Debugging GOAL Agents at the Source Level in the Eclipse IDE

Vincent J. Koeman, Koen V. Hindriks

Delft University of Technology, The Netherlands
{v.j.koeman,k.v.hindriks}@tudelft.nl

1 Introduction

Debugging is the process of detecting, locating, and correcting faults in a computer program [5]. A large part of the effort of a programmer consists of debugging a program. This makes efficient debugging an important factor for both productivity and program quality [10]. Typically, a defect is detected when a program exhibits unexpected behaviour. In order to locate the cause of such behaviour, it is essential to explain how and why it is generated [4]. A source-level debugger is a very useful and important tool for fault localization that supports the suspension and single-step execution of a program [9]. Single-step execution is based on breakpoints, i.e., points at which execution can be suspended [5]. Stepping through program code allows for a detailed inspection of the program state at a specific point in program execution and the evaluation of the effects of specific code sections.

Debuggers typically are source-level debuggers. However, *most debuggers available for agent programs do not provide support for suspending at a particular location in the source code.* Instead, these debuggers provide support for suspension at specific points in the reasoning or decision cycle of an agent. Although the role of an agent's decision cycle in the generation of an agent's behaviour is very important, we believe that source-level debugging is also very useful for agent-oriented programming.

2 Main Purpose: a Source-Level Debugger for Agent Programming

The main aim of the demonstration is to *demonstrate a source-level debugger that supports single-step execution of a cognitive agent program.* Arguably, such a tool provides an agent programmer with a better understanding of the relation between an agent's program code and its behaviour. A source-level debugger for agent programs should support single-step execution through both code-based as well as cycle-based suspension. The possible code-based breakpoints can be defined by using the programming language's syntax. In addition, the agent's decision cycle can be evaluated for important events that are not represented in the agent's source in order to determine cycle-based breakpoints. These points can then be used to define a stepping flow, i.e., identifying the result of a stepping

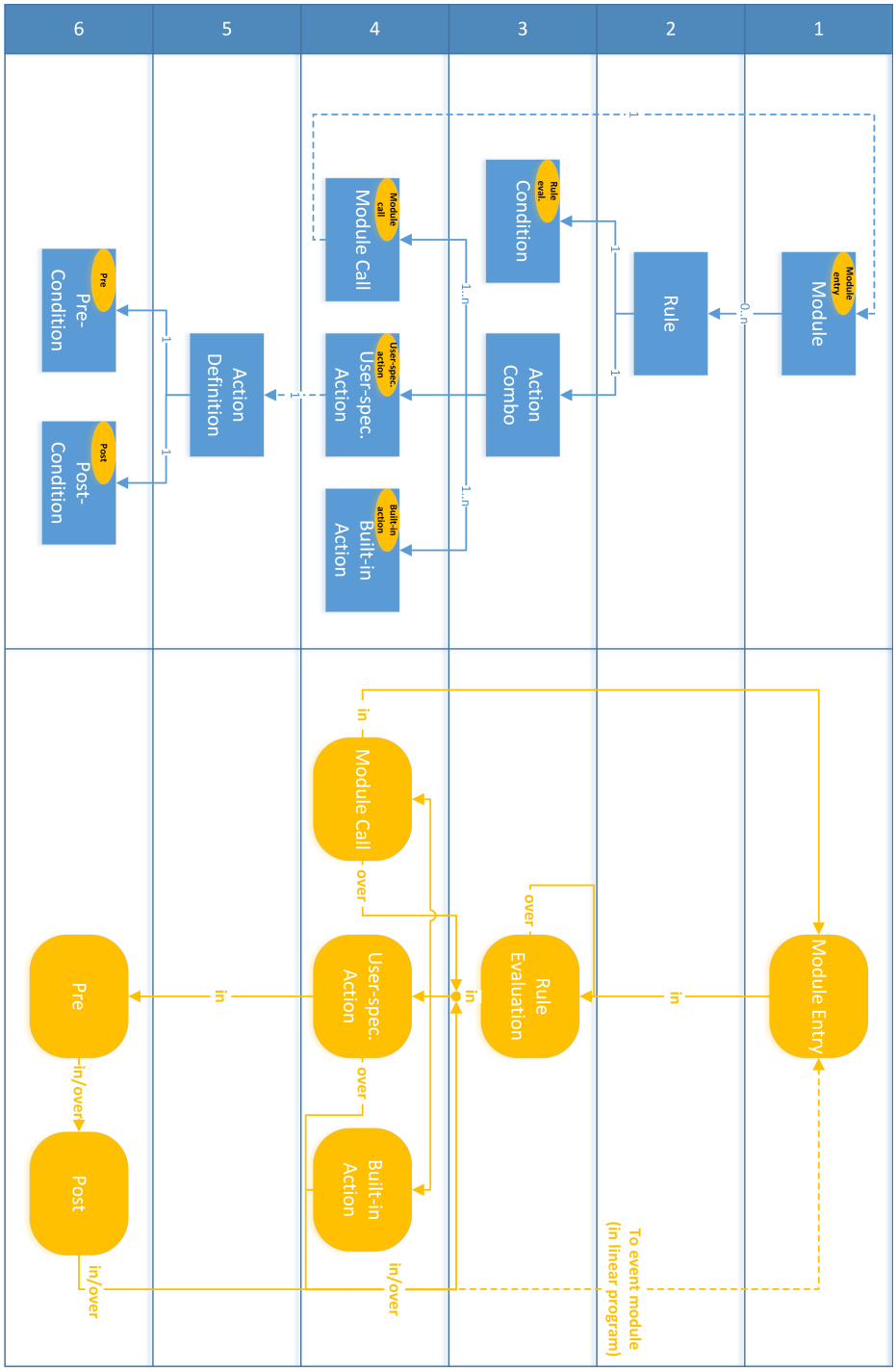

Fig. 1. A GOAL syntax tree with the relevant breakpoints indicated on the nodes that are present at the different levels on one side, and the stepping flow between those breakpoints illustrated on the other.

action on each of those points. Additionally, other required features such as user-defined breakpoints, visualization of the execution flow, and state inspection should be handled.

