A Proposal for Multi-Agent System based Modeling and Validation of Self-Organization

Unai Arronategui
University of Zaragoza, Centro Politecnico Superior,
C/ Maria de Luna, 1, 50018 Zaragoza
Email: unai@unizar.es

Daniel Moldt
University of Hamburg, Department of Computer Science,
Vogt-Kölln-Str. 30, D-22527 Hamburg
Email: moldt@informatik.uni-hamburg.de

***Abstract*— Dynamic self-organization is a basic feature to *Autonomic Computing Systems* (ACS). But its modeling and validation, being important issues, remain complex. Here, the conceptual integration of multi-agent systems and high-level Petri Nets can help with their powerful concepts and tools.**

We propose an homogeneous modeling technique integrating three concepts: agents, environments and Nets within Nets which are supported by the RENEW tool, allowing direct execution of the model.

This paper gives some insight in the proposed approach. The dynamic self-integration of a *service of code mobility* in an operating system is used as an illustrating example.

Keywords: IDE, high-level Petri nets, nets within nets, patterns, Reference nets, Renew, workflow, workflow patterns

I. INTRODUCTION

A main goal in Autonomic Computing Systems (ACS) is transforming current and future computer systems in gradually more and more self-organized dynamic systems. This goal has been defined to solve the steadyly rising complexity in the design, integration and management of networked computer systems [6]. A more self-organized system means, on the one hand, system administrators can focus on higher levels of system management tasks and, on the other hand, they will be able to deal with future pervasive computer systems.

In this paper, we explore modeling and validation issues in ACS.

At the operating system level work has been done by reflective approaches like in Apertos [17] for self-organization through reflection. An architecture is provided to deal with runtime modification of operating system functionality. In this case a reflective object-oriented modeling approach has been taken.

Validation of this kind of systems has been neglected in the past. Thus, there is a need to validate a new state reached after self-modification of a system has occurred. Also, in object-oriented the modeling of concurrent and distributed activities which are required in operating systems is not very well integrated.

We propose to solve these obstacles by modeling an operating system architecture with the integration of three powerful concepts: agents, environments and Nets within Nets. A multi-agent approach seems appropriate for the concurrent and distributed nature of operating system services. A reflective approach has been proposed in [4]. This already has references to the advantages of agent systems. In addition we propose to use the Nets within Nets paradigm [16]. It allows both, modeling the key concepts and directly representing recursion and reflection. The main problem is the missing restrictions of such a powerful technique. Therefore, agents and Multi-agent systems (MAS) are used to structure such models. Here we rely on MULAN [7], our architectural reference framework for MAS.

In the next section we will briefly introduce the Nets within Nets paradigm, followed by the concept of multi-agent systems. Since self-organization is a key concept, it is presented in a separate section. The last section contains conclusions on the results.

II. NETS-WITHIN-NETS

Autonomic Computing Systems can be viewed as Discrete Event Systems (DES). This kind of systems can be modeled with Petri Nets [14] that provide an intuitive graphical representation and a formal semantics of concurrent distributed processes. In this formal representation, complex and concurrent systems execution can be validated and system properties can be verified (mutual exclusion, deadlocks, livelocks, liveness, boundness of resources, etc).

A Petri Net is composed of places, transitions and arcs. Places represent resources that can be available or not, or conditions that can be fulfilled. Places are denoted in diagrams as circles or ellipses. Transitions are the active part of a net. Transitions are depicted as rectangles or squares and they connect different places. A transition that fires (or occurs) removes resources or conditions (for short: tokens) from places and inserts them into other places. This is determined by arcs that are directed from places (input places) to transitions and from transitions to places (output places).

The paradigm of *Nets within Nets* [16] belongs to the family of high-level Petri Nets [5]. So, it provides the intuitive graphical representation and formal semantics found in Petri Nets formalism. In this paradigm, the tokens of a Petri Net can again be Petri Nets. In this way, hierarchical and/or recursive structures and systems can be modeled in an elegant way.

