Modelling Mobility and Mobile Agents using Nets within Nets

Michael Köhler and Heiko Rölke

Universität Hamburg, Fachbereich Informatik
Vogt-Kölln-Str. 30, D-22527 Hamburg
roelke@informatik.uni-hamburg.de

Abstract. Mobility creates a new challenge for dynamic systems in all stages of modelling, execution, and verification.
In this work we present an application area of the paradigm of "nets within nets". Nets within nets are well suited to express the dynamics of open, mobile systems. The advantages of Petri nets – intuitive graphical representation and formal semantics – are retained and supplemented with a uniform way to model mobility and mobile (agent) systems.
First the modelling of mobility is introduced in general, the results are carried forward to model mobility in the area of agent systems. The practicality of the approach is shown in a second step by modelling a small case study implementing a household robot system.

Keywords: agent, mobile agent system, mobility, nets within nets, Petri nets, Mulan, Renew

1 Introduction

The general context of this paper is mobility in open multi agent systems. The main question is how to model mobility in an elegant and intuitive manner without losing formal accuracy. The modelling language should feature a graphical representation to be used in a software engineering process. The modelling paradigm should be capable of expressing the different kinds of agent mobility. The models should build upon a formalism that has a formal semantics to support verification and execution. The (direct) execution of the models prohibits errors of manual translation from models e.g. to program code. Executable models support the validation process.

Here, we present a proposal, how the paradigm of "nets within nets" can be used to describe mobility. The paper consists of two parts: First it is shown how "mobility" in general can be expressed (environment, moved entity, different types of movement). In the second part these results are used to support the engineering of mobile agent systems.

Most of the work on mobility is based on calculi. Mobility calculi like in [CGG99], [VC98] or in [MPW92] formalise mobility without having a visual representation, thus being unsuited for the software-technical modelling of mobile entities. Mobile Petri nets [AB96] provide a description of mobility by embedding

the π-calculus into the formalism of ordinary Petri nets. Since the formalism is far from the general intuition of a token game, it cannot be used here for the proposed goal of supporting the specification process of complex software systems.

Structure of the paper In section 2 general statements on mobility are given: minimal preconditions for a formalism to express mobility are suggested as well as our needs for an intuitive modelling. A selection of formalisms is compared on this basis. We present a short introduction to the paradigm of nets within nets and show how this paradigm can be used to model mobility. In section 3 we give a description how this paradigm is used in the specific context of open multi agent systems. Section 4 presents a small case study of a mobile agent system using the modelling proposals. The paper closes with a conclusion and an outlook to further work.

2 Nets within Nets and Mobility

Figure 1 shows the mandatory elements of a system to become a *mobile* system. The overall system is divided into separate locations. At least two different locations are necessary. Locations host entities (some of) which are able (under certain circumstances) to undertake a movement from one location to another, which means that the environment of the entity changes.[1]

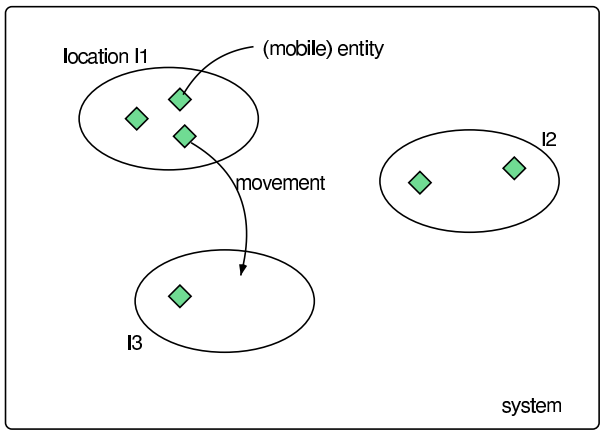

Fig. 1. Elements of mobile systems

[1] This changing may either be logical or physical. The modelling of real-world scenarios makes it necessary to cope with the additional problems of physical movements – which may be mapped to a logical changing of the environment.

An important point of mobility is the embeddedness of the mobile entity: each entity is embedded in a local environment (location) that assists the entity by offering some services and restricts it by declining others (or not having the potential to offer them). If all locations look the same to the mobile entity and no difference is made in local versus remote communication, movements are transparent to the mobile entity. This is an important feature for example in load balancing systems. The systems we are considering usually show several differences between the locations and hence are more interesting to model, since these differences cause the complexity of the systems.

