Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

ALEXANDER REINEFELD, HINNERK STÜBEN,
FLORIAN SCHINTKE, GEORGE DIN

GuiGen: A Toolset for Creating Customized Interfaces for Grid User Communities

ZIB-Report 02-06 (February 2002)

GuiGen: A Toolset for Creating Customized Interfaces for Grid User Communities

Alexander Reinefeld [a], Hinnerk Stüben [a], Florian Schintke [a], George Din [b]

[a] *Zuse Institute Berlin (ZIB), Takustraße 7, 14195 Berlin, Germany*

[b] *FOKUS Berlin, Kaiserin-Augusta-Allee 31, 10589 Berlin, Germany*

Abstract

GuiGen is a comprehensive set of tools for creating customized graphical user interfaces (GUIs). It draws from the concept of computing portals, which are here seen as interfaces to application-specific computing services for user communities. While *GuiGen* was originally designed for the use in computational grids, it can be used in client/server environments as well.

Compared to other GUI generators, *GuiGen* is more versatile and more portable. It can be employed in many different application domains and on different target platforms. With *GuiGen*, application experts (rather than computer scientists) are able to create their own individually tailored GUIs.

Key words: Grid computing; customized user interfaces; grid user communities; web portals; XML.

1 Introduction

Grid environments provide an advanced infrastructure for the use of distributed computing resources. Such systems are most beneficial when the technical details of the target platforms are hidden from the user. Ideally, a user should just have to deal with the application rather than with the type and location of the computer the application is run on. For this purpose, grid user communities have developed graphical user interfaces for specific application domains.

⋆ Part of this work was funded by the German BMBF project UnicorePlus.

To be published in J. Future Generation Computer Systems (FGCS), 2002.

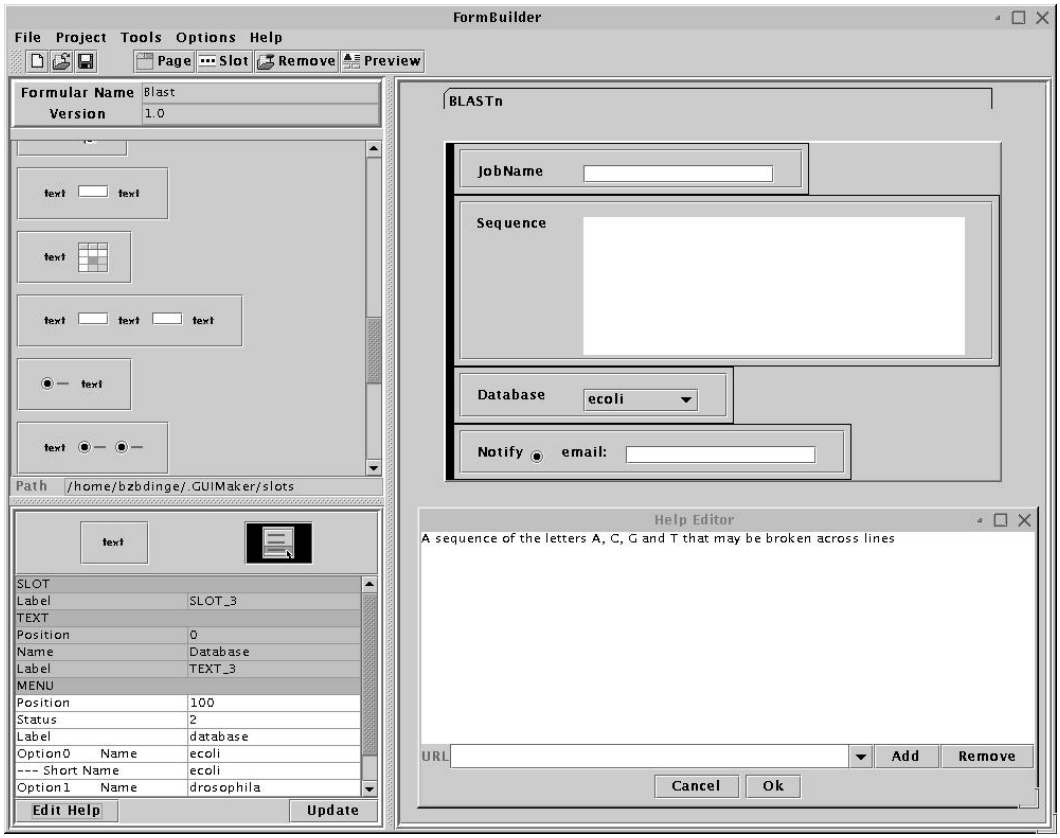

Fig. 1. *GuiGen*'s FormBuilder, here used for creating an application-specific user interface for the BLAST sequence search.

