In *Applied Parallel Computing. State-of-the-art in Scientific Computing.*
Springer-Verlag, Lecture Notes in Computer Science, (to appear).

# Semi-Automatic Generation of Grid Computing Interfaces for Numerical Software Libraries

Erik Elmroth and Rikard Skelander

Dept. of Computing Science and HPC2N, Umeå University, SE-901 87 Umeå, Sweden.
{elmroth, rikard}@cs.umu.se

**Abstract.** There is an immediate need to develop Grid interfaces for a large set of numerical software libraries, in order to make popular software of today available in the computing infrastructure of tomorrow. As this development work tend to be both tedious and error-prone, this contribution presents a semi-automatic process for generating the interfaces. The underlying principle is to use a front-end tuned for each numerical library and a back-end for each Grid environment considered. Then all library—Grid environment combinations can be generated with a small amount of manual work. The presentation of the main ideas is followed by a proof-of-concept implementation that generates NetSolve interfaces for the complete SLICOT software library, a numerical library comprising nearly 400 Fortran subroutines for numerical computations in the design and analysis of control systems.
**Keywords:** Grid computing, numerical software libraries, remote computing, interface, SLICOT, NetSolve.

## 1 Introduction

The rapid development of the Grid computing infrastructure puts strong demands on development of Grid-enabled application software and software libraries. Some of the typical Grid resource usage scenarios are based on the underlying idea that all small computations are performed on a single local computer, while large-scale computations are automatically distributed to appropriate and more powerful computing resources on the Grid. Examples include Grid-empowered problem solving environments and application-oriented web-based Grid portals, so-called science portals.

For both these scenarios, there is an immediate need for interfacing standard software libraries on remote resources. The development of such interfaces tend to be both tedious and error-prone. This contribution presents a semi-automatic process for generating the interfaces, and a proof-of-concept implementation that generates NetSolve interfaces for the complete SLICOT software library.

Financial support has been provided by the Swedish Foundation for Strategic Research under the frame program grant A3 02:128 and The Swedish Research Council (VR) under contract 343-2003-953.

The front-end of the prototype performs a parsing of calling sequences, variable declarations, and source code documentation in order to automatically determine the subroutine interfaces. Notably, the front-end needs to be tuned to the conventions of each specific library and the programming language used. Depending on the level of consistency of the documentation, some parts of the calling sequence may not be uniquely determined by the automatic procedure. Such cases are reported and taken care of by hand.

The back-end, which in our proof-of-concept implementation generates so called NetSolve Problem Description Files, is general with respect to which library that has been processed by the front-end, and the interfaces generated are completely portable. As the original SLICOT software includes routines for testing and timing, they can directly be used to verify the correctness of the NetSolve interfaces, by simply calling routines in the NetSolve-enabled library instead of the standard library.

The prototype is demonstrated by generating NetSolve interfaces for the complete SLICOT library. SLICOT is a subroutine library comprising nearly 400 Fortran routines for numerical computations in the design and analysis of control systems.

The outline of the rest of the paper is as follows. The remote computing scenario and the motivation for this work is presented in Section 2. Here, we also exemplify the types of Grid middleware and software libraries considered. The general process for generating interfaces is described in Section 3. The usage of this process in practice, and some lessons learned follows in the description of our proof-of-concept implementation in Section 4. Section 5 briefly describes how to use the NetSolve version of the SLICOT library on more powerful remote resources without having to install more than a small set of NetSolve client software on the local computer. We finally make some concluding remarks in Section 6.

## 2 Background and Motivation

Several common Grid middleware solutions are based on the underlying principle that data should be sent to some remote resource where the software resides. This principle is often referred to as *remote computing*. In alternative approaches, both data and software are sent from the client machine to a resource (*code shipping*), or both software and data are sent from different locations to a third, computational resource (*proxy computing*).

Our focus is on the generation of interfaces for software libraries in the remote computing scenario. The middleware solution used for our proof-of-concept implementation is NetSolve [3]. NetSolve basically makes it possible to call subroutines on remote systems with interfaces for Fortran, C, Matlab, Mathematica, and Octave (see also Section 4). Similar functionality using alternative interfaces is provided by the Ninf (Network-based information library) software [15], and a recent enhanced version, named Ninf-G [20], that is built on top of standard Globus-based Grid services. Other related efforts include the Purdue Network

Computing Hub (PUNCH) [11], Network Enabled Optimization Server (NEOS) [14], the Remote Computing System (RCS) [2], etc. Moreover, the Open Grid Services Architecture [8] defines a more general framework for specifying Grid services based on Web services, that also can be used in the specific scenario outlined here [16].

