Compiling P–GRADE programs for the JGrid architecture

László Lövei

June, 2003

P–GRADE[1] is an interactive, graphical software development tool, designed to build parallel programs using message passing for communication. It supports the graphical construction of the communication topology of the program by defining processes, communication ports and channels between them. The main algorithmical structures of the processes can be defined graphically based on communication actions, and details can be given by arbitrary textual code fragments. This technique hides the underlying communication infrastructure, while it allows the programmer to get the full power of the used programming language.

JGrid[2] is a Grid infrastructure built on the Java-based Jini[3] technology. It defines a so-called Compute Service, which can be used to execute arbitrary Java code wrapped in a special Java object, and a management infrastructure, which simplifies the usage of Compute Services by caching and by providing search facilities. It also provides methods for communication between the running tasks, supporting distributed programs this way.

One of the JGrid projects' goals is to make the P–GRADE development environment available on the JGrid platform, with full support of the monitoring and debugging capabilities. The key task to achieve this is to develop a compiler, which translates GRAPNEL, the language of the P–GRADE environment, into Java code, using the JGrid platform for program running, communication, monitoring and debugging.

The first important thing about the generated code is that large portions of it can be generalized and written as a support library. This approach gives opportunity to generate Java code that is independent of the target platform. The other role of the library is to provide an interface that makes code generation easier and cleaner.

An other classification of the code can be based on the distributed nature of the program. In a JGrid system, the programs are Java objects; they correspond to P–GRADE processes. These object communicate with each other, based on a communication topology, which is defined by ports, channels, communication groups etc. The topology provides information on where each message should be sent; this information can be attached to the ports, and must be available to the running process. The other parts of the topology are only needed for the

initialization of the program, and need not be present in a running process.

The generated code of the initialization part can be reduced to a declarative listing of the topology: each process ang group has a class, which lists the ports, channels, subgroups and subprocesses. The details of the initialization can be hidden in the library, which creates serializable process objects with the knowledge of their communication connections. These object are sent into the Grid, and are run by Compute Services.

Special support is needed for the communication templates. These are special, scalable process groups with predefined communication topology (eg. pipeline). The group classes defined for the templates handle the process instantiations and build the communication structures themselves.

The actually running codes of the processes are generated into special Java methods of the process classes. There are basically three kinds of generated instructions: control constructs, which are primitive Java elements; textual code fragments, which are copied verbatim into the final code; and communication actions, which are the most complex parts of the system.

There are three types of communication actions: output, input, and alternative input. The first two can be point-to-point or group communication, the last is only meaningful with groups; besides, the output action may have synchron or asynchron semantics. After handling these properties, the message data must be interpreted according to the so-called protocols given in the P–GRADE program; these are the descriptions of the message format. The generic properties can be handled by the library, but each of the protocols must have specially generated insertion and extraction code, which is handled by message buffers and packing and unpacking primitive functions.

The last topic which the compiler must deal with is the support of the external tools, the monitor and the debugger. The monitoring is done by instrumentating the code: every graphically defined program step calls the corresponding instrumentation function, which handles the monitoring. The debugger requires even less support, only a cross reference file need to be created, which contains the positions of the graphically defined steps in the Java source files.

References

[1] P–GRADE User's Manual
http://www.lpds.sztaki.hu/pgrade/p_grade/manual.html

[2] JGrid: Grid system based on Jini
http://pds.irt.vein.hu/jgrid

[3] Jini Network Technology
http://www.jini.org