GuiGen helps application experts to design customized GUIs for different application domains. In *GuiGen*, the user interface is split into two parts: a platform independent GUI that can be easily tailored to the specific needs of a user community, and the machine specific back-ends, one for each potential target system. The GUIs (called Forms) and the back-ends (called JobTemplates) can be maintained in repositories for later re-use.

Fig. 1 illustrates how the graphical FormBuilder is used to create a GUI for an application, in this case a BLAST [1] software package for DNA sequence searches. As can be seen, the FormBuilder provides predefined widgets that facilitate the design of GUIs by novices. This is an important aspect, because we assume the application experts rather than computer scientists to create the GUIs, because they know best about the work-flow in their application domain.

The customized GUIs are encoded in XML. This allows to store them in libraries and to send them from one system to another, which is especially important in heterogeneous distributed systems. The GUIs' back-ends are kept separately. They usually contain just a few script commands for starting the job with the given settings on the target machine. For a given GUI there may

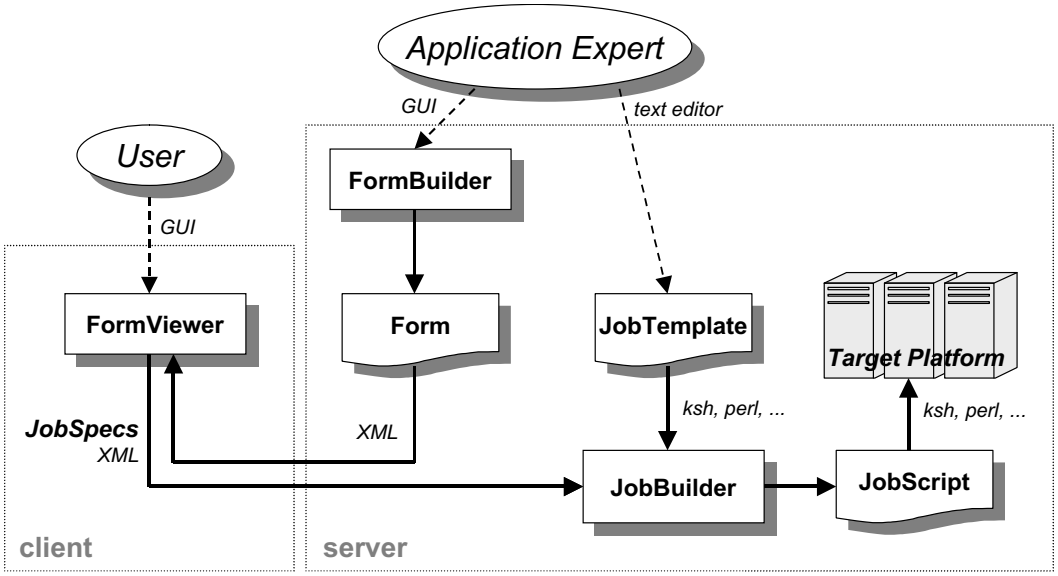

Fig. 2. The components of *GuiGen* and their interaction in a client/server environment.

be more than one back-end, depending on the number of target machines. Back-ends can be programmed in an arbitrary scripting language like sh, csh, ksh, or Perl. New scripting languages can be easily added.

GuiGen does not substitute the grid middleware. It rather builds on the grid layer by accessing job management, file transfer, authorization, security, and other services. Hence, there is no need for including these services in *GuiGen* itself. The number and quality of services solely depends on the underlying grid system.

GuiGen is open, generic, and extensible. Most of its tools are implemented in Java and Perl. By using existing Java and XML components, we were able to implement the complete *GuiGen* system with only 15.000 lines of code.

In the following sections, we present some architectural and implementational details of *GuiGen* and discuss thereafter the use of *GuiGen* in the Grid.

2 Software Architecture

2.1 Building Blocks

In *GuiGen*, there are four different types of documents and three kinds of software components, all of them illustrated in Fig. 2. In this section, we first introduce these seven components and describe their interactions thereafter.