Reference nets [9] are an implementation of some aspects of Nets within Nets. With this kind of nets a referential semantics are assumed, tokens in one net are the references to other net instances. As for other Petri Nets formalisms the tool RENEW

0-7803-8513-6/04/$20.00 ©2004 IEEE

[10], an integrated development environment and simulator for references nets, is provided. The benefit of this feature for the RENEW tool is that it is modular and extensible. In this context, Reference net models are executable which allows the validation of the modeled systems.

The property of reflection could be expressed in the fact that nets could be created, modified or destroyed, if they are expressed as tokens running inside another net. This structure can be layered, as required.

III. CONCEPTS FOR MULTI-AGENT SYSTEM MODELS

Autonomic computing systems can be viewed as a set of interacting autonomic agents. Autonomy, proactivity and goal-oriented behaviour in an open environment are basic features of these agents. Moreover, concurrency and distribution are inherent features in multi-agent systems modeling.

In the context of open systems, the environment is a basic abstraction that denotes the outside world. From the view-point of the model the environment is everything that does not belong to the modeled system. Inside the model, the interactions with the environment are represented with basic general properties and input/output functions that are needed as a part of the modeled system. In some cases, the environment can be represented as a special agent whose basic interactions and properties are modeled only.

We propose a modeling paradigm based on a variant of the MULAN [7] multi-agent system architecture. MULAN is built on *Nets within Nets*, which is used to describe hierarchies in an agent system. Moreover, MULAN is implemented in the RENEW tool. Our proposed modeling takes advantage of these features, since it allows for the validation of our models.

MULAN has a hierarchical structure with four levels of abstraction: multi-agent system, agent platform, agent and protocol. In this contribution we make an extension to our successful MAS reference architecture by adding an environment. Each level covers some relevant properties of a MAS or some specific behavior. The model for this is a reference net instance which has a reference to another reference net instance, as described above.

The environment is the first bottom level in an open system. It is usually *not modeled explicitly*, so isn't it here. In Petri nets, it can be modeled as transitions that have only output places or only input places if there are no further assumptions about the behavior of the environment. Any kind of properties and behavior can of course be added if this is appropriate for the model. The general concept of an environment is applied on all levels. The following nesting of the different layers can always be viewed as a specialization of the relation of an agent in its environment. In terms of the modeling technique: an object net that lies within a system net. The main purpose of the different layers is to illustrate explicitly different aspects of multi-agent systems. The possibility of combining and merging the different properties and behaviors is discussed in the following.

The second level describes a multi-agent system, a Petri net whose places contain agent platforms as tokens (see the top left part of Figure 1). The transitions between places describe communication or mobility channels. This level models the infrastructure upon which the modeled system is modeled.

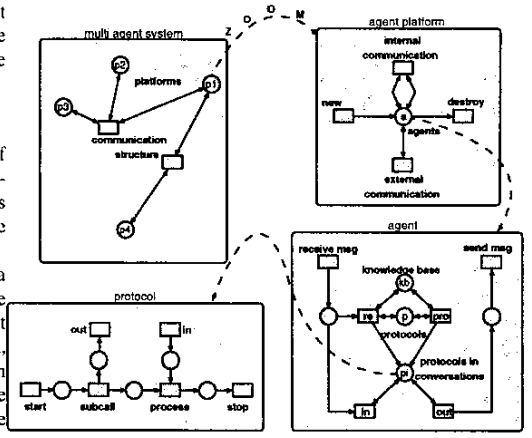

Fig. 1. Multi agent systems as Nets within Nets; taken from [7]

Agent platform structure forms the third level of abstraction (see the top right part of Figure 1). It defines the net within agents run as tokens. We have places representing states of (running) agents. Also, transitions dealing with reflective actions upon agents. They could be agent creation, agent destruction, intra-platform communication between agents, inter-platform communication between agents running on different platforms, send and receive agent (for mobility), etc. The creation transition is synchronized e.g. with the environment. The interpretation of a platform can be a location (physical or conceptual). The transportation or communication channels on the system levels are synchronized with the external actions of a platform. Internal actions of the platform can be synchronized with several local agents.

