Adding Runtime Net Manipulation Features to MulanViewer

Jan Schlüter, Lawrence Cabac, Daniel Moldt

University of Hamburg, Department of Computer Science,
Vogt-Kölln-Str. 30, 22527 Hamburg
`http://www.informatik.uni-hamburg.de/TGI/`

Abstract. MulanViewer, a Mulan inspection tool, is focused on gathering information from a Petri net-based multi-agent system and greatly helps finding bugs, but fixing them is overly time-intensive. We overcome this limitation by extending MulanViewer with runtime net manipulation features. To do so, we analyze a typical debugging cycle, point out bottlenecks and implement the most promising additions. The new features considerably accelerate the identification and fixing of bugs frequently encountered in Mulan applications. Overall the enhancements complement MulanViewer's features to navigate large and complex Petri net implementations by adding manipulation capabilities.

1 Introduction

The Paose approach is based on high-level Petri nets forming an agent-oriented structure, embedding other successful techniques, such as Java and UML. The development process is highly complex. However, it is supported by a set of powerful tools and we improve the development process by continuous investigation of techniques, tools and methods. In our last project, the time consuming debugging process was one of our targets. In Section 2 we sketch the Paose development setting, highlight the challenges in debugging agent systems and describe current solutions within Mulan. Subsequently, we identify debugging bottlenecks by timing common tasks during Paose debugging processes and suitably extend MulanViewer in Section 3. Section 4 closes with a short summary and an outlook of future work.

2 Developing Multi-Agent Systems with Mulan

Before tackling our goal, we will look at the existing work we can build on: the current development environment, the challenges during debugging multi-agent systems and the toolset meeting them.

2.1 The Paose Development Environment

The multi-agent system development environment used for Petri net-based agent-oriented software engineering is based on Renew (Reference net workshop [4]).

Renew provides a graphical user interface for creating and editing Petri nets and a simulator supporting different formalisms. In the Java net formalism, every token can be a reference to another net or an arbitrary Java object. Nets can communicate with embedded nets through synchronous channels and directly work with Java objects using our Java-based inscription language. During the simulation, Renew visualizes the token game, allowing its user to interactively pause and continue the simulation, fire transitions manually, set breakpoints on transitions or places and inspect the token in detail.

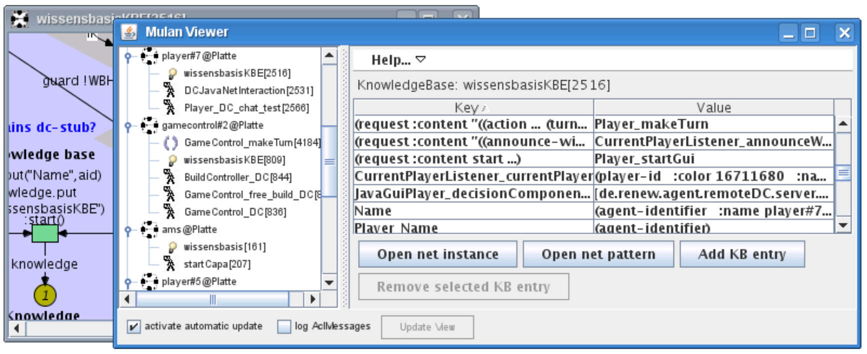
Mulan/Capa is our FIPA-compliant framework for developing multi-agent systems (MAS) based on reference nets and Java. For Mulan, we identify three orthogonal views on a MAS [1]: structure, behavior and terminology. The framework respects this by separating the agents and their knowledge from their behavior, which is encapsulated in protocol nets, and from the Java-based ontology. The implementation of Mulan/Capa extensively uses the nets-within-nets paradigm: The system infrastructure net holds all platform nets, providing global communication and agent mobilization services. Each platform net hosts a number of agents, which can use the platform to exchange messages. An agent, in turn, can be a reference net, too. The default agent holds a declarative knowledge base and *active knowledge* encapsulated in decision component nets as well as a protocol factory, which instantiates protocol nets in reaction to incoming messages based on associations in the knowledge base (reactive behavior) or through internal triggers (proactive behavior). Protocol nets are reference nets defining a structured schedule of internal actions (knowledge base and decision component access) and external actions (messages send out and received). A protocol usually corresponds to a column in an interaction diagram defining the interactions taking place with one or multiple other agents (roles).

2.2 Challenges in Debugging Multi-Agent Systems

Apart from the common pitfalls [6] shared with object-oriented software development, developers of multi-agent systems have to cope with some specific problems, the most important one being that “[m]ultiagent systems tend to lack any central control” [7, p.3]: The system consists of autonomous, possibly distributed agents acting and interacting concurrently without any global instance controlling it. This imposes a number of challenges to be overcome during debugging a MAS. Due to the dependencies between agents, locally inspecting them often does not suffice to track the source of a problem, but the complexity of the system and the lack of a central instance makes it difficult to gain a global overview of the system’s state [2]. Furthermore, the concurrent processes within the system may induce nondeterministic behavior [5], preventing the clear reproduction of observed problems.

