

JCHRIDE: An Integrated Development Environment for JCHR

Slim Abdennadher and Shehab Fawzy

Computer Science Department, German University in Cairo
[slim.abdennadher]@guc.edu.eg
<http://www.cs.guc.edu.eg>

Abstract. The rule-based programming language Constraint Handling Rules (CHR) has been introduced to ease the development and implementation of constraint solvers. Currently, several CHR libraries exist in languages such as Prolog, Haskell and Java. The K.U.Leuven JCHR system is a high-performance integration of Constraint Handling Rules (CHR) and Java. JCHR is currently by far the most efficient implementation of CHR in Java. Its performance is competitive with state-of-the-art CHR systems in e.g. HAL and Prolog. To ease the duty of a CHR programmer, we introduce in this paper an integrated development environment (IDE) for the K.U.Leuven JCHR. The IDE is implemented as a plug-in in Eclipse.

1 Introduction

Constraint Handling Rules (CHR) has been designed to be a high-level, committed-choice, online and concurrent programming language, which consists of a set of rules that transform constraints into simpler ones until they are solved [1].

CHR defines both simplification of and propagation over userdefined constraints. Simplification rules replace constraints by simpler constraints while preserving logical equivalence (e.g., $X \leq Y \wedge Y \leq X \Leftrightarrow X = Y$). Propagation rules add new constraints, which are logically redundant but may cause further simplification (e.g. $X \leq Y \wedge Y \leq Z \Rightarrow X \leq Z$). Repeated application of rules incrementally solves constraints. For example, with the propagation rule we can transform $A \leq B \wedge B \leq C \wedge C \leq A$ to $A \leq B \wedge B \leq C \wedge A \leq C \wedge C \leq A$ and this leads after the application of the simplification rules to $A = B \wedge B = C$. Multiple atoms in a rule head are a feature that is essential in solving conjunctions of constraints.

CHR can be used as a stand alone programming language or can be embedded in any host language. There exist several libraries in various languages, such as Prolog, Haskell and Java. The K.U.Leuven JCHR is currently by far the most efficient implementation of CHR in Java [2].

So far the JCHR programmer had to use a text editor to write the code, edit and save it. Then, the user compiles on the console, sees the output errors, and then returns back to the editor to fix them going back and forth between the code and the console. After implementing the constraint solver in JCHR, the

user has to open a Java editor to implement the application using the constraint solver. Thus, there was a huge need for an integrated development environment (IDE) for JCHR.

In computing, an IDE is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor, a compiler and/or interpreter, build automation tools, and (usually) a debugger. IDEs are designed to maximize programmer productivity by providing tightly-knit components with similar user interfaces, thus minimizing the amount of mode switching the programmer must do comparing to loose, discrete collections of disparate development programs.

Typically, an IDE is dedicated to a specific programming language, so as to provide a feature set which most closely matches the programming paradigms of the language. However, some multiple-language IDEs are in use, such as Eclipse. Thus, our solution was the development of an IDE for JCHR as a plug-in in Eclipse 3.3 benefiting from its ability of adding plug-ins, it being a Java editor and the benefits of the regular IDE discussed above. Such an IDE supports the software-developer in the different stages of the development process. JCHRIDE was implemented after looking at JBossIDE [3].

In this paper, we present an IDE for JCHR, called JCHRIDE. In Section 2, we present JCHR by example. In Section 3, we describe some of the implemented features.

2 Java Constraint Handling Rules

A CHR solver consists of rules. We distinguish between three kinds of rules: simplification rules, Propagation rules, and Simplagation rules. A CHR rule consists of a head and a body and may contain a guard. CHR allows multiple heads and a conjunction of zero or more atoms in the guard as well as in the body of the rule. In the following, the syntax and the semantics of CHR are introduced by example.

2.1 JCHR by Example

Following is an example of a typical JCHR program defining the partial order relation \leq . $A \leq B$ holds if variable A is less than or equal to variable B . Compared to the classical CHR syntax, a JCHR solver is written in a Java-like syntax in order to enhance the acceptance of rule-based constraint solving in the Java community. A solver is defined using the keywords `public handler Name` that resembles implementing a class in Java.

```
package examples.leq;
import runtime.*;
public handler leq<T> {
    solver EqualitySolver<T>;
    public constraint leq(Logical<T>, Logical<T>) infix =<;
```

```

rules {
    reflexivity @ X =< X <=> true.
    antisymmetry @ X =< Y, Y =< X <=> X = Y.
    idempotence @ X =< Y \ X =< Y <=> true.
    transitivity @ X =< Y, Y =< Z ==> X =< Z.
}
}

```

The first rule, which is called reflexivity (rule names are optional), is a single-headed simplification rule. Simplification rules correspond to logical equivalence. It removes constraints of the form $A=<A$ from the constraint store. The second rule, antisymmetry rule, is a simplification rule with two atoms in the head. It replaces two $=<$ constraints by an equality constraint. The equality constraint is usually handled by the host language.