Application-oriented Grid portals represent another type of Grid environments that in one aspect may have similar requirements. Also a portal may be seen as a remote interface to the software. Given a semi-automated process for generating other Grid interfaces for remote computing, also the set of application-oriented portals may be considered as target. We have already seen both automated efforts and different types of toolkits for constructing application-specific portals, including an automatic tool for Ninf-coupled portals [19] and the SLICOT web portal [7, 10].

Our aim is to Grid-enable scientific software libraries typified by LAPACK, SLICOT, Linpack, IBM ESSL [1, 4, 9, 18], and several others. This type of large numerical libraries have been developed for many years, and have proved to be robust, popular and long-lived. Over the years, we have seen that many such libraries, e.g., written in Fortran 77, have continued to stay popular even by users preferring more modern programming languages. Hence, we expect that there will be a demand for using todays numerical libraries also in the future and from the next generation of computing environments, including Grids and web-based Grid portals.

Over time, the relative overhead for transferring the data to a remote resource before performing the computations is decreasing as network performance is increasing more rapidly than the performance of the computers. Today, we see a doubling of network bandwidth every 9–12 months which should be related to the doubling of computer performance every 18 months. As this trend continues, the overhead of data transfer may well be compensated by the reduced manual work for software library installations and tuning.

Clearly, we can foresee the need for at least a number of different interfaces for each of quite a few different libraries, leading to a significant number of library-interface combination. If manually generated one-by-one, without making any attempt to reuse results between the different interfaces generated, we can easily foresee an enormous amount of both tedious and error-prone work.

Hence, we propose to use a semi-automated process where we only need one front-end process for each library and one back-end process for each Grid environment considered, in order to generate all software library—Grid environment combinations requested.

## 3 Semi-Automatic Interface Generation

The interface generation process can basically be viewed as two nearly independent processes performed in sequence. First, a semi-automatic process parses the numerical software library in order to extract all library-specific information required to define the correct calling sequences for all subroutines. This

information is then stored in a basic internal format. The second process is the generation of Grid middleware specific interfaces for all subroutines.

The parsing process of the front-end described below is aiming at numerical libraries written in Fortran, as these still are dominating among existing numerical libraries. The general idea, however, is applicable to libraries written in other programming languages as well.

### 3.1 Front-End: Extracting Software Library Information

The front-end parses the source code of the numerical software library and extracts the required information from the

- Calling sequences.
- Parameter declarations.
- Inline documentation.

The information that can be extracted from the calling sequence is obviously the name and the order of all parameters. The parameter declarations give additional information about the data types, the number of dimensions of array elements, and the leading dimension of at least all but the last dimension of the array arguments. As parameters are passed by reference, the declarations do not have to (and do not in general) contain information about the last dimension of array arguments, which is required to determine the total size of such objects. This information is, however, typically included in the inline documentation of the code. The inline documentation together with the declarations may also tell if the parameters are of type input, output, or both.

The parsing of the inline documentation is by nature more difficult than that of the calling sequence and the parameter declarations, as it does not follow any well-defined syntactical and semantical rules and definitions. Despite this fact, it is typically possible to automatically extract a vast majority of the information required from this documentation by taking advantage of the more or less strict conventions that often are used in state-of-the-art libraries.

If the size of a dimension is expressed in the inline documentation as a constant or as a single input parameter, the automated process has no problems. However, it becomes more complicated if the dimension size is expressed as some function of input parameters. For example, the automatic process may determine from the inline documentation, that a parameter declared as `A(LDA, *)` is to be used as `A(LDA, MAX(M, N))`, where `M` and `N` are included in the list of parameters. In order to perform the communication correctly, both the client and the server resource need to know the exact size the array `A`. Since how to handle this type of issues depends on the functionality of the Grid software, and we strive to make the internal format general with no dependencies on the back-end, we add one extra variable to the internal storage format for each function value requested (e.g., one extra parameter for the function value `MAX(M, N)` in the example above).

