A Monitoring Toolset for PAOSE

Lawrence Cabac, Till Dörges, Heiko Rölke

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

Abstract PAOSE (Petri net-based Agent-Oriented Software Engineering) combines the paradigm of AOSE (Agent-Oriented Software Engineering, see [10]) with the expressive power of Petri nets – reference nets [12] to be more precise. While AOSE is a powerful approach when it comes to designing and developing distributed (agent) applications, it does not address the problems specific to debugging, monitoring, and testing of these applications, i.e. no global state of the system and very dynamic operating conditions. To tackle these problems, two tools have been developed in the context of PAOSE, which are presented in this work.
Firstly, this paper will give a short overview over the interrelated set of tools, which exists already and supports Petri net-based AOSE. The tools are centered around the Petri net-based multi-agent system development and runtime environment RENEW / MULAN / CAPA.
Secondly, MULAN-Viewer and MULAN-Sniffer will be presented in more detail – two tools to address the issues encountered during debugging, monitoring, and testing agent applications. Both tools are first class members of the aforementioned family. The first tool, MULAN-Viewer, deals with the introspection of agents and agent behaviors, while it also offers rudimentary features for controlling the agent-system. The MULAN-Sniffer as the second tool places emphasis on tracing, visualizing, and analyzing communication between all parts of the multi-agent application and offers interfaces for more advanced methods of analysis, such as process mining. Both MULAN-Viewer and MULAN-Sniffer are realized as RENEW plugins that can also be extended by other plugins.

Keywords reference nets, RENEW, monitoring, testing, debugging, inspection, analysis, multi-agent systems, PAOSE

1 Introduction

Developers of multi-agent applications – or distributed systems in general – have to cope with many aspects of such systems that increase the difficulty of activities like debugging, monitoring, and testing. Particularly multi-agent applications sport decentralized control, very dynamic operating conditions, and they are inherently complex. Therefore these tasks are hard [19,17].

The main underlying problem is that distributed systems do not allow for a simple definition of their global state, as the state depends for example on the location of the observing agent. And even if the definition succeeds, gathering the state of a distributed system is far from trivial – leave alone single-stepping through a sequence of states – because it is hard to freeze the system. In addition to this, system access permissions have to be taken into account, e.g. a developer may not have access to all parts of the system even though these very parts trigger problems with the sub-system being developed by him. Concurrency can also lead to errors that only occur in special settings or are not reproducible.

First of all this paper will give a brief overview over the family of interrelated tools that are used for Petri net-based AOSE (for other approaches compare [1] or [16]). Apart from the base set there are tools for designing and creating, as well as tools for debugging, monitoring, and testing[1] multi-agent applications. From the latter set of tools for inspecting the Mulan-Viewer focuses on the inspection of individual agents, their behavior, and the platforms. The Mulan-Sniffer focuses on the communication between agents. Here communication is traced directly at the transport service layer. A third inspection mechanism actually is based on the inspection of Petri net instances at runtime provided by the virtual machine Renew as explained in Section 2.

Mulan-Viewer and Mulan-Sniffer are implemented as extensible Renew plugins. Both are able to connect to remote platforms using TCP/IP connections. By this means they support distributed and independent system inspection.

Section 2 will present the interrelated set of tools crucial for our Petri net-based AOSE process. Sections 3 and 4 will introduce the Mulan-Viewer and the Mulan-Sniffer respectively.