2.3 Solutions in Mulan

Renew principally provides means to inspect a running multi-agent system, gaining insight into platforms, its agents and their states in terms of net in-



KnowledgeBase: wissensbasisKBE[2516]

Key	Value
(request :content "((action ... (turn...	Player_makeTurn
(request :content "((announce-wi...	CurrentPlayerListener_announceW...
(request :content start ...)	Player_startGui
CurrentPlayerListener_currentPlayer	(player-id :color 16711680 :na...
JavaGuiPlayer_decisionComponen...	[de.renew.agent.remoteDC.server....
Name	(agent-identifier :name player#7...
Player_Name	(agent-identifier)

Fig. 1. MulanViewer displaying an agent's knowledge base

stances. However, actually finding a certain agent's knowledge base or a particular protocol instance within the running system requires navigating a deep hierarchy of heterogeneous nested nets. This may often be confusing and time-consuming and makes it difficult to grasp a thorough overview about what is happening. The inspection tool called MulanViewer [3,2] fills this gap. Generally spoken, this tool enables its user to easily access the tokens of interesting places and (some) transitions in the complex net system. It does so by browsing the net structure, registering for change notifications at the Renew simulation engine and building a hierarchical model of the system's accumulated state, i.e. the platforms, agents, their knowledge bases, decision components and protocols. This model is visualized in a graphical user interface providing an overview of the active components in the MAS as well as adequate views into these components such as a table listing an agent's declarative knowledge (see Figure 1). For investigation on a lower level MulanViewer allows opening a component's corresponding net in Renew.

The two tools MulanViewer and Renew form a powerful inspection toolset for multi-agent systems implemented in Mulan/Capa. As the system's components are reference nets, most types of errors lead to a transition not being able to fire, which in turn leads to a protocol being blocked. Blocked protocols can easily be found in MulanViewer's overview. Renew's visualization of the protocol net instance allows to quickly locate the problematic transition: The developer just has to look for the lifeline token in the protocol's schedule of actions. The following transition's inscription reveals whether the protocol is blocked because of a missing knowledge base entry, an erroneous decision component channel or other unmet preconditions.

3 Extending MulanViewer

While finding the problem in a MAS is rather easy, fixing it is tedious. As Renew does not provide a way to change a net or its tokens during simulation – which, by the way, might introduce more problems than it solves – developers have to stop the MAS, modify the nets or the agent's initial knowledge base and recompile the project, leading to an overly time-consuming debugging cycle.

We propose that extending the toolset with manipulation features for a running MAS would greatly optimize the process of debugging: Modifying a running MAS dynamically until it works should be faster than modifying the static MAS code, recompiling and restarting it each time. To find out which kinds of features particularly help saving time, we analyze a typical debugging cycle to identify bottlenecks. Subsequently, we extend our toolset to eliminate these bottlenecks by providing means to carefully modify net instances during the simulation.

3.1 Identifying Debugging Bottlenecks

Reflecting on a project course held in Winter 2007/2008, we analyze how the tools described in Section 2.3 are usually used to resolve problems in a Mulan/Capa MAS in order to find the most time-intensive steps. We use a MAS implementing the German board game "Siedler", running on a 2.8 GHz dual core machine with 3 GB working memory to take the time for the tasks. Figure 2 depicts our results: a Petri net modeling the dependencies between debugging tasks and the outcomes of tests.

To be able to debug a component of the MAS, its developers have to ensure they have the latest versions of the project, compile it and bring the MAS into a state that uses their component to see whether it works as intended (steps 1 to 5). Although we assumed the project can be quickly retrieved from a shared repository and the developers can easily activate their component (here: a protocol of an interaction), these steps take up to three minutes.

As we can see in the debugging cycle, these steps have to be repeated on each knowledge base, protocol or decision component related problem. Knowledge base problems comprise an additional bottleneck: The design artifact cannot be merged, thus distributed manipulation synchronized through a repository is not possible, yet.

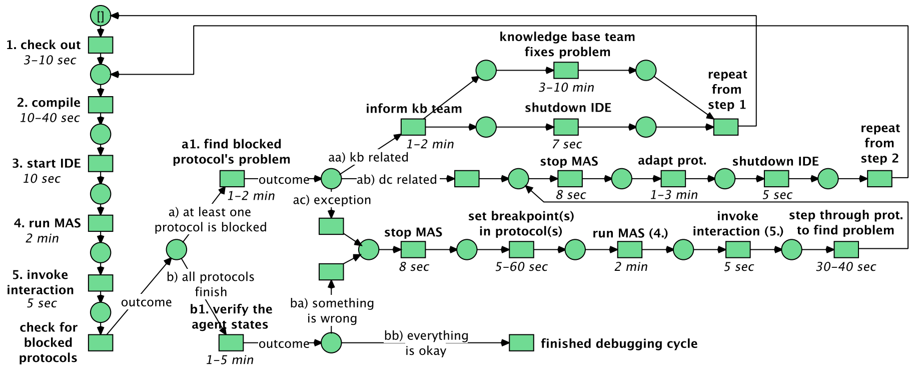


Fig. 2. Petri net modeling a typical Paose debug cycle.