Modelling opportunities To argue why we are not using a common mobility calculus, we take the Seal Calculus [VC98] as an example. In the Seal Calculus a process term P with locality is denoted as $E[P]$. Mobility is described by sending seals over channels. The term $\bar{x}^*\{y\}.P$ describes a process, which can receive a seal y over the channel $\bar{x}^*$ and then behaves like P. For receiving the term $x^*\{z\}.Q$ describes a process, which receives the seal z, substitutes this seal for every occurrence of z in Q and then behaves like this new Q.[2]

Mobility calculi like the Seal Calculus offer about the same modelling power while being (partly) easier to analyse than our extended Petri net formalism. We favour nets within nets for the description of mobility, since Petri net models are easier to read without losing the exactness. This is the main difference to models based upon calculi or logic, like HiMAT of Cremonini et al. [COZ99]. Nets within nets can be used for modelling purposes in our MULAN[3] architecture as well as for analysis purposes. Additionally, our formalisation allows the description of location and concurrency in an integrated way, which would not have been possible with simple Petri nets or mobility calculi alone.

Buchs and Hulaas [HB01] use an extended variant of high-level algebraic Petri nets to model mobile (agent) systems. The difference to this approach is the approach to mobility: Their system uses a π-calculus like style, e.g. mobility via passing of names. In our opinion it is more intuitive to remove the mobile entity from one location and let it enter a new one. This will be shown in subsection 2.2. Nevertheless, it is possible to simulate our approach with π-calculus like mobility and vice versa.

The use of widespread (graphical) modelling formalisms like UML is not possible, because all included types of diagrams (e.g. class diagrams, activity diagrams, state charts) are static and/or do not offer a notion of (dynamic) embeddedness. Additionally they lack a formal semantics. This is especially the case for the deployment diagram, that was originally used to model the *static* allocation of objects among different computers (clients and servers).

In the majority of cases mobile agents are directly implemented using a programming language like Java [Sun]. As an example we cite Uhrmacher et

[2] The Seal Calculus thus describes a *spontaneous move*. See Figure 4 in subsection 2.3.

[3] MULAN stands for MULti Agent Nets and is an approach to cover a complete agent framework (specification, design, implementation, and execution) on Reference nets and Java [Röl02].

al. [UTT00] (representative for other publications): The implementation is done using Java whilst the mobility aspect of the system is presented using simple graphics (without semantics). In contrast to this approach we try to unify illustration and implementation using the same (Petri net) model for both purposes.

Summing up these arguments we decided to use an extended Petri net formalism, that will now be introduced.

2.1 Nets within Nets

The paradigm of "nets within nets" due to Valk [Val98,Val00] is based on the former work on task-flow nets [Val87]. The paradigm formalises the aspect that tokens of a Petri net can also be nets. Taking this as a view point it is possible to model hierarchical structures in an elegant way.

We will now give a short introduction to the paradigm of Reference nets [Kum02,KW98], an implementation of certain aspects of nets within nets. It is assumed throughout this text that the reader is familiar with Petri nets in general as well as coloured Petri nets. Reisig [Rei85] gives a general introduction, Jensen [Jen92] describes coloured Petri nets.

A net is assembled from places and transitions. Places represent resources that can be available or not, or conditions that may be fulfilled. Places are depicted in diagrams as circles or ellipses. Transitions are the active part of a net. Transitions are denoted as rectangles or squares. 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 to transitions and from transitions to places.

Reference nets [Kum98] are so-called high-level Petri nets, a graphical notation that is especially well suited for the description and execution of complex, concurrent processes. As for other net formalisms there exist tools for the simulation of reference nets [KWD01]. Reference nets offer some extensions related to "ordinary" coloured Petri nets: net instances, nets as token objects, communication via synchronous channels, and different arc types. Beside this they are quite similar to coloured Petri nets as defined by Jensen. The differences are now shortly introduced.

Net instances Net instances are similar to objects of an object oriented programming language. They are instantiated copies of a template net like objects are instances of a class. Different instances of the same net can take different states at the same time and are independent from each other in all respects.

Nets as tokens Reference nets implement the "nets within nets" paradigm of Valk [Val98]. This paper follows his nomenclature and denominates the surrounding net *system net* and the token net *object net*. Certainly hierarchies of net within net relationships are permitted, so the denominators depend on the beholders viewpoint.