2 A Family of Tools to Support Petri net-based AOSE

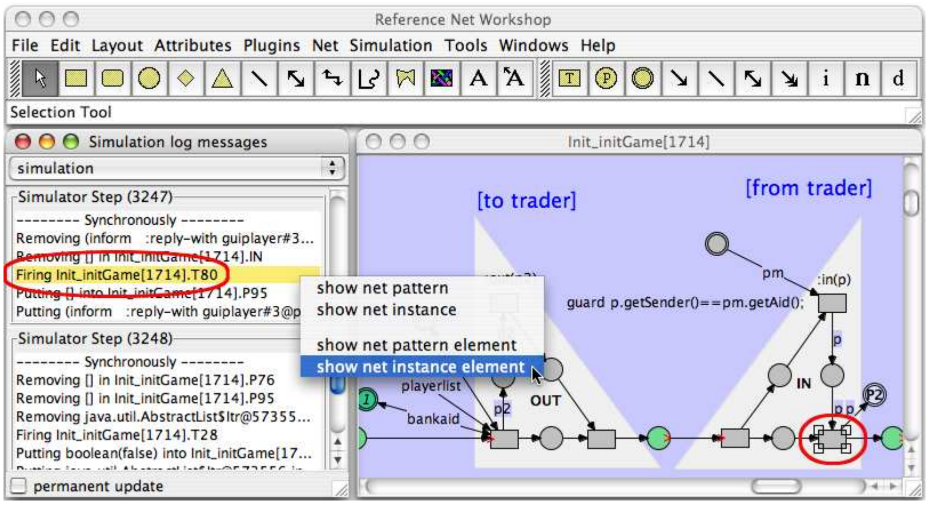
This section will briefly describe the "relatives" of the inspecting tools, which constitute the main focus of this paper. All tools are under constant development and key for the research activities conducted in our group.

2.1 Tool Basis

Renew [14,13], Mulan [11,18], and Capa [7] are the condicio sine qua non for the entire Petri net-based AOSE process. Renew can be thought of as both the virtual machine (VM) and the integrated development environment (IDE) for all our Petri net-based development. It features an extensible plugin architecture, which helps in hooking all the members of our tool family together.

Apart from being the VM it also provides an inspection mechanism for Petri net instances at runtime. The simulator allows to view the token game, run in continuous or single-step mode, allows to set locally or globally and statically or dynamically defined breakpoints and has been enhanced to inspect token objects in a UML-like viewer in version 2.1. In the context of testing and debugging of control-flow and data, handling this kind of inspection on a very low level is of great advantage when it comes to agent behavior (protocol nets).

[1] We regard all activities (*debugging*, *monitoring*, *testing*) as forms of *inspection.*



```
Simulator Step (3247)
-------- Synchronously --------
Removing (inform   :reply-with guiplayer#3...
Removing [] in Init_initGame[1714].IN
Firing Init_initGame[1714].T80
Putting [] into Init_initGame[1714].P95
Putting (inform   :reply-with guiplayer#3@p
Simulator Step (3248)
-------- Synchronously --------
Removing [] in Init_initGame[1714].P76
Removing [] in Init_initGame[1714].P95
Removing java.util.AbstractList$Itr@57355...
Firing Init_initGame[1714].T28
Putting boolean(false) into Init_initGame[17...
```

Figure 1. The IDE part of RENEW showing menu, two palettes (drawing tools and Petri net tools), a part of the simulation log in the log view, and a fragment of a protocol net instance with a highlighted figure (compare with context menu).

Figure 1 shows the graphical interface of RENEW together with a net instance during execution. On the left one can also see the simulation log which permits going through all simulation steps and inspecting the transitions or places.

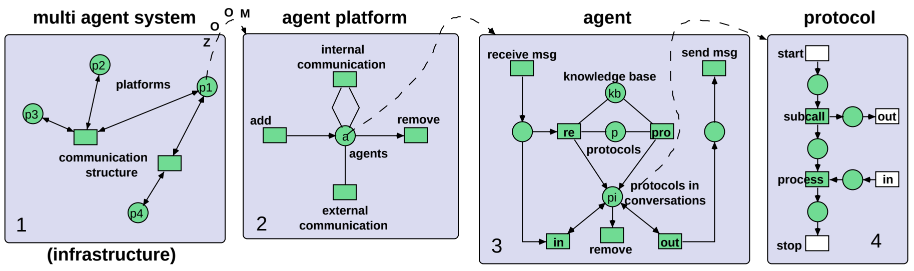


Figure 2. The MULAN architecture; from infrastructure to agent protocol.

MULAN describes the reference architecture for multi-agent nets. Strictly speaking an executable conceptual framework. It is heavily based on reference nets and was designed with the corresponding FIPA [8] standards in mind. Figure 2 shows the key architectural approach. CAPA finally is the current reference implementation for MULAN. It is FIPA-compliant and uses *Java* as well as reference nets (via RENEW) to implement an infrastructure for multi-agent systems.