There will normally also be cases where the automated process fails to determine, e.g., the size of an array, possibly because its size is expressed in terms

of natural language. Such cases are simply identified by a "flag", telling where the user needs to put in some manual work.

The fact that the conventions are different for different libraries makes it necessary to adapt the front-end for each case. Then, the extent to which all routines in a library follows these conventions determines how much manual work is needed after the automatic process.

As the result of the parsing, all information required for each subroutine call is extracted and stored in an internal format. The information includes the subroutine name, the lists of input and output parameters, data types for the parameters, array dimensionality, etc.

### 3.2 Back-End: Generating Grid Interfaces

The input data for the back-end process is the data generated by the front-end, stored in the internal format. From this data, the back-end can be configured and tuned to automatically generate all the interfaces in the exact format requested for a specific Grid middleware. Of course, this is a process that can be rather different for different Grid middleware, but in most cases the code to be generated has rather limited syntax and semantics which makes the back-end easily developed. Hence, most of this generation is trivial, but we will in the following proof-of-concept implementation also illustrate some technical problems that need to be handled.

Notably, given a back-end for one middleware, it can be used for a automatic translation for any software library for which the requested information is available in the internal format.

## 4 Proof-of-Concept Demonstration

The feasibility of the process described above have been investigated in a proof-of-concept implementation, that makes the complete SLICOT library available from remote resources via NetSolve.

### 4.1 Aggregation of Interface Data for the SLICOT Library

SLICOT is a numerical software library for computations in systems and control theory [18], freely available for non-commercial use. The library provides Fortran 77 implementations of algorithms and methods for the design and analysis of control systems. Among the more pronounced design principles for SLICOT is the strive to provide robust, stable and accurate algorithms, to take both memory requirements and floating point performance issues into account, and to follow strict programming and documentation conventions throughout the library.

In total, SLICOT comprises nearly 400 user-callable and computational routines. Around 200 of the routines have associated example programs, example data files and results for illustrations and comparisons. The Basic Linear Algebra

Subprograms (BLAS) [5, 6, 13] and LAPACK [1] are used for underlying linear algebra computations.

SLICOT is organized in eleven groups of routines, depending of their applicability or type:

A: Analysis Routines
B: Benchmark and Test Problems
C: Adaptive Control
D: Data Analysis
F: Filtering
I: Identification
M: Mathematical Routines
N: Nonlinear Systems
S: Synthesis Routines
T: Transformation Routines
U: Utility Routines

The parsing of the library has been performed as described in Section 3.1. It should be remarked that the documentation style in SLICOT is not completely homogeneous and does not follow one unified convention as strictly as the SLICOT design goal suggests. This implies some extra efforts in designing and tuning the front-end parser, but it should be noted that almost all information required has been gathered through the automatic process.

The only type of information for which we do not think it is worth the effort to build an automatic tool, is for identifying array dimensions that are not only parameter values. A typical example is to determine the size of an array declared as `WORK(*)`, where the documentation tells that the array is of size `LWORK` and `LWORK` is not a parameter itself but a more complex function of the input parameters. In these cases the automated process simply indicates that manual work is required to add this information.

After completing the automatic parsing and some work by hand, all the requested information is stored in the internal format. Notably, this information is now gathered once and for all and can be use to generate any number of different Grid interfaces using differently configured back-ends. In this proof-of-concept implementation we generate interfaces for NetSolve.

### 4.2 Generating the NetSolve-SLICOT Interfaces

NetSolve is a Network-enabled solver that gives clients transparent access to software on remote servers [3]. NetSolve *agents* match service requests with the resources available. In order to make software available via NetSolve, each subroutine interface must be specified in a NetSolve Problem Description File (PDF). Based on such files, NetSolve provides interfaces to be called from Fortran, C, Matlab, Mathematica, and Octave. The actual software made available via NetSolve can be written in C or Fortran.