The four document types are:

JobScript: the final, executable script that is generated by the JobBuilder. The JobScript is used to submit the job to the resource management system on the target computer.

JobTemplate: an almost complete job script written in a native scripting language. The missing variable initializations are inserted by the JobBuilder at marked positions. JobTemplates for a particular service are stored in all computing centers that provide this service. Center-specific policies may be encoded.

Form: an XML document describing a form in the common sense. Forms are generated by the graphical FormBuilder and displayed on the screen by the FormViewer. Forms can be stored in repositories, web sites or local file systems. Version information in a Form allow Forms to evolve over time.

JobSpecs: a temporary XML document containing the user input (key/value pairs) of the corresponding Form. JobSpecs are generated by the FormViewer, when a user submits a filled Form.

These documents are processed by the following software components:

FormBuilder: a graphical editor used by an application expert to create a customized Form.

FormViewer: an execution environment for displaying and filling in a customized Form on the screen. The FormViewer checks the user input for correct syntax and transforms it into XML JobSpecs.

JobBuilder: reads a JobTemplate and the JobSpecs, translates the JobSpecs into the specified scripting language and inserts the resulting initializations into the JobTemplate to produce a native script.

2.2 Creating GUIs with GuiGen

Before being able to use *GuiGen*, we first need to create a GUI with the FormBuilder. Fig. 1 illustrates the use of the FormBuilder to arrange graphical widgets like buttons, menus, editable lines, ticks, text fields, and tables on a Form. Each field in the Form gets a unique label. Fig. 1 shows a snapshot of a session, where a simplified GUI for the BLAST [1] sequence search is developed. This GUI has just one page with the entries 'JobName', 'Sequence', 'Database', and 'Notify'.

The FormBuilder stores the Form in XML notation with all necessary information like help texts, labels, constraints, and syntactic information on the possible values of each entry given as regular expressions. We have chosen regular expressions as a compromise, because they are easy to encode and they are sufficient in most cases.

Fig. 3 shows the XML code of the above BLAST Form. As can be seen, not only all fields but also every widget in the form gets a unique label which can be modified by the user. These labels are used to handle internal dependencies in the form, i.e., for dimming parts of the form depending on the previous settings. Versioning of Forms is also supported to allow the evolution of Forms.

```
<?XML version="1.0"?>
<form label="Blast" version="1.0">
  <page title="BLASTn" label="PAGE_0">
    <slot label="SLOT_0">
      <text label="TEXT_0">JobName</text>
      <editable-line tab="100" label="jobname" width="15"/>
    </slot>
    <slot label="SLOT_1">
      <help-text>A sequence of the letters A, C, G and T that may be
                 broken across lines</help-text>
      <text label="TEXT_1">Sequence</text>
      <editable-area label="sequence" tab="100" width="30" height="10"
         typeDescription="^[ACGT]+(\n[ACGT]+)*\n?$"/>
    </slot>
    <slot label="SLOT_3">
      <text label="TEXT_3">Database</text>
      <menu label="database" tab="100">
        <menu-entry entryLabel="ecoli">ecoli</menu-entry>
        <menu-entry entryLabel="drosophila">drosophila</menu-entry>
      </menu>
    </slot>
    <slot label="SLOT_4">
      <text label="TEXT_4">Notify</text>
      <button label="notify" state="true" tab="50"></button>
      <text label="TEXT_5" tab="80">email:</text>
      <editable-line tab="140" label="email" width="15"/>
    </slot>
  </page>
  <ignore ref="TEXT_5" if_status_of="notify" equals="false"></ignore>
  <ignore ref="email" if_status_of="notify" equals="false"></ignore>
</form>
```

Fig. 3. XML representation of the BLAST Form.

The second step is to create a JobTemplate. This is a script for starting the execution of a job. JobTemplates (example shown in Fig. 4) are machine and application-dependent. They use variables with the names of the labels defined in the Form. JobTemplates contain all details on the local installation and usage policies of the computers. Different sites may have different JobTemplates to get the jobs started correctly in their special environment. JobTemplates can

```
#!/bin/ksh

@@@include-list ksh

blastn -s "$sequence" -d $database

if [[ $notify = true ]]
then
    print "job $jobname finished at $(date)" | mail $email
fi
```

Fig. 4. JobTemplate corresponding to the Form in Fig. 3.

be written to support several versions of a Form by checking the given version to trigger version dependent actions.