2.2 Tools for Design and Code Generation

The tools introduced in Section 2.1 have been used numerous times to design large Petri net-based multi-agent applications (MAA). Throughout these processes several tools were spawned that greatly facilitate the creation of applications, e.g. by automatic code generation (see [6]).

Use Cases model the relations between actors and their interactions. With the corresponding tool for PAOSE we model agents of the system and their interactions. Thus the diagram provides an overview of the system and we can generate the necessary directory structure for sources of the MAA as well as the file stubs (build files, ontology and knowledge base files). The corresponding palette can be seen in the upper right corner of Figure 3.

Agent Interaction Protocol Diagrams (AIP) [6] detail the interactions identified during use case (coarse system) design. Tool support consists of rapid modeling support and generation of protocol nets (see menu in Figure 3) to be executed by the agents. The resulting protocol code is made up of directly runnable Petri nets, i.e. reference nets. Figure 3 shows an AIP fragment on the left and a generated agent protocol net (Petri net code) on the right.

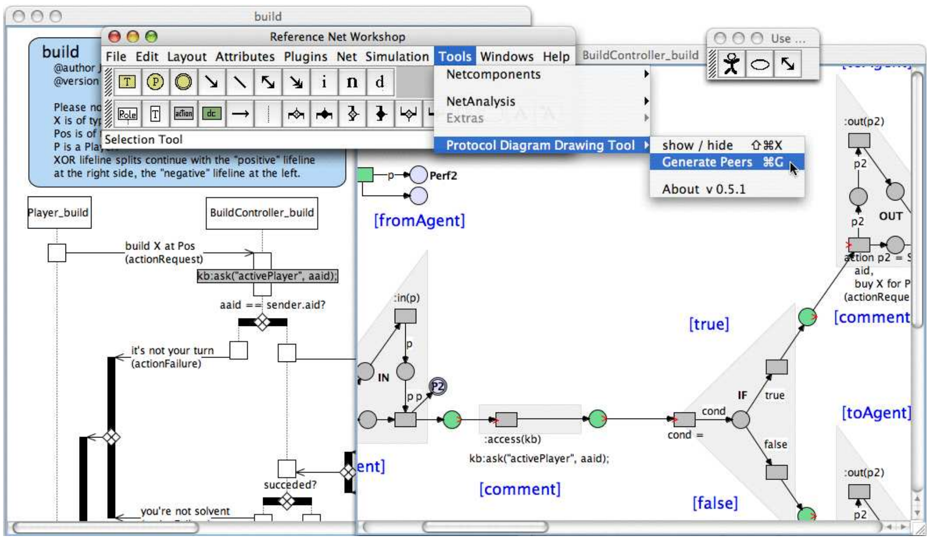
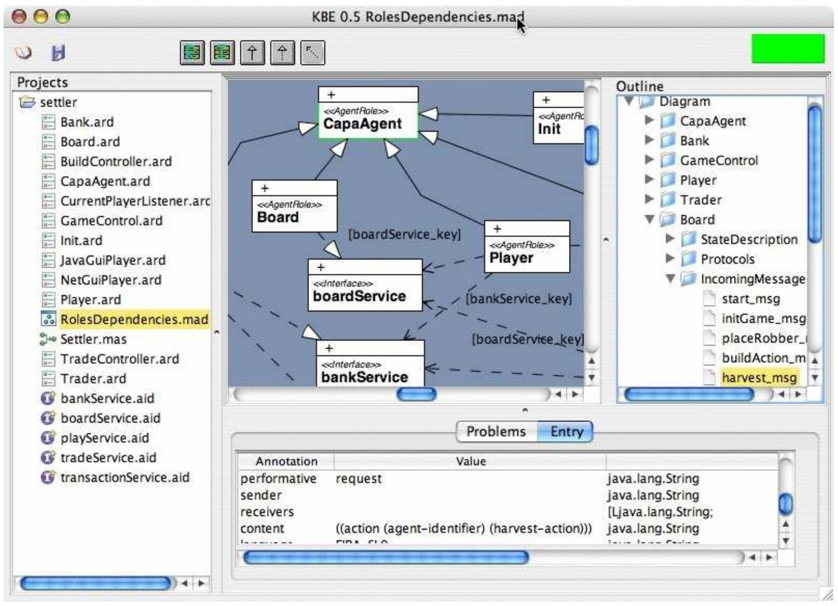


Figure 3. RENEW menu for generating protocols nets from AIP diagrams.