Almost all of the translation from the internal format to the rather basic language of the PDF files can be done automatically. Again, it is only some

of the more complicated array dimensions that cause problems. Sizes that are parameter values, e.g., `LDA` in Section 3.1, or unconditional functions of the input parameters, e.g., `LDA*M`, do not cause any problems in the back-end process. For conditional expressions, it is possible to use so called "`@COMP`" expressions in the PDF file. The "`@COMP`" expressions can be used to handle cases, where the array size depends on, e.g., if another parameter is `TRANS = 'Y'` or `TRANS = 'N'` (i.e., if the matrix transpose should be used or not). However, the "`@COMP`" functionality can currently only handle conditional expressions including tests of equality. Apparently, this limitation of the "`@COMP`" expressions will be removed in future NetSolve versions, but for the current version we overcome the problem by adding an extra parameter where the user specifies the array size in these rare cases.

The output from this back-end is one NetSolve PDF-file for each SLICOT subroutine. These files are then to be stored on a server resource that has the NetSolve server software running, and access to the complete SLICOT library including the underlying BLAS and LAPACK routines. In addition to this, the server must register with a NetSolve agent, which then can direct the user's requests to the server.

In order to test the generated interfaces, we have modified the nearly 200 test routines available with SLICOT to call the NetSolve versions or the routines instead of routines in a local SLICOT library. We remark that for libraries where test routines are available, this is in general a very convenient way of testing also the new interfaces.

## 5 Using SLICOT via NetSolve

In order to use the NetSolve version of SLICOT the user must install the NetSolve client routines on the local computer, set the appropriate environment variable and link with the appropriate NetSolve library. Notably, there is no need to install SLICOT, BLAS, and LAPACK on the local computer as these libraries will only be accessed on remote resources. NetSolve also provides commands for querying an agent about available servers.

With the basic installations in place, there are only marginal changes that have to be made to an application software in order to call the NetSolve version of SLICOT on a remote computer system instead of calling a locally installed SLICOT library. This is illustrated by following small example, showing how a standard SLICOT routine is called via NetSolve from a Fortran or C to application program.

The subroutine head of the original Fortran code for the SLICOT routine SB02MD is outlined below. SB02MD is a routine implementing a Schur vectors method for solving algebraic Riccati equations [12, 17, 21].

```
SUBROUTINE SB02MD( DICO, HINV, UPLO, SCAL, SORT, N, A, LDA, G,
                   LDG, Q, LDQ, RCOND, WR, WI, S, LDS, U, LDU,
                   IWORK, DWORK, LDWORK, BWORK, INFO )
```

A remote NetSolve call to this routine from a Fortran program is made through a subroutine call to FNETSL with the routine name (`SB02MD`) and a NetSolve status variable as two additional arguments preceding the list of the parameters for the standard SB02MD routine:

```
CALL FNETSL( 'SB02MD()', STATUS, DICO, HINV, UPLO, SCAL, SORT, N,
            A, LDA, G, LDG, Q, LDQ, RCOND, WR, WI, S, LDS, U, LDU,
            IWORK, DWORK, LDWORK, BWORK, INFO )
```

A corresponding call from a C program is made as a function call to "`netsl`". Here the NetSolve status variable is the function return value:

```
status = netsl( "SB02MD()", &DICO, &HINV, &UPLO, &SCAL, &SORT, &N,
               A, &LDA, G, &LDG, Q, &LDQ, &RCOND, WR, WI, S, &LDS,
               U, &LDU, IWORK, DWORK, &LDWORK, BWORK, &INFO);
```

## 6 Concluding Remarks

The semi-automatic process described aims at minimizing and improving the tedious and error-prone work required to develop a set of different Grid interfaces for each of a large number of numerical software libraries. The underlying idea is to develop one front-end for each numerical library and one back-end for each Grid environment considered, and then with a small amount of work be able to obtain all library—Grid environment combinations. The extent to which this process can be made automatic or needs additional manual work depends both on how well structured and consistent the inline documentation is and the features provided by the Grid software. However, our experience from this proof-of-concept implementation is that a vast majority the tedious and error-prone work can better be done automatically.

The proof-of-concept implementation illustrates this process for one libary (SLICOT) and one Grid environment (NetSolve). In order to make SLICOT available in some other Grid environment or via a web portal as done in [7], the same front-end could be used with a new back-end generating, e.g., PHP-scripts for a web portal.

We remark that new numerical libraries could benefit from having the interfaces specified in, for example, XML. Based on such specification, the front-end process could be made trivial and completely automatic.