Before a JobTemplate can be executed the JobSpecs must be inserted by the JobBuilder to create an executable JobScript. The JobBuilder will insert assignments of values to variables in the language of the JobTemplate. To find the right place for the variable assignments in the JobTemplates, the JobBuilder scans the JobTemplates for command lines of the form:

`@@@include-list` *scripting-language*

and replaces them by variable assignments in the syntax of the chosen scripting language. Our current implementation of the JobBuilder supports sh, ksh, csh, and Perl.

In case a job needs more than just a script to be started, additional environment variables or temporary input files can be generated by the JobScript. This allows more complex PBS or Condor jobs to be started.

2.3 Using the GUIs

For displaying a Form on the screen, the user just has to start the FormViewer, which is a Java application or an applet running in a standard web browser. The FormViewer reads the XML Form and displays it on the screen, as shown in Fig. 5.

When the user types his input into the Form, the FormViewer checks the constraints and syntax as specified in the Form. The syntax checks are done in *GuiGen* with JFlex[1]. Plain HTML pages cannot be used because there is no mechanism for online syntax checks.

Finally the FormViewer puts the user input together with the corresponding labels into a JobSpecs XML document as shown in Fig. 6.

2.4 Properties of Forms

Forms are created once and can be used arbitrarily often. Each Form is subdivided into smaller objects: pages, slots, and elements. We have chosen this structure because it can be easily expressed in an XML Document Type Definition (DTD).

[1] JFlex: `http://www.jflex.de/`

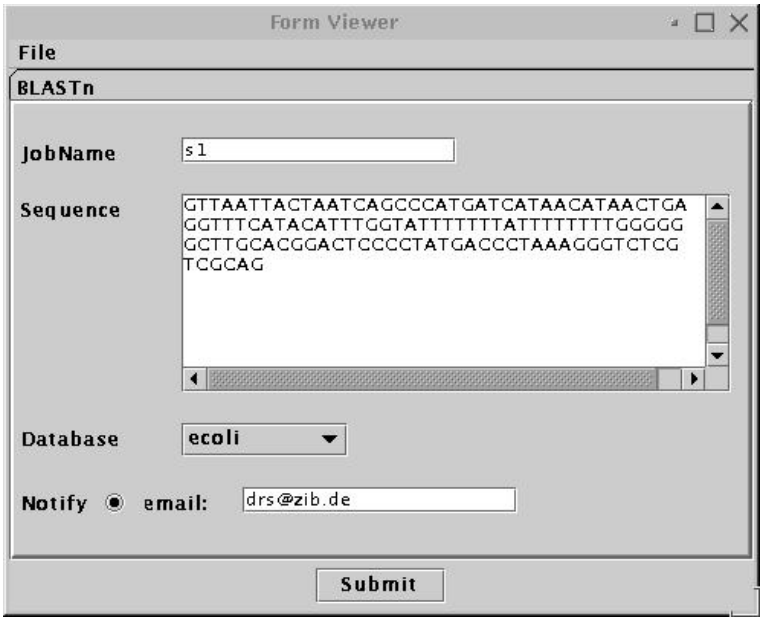

Fig. 5. The BLAST Form of Fig. 3 displayed by the FormViewer.

```
<?XML version="1.0"?>
<jobSpecs form="Blast" version="1.0">
  <string label="jobname">s1</string>
  <text label="sequence"><![CDATA[GTTAATTACTAATCAGCCCATGATCATAACATAACTGAGGTTTCATACATTTGGTAT
TTTTTTATTTTTTTTGGGGGGCTTGCACGGACTCCCCTATGACCCTAAAGGGTCTCGTCGCAG]]></text>
  <string label="database">ecoli</string>
  <string label="notify">true</string>
  <string label="email">drs@zib.de</string>
</jobSpecs>
```

Fig. 6. XML representation of the contents (JobSpecs) of the form in Fig. 5.

Note that Forms do not contain any information on the positioning of the elements on the screen. Only the logical order is specified by the hierarchy of the elements in the XML file. For each element a context sensitive help text may be specified. The FormViewer pops up the text when the element is clicked with the right mouse button. Hyperlinks are also supported.

2.5 Generating JobScripts

The JobBuilder translates the JobSpecs from XML into a native scripting language. This could have been done with XSLT, the XSL Transformations of the Extensible Stylesheet Language [2]. We have implemented the JobBuilder in Perl which is available on supercomputers as well.