Knowledge Base Editor [3] is somewhat misleading as a name since it can not only be used to generate agents' knowledge bases. It can also model roles to be assumed by agents, services provided and required by agents as well as the necessary messages. Figure 4 shows the tool with a fragment of the roles and services dependencies modeled in the center. The outline on the right permits selection of specific entries which are displayed on the bottom.



Annotation	Value	
performative	request	java.lang.String
sender		java.lang.String
receivers		[Ljava.lang.String;
content	((action (agent-identifier) (harvest-action)))	java.lang.String

Figure 4. KBE tool showing modeling of roles and service dependencies.

Ontology Generation [4] provides tool support for creating *Java* classes that contain the ontology. These classes can then be easily used across the entire multi-agent application.

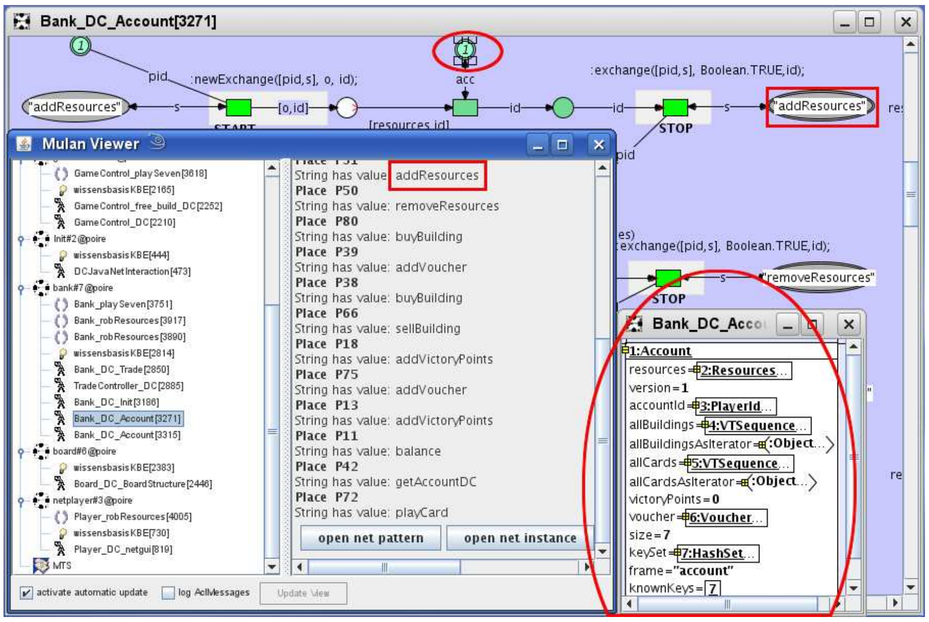
The following sections will present two tools for inspection: MULAN-Viewer and MULAN-Sniffer. It is important to note that both tools have been realized as RENEW plugins and their functionality can be extended by other plugins.

3 MULAN-Viewer

The MULAN-Viewer allows to inspect the state of a multi-agent system (MAS) or several multi-agent systems and its agents, their knowledge bases, and their active protocols and decision components (internal protocols missing external communication features). The main components of the MULAN-Viewer are the platform inspector and the graphical user interface. An arbitrary number of platforms can be inspected both locally and remotely at the same time.