Synchronous channels A synchronous channel [CH94] permits a fusion of transitions (two at a time) for the duration of one occurrence. In reference nets (see [Kum98]) a channel is identified by its name and its arguments. Channels are directed, i.e. exactly one of the two fused transitions indicates the net instance in which the counterpart of the channel is located. The other transition can correspondingly be addressed from any net instance. The flow of information via a synchronous channel can take place bi-directional and is also possible within one net instance. It is possible to synchronise more than two transitions at a time by inscribing one transition with several synchronous channels.

Arc types In addition to the usual arc types reference nets offer *reservation arcs* and *test arcs*. Reservation arcs carry an arrow tip at both endings and reserve a token solely for one occurrence of a transition. They are a short hand notation for two opposite arcs with the same inscription connecting a place and a transition. Test arcs do not draw-off a token from a place allowing a token to be tested multiple times simultaneously, even by more than one transition (test on existence).

2.2 Modelling Mobility

The intuition of nets within nets is, that the token nets are "lying" as tokens in places just as one is used to have ordinary tokens. This is illustrated in the figures 2 and 3. When modelling more widespread nets a displaying as in the mentioned figures is not practical. Therefore the modelling tool Renew implements a kind of pointer concept: net tokens are references (hence the name) to nets each displayed in a window of their own.[4]

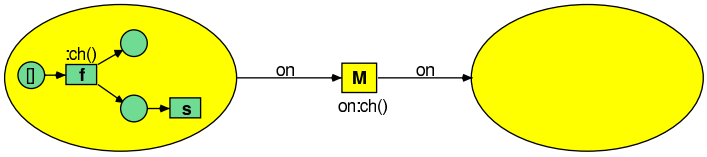

Fig. 2. Object net embedded in system net

To give an example we consider the situation where we have a two-level hierarchy. The net token is then called the "object net", the surrounding net is called the "system net". Figure 2 illustrates this situation: The object net in the left place of the system net can be bound to the arc inscription on. Doing so, transition M is activated with this possible binding. In addition M is inscribed

[4] There is another difference between the Reference nets of Renew and the intuitive modelling, namely the use of reference versus value semantics. This topic is covered by Valk [Val00].

with a synchronous channel (on:ch()). This inscription means that for the object net on to become an actual binding of transition M an adequate counterpart has to be found within on – an enabled transition inscribed with the channel :ch(). This precondition is fulfilled by transition f of the object net. So the synchronous firing of object and system net can take place and leads to the situation in Figure 3.

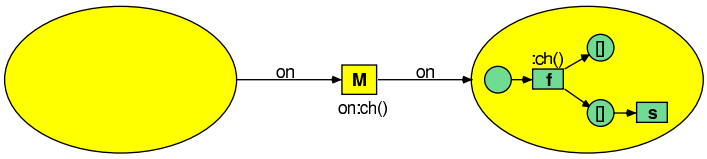

Fig. 3. Object and system net after firing

The object net is moved to the right place of the system net. Synchronously the marking of the object net changed so that another firing of transition f is not possible.

The example gives an idea how the interplay between object net and system net can be used to model mobile entities manoeuvring through a system net, where the system net offers or denies possibilities to move around while the mobile object net moves at the right time by activating a respective transition that is inscribed with the counterpart of the channel of the move transition of the system net.

Without the viewpoint of nets as tokens, the modeller would have to encode the agent somehow. This has the disadvantage, that the inner actions cannot be modelled directly, so, they have to be lifted up to the system net, which seems quite unnatural. By using nets within nets we can investigate the concurrency of the system and the agent in one model without losing the needed abstraction.

2.3 Types of Mobility

The interplay between object net and system net induces four possibilities for an object net to move or to be moved, respectively.