To allow for future extensions, the JobBuilder is able to generate code for any language defined in a configuration table as shown in Fig. 7. We use a very simple grammar for specifying the syntax of variable initializations in the

language	*prefix*	*assignment*	*delim*	*delim repr.*	*postfix*
sh, ksh		=	'	'"'"'	
csh	set␣	␣=␣	'	'"'"'	
Perl	$	␣=␣	'	\'	;
Java	String␣	␣=␣	"	\"	;
C	char␣*	␣=␣	"	\"	;
cpp	#define␣	␣	"	\"	

Fig. 7. Translation table for the JobBuilder. A '␣' represents a blank and a missing entry denotes the empty string.

scripting languages consisting of just one rule:

initialization ::= prefix name assignment delim value delim postfix

The elements are concatenated without any white space in between. In this rule *name* is substituted by the label specified in the Form and JobSpecs. *Value* is substituted by the given value.

All other elements *prefix, assignment, delim*, and *postfix* are language dependent, as shown in Fig. 7. The user input (including numbers) is mapped to character strings. Delimiters are quoted.

2.6 Implementation of GuiGen

The FormBuilder is the major part of the *GuiGen* toolset. Like the FormViewer it was implemented with Java JDK 1.3 and the Swing graphics library. For parsing XML [2,16] we used the JAXP [2] implementation of DOM [5]. In the JobBuilder we have used the 'expat' library and XML::Parser, a Perl implementation of SAX, for parsing the XML code.

To provide a preview option in the FormBuilder the Java classes of the FormViewer were re-used. With the help of the above mentioned tools we were able to implement the whole *GuiGen* system with 15.000 lines of code.

[2] JAXP: Java^{TM} APIs for XML Processing

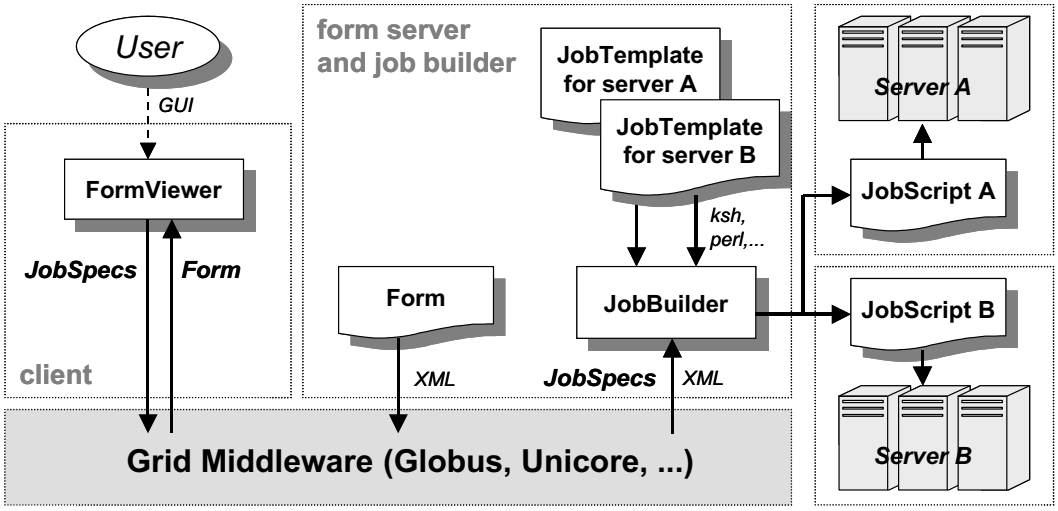

Fig. 8. *GuiGen* used in a grid environment.

3 Using *GuiGen* in Grid Environments

To keep things simple, we have described *GuiGen* in the context of a client/server environment. In practice, *GuiGen* is most beneficial when applied to distributed environments like computational grids. Here the user specifies the job (service) he wants to be done and the grid middleware autonomously decides on which system it is going to be executed—so much for the theory.

In practice, several scenarios are possible. The key questions are from where to retrieve the Form and on which machine to execute the JobScript. In the scenario shown in Fig. 8 the target machine is determined by the grid middleware. Here the JobBuilder selects the corresponding JobTemplate according to the label and version in the `<form>` tag.