The user interface consists of two views: a MAS overview on the left and the detail view on the right (see Figure 5). The hierarchical structure of the multi-agent system is represented as a tree view. The levels of the tree view correspond directly to three of the four levels known from the MULAN model (see Figure 2): agent platform, agent, and internal agent components (knowledge base, protocols, decision components[2]). The message transport system agent

[2] Decision components are omitted in the model, as they are special types of protocols.



```
GameControl_playSeven[3618]
wissensbasisKBE[2165]
GameControl_free_build_DC[2252]
GameControl_DC[2210]
Init#2@poire
  wissensbasisKBE[444]
  DCJavaNetInteraction[473]
bank#7@poire
  Bank_playSeven[3751]
  Bank_robResources[3917]
  Bank_robResources[3890]
  wissensbasisKBE[2814]
  Bank_DC_Trade[2850]
  TradeController_DC[2885]
  Bank_DC_Init[3186]
  Bank_DC_Account[3271]
  Bank_DC_Account[3315]
board#6@poire
  wissensbasisKBE[2383]
  Board_DC_BoardStructure[2446]
netplayer#3@poire
  Player_robResources[4005]
  wissensbasisKBE[730]
  Player_DC_netgui[819]
MTS
```

[✓] activate automatic update [ ] log AclMessages

```
String has value addResources
Place  P50
String has value: removeResources
Place  P80
String has value: buyBuilding
Place  P39
String has value: addVoucher
Place  P38
String has value: buyBuilding
Place  P66
String has value: sellBuilding
Place  P18
String has value: addVictoryPoints
Place  P75
String has value: addVoucher
Place  P13
String has value: addVictoryPoints
Place  P11
String has value: balance
Place  P42
String has value: getAccountDC
Place  P72
String has value: playCard
```

```
1:Account
resources=2:Resources...
version=1
accountId=3:PlayerId...
allBuildings=4:VTSequence...
allBuildingsAsIterator=<:Object...>
allCards=5:VTSequence...
allCardsAsIterator=<:Object...>
victoryPoints=0
voucher=6:Voucher...
size=7
keySet=7:HashSet...
frame="account"
knownKeys=7
```

Figure 5. MULAN-Viewer depicting several agents and their protocols. A net instance (opened from the MULAN-Viewer) can be seen in the background. On the right bottom the content of the place from the top is inspected.

(MTS) associated with each platform can be seen on the bottom left. If desired, the messages can be listed. All elements can be inspected in the detail view. Additionally, underlying Petri nets (as well as Petri net instances) of agents, protocols, decision components and knowledge bases can be directly accessed.

In Figure 5 the running instance of the net *Bank_DC_Account* from agent *bank#7* on platform *poire* has been opened in the detail view (right hand side of the MULAN-Viewer) and can be seen in the background. The superimposed rectangles[3] indicate which elements from the detail view correspond to those in the inspected nets. In this case the string *addResources* shown in the detail view of the MULAN-Viewer is contained by the place in question. The parts marked by superimposed ellipses[3] show how inspection levels can be nested: First the MULAN-Viewer allows for navigation (left hand side of the MULAN-Viewer) to the desired agent and its protocols. Then the Petri nets can be inspected using RENEW. In the example the place on the top contains one token of type *Account*, which is inspected as UML object hierarchy (*token bag*). The MULAN-Viewer also allows for simple means of control like launching or stopping an agent.

The interaction between the MULAN-Viewer and the agent platform that is being inspected works according to a server/client architecture, with the MULAN-Viewer being the client. The server is realized by an observation module in the

[3] Note that the original color of superimposed elements is red.

platform. More precisely the observation module registers with the underlying simulation engine provided by Renew. The simulation engine keeps a model of the simulated platform, that can be accessed by modules like the observation module. Changes in simulation state are directly updated in the model and propagated through events. The entire communication is based on *Java* Remote Method Invocation (RMI).

4 Mulan-Sniffer

In every distributed system coordination between the involved sub-systems is important. Coordination usually is accomplished through the exchange of messages. From the developers' points of view it is therefore crucial that they are able to analyze this exchange and inspect the corresponding messages.