1. The object net is moved inside the system net, neither object nor system net controls the move. (Spontaneous Move)
2. The object net triggers the movement, the system net has no influence. (Subjective Move)
3. The system net forces the object net to move. (Transportation, Objective Move)
4. Both nets come to an agreement on the movement. (Consensual Move)

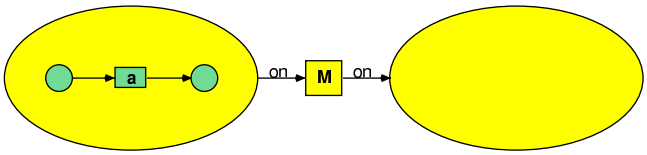

Fig. 4. Spontaneous Move

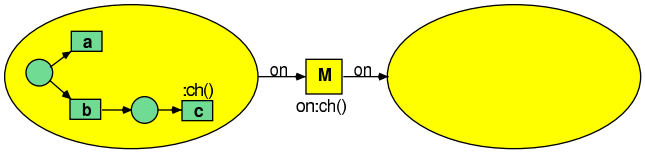

Fig. 5. Subjective Move: Object net triggers movement

If neither object net nor system net influence the movement it may happen *spontaneously*. This is the situation in Figure 4. There is no pre- or side condition for the movement.

One may argue that the second possibility does not exist, because the system net offers the ways for object nets to move from one location to another. So the system net "decides" which movements can be carried out and which can not. But in a *given* system net it is possible for an object net to control the movement as shown in Figure 5.

In the figure the only condition for the movement to be carried out is the synchronous channel (see transition M). The synchronous channel is activated if its counterpart inside the object net is also activated (that means, it is inscribed to an enabled transition). So the movement depends on the (firings of the) object net only. This movement is called *subjective* because the mobile entity itself is subject of the execution.

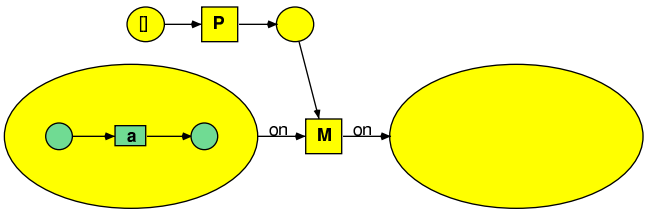

Fig. 6. Transportation: System net triggers movement

If Figure 4 is extended with some kind of condition for transition M, the system net may control the movement, the object net is *transported* from one location to another. Figure 6 shows only one possibility of how the system net may control the transition M, another one is for example a guard (see Figure 7).

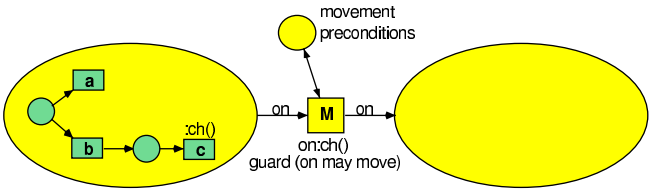

Fig. 7. Consensual Move

By combining Figure 5 and Figure 6 both object and system net influence transition M. For the transition to be enabled, they have to agree upon the movement. For this reason this kind of movement is called *consensual*. An example for such a move is shown in Figure 7.

The figure shows another way of modelling a (side) condition for transition M: a combination of a place holding movement conditions and an appropriate guard. The guard monitors the movement conditions, transition M is only enabled if the object net is allowed to move.

3 Agent Systems

In the following section we lift the general insights of how mobility can be modelled to a special form of multi agent systems. The modelling of a mobile agent (object net) moving through a "world" of several locations (system net) allows for an intuitive reproduction of real-world scenarios.

First the multi agent architecture MULAN is introduced, that also profits from the use of nets within nets.

3.1 Multi Agent Architecture Mulan

The multi agent system architecture MULAN [KMR01] is based on the "nets within nets" paradigm, which is used to describe the natural hierarchies in an agent system. MULAN is implemented in RENEW [KW98], the IDE (Integrated Development Environment) and simulator for reference nets. MULAN has the general structure as depicted in Figure 8: Each box describes one level of abstraction in terms of a system net. Each system net contains object nets, which structure is made visible by the ZOOM lines.[5] The figure shows a simplified

[5] This zooming into net tokens should not to be confused with place refining.

version of MULAN, since for example several inscriptions and all synchronous channels are omitted. Nevertheless this is an executable model.

The net in the upper left of Figure 8 describes an agent system, which places contain agent platforms as tokens. The transitions describe communication or mobility channels, which build up the infrastructure. The multi agent system net shown in the figure is just an illustrating example, the number of places and transitions or the interconnections have no further meaning.