The third rule is a simpagation rule which removes redundant copies of the same constraint. Such rules are often needed because of the multi-set semantics of JCHR. Finally, the last rule (transitivity) is a propagation rule that adds redundant constraints. Propagation rules correspond to logical implication.

Execution proceeds by exhaustively applying the rules to a given input query. For example, given the query $A=<B$, $B=<C$, $C=<A$ the transitivity rule adds $A=<C$. Then, by applying the antisymmetry rule, $A=<C$ and $C=<A$ are removed and replaced by $A=C$. Now the antisymmetry rule becomes applicable on the first two constraints of the original query. Now all JCHR constraints are eliminated so no further rules can be applied, thus the answer $A=C$, $A=B$ is returned.

Once the solver is implemented and compiled in JCHR, the user can use the constraints in any Java program. For more details, the reader can refer to the JCHR manual [2].

3 Implemented Features of the IDE

JCHRIDE is an IDE for JCHR implemented as a plug-in in Eclipse 3.3. Since the aim of an IDE is to support the software-developer in the different stages of the development process, JCHRIDE has been implemented with seven features.

- **Syntax highlighting** is a sort of rule-based scanner that highlights some predefined words to be colored with certain colors. The editor highlights keywords, types, constants, single line comments, multi-line comments, Doc comments and text inclosed between the single and double quotes (in other words strings and characters). See Figure 1 for more illustration.
- **Basic functions** are the very basic functions found in any IDE or any text editor like find/replace, copy, cut, paste, ... (see Figure 2).

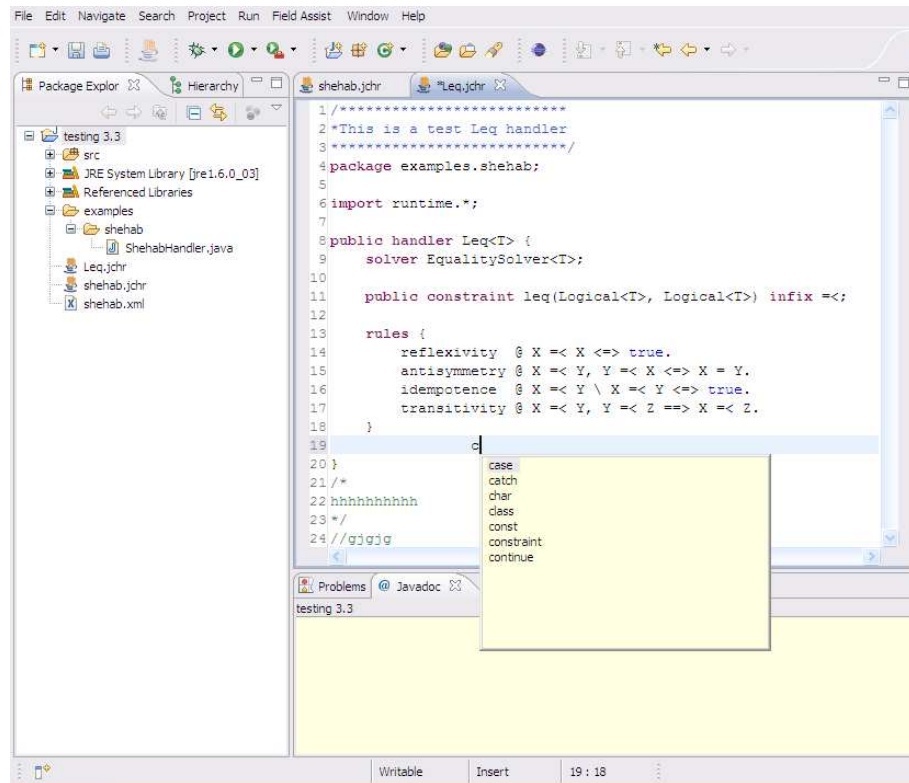


Figure 1: A snapshot from the IDE displaying most of its features

Edit	Navigate	Search	Project	Run	Field
	Undo Typing				Ctrl+Z
	Redo				Ctrl+Y
	Cut				Ctrl+X
	Copy				Ctrl+C
	Paste				Ctrl+V
	Delete				Delete
	Select All				Ctrl+A
	Find/Replace...				Ctrl+F
	Find Next				Ctrl+K
	Find Previous				Ctrl+Shift+K
	Incremental Find Next				Ctrl+J
	Incremental Find Previous				Ctrl+Shift+J
	Add Bookmark...				
	Add Task...				
	Word Completion				Alt+/