In the Unicore project [14], which gave the impetus for our work [4], there is no brokerage service. The user simply selects the target machine by himself. In this respect, Unicore is not a computational grid in the common sense, but it provides an improved, uniform access to distributed high-performance computer resources [13]. In Unicore a *Job Preparation Agent (JPA)* is used to create and to submit the jobs to a participating site, where a *Network Job Supervisor (NJS)* incarnates the jobs on the target platform(s). The user gets a machine independent, application-specific Form, fills it in and sends it via the JPA to the NJS of the target machine. There the JobBuilder selects the corresponding JobTemplate and builds the JobScript.

In other grid systems, like Globus [8] or DataGrid [3] for example, a broker agent decides where to execute the job. The grid middleware sends the JobSpecs together with some information on the type and size of the selected target machine to a JobBuilder. Note that the JobBuilder may be run on any

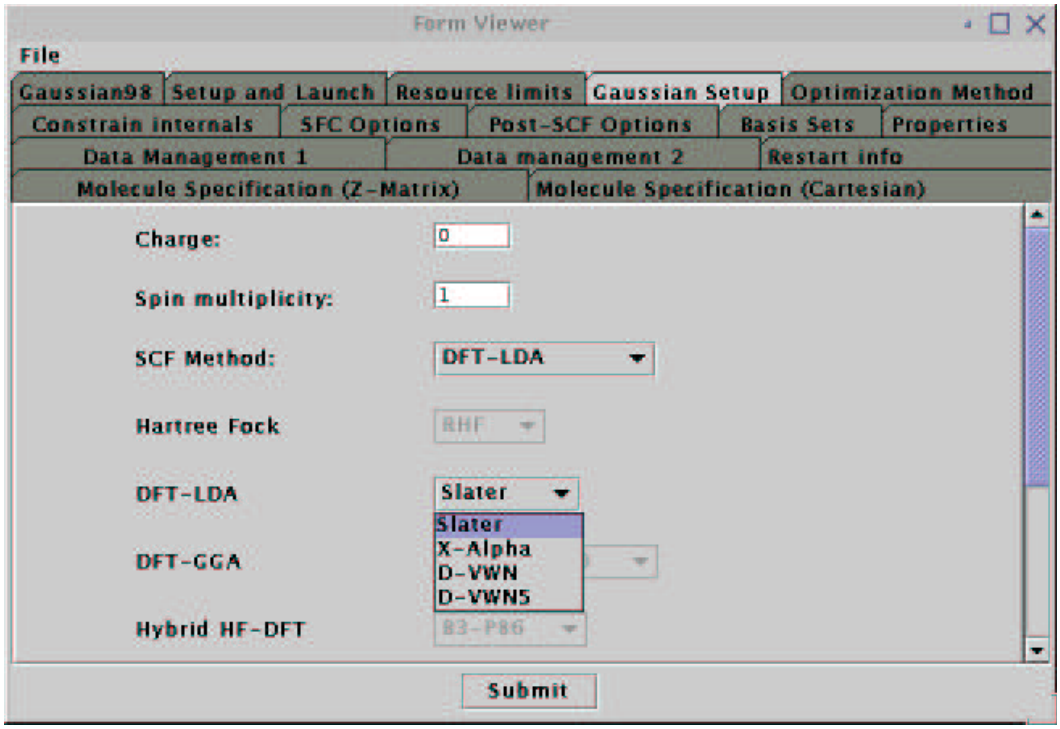

Fig. 9. Screenshot of a Gaussian form with sub-forms.

machine in the grid. Based on the given information the JobBuilder selects the appropriate JobTemplate to generate the executable JobScript. The JobTemplates can be retrieved, e.g., from the *Globus Information Services (GIS)*, which is distributed among the participating servers.

Also the application-specific Forms could be kept in a logically central, but physically distributed global depository. This allows to maintain hierarchies of Forms for the various application domains. As an example, Fig. 9 shows a more complex Gaussian [6] form with several pages which could be used as a sample by the user community. Users could then build on such examples to create customized Forms for their specific work models.

4 Related Work

There exists a variety of tools for the interactive design of graphical user interfaces (Ilog Views, Qt Designer) and integrated development environments (Forte, JBuilder, KDevelop). Most of them support only one widget library and programming language for initializing, arranging and starting the elements in the window. The GUIs are typically stored in a proprietary format. The connection between the GUI and the application is usually done with a callback mechanism that is implemented in the same programming language as the callback stubs.

From these approaches, our *GuiGen* toolset differs in the following ways:

- it uses standard XML rather than a proprietary data format for handling the GUIs,
- it uses key-value pairs instead of callback functions,
- it is able to handle applications implemented in arbitrary programming languages.