The fourth level of abstraction is built with an agent net model, which are tokens at the platform level (see the bottom right part of Figure 1). Interactions between agents are provided by message passing, with an incoming transition and an outgoing transition. This ensures an active autonomy of the agent which has to explicitly execute such a communication. A place stores the knowledge base where the persistent states of the agent are defined and changed. The potential behaviour of an agent is described in terms of protocols which are also stored persistently. Instantiated protocols are part of conversations, which manage interaction, and are developed in the conversations place. A reactive and a proactive transition represent the possibility to create or start conversations inside the agent. The reactive transition creates or starts protocols in response to the messages received from outside. The proactive transition runs protocols based on internal objectives provided

by the knowledge base place. Active protocols (=conversations) are again conceptually modeled by separate reference net instances.

Finally, protocols are basic Petri Nets, modeling the ongoing conversation of an agent (see the bottom left part of Figure 1. Usually this structure is very much like a workflow, since it is described by an explicit start, middle and end part. The middle part usually contains some action synchronized with the agent to send or receive information. However, protocols can have an arbitrary net structure if necessary.

Reflection is obtained through each level in the adjacent layer. The multi-agent system layer gives the reflective level for agent platforms. Agent platforms can represent the reflective level for agents. But also, at the agent level, we can use an organization-based reflection [4] through system level "agentification"and agent representatives for delegation of tasks. This latter form of reflection, due to its relative simplicity, is interesting in self-organization as we explain in the next sections.

IV. Modeling Self-organization

The following sections explain the proposed modeling approach. This is done using an illustrating example, operating system modelling. More explicitly, this is developed in the form of the self integration of a service of code mobility into a flexible operating system.

This will be developed, in the next subsections, with this methodology :

- Describe a basic model of a computing system with a special focus on the operating system layer.
- Build a general model of self-organization in operating system structured with a μ-kernel design.
- Explain the validation approach with the tools that allow so much the modeling as the execution of the system.

A. A Basic Model of a Computing System

Here, the multi-agent system level abstraction represents a set of a finite number of networked computers. In this case, the environment is everything not modeled at any level, and corresponds to elements inside and around the computer system: users, disk data, detailed hardware elements, etc.

The upper levels model the operating system architecture.

If we want to deal with self-organization, second generation μ-kernel designs, like L4 [12] and Exokernel [3], show real possibilities [11] in flexible and efficient execution. Flexibility is aimed to tailor most system services (memory manager, scheduler, device drivers, etc.) to applications directly in the user space. Efficiency is obtained giving minimal but very efficient primitive services to system and application designers. For instance, L4 works only with the concepts of threads, IPC (InterProcess Communication) and addresses spaces implemented in very few and fast primitives. In this architecture, system services can be reconfigured on the fly in a running system. This functionality opens doors to self-organization at different levels in computer systems.

With this kind of architecture in mind, the μ-kernel is modeled as an agent platform in our approach. Agents represent all user-level programs: system services and applications.

On the one hand, the agent platform should be enhanced to represent, at least, two places. One contains running agents and the other stopped agents waiting for a processor to be available. This would be the simplest model, sufficient to represent a very simple μ-kernel. Still, we do not deal, at this level, with different states related to scheduling policies. That is left to the scheduler, modeled like an "agentified"system service running on the agent platform.

On the other hand, the "agentified"system services (scheduler, memory manager, security manager, file system manager) have a special relationship with application agents and the agent platform (the net supporting the agents). System agents control what application agents can do and what resources they can get. Application agents can ask for services directly to each system service agent or, easier for them, they can ask for all operating system services to a system manager agent. This agent knows about all the other system agents competences, so it can delegate each service demand to the appropriate system service agent.

Also, more generally, agents can be grouped by affinity and have representatives to interact with other agents. This way, interaction complexity can be better managed.