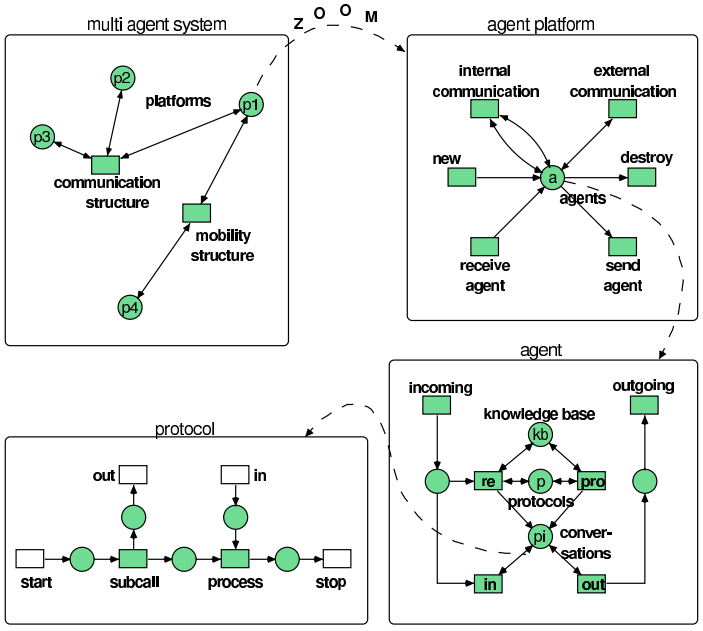

Fig. 8. Agent systems as nets within nets

By zooming into the platform token on place p1, the structure of a platform becomes visible, shown in the upper right box. The central place agents hosts all agents, which are currently on this platform. Each platform offers services to the agents, some of which are indicated in the figure.[6] Agents can be created (transition new) or destroyed (transition destroy). Agents can communicate by message exchange. Two agents of the same platform can communicate by the transition internal communication, which binds two agents, the sender and the receiver, to pass one message over a synchronous channel.[7] External communi-

[6] Note that only mandatory services are mentioned here. A typical platform will offer more and specialised services, for example implemented by special service agents.

[7] This is just a technical point, since via synchronous channels provided by RENEW asynchronous message exchange is implemented.

cation (external communication) only binds one agent, since the other agent is bound on a second platform somewhere else in the agent system. Also mobility facilities are provided on a platform: agents can leave the platform via the transition send agent or enter the platform via the transition receive agent.

A platform is therefore quite similar to a location in the general mobility scenario as it was introduced in the beginning of section 2.

Agents are also modelled in terms of nets. They are encapsulated, since the only way of interaction is by message passing. Agents can be intelligent, since they have access to a knowledge base. The behaviour of the agent is described in terms of protocols, which are again nets. Protocols are located as templates on the place protocols. Protocol templates can be instantiated, which happens for example if a message arrives. An instantiated protocol is part of a conversation and lies in the place conversations.

The detailed structure of protocols and their interaction have been addressed before in [KMR01], so we skip the details here and proceed with the modelling of agent migration.

3.2 Mobility as a system view

When modelling a complex system it is often undesirable to see the overall complexity at every stage of modelling and execution (simulation in our case). Therefore the notion of a system view is introduced. Several views on an agent system are possible, for example the history of message transfer (message process), the ongoing conversations and/or active protocols (actual dynamic state) or the distribution of agents among a system of several locations (platforms).

Using nets within nets as a modelling paradigm allows for the direct use of system models at execution time. This can be exploited as follows: The overall system is designed as a system net with places defining locations and transitions representing possible moves from one location to another. This is a direct transformation of the general mobility modelling ideas of section 2. The adaption to host platforms (that encapsulate the mobile agents) instead of agents directly is straightforward (adding one level of indirection) and can be omitted here.

Figure 9 shows an example of such a system net. Places are locations (rooms) in or in front of a house, transitions model possible movements between the rooms. The walls of the house are drawed in for illustrationary purposes only. This example is carried forward in the next section (4).

The interesting point is that it is possible to "decorate" the places in the system net with additional interesting features, for example a characteristic subset of the services of the hosted platform together with some of the pre- and postconditions of these services. The beholder of such an enhanced system net is provided with a complete view of important system activities without having to deal with the underlying complexity.[8]

[8] This is illustrated in the next section: The mobility system net is just a small part of the overall system consisting of several agents, platforms, technical substructure e.g. for the remote communication and so on. This is hidden from the user as long as the user does not wish to see the implementation details.