For the Petri net-based multi-agent system Mulan / Capa a suitable tool, the Mulan-Sniffer, has been developed. It was inspired by the JADE sniffer [9]; other related tools and approaches are the ACLAnalyser [2] and the sniffer in MadKit [15]. It uses (agent) interaction protocols (AIP) for visualization and debugging [6,17]. The Mulan-Sniffer focuses on analyzing messages sent by agents in a multi-agent system. The key features are:

portability The current implementation – realized in *Java* – has been tested with the Petri net-based multi-agent system Mulan / Capa, but adaption to other FIPA compliant multi-agent systems is easily possible. Theoretically nothing about the target multi-agent system needs to be known, as SL0 content could be directly read from the wire[4] via libpcap [5] or comparable means.

modular architecture Not only the input system is extensible but filtering, analysis and presentation of the messages can be adapted through a powerful plugin system as well. Extensions can even be realized using Petri nets.

distribution The Mulan-Sniffer is able to gather messages from both local and remote platforms.

filtering Messages can be selected using stateful or stateless filters. Basic filtering primitives (*from*, *to*, ...) are provided. More sophisticated filters are to be added via the plugin system. Apart from online filtering offline filtering is also possible.

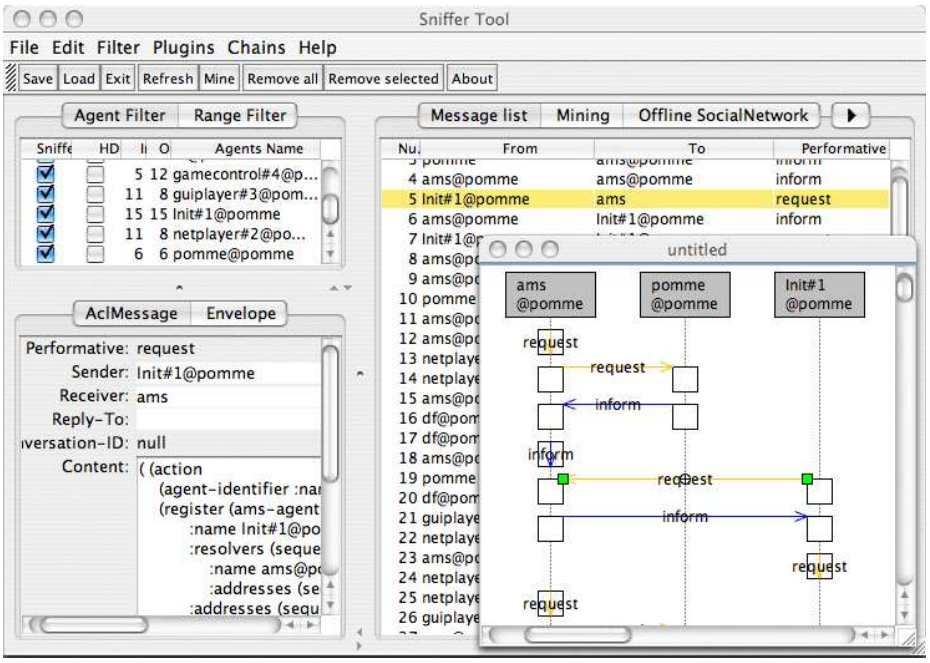
analysis *Mining-chains* can be used to apply arbitrary analysis algorithms to the messages. Examples are given in [5].

visualization Apart from showing elementary statistics (total number of messages sent, ...) each message can thoroughly be inspected. Moreover sequence diagrams are auto-generated (in function of the filters applied) on the fly. More complex visualizations can – of course – be realized as plugins. It is interesting to note that the sequence diagrams are actually Agent Interaction Protocol Diagrams. From these AIP Petri net code stubs for agent

[4] Unless cryptography is employed.

[5] `http://www.tcpdump.org/`, `http://sourceforge.net/projects/libpcap/`

protocols can be generated again [6]. Thus, agent behavior can be defined by observing a running system turning user interaction (manually) into agent behavior or even allowing the agents to use these observations to adapt their own behaviors.