B. Modeling Self-organization of Services in an Operating System

So far, a traditional computing system model has been described.

We can think of two ways to model self-organization. On the first one, we can model each system service agent with its own self-managing behaviour defined in its protocols. On the second one, the system manager agent described in the previous subsection could enhance its competences to deal with self-organization of its group, the system agents group. As said before, the latter can be a more manageable approach, albeit, the first one can have better resilience features due to control distribution.

A self-organization feature is the capacity to upgrade or enhance its competences in an autonomic way. An application agent asks for a task execution not available in the corresponding system service agent. In current systems, this event leads to an error. However, the autonomic behaviour of system agents looks for the existence of this task protocol in secure, well known and regularly updated repositories. If there exists a protocol for this new task, the system service agent loads a new protocol in its own protocol place. Next, this system service executes the required task. This process can be kept transparent. The application service has only an additional delay in the requested behaviour execution.

Also, protocol upgrading can be planned progressively when repositories get enhanced with new protocols or new versions of existing protocols. The autonomic computing system plans when and where upgrades should be done automatically.

Moreover, this can be also designed in a more crude context. E.g., an application agent asks for a task which can't be served by any of the system service agents and does not correspond to any system service agent domain. Thus, in some cases, the operating system needs to get new system agents for new services being provided. That means a more complex organization behaviour. The operating system manager agent fetches the new system agent. Other system agents are asked for an upgrade of protocols needed to interact with the new system agent. And finally, the integration ends with the new system agent running smoothly and providing the service required by the application agent.

In this last case, a system manager agent carries out the tasks involved with a new system service agent integration. For the sake of simplicity, a first version of the self-organization architecture is built into the system manager agent. The other system agents only have the capacity to upgrade or get new protocols of behaviour. This is a group-based self-organization where a representative (the system manager agent) deals with self-management tasks for the group. The example developed in the next subsection is based on this design.

Self Integration of a service of Code Mobility

An illustrating example can be provided with the integration of a strong code mobility service in a running operating system. This mobility service can be requested, for instance, from a mobile agent application or, also, it can be demanded by a new computer node in the network that asks for help to reduce activity load by means of a load balancing service agent. (For the modeling of mobility using nets within nets see [8]).

The integration of this new service means that, at least, the scheduler agent and the memory manager agent needs to be upgraded. Both need, in this new context, to provide new competences. The scheduler must offer a new migration state in scheduling. The memory manager must give complete information of the running state of processes (variables, stack and registers) found in the migration scheduling state.

Once upgraded, the migration service interacts with the scheduler to get processes in the migration scheduling state and with the memory manager to obtain their running state. Then, the state and code of the migrating agent is serialized and sent to the targeting computer node. When the migration system agent receives confirmation of successful execution of the migrated agent in the remote node, it can ask for the releasing of local resources have been used by the migrated agent.

From this point, at the same time that a new service has been executed transparently to the application, the system has been upgraded for latter uses. Unless the system manager has been told, by the human system administrator or a higher level computer manager agent, to revert the system configuration always to the same state. This kind of on-demand services could be individually provided, for instance, in little memory footprint computing platforms.

How to Model such Systems based on MULAN

What is new with respect to traditional systems modeling? The models proposed here can be completely described within the RENEW-tool. The special structure of the Petri net models is supported by a technical infrastructure called CAPA [2]. It allows for a FIPA-compliant implementation and therefore an easy interconnection over distributed systems based on a standard communication protocol. Our proprietary framework is therefore open to other standardized frameworks and implementations (see e.g. the integration into the Agentcities context [15] and [1]).