## 7 Acknowledgements

We acknowledge the anonymous referees for their comments and suggestions.

## References


1. E. Anderson, Z. Bai, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide, Third Edition*. Society for Industrial and Applied Mathematics, Philadelphia, 1999.
2. P. Arbenz, W. Gander, and M. Oettli. The remote computation system. *Parallel Computing*, 23:1421–1428, 1997.
3. D.C. Arnold, H. Casanova, and J. Dongarra. Innovations of the NetSolve grid computing system. *Concurrency and Computation: Practice and Experience*, 14(13-15):1457–1479, 2002.
4. J. Bunch, J. Dongarra, C. Moler, and G.W. Stewart. *LINPACK User's Guide*. SIAM, Philadelphia, PA, 1979.
5. J. Dongarra, J. Du Croz, I. Duff, and S. Hammarling. A proposal for a set of level 3 basic linear algebra subprograms. *SIGNUM Newsletter*, 22(3):2–14, February 1987.
6. J. Dongarra, J. Du Croz, S. Hammarling, and Richard J. Hanson. An extended set of Fortran basic linear algebra subroutines. *ACM Transactions on Mathematical Software*, 14(1):1–17, March 1988.
7. E. Elmroth, P. Johansson, B. Kågström, and D. Kressner. A Web Computing Environment for the SLICOT Library. In P. Van Dooren and S. Van Huffel, editors, *The Third NICONET Workshop on Numerical Control Software*, pages 53–61, 2001.
8. I. Foster, C. Kesselman, J. Nick, and S. Tuecke. Grid services for distributed systems integration. *IEEE Computer*, 35(6):37–46, 2002.
9. IBM. *Engineering and Scientific Subroutine Library, Guide and Reference*. Ver. 3, Rel. 1.
10. P. Johansson and D. Kressner. Semi-Automatic Generation of Web-Based Computing Environments for Software Libraries. In *Proceedings of The 2002 International Conference on Computational Science (ICCS2002)*, 2002.
11. N. Kapaida and J. Fortes. An architecture for Web-enabled wide-area network-computing. *Journal of Networks, Software Tools and Applications*, 2(2):153–164, 1999.
12. A. Laub. A Schur method for solving algebraic Riccati equations. *IEEE Trans. Autom. Contr.*, AC-24:913–921, 1979.
13. C. Lawson, R. Hanson, D. Kincaid, and F. Krogh. Basic linear algebra subprograms for Fortran usage. *ACM Transactions on Mathematical Software*, 5:308–323, 1979.
14. J. More, J. Czyzyj, and M. Mesnier. The NEOS server. *IEEE Journal on Computational Science and Engineering*, 5:68–75, 1998.
15. M. Sato, H. Nakada, S. Sekiguchi, S. Matsuoka, U. Nagashima, and H. Takagi. Ninf: A network based information library for global world-wide computing infrastructure. In *HPCN Europe*, pages 491–502, 1997.
16. S. Shirasuna, H. Nakada, S. Matsuoka, and S. Sekiguchi. Evaluating Web services based implementations of GridRPC. In *11th IEEE International Symposium on High Performance Distributed Computing, Edinburgh, Scotland*. IEEE Computer Society Press, Los Alamitos, CA, 2001.
17. V. Sima. *Algorithms for Linear-Quadratic Optimization*, volume 200 of *Pure and Applied Mathematics: A Series of Monographs and Textbooks*. Marcel Dekker, Inc., New York, 1996.
18. SLICOT. The SLICOT library and the numerics in control network (NICONET) website. http://www.win.tue.nl/niconet/.

19. T. Suzumura, H. Nakada, M. Saito, S. Matsuoka, Y. Tanaka, and S. Sekiguchi. The Ninf portal: An automatic generation tool for Grid portals. In *Proceedings of the 2002 joint ACM-ISCOPE conference on Java Grande*, pages 1–7. ACM Press, 2002.
20. Y. Tanaka, H. Nakada, S. Sekiguchi, T. Suzumura, and S. Matsuoka. Ninf-G: A reference implementation of RPC based programming middleware for Grid computing. *Journal of Grid Computing*, 1(1):41–51, 2003.
21. W.M. Wonham. On a Matrix Riccati Equation of Stochastic Control. *SIAM J. Contr.*, 6:681–697, 1968.