3 Demonstration: Debugging GOAL Agents in Eclipse

The GOAL agent programming plug-in for Eclipse [1] provides a full-fledged development environment for agent programmers, integrating all agent and agent-environment development tools in a single well-established setting [7]. The Eclipse platform is based on an open architecture that allows for building on top of well-known existing frameworks [2]. By using Eclipse and the DLTK framework [3], for example, a state-of-the-art editor for GOAL has been created. Its features include syntax highlighting, auto-completion, a code outline, code templates, bracket matching, and code folding. By using a newly designed ANTLR 4 grammar [8], specifically its new 'lexical modes' feature for 'language islands', exchangeable support for embedded knowledge-representation (KR) languages is provided. In addition, a testing framework for the validation of agents based on temporal operators has been created.

The source-level debugger for GOAL fully implements the designed stepping flow [6] (see Figure 1) within the Eclipse debugging framework based on the DeBugGer Protocol (DBGP), which is a common debugger protocol for languages and debugger UI communication [1]. When execution is suspended, the debugger highlights the code that is about to be executed, showing its evaluation (i.e., the values of variables referenced in a rule), and allowing for a detailed inspection of the relevant agent's mental state. Such a mental state can be sorted, filtered, queried, and modified. The debugger can also be used in combination with the testing framework. For example, when a test fails, the same functionalities as when stepping are available, allowing a user to investigate the piece of code that was responsible for the failure of a certain temporal condition.

In our demonstration, we will start by showing how to install the GOAL agent programming plug-in in Eclipse. By using two running examples (i.e., two examples of agent systems), the latest version of the GOAL programming language itself will be demonstrated directly from the state-of-the-art code editing facilities that the plug-in contains. Next, the features of the source-level debugger that is integrated within the Eclipse plug-in will be illustrated for a very simple agent that does not contain any bugs. In this way the implementation of the designed stepping flow will be visualized in a concrete and direct manner, by for example showing the results of the different stepping actions or the way a mental state can be inspected, filtered, queried, and modified. After this 'dry run' we will demonstrate the usability and thus effectiveness of the source-level debugger by introducing different kinds of bugs, and then using the debugger to detect them. Finally, some additional features like customizable logging will be highlighted as well.

[1] See `http://goalhub.github.io/eclipse` for a demonstration of the debugger implementation and instructions on how to install GOAL in Eclipse.

4 Conclusion

We have proposed a source-level debugger design for agents that takes code stepping more serious than existing solutions, aimed at providing a better insight into the relationship between program code and the resulting behaviour. We identified two different types of breakpoints for agent programming: code-based and cycle-based. The former are based on the structure of an agent program, whereas the latter are based on an agent's decision cycle. We proposed concrete design steps for designing a debugger for cognitive agent programs. By using the syntax and decision cycle of an agent programming language, a set of pre-defined breakpoints and a flow between them can be determined in a structured manner, and represented in a stepping diagram. Based on such a diagram, features such as user-defined breakpoints, visualization of the execution flow, and state inspection can be handled as well.

By using a concrete design for the GOAL agent programming language, a source-level debugger was created within the Eclipse environment, fully implementing the design within a state-of-the-art setting. This implementation is publicly available and used in our demonstration.

References

1. Caraveo, S., Rethans, D.: Dbgp - a common debugger protocol for languages and debugger ui communication (2007)
2. Geer, D.: Eclipse becomes the dominant java ide. Computer 38(7), 16–18 (July 2005)
3. Gomanyuk, S.: An approach to creating development environments for a wide class of programming languages. Programming and Computer Software 34(4), 225–236 (2008)
4. Hindriks, K.V.: Debugging is explaining. In: Rahwan, I., Wobcke, W., Sen, S., Sugawara, T. (eds.) PRIMA 2012: Principles and Practice of Multi-Agent Systems, LNCS, vol. 7455, pp. 31–45. Springer Berlin Heidelberg (2012)
5. ISO: ISO/IEC/IEEE 24765:2010 systems and software engineering - vocabulary. Tech. rep., Institute of Electrical and Electronics Engineers, Inc. (2010)
6. Koeman, V.J., Hindriks, K.V.: Designing a source-level debugger for cognitive agent programs. In: PRIMA 2015: Principles and Practice of Multi-Agent Systems. LNCS, Springer International Publishing (2015)
7. Koeman, V.J., Hindriks, K.V.: A fully integrated development environment for agent-oriented programming. In: Demazeau, Y., Decker, K.S., Bajo Pérez, J., de la Prieta, F. (eds.) Advances in Practical Applications of Agents, Multi-Agent Systems, and Sustainability: The PAAMS Collection, LNCS, vol. 9086, pp. 288–291. Springer International Publishing (2015)
8. Parr, T.: The Definitive ANTLR 4 Reference. Pragmatic Bookshelf, 2nd edn. (2013)
9. Romero, P., du Boulay, B., Cox, R., Lutz, R., Bryant, S.: Debugging strategies and tactics in a multi-representation software environment. International Journal of Human-Computer Studies 65(12), 992–1009 (2007)
10. Zeller, A.: Why Programs Fail, Second Edition: A Guide to Systematic Debugging. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edn. (2009)