At the same time, agents bring a new and interesting modeling concept into the application area. The way systems can be structured and organized can be changed. MAS have their strength especially in those areas were the demands are high: flexibility, robustness, self-adaptation, etc. However, if agents are only considered to be special flexible components, then these advantages cannot be used completely. For the complex handling of such systems there is no complete theory. The interaction of such MAS requires techniques able to handle concepts of social interactions. Some contributions have been developed during the last years at Hamburg. The MULAN-architecture has been extended on the conceptual level by SONAR-agents. Each SONAR-agent is implemented as a MULAN-agent. The technical implementation is therefore homogeneous. However, the SONAR-agent can contain arbitrarily nested SONAR-agents. These cover important sociologically relevant features like system structures, processes and actors. These are used to reflect the external relations of an agent which contains the SONAR-agent. This holds for all kind of agents. All this allows for an efficient implementation of the overall structure of the system. (Some details can be found in chapters 11-12 in [13]).

As explained above, MAS can be viewed from different angles, which we called environment, system, platform, agent and protocol. Each view only emphasizes some important features. On the application level several views can be integrated on one implementation level. This means that the platform can also be viewed as an agent. Usually this is the case since the platform can be addressed and treated as being an agent when seen from the upper (e.g. the system) level. Furthermore, as mentioned above, the agents on a platform can again contain agents, where the agent represents a platform. This allows for an arbitrary nesting of agents that can aggregate their necessary features or services by hosting other agents and serving as a platform for them and offering services to the environment at the same time (with any kind of modifications).

C. Validation

Validation becomes possible since the RENEW-tool allows to implement directly, the models built according to the proposed approach. Each reference net can be executed directly. Java code can be used for inscriptions. External Java code can be integrated, based on the fact that Java objects and reference net instances can be arbitrarily mixed in an efficient way, even

in a concurrent and distributed fashion (we don't explain here the technical details).

Since the sophisticated models can cover, easily and intuitively, important features, the execution of the models allows to directly validate the specification, which is at the same time the implementation of the prototypes. To make really efficient systems with time-critical components like an operating system a reimplementation is of course necessary. However, this can be related to a thoroughly defined and investigated prototype. The exploration of this modeled prototype can be of invaluable help to better design and guide the effective implementation of the modeled system. For instance, exploration can be made in different designs. Also, tuning of the modeled system can be made before the implementation.

To what extend the advanced and complex concept of a MAS can be applied in this area, still has to be checked. At least the overall structure can be enhanced considerably.

V. Conclusion

Modeling and validation are important issues in Autonomic Computing. Some important features have been presented by means of the sketched example, the modeling of an operating system. In the area of operating systems, efficiency/real-time aspects are important, which prohibited to apply several advanced features. However, very similar concepts have to be used when working on the modeling and validation of different aspects of ACS. Here however, the goal is often to understand these systems. This allows to directly apply the approach as proposed here. What might be even more interesting is that a smooth integration of ACS and MAS can be reached due to the concepts elaborated in the area of hybrid systems (having humans and agents at the same conceptual level). Again we have to confess that here the HCI (Human-Computer-Interface) community is still working hard, however, the goals of many developers can already be seen.

The concept of environment for agents is important. There is a need to find the system borders and the respective interfaces. Since it can be applied recursively, a homogeneous structure can be used on the technical level. At the same time conceptual views can be emphasized as they become necessary from the modeler point of view. The underlying paradigm of *Nets within Nets*, with its instantiation by reference nets and the related tool set RENEW, is very powerful and can be used to describe all relevant features. The restriction to agents and MAS is necessary since otherwise the overview of models quickly becomes impossible, even for experienced developers. In our practical implementations we also apply (A)UML, Java and many other traditional software engineering techniques. At the same time, we have made very promising experiments with the inclusion of techniques coming from Artificial Intelligence (AI) and especially Distributed AI (DAI), like Prolog or BDI-Architectures.

In the near future we will work on some further case studies to experiment with the applicability of the approach. The area of flexible manufacturing systems has been used and will be used. Also we will investigate possibilities of applying formal results from the Petri nets research community.

Acknowledgment

The authors would like to thank the colleagues in our respective departments for valuable discussions and a first reviewing help, specially Jose Manuel Colom and Joaquin Ezpeleta.

