The Design and Implementation of JLucid

Weichang Du

Faculty of Computer Science
University of New Brunswick
Fredericton, N.B.
Canada E3B 5A3
wdu@unb.ca

Abstract

JLucid as a hybrid intensional object-oriented language integrates intensional language Lucid and object-oriented language Java. A JLucid program consists of Java class definitions, and a set of Lucid definitions to specify object collaborations in high level. In addition to Lucid operators, Java's new operator and class/instance methods in defined Java classes are also allowed in Lucid definitions. The semantics of a Lucid variable in a context is either a Java basic value or a reference to a Java object. Lucid variables are strongly typed with Java type declarations. A Lucid variable is defined by an expression, which defines the value of the variable in each context. The expression consists of a combination of Lucid's intensional and pointwise operators and Java method calls. Typically a Lucid variable in a context is defined as a reference to an Java object which may be shared by reference values of this or other variables in different contexts. The program semantics of JLucid can be considered as an extension to the original Lucid program with Java classes as extended abstract data types.

JLucid supports three kinds of applications programming. In multidimensional object-oriented programming, one can use Lucid's multidimensionality to specify multidimensional problem/solution structure, and use Java classes to specify states and behaviors of computing engines or agents that work together in the multidimensional space to solve the problem. Collaborations of agents in different contexts are specified by intensional context switching operations combined with message passing. In implicitly parallel object-oriented programming, the eduction computing model and context parallelism are supported in JLucid at top objects level. That is, parallel computations in form of executions of object methods can be initiated by demands in Lucid expression evaluations in different contexts. JLucid can also be used for specifying coordination of distributed components such as Java beans. In this case, the Lucid part of a JLucid program specifies an executable or nonexecutable component coordination algorithm. The program's spatial context space defines
the coordination topology; the temporal context space defines the coordination timing; and the intensional definitions specify the coordination mechanism among components.

JLucid is implemented with various user-definable implementation strategies. The sequential implementation is based on an object-oriented eduction model. In this model, we compile each Lucid variable definition to a Java class. An instance of the class corresponds to the variable in a particular context. The main attribute of the class is an instant variable which is typically a reference to an objects of Java classes defined in the JLucid program. The Lucid and Java parts of a JLucid program are thus integrated seamlessly into a single Java program. Computations of the program are driven at top level by demands for method calls to objects referenced by instances of Lucid variable classes. The multithread implementation allows two instances of Lucid variable classes running in parallel as separate, controllable threads. Two-level thread control managers at program level and class level manage thread creation and synchronization for performance optimization. The distributed parallel implementation is based on the dynamic generator/executor parallel program architecture and implemented using Java RMI. Parallel computation tasks are dynamically distributed to remote generators/executeors with different scheduling strategies: dynamic load balancing, class partition, context partition, and random computation.