The Eclipse Platform [12], for example, is an emerging framework for integrated development environments, backed by over 35 tool providers that try to eliminate the drawbacks of the existing development environments described in the beginning. Compared to our approach the GUIs in Eclipse can not yet be used in grid environments.

There are also some web-based GUIs for batch systems like PBSWeb [11], but they do not provide application specific interfaces. With our approach we can hide the technical details of the underlying batch system by displaying Forms in the user's own terms and notations.

Other current projects like `qprep` [15] and the Uniform Job Submission Script Syntax (UJSSS) initiative of the Global Grid Forum's Scheduling Working Group [7] define a command line interface for job management. On the one hand, this API provides a common access point to different job management systems, but on the other hand site-specific settings, like file access, system paths, policies and lifetime of intermediate storage, etc. are not dealt with.

5 Summary and Future Plans

GuiGen allows to quickly design graphical user interfaces for grid applications. While the *GuiGen* software can also be deployed as an easy-to-use toolset on a local computer, it was designed for establishing customized interfaces for applications in computational grids, like Globus or Unicore [14]. Here, the user is normally not interested in accessing a specific computer in the grid, but rather in obtaining a service, no matter on what computer that service is being supplied.

From another point of view, *GuiGen* may be seen as a tool for generating portals [9,10]. We believe that, after the tremendous success of portals in the commercial world (e.g., the enterprise information portals and the community portals) it is now time to adapt and utilize the portal idea in the science community.

GuiGen is just a first step to make high-performance computing more user-

friendly by better satisfying the needs of the users. We now consider an even more user-centric approach by introducing the cookie concept known from the world wide web. Cookies would allow the system to recognize user preferences and to act correspondingly. If, e.g., a BLAST user has previously used certain parameter settings, the GUI should show up with the same settings in the next session as well. This should apply to all preferences, including the choice of server and pre- and post-processing tools.

References

[1] S.F. Altschul, T.L. Madden, A.A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D.J. Lipman. *Gapped BLAST and PSI-BLAST: A new generation of protein database search programs*, Nucleic Acids Res. 25 (1997), 3389–3402.

[2] R. Anderson *et al.*, *Professional XML*, Wrox Press Ltd., 2000.

[3] *DataGrid project*. `http://www.eu-datagrid.org/`.

[4] G. Din. *Service Description in Unicore*. Diploma Thesis, Polytechnica University of Bucharest, Romania, June 2001.

[5] *DOM: Document Object Model Level 1 Specification*, W3C Recommendation, `http://www.w3.org/TR/REC-DOM-Level-1/`, October 1998.

[6] M.J. Frisch *et al. Gaussian 98, Revision A.7.*, Gaussian Inc., Pittsburgh PA, 1998.

[7] *Global Grid Forum*, `http://www.globalgridforum.org/`.

[8] *Globus project*, `http://wwww.globus.org/`.

[9] T. Haupt, E. Akarsu, G. Fox, and C.H. Youn. *The Gateway system: Uniform web based access to remote resources.* Concurrency – Practice and Experience 12(8), (2000), 629–642.

[10] G. v. Laszewski, I. Foster, J. Gawor, P. Lane, N. Rehn, and M. Russell. *Designing Grid-based problem solving environments and portals.* Procs. 34$^{\text{th}}$ Hawaii Intern. Conf. on System Sciences, 2001.

[11] G. Ma and P. Lu. *PBSWeb: A Web-based Interface to the Portable Batch System*, In Proc. 12$^{\text{th}}$ IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS), Nevada, 24–30 (2000).

[12] Object Technology International, Inc. *Eclipse Platform Technical Overview.* White Paper, `http://www.eclipse.org/`, July 2001.

[13] A. Reinefeld, H. Stüben, T. Steinke, and W. Baumann. *Models for Specifying Distributed Computer Resources in UNICORE.* 1$^{\text{st}}$ European Grid Forum Workshop, ISTmus Conference Proceedings, Poznan, Poland, 313–320 (2000).

[14] *UNICORE project*, `http://www.unicore.de/`.

[15] J. Werne, C. Bizon, and M. Gorlay. ***`qprep`**: A facility-independent tool for job submission.* `http://www.pstoolkit.org/`. December 2001.

[16] *XML: Extensible Markup Language*, `http://www.w3.org/XML/`.