This work has been supported by the German-Spanish Integrated Action HA2000-0047, Deutscher Akademischer Austauschdienst (DAAD) and the Spanish CICYT-FEDER TIC2001-1819.

References

[1] Michael Duvigneau, Michael Köhler, Daniel Moldt, Christine Reese, and Heiko Rölke, *Agent-based settler game*, In Proceedings of *Agentcities Agent Technology Competition, Barcelona, Spain*, 2003.

[2] Michael Duvigneau, Daniel Moldt, and Heiko Rölke, *Concurrent architecture for a multi-agent platform*, In Fausto Giunchiglia, James Odell, and Gerhard Weiß (eds.): *Third International Workshop, AOSE 2002, Bologna, Italy, July 15, 2002*, volume 2585 of *LNCS*, Springer-Verlag, Berlin Heidelberg New York, 2003.

[3] D. R. Engler, M. F. Kaashoek and J O'Toole Jr., *Exokernel: An Operating System Architecture for Application-Level Resource Management*, 15th SOSP'95 Proceedings, ACM, Dec. 1995, pp. 251-266.

[4] J. Ferber, O. Gutknecht, *A Meta-Model for the Analysis and Design of Organizations in Multi-Agent Systems*, ICMAS'98, Jul. 1998.

[5] K. Jensen and G. Rozenberg (eds.), *High-level Petri Nets – Theory and Application*. Springer-Verlag, Berlin Heidelberg, 1991.

[6] J. O. Kephart and D. M. Chess, *The Vision of Autonomic Computing*, Computer, Jan. 2003, pp. 41-50.

[7] Michael Köhler, Daniel Moldt, and Heiko Rölke, *Modelling the structure and behaviour of Petri net agents*, Application and Theory of Petri Nets, volume 2075 of LNCS, Springer Verlag, 2001, pp. 224-241.

[8] Michael Köhler, Daniel Moldt, and Heiko Rölke, *Modelling mobility and mobile agents using nets within nets*, In Wil van der Aalst and Eike Best (eds.), *Proc. of 24nd International Conference on Application and Theory of Petri Nets 2003 (ICATPN 2003), Eindhoven, NL*, Berlin Heidelberg New York, 2003, volume 2679 of LNCS, Springer-Verlag.

[9] Olaf Kummer, *Referenznetze*, Logos-Verlag, Berlin, 2002.

[10] Olaf Kummer and Frank Wienberg, *Reference net workshop (Renew)*, University of Hamburg, http://www.renew.de, 1998.

[11] L4KaTeam, *L4 X.2 Reference Manual*, Department of Computer Science, University of Karlsruhe, http://l4ka.org/documentation/l4-x2.pdf, Apr. 2003.

[12] J. Liedtke, *The performance of μ-Kernel-based Systems*, 16th SOSP'97 Proceedings, ACM, Oct. 1997.

[13] R. v. Lüde, D. Moldt, R. Valk, M. Köhler, R. Langer, H. Rölke, and D. Spresny, *Sozionik: Modellierung soziologischer Theorie*, Wirtschaft – Arbeit – Technik, Lit-Verlag, Münster, 2003.

[14] T. Murata, *Petri Nets: Properties, Analysis and Applications*, Proceedings of the IEEE, vol. 77, n.4, April 1989.

[15] Christine Reese, *Multiagentensysteme: Anbindung der petrinetzbasierten Plattform CAPA an das internationale Netzwerk Agentcities*, diploma thesis, University of Hamburg, Department for Computer Science, Vogt-Kölln Str. 30, 22527 Hamburg, Germany, 2003.

[16] Rüdiger Valk, *Petri nets as token objects: An introduction to elementary object nets*, In Jörg Desel and Manuel Silva (eds.), *Application and Theory of Petri Nets*, volume 1420 of *LNCS*, Springer-Verlag, Berlin Heidelberg New York, pp. 1–25, June 1998.

[17] Y. Yokote, *The Apertos Reflective Operating System*, OOPSLA'92 Proceedings, ACM, Oct. 1992, pp. 414-434.