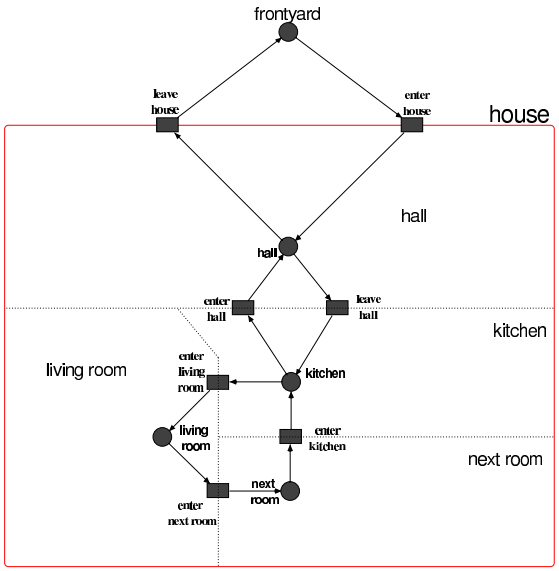

Fig. 9. Example system net

The difference between this proposal and a visualisation tool that shows some activities of e.g. a program running in background is twofold: First, using our proposal, the modelling process concludes with a running system model. A normal modelling process requires at least three stages to gain a comparable result: (a) model the system, (b) implement the model, and (c) write a visualisation for the program. Second, the visualisation of a system model at execution time is indeed the implementation of the system. This eliminates several potential sources of errors shifting from model to implementation to visualisation in an ordinary software design process.

4 Mobile Robot Case Study

To illustrate the modelling method introduced before, we introduce a small case study, a mobile household robot, unfortunately just a *software* bot.

4.1 Specification

The robot is internally implemented according to the MULAN architecture introduced in subsection 3.1. But that is not what the supervisor of the robot wants to know and see. What he requires is a simple view on his household, the robots location and state, and maybe some supplementary information. That is just what we provide with the extended system net view on the agent system.