Sniffer Tool
File Edit Filter Plugins Chains Help
Save Load Exit Refresh Mine Remove all Remove selected About
Agent Filter Range Filter
Message list Mining Offline SocialNetwork
AclMessage Envelope
untitled
ams @pomme
pomme @pomme
Init#1 @pomme
request
request
inform
inform
request
inform
request
request

Sniffe	HD	I	O	Agents Name
✓		5	12	gamecontrol#4@p...
✓		11	8	guiplayer#3@pom...
✓		15	15	Init#1@pomme
✓		11	8	netplayer#2@po...
✓		6	6	pomme@pomme

Performative:	request
Sender:	Init#1@pomme
Receiver:	ams
Reply-To:	
ıversation-ID:	null
Content:	((action (agent-identifier :nai (register (ams-agent :name Init#1@po :resolvers (seque :name ams@pc :addresses (se :addresses (sequ

Nu.	From	To	Performative
4	ams@pomme	ams@pomme	inform
5	Init#1@pomme	ams	request
6	ams@pomme	Init#1@pomme	inform
7	Init#1@p		
8	ams@pc		
9	ams@pc		
10	pomme		
11	ams@pc		
12	ams@pc		
13	netplaye		
14	netplaye		
15	ams@pc		
16	df@pom		
17	df@pom		
18	ams@pc		
19	pomme		
20	df@pom		
21	guiplaye		
22	netplaye		
23	ams@pc		
24	netplaye		
25	netplaye		
26	guiplaye		

Figure 6. MULAN-Sniffer main window with generated sequence diagram.

Figure 6 shows the MULAN-Sniffer's main window, while sniffing the messages from a teaching project. The main window is divided in three major areas. The top-left one displays the agents known from sniffing the messages. Here, agents can be selected for simple filtering. The right area shows the *Message list*. The currently selected message is displayed in detail in the bottom left area of the main window. Next to the Message list tab one can select from a couple of viewer plugins loaded already. *Online SocialNetwork* (accessible via the arrow next to *Offline SocialNetwork*) for example allows to visualize the frequency of message exchange by pairs of agents. Additionally a part of the on-the-fly auto-generated sequence diagram is shown. Selecting a message arrow in the diagram will highlight the corresponding message in the message list and display the content in the message detail view (tabs: *AclMessage* and *Envelope)* and vice versa.

The MULAN-Sniffer uses the same interface of the platform as the MULAN-Viewer for the collection of messages.

5 Conclusion

In this paper we presented two tools for debugging, monitoring, and testing multi-agent applications. The tools are related with an entire family of tools that form the key elements of our Petri net-based AOSE process. The Mulan-Viewer provides a hierarchical overview of all agents in the system. It also permits inspecting the state of all elements in the system, whereas the Mulan-Sniffer focuses on the communication of agents and its graphical visualization. Both have been used and enhanced extensively in our teaching projects.

Further possible improvements of the tools are more and better representation of element details in the Mulan-Viewer's interface and increasing the ease of debugging multi-agent applications by adding features for the manipulation of communications (injecting messages), conversations (starting protocols), states (changing knowledge base content), and organizations (migrating agents). To improve the handling of very large scale message logs the integration of an efficient data base is desirable for the Mulan-Sniffer. An interesting topic is the log analysis through advanced techniques such as process mining, for which a prototypical plugin already exists. Further areas of research are ad-hoc systems, support for other platforms, dynamic reconfiguration, and the integration of a security model in the overall architecture as well as the supporting tools.

Acknowledgments

Several authors have contributed to the development of the presented tools. Among these are Timo Carl, Michael Duvigneau, Frank Heitmann, Dr. Daniel Moldt, Florian Plähn, and Jörn Schumacher.

References

1. Ali Al-Shabibi, Didier Buchs, Mathieu Buffo, Stanislav Chachkov, Ang Chen, and David Hurzeler. Prototyping object oriented specifications. In Wil M. P. van der Aalst and Eike Best, editors, *ICATPN*, volume 2679 of *Lecture Notes in Computer Science*, pages 473–482. Springer, 2003.
2. Juan A. Botía, Juan M. Hernansaez, and Fernando G. Skarmeta. Towards an approach for debugging mas through the analysis of acl messages. In Gabriela Lindemann, Jörg Denzinger, Ingo J. Timm, and Rainer Unland, editors, *MATES*, volume 3187 of *Lecture Notes in Computer Science*, pages 301–312. Springer, 2004.
3. Lawrence Cabac, Ragna Dirkner, and Heiko Rölke. Modelling service dependencies for the analysis and design of multi-agent applications. In Daniel Moldt, editor, *Proceedings of the Fourth International Workshop on Modelling of Objects, Components, and Agents. MOCA'06*, number FBI-HH-B-272/06 in Reports of the Department of Informatics, pages 291–298, Vogt-Kölln Str. 30, D-22527 Hamburg, Germany, June 2006. University of Hamburg, Department of Informatics.
4. 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.

