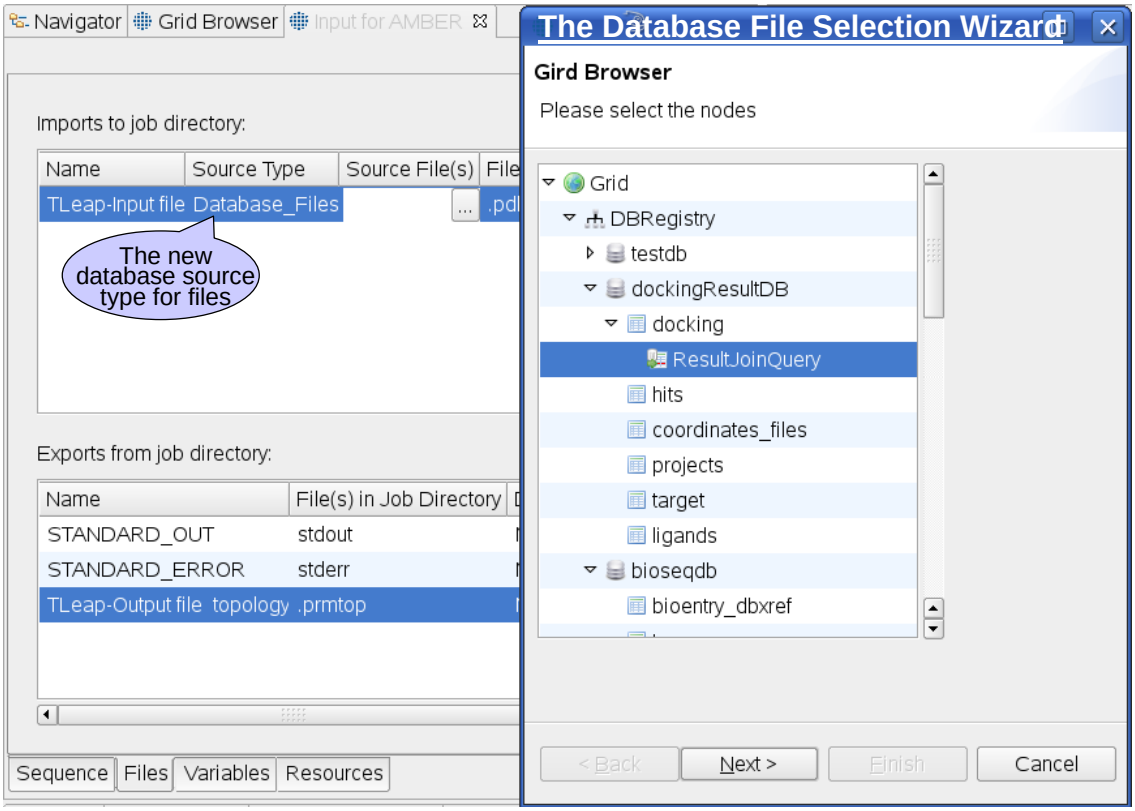

Figure 4: The screenshot shows the selection of a file, whose source is a database. The wizard guides the user through the selection. The selected file is used as input for the job.

Appendix D - The UNICORE Rich Client and AMBER

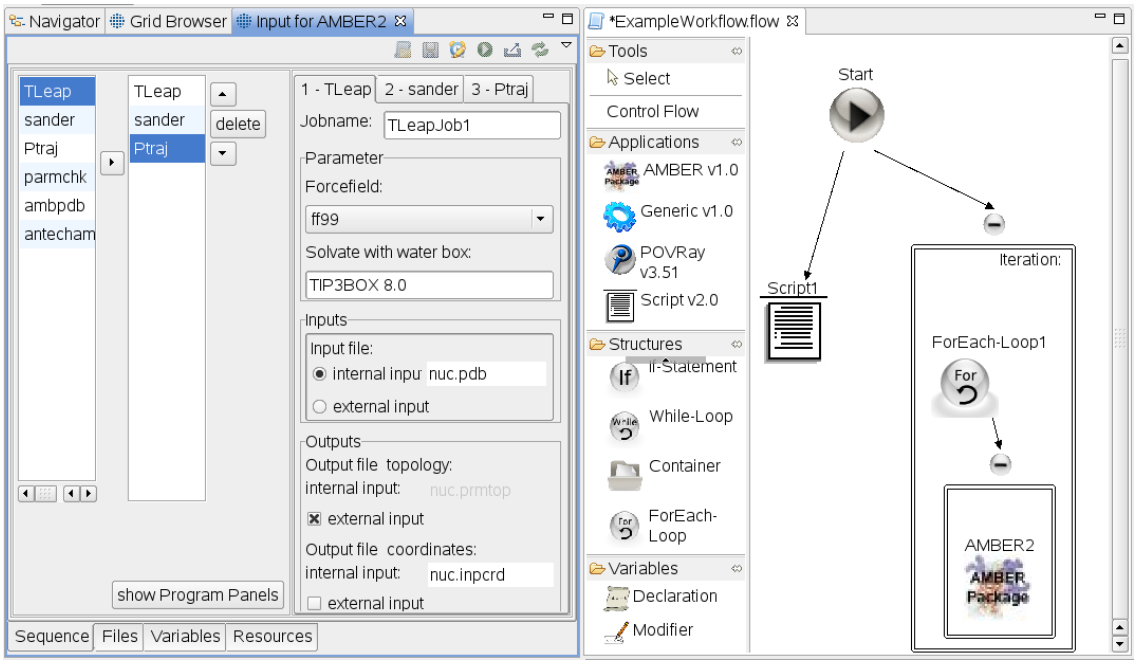

Figure 5: The right hand side of the screenshot shows the creation of a workflow, containing AMBER in a ForEachLoop and a Script. On the left hand side the scientist can conveniently configure the sequence and respective programs.

Jül-4303
Juli 2009
ISSN 0944-2952