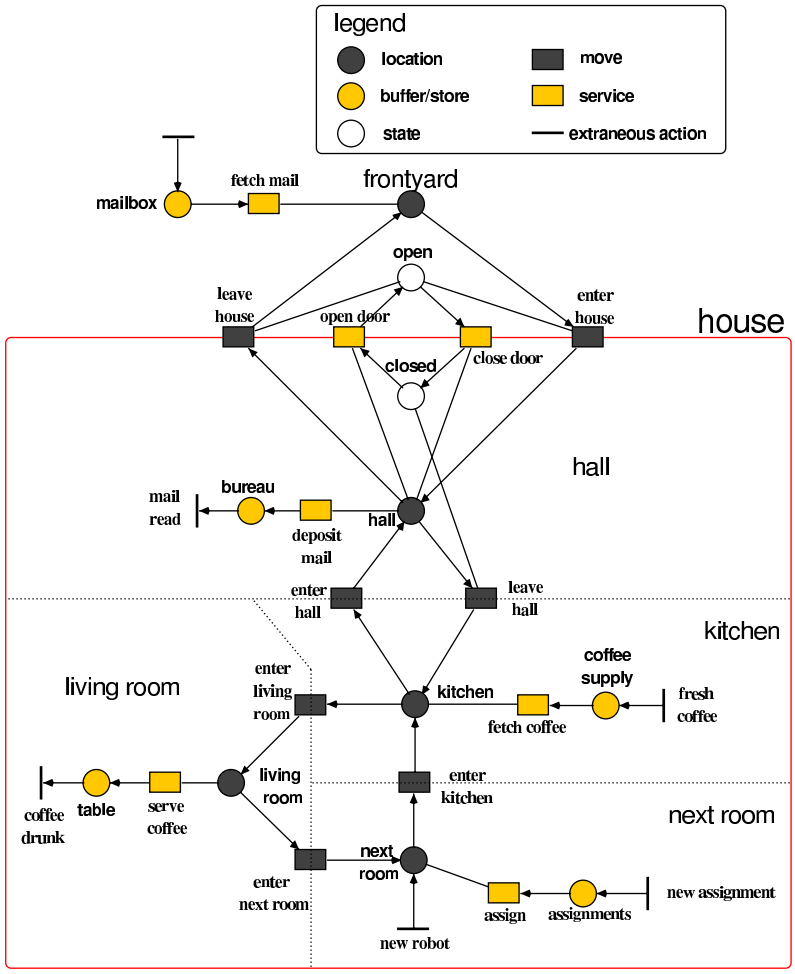

Fig. 10. Household System Net

The household is represented by the system net in Figure 10.[9] The household consists of several rooms (locations): hall, living room, kitchen, next room, and the front yard (dark places). Each room offers special services to the robot: it can fetch coffee in the kitchen, serve it in the living room, fetch mail in the front yard, open and close the door in the hall, and so on (light transitions). The possible movements from one location to another are displayed as dark transitions. Note that moving from room to room is not symmetric in this scenario. For example it is not possible to move directly from the kitchen to the next room. Service transitions are supplemented with additional information (service state/buffer, light places) showing for instance if new mail has arrived, coffee is available and so on. Extraneous actions not accessible for the robot are displayed as thin-lined transitions: arrival of new mail, new assignments for the robot etc.

The door of the house is used to show another possibility of viewing special parts of the system: the state of the door (open/closed) is directly modelled. This system state is not belonging to a single service (as for example the state of the mailbox), but is queried by a couple of service transitions including the movements into and out of the house.

This model of the household is filled with life by implementing an appropriate robot agent and defining the desired services for the platforms. The behaviour modelling for this kind of agents has been introduced elsewhere [KMR01].

Having defined the functionality of platforms and agents the parts are inserted in the household system net, which is a straightforward operation that claims for automation (tool support). After that the system is ready for execution.

4.2 Execution

Figure 11 shows an early state of a sample simulation of the agent system. For illustration purposes some nets that normally work "behind the scenes" are made visible in this screen shot. An ordinary simulation would just show the net house2 in the middle of the figure. It is outside the scope of this paper to explain the full functionality of the MULAN agent framework, so just the mobility overview net is regarded. [10]

Taking a closer view on the central household system net an additional feature becomes visible: Tokens can be visualised by an intuitive image. In the figure the incoming mail is visible in the upper left, and a fresh coffee on the lower right side. This is a one-to-one matching between token and image – whenever such a token is produced, moved, or deleted, the image follows. The case of the robot in the next room (lower middle) is a little bit more tricky (to implement), because what is really lying on the location place is a platform hosting the robot

[9] The use of colour greatly supports the differentiation of different types of places, transitions, or arcs. Unfortunately this is – even in the adapted form as in the figures – not so obvious in a black and white representation.

[10] Details of the MULAN framework are topic of an ongoing diploma thesis [Duv02] and a technical report [Röl02] as well as other papers, e.g. [DMR02]