5. Lawrence Cabac, Nicolas Knaak, Daniel Moldt, and Heiko Rölke. Analysis of multi-agent interactions with process mining techniques. In *Multiagent System Technologies. 4th German Conference, MATES 2006 Erfurt, Germany. Proceedings*, volume 4196 of *Lecture Notes in Computer Science*, pages 12–23, Berlin, Heidelberg, New York, 2006. Springer.
6. Lawrence Cabac and Daniel Moldt. Formal semantics for AUML agent interaction protocol diagrams. In *Agent-Oriented Software Engineering V: 5th International Workshop, AOSE 2004, New York, NY, USA, July 19, 2004. Revised Selected Papers*, volume 3382 of *Lecture Notes in Computer Science*, pages 47–61. Springer, January 2005.
7. Michael Duvigneau, Daniel Moldt, and Heiko Rölke. Concurrent architecture for a multi-agent platform. In Fausto Giunchiglia, James Odell, and Gerhard Weiß, editors, *Agent-Oriented Software Engineering III. Third International Workshop, Agent-oriented Software Engineering (AOSE) 2002, Bologna, Italy, July 2002. Revised Papers and Invited Contributions*, volume 2585 of *Lecture Notes in Computer Science*, pages 59–72, Berlin, Heidelberg, New York, 2003. Springer.
8. Foundation for Intelligent Physical Agents (FIPA) – homepage. `http://www.fipa.org/`. Foundation for Intelligent Physical Agents.
9. The Sniffer for JADE. Online documentation. `http://jade.cselt.it/doc/tools/sniffer/index.html`, January 2008.
10. Nicholas Robert Jennings. On agent-based software engineering. *Artificial Intelligence*, 117(2):277–296, 2000.
11. Michael Köhler, Daniel Moldt, and Heiko Rölke. Modelling the structure and behaviour of Petri net agents. In J.M. Colom and M. Koutny, editors, *Proceedings of the 22nd Conference on Application and Theory of Petri Nets 2001*, volume 2075 of *Lecture Notes in Computer Science*, pages 224–241. Springer, 2001.
12. Olaf Kummer. Introduction to Petri nets and reference nets. *Sozionik Aktuell*, 1:1–9, 2001. ISSN 1617-2477.
13. Olaf Kummer, Frank Wienberg, and Michael Duvigneau. Renew – the Reference Net Workshop. Available at: `http://www.renew.de/`, May 2006. Release 2.1.
14. Olaf Kummer, Frank Wienberg, Michael Duvigneau, Jörn Schumacher, Michael Köhler, Daniel Moldt, Heiko Rölke, and Rüdiger Valk. An extensible editor and simulation engine for Petri nets: Renew. In Jordi Cortadella and Wolfgang Reisig, editors, *Applications and Theory of Petri Nets 2004. 25th International Conference, ICATPN 2004, Bologna, Italy, June 2004. Proceedings*, volume 3099 of *Lecture Notes in Computer Science*, pages 484–493, Berlin, Heidelberg, New York, June 2004. Springer.
15. MadKit. `http://www.madkit.org`, January 2008.
16. R.S. Mans, Wil M.P. van der Aalst, P.J.M. Bakker, A.J. Moleman, K.B. Lassen, and Jens Bæk Jørgensen. From requirements via colored workflow nets to an implementation in several workflow systems. In *Proceedings of the International Workshop on Coloured Petri Nets (CPN 2007)*. Computer Science Department, Aarhus University, October 2007.
17. 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.
18. Heiko Rölke. *Modellierung von Agenten und Multiagentensystemen – Grundlagen und Anwendungen*, volume 2 of *Agent Technology – Theory and Applications*. Logos Verlag, Berlin, 2004. (German).
19. Marc H. Van Liedekerke and Nicholas M. Avouris. Debugging multi-agent systems. *Information and Software Technology*, 37:103–112, 1995.