3.2 Implementing Manipulation Features

Although the debugging cycle could be sped up by minimizing the time needed for the repeated steps (most notably steps 1 to 5), this would not change its problematic *structure*. Improving the knowledge base editor to eliminate the dependency on the knowledge base team would arguably help, but we would rather like to completely remove the need for changing the initial knowledge base content just to continue testing. In the same line, decision component and protocol problems should be solvable in the running system.

A general solution for this would be extending Renew to allow editing a net's structure during a simulation, automatically updating all of the net's instances (*Hot Code Replacement*), and to allow modification of a token (*Token Injection*) – this way all problems in knowledge bases, protocols and decision components could be fixed or at least worked around on the fly without needing to restart the system. However, this would require substantial changes to Renew, which are outside the scope of this work. Instead, we will cope with the different types of problems separately, trying to find and implement solutions to fix the most common problems by carefully modifying the running system.

The most usual knowledge base related problems are missing or misspelled entries, causing protocols or decision components accessing them to be blocked. To resolve them, we extend MulanViewer's knowledge base view with capabilities to interactively add, change and remove knowledge base entries. MulanViewer does not directly modify the knowledge base net, but asks Renew's simulation engine to fire the knowledge accessing transitions on the agent's behalf, thus avoiding concurrency problems.

For the specific, but frequent case of erroneous message-to-protocol associations merely fixing the entry often does not suffice: When the problem is spotted by a developer, it usually is because the agent could not interpret a message and suspended an interaction. After correcting the entry, recreating the state of the MAS prior to the misinterpretation may require a restart, which we want to avoid. So we extend MulanViewer to allow moving a message from the protocol factory's *not understood* place to the place of incoming messages, forcing the agent to reinterpret the message without the other agent needing to resend it.

In the case of missing or misnamed DC channels, being able to modify the affected agent's DC certainly would provide the greatest relief. However, as this is out of scope, we implement a feature that loads additional DCs into an agent by instantiating them and injecting them into the agent net. This way misspelled and missing channels can be provided as wrapper and stub channels in a new DC, which can be drawn and loaded by the developer on the fly.

As an all-round solution for circumventing problems during debugging a particular component that are caused by another agent not acting as supposed, we extend MulanViewer to enable developers to start a protocol interactively in the context of an agent. While this does not solve the real problem, it allows a developer to finish testing the respective component independently of the other agent's bugs.

4 Conclusion

We evolved MulanViewer from a mere monitoring tool into an inspection tool capable of manipulating a MULAN multi-agent system during runtime by directly working on the underlying net instances. By this, we reduced the time needed for fixing particularly common problems found during the debugging phase from about 3 minutes to less than a minute, which was our main source of motivation. Apart from that, the new manipulation features help testing a single MAS component outside its usual environment, partly eliminating the dependency between developers when working in a team.

Future work will include additional manipulation features and tool support for repeated tasks in order to further accelerate the debugging cycle. Additionally, we will extend MULANVIEWER to allow its users to apply runtime manipulations – like editing the knowledge base – to the static code, making changes persistent. As our tool now can considerably interfere with a running MAS, we need to develop a security model, allowing a system or its components to be protected from being manipulated. The MULANVIEWER idea of directly accessing particular places and transitions provokes a strong dependence on the MULAN implementation anyway, so we plan to rework the architecture, shifting as much responsibility as possible to the MAS, allowing it to determine which and how much information may be published and which states may be modified from outside.

References

1. Lawrence Cabac, Till Dörges, Michael Duvigneau, Christine Reese, and Matthias Wester-Ebbinghaus. Application development with Mulan. In Daniel Moldt, Fabrice Kordon, Kees van Hee, José-Manuel Colom, and Rémi Bastide, editors, *Proceedings of the International Workshop on Petri Nets and Software Engineering (PNSE'07)*, pages 145–159, Siedlce, Poland, June 2007. Akademia Podlaska.
2. Lawrence Cabac, Till Dörges, and Heiko Rölke. A monitoring toolset for Petri net-based agent-oriented software engineering. In Rüdiger Valk and Kees M. Valk van Hee, editors, *29th International Conference on Application and Theory of Petri Nets, Xi'an, China*, volume 5062, pages 399–408, June 2008.
3. Timo Carl. Evaluation und beispielhafte Erweiterung einer referenznetzbasierten Agentenumgebung. Studienarbeit, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln Str. 30, D-22527 Hamburg, August 2003.
4. Olaf Kummer, Frank Wienberg, and Michael Duvigneau. Renew – the Reference Net Workshop. Available at: http://www.renew.de/, May 2006. Release 2.1.
5. David Poutakidis, Lin Padgham, and Michael Winikoff. Debugging multi-agent systems using design artifacts: the case of interaction protocols. In *AAMAS*, pages 960–967. ACM, 2002.
6. Bruce F. Webster. *Pitfalls of object-oriented development.* M & T Books, New York, NY, USA, 1995.
7. M.J. Wooldridge and N.R. Jennings. Software engineering with agents: pitfalls and pratfalls. *Internet Computing, IEEE*, 3(3):20–27, May/Jun 1999.