Figure 2: Basic functions

- **JCHR format recognition** is an option to recognize automatically a new created file with extension `.JCHR` to be a JCHR program. The Eclipse IDE automatically opens the editor page of the file and changes its icon to the JCHR icon as shown in Figure 1.
- **Word completion** is a feature that helps the user to complete any word with different matching proposals of already written words. This feature is activated whenever you write any number of letters of any written word in the page and if the "Alt" and "/" buttons are pressed.
- **Content assist** is an auto-completion feature. Once you press "Ctrl" and "Space", it opens a pop-up list containing proposals. It also completes or decreases the number of choices as the user writes part of the word as illustrated in Figure 1.
- **JCHR new page wizard** is a feature that enables the user to add a new JCHR file with capabilities like adding a package, a handler and a rule set from the GUI (see Figure 3).

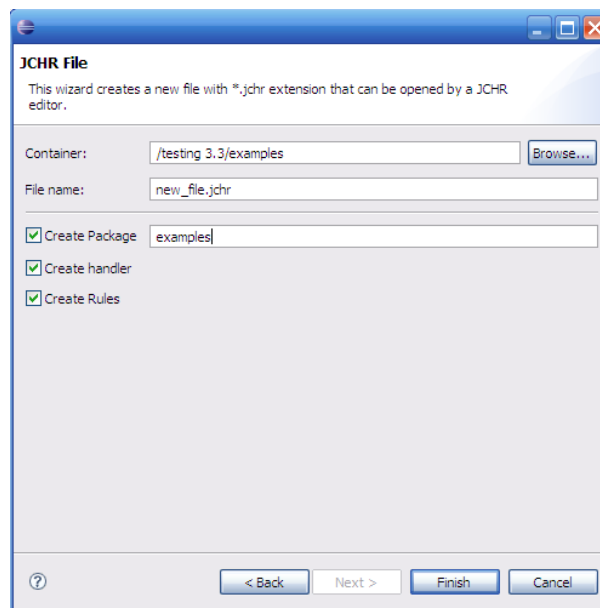


Figure 3: New wizard page

- **A compiling button** is a button containing the JCHR logo. When the user clicks on the button, then first it checks whether the user performed any changes and asks to save the new changes. Then it compiles returning the output in a new window as seen in Figure 4.

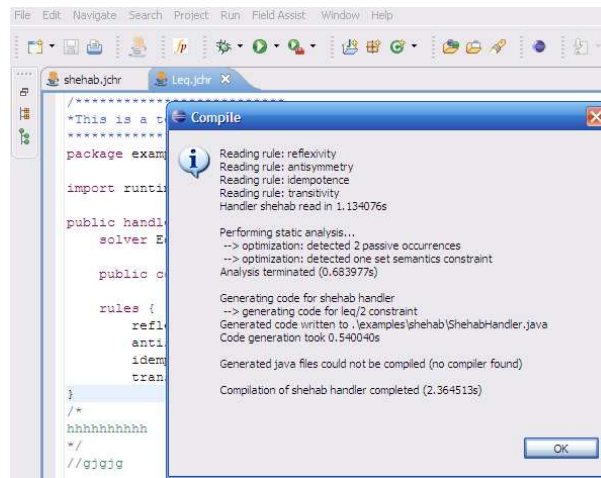


Figure 4: Compile button sample output

4 Conclusion and Future Work

The K.U.Leuven JCHR is a user friendly, flexible and efficient CHR system for Java. These three advantages were achieved by a high-level syntax that is familiar to both Java programmers and users of the other CHR embeddings, a well thought out design and an optimized compilation to Java with the use of very efficient constraint store. One of the drawbacks of the system was the lack of an Integrated Development Environment. So the aim of this project was to ease the duty of the CHR programmer by providing an IDE for JCHR. The current status of the project is plug-in IDE in Eclipse with seven features: Syntax highlighting, Compile button, Dynamic content assist, Word completion, Basic functions (copy, cut, paste, find, etc.), JCHR format recognition and the new file wizard feature with package, rule and handler input functions.

We are currently enhancing JCHRIDE with a debugger and visualization tool for the execution of JCHR solvers. These features will help in enhancing the understanding of the language.

References

1. T. Frühwirth. Theory and practice of constraint handling rules, special issue on constraint logic programming. *Journal of Logic Programming*, 37(1-3):95–138, October 1998.
2. P. Van Weert. JCHR: Constraint Handling Rules in Java. *Manual*, 2006
3. M. Culpepper, L. Etienne, H. Dockter. An introduction and walkthrough of JBoss Eclipse IDE <http://docs.jboss.com/jbosside/tutorial/build/en/html/index.html>, 2002

This article was processed using the L^AT_EX macro package with LLNCS style