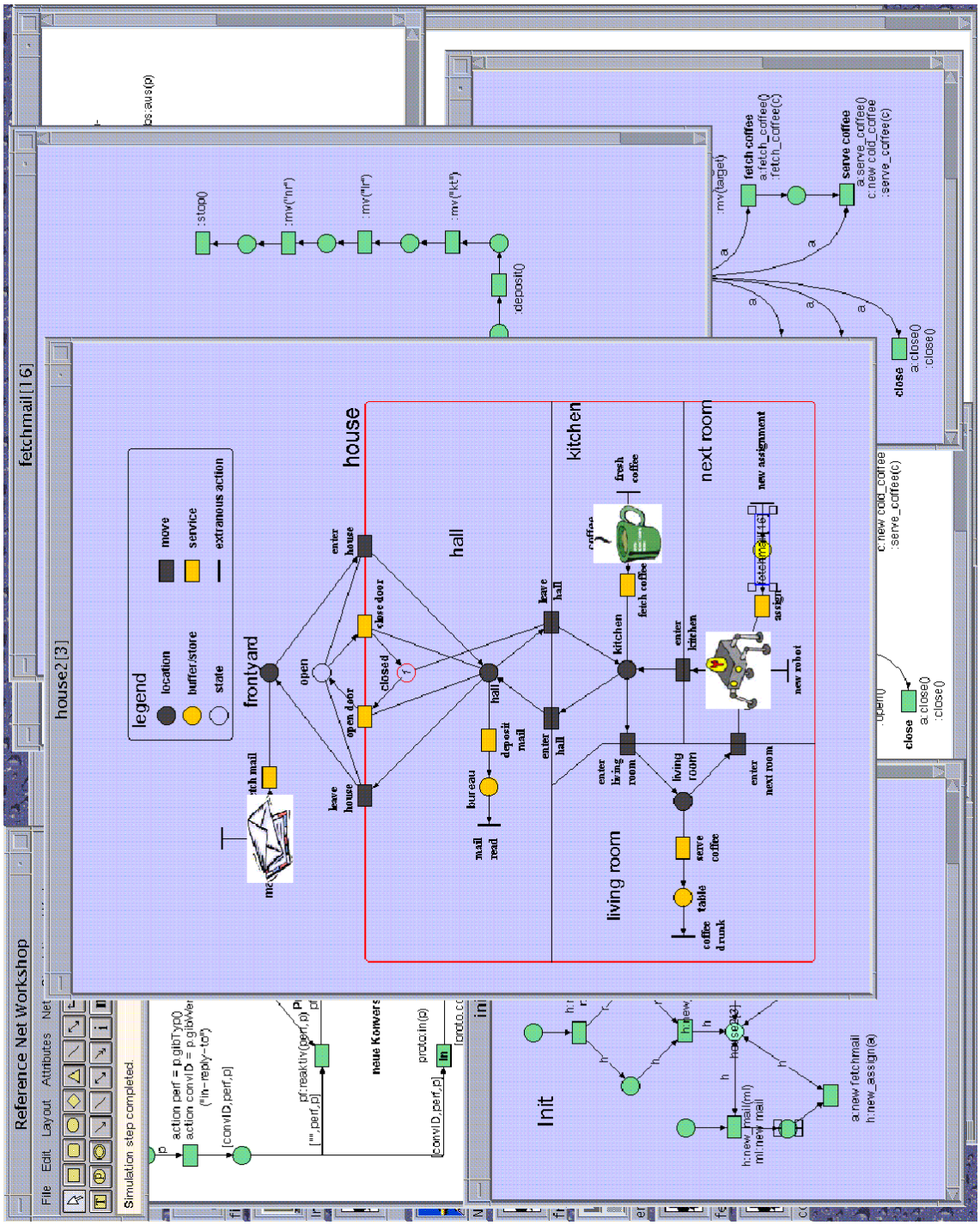

Fig. 11. System in Action

agent, and not the robot itself. What can be seen in the system net is actually a dynamic token figure of the platforms showing representatives of the agents hosted by the respective platform (empty platform means no image).

It is now easy to see the state of the overall system by looking at the system net. All the simplifications in presentation do not mean a loss of generality in the system's implementation. It is possible to take control of the whole agent system implementing the robot scenario by just double clicking on one of the tokens. The tool RENEW will then display the respective net (similar to those introduced in subsection 3.1), allowing for a complete inspection of the running system without having to interrupt.

In further execution steps the robot will receive one or more assignments, for example to serve fresh coffee in the living room. Its internal representation says that coffee is only available in the kitchen, so it moves there to look for the coffee service of the kitchen platform. In the scenario of Figure 11 this service is available, so the robot fetches the coffee and moves to the living room, where it serves the coffee. The robot ends the assignment by moving back to the next room and waiting for orders.

The behaviour modelling and planning the robot carries out to perform its tasks is outside the scope of this paper. The mobility system net helps greatly in validating a certain robot planning algorithm, for instance by unmasking bad habits of the robot like fetching the coffee, leaving the house for mail, returning to the living room later and serving cold coffee.

5 Conclusion and Outlook

In this article we have presented the paradigm of "nets within nets" for the modelling of mobility. Nets within nets are attractive, since the concept allows for an intuitive representation of mobile entities as well as an operational semantics which is implemented in the RENEW-simulator.

We have shown how this approach can help in modelling and executing mobile agent systems and underpinned our proposal with a small case study.

Besides the software-technical aspects, the exact semantics of nets within nets – in principal – allows for an analysis of mobile applications. This still needs more work, a starting point is [Köh02]. Formal treatment of mobile (agent) systems may prevent "classical" problems like deadlocks because of insufficient resources, but also helps in "modern" problems like security.

The use of an intuitive graphical formalism like Petri nets offers the room for a more widespread use as a modelling tool in agent oriented software engineering (AOSE) as opposed to text-based formalisms (mobile calculi), as the use of graphic modelling is an accepted technique in mainstream software engineering (see for example the Unified Modelling Language (UML) [Fow97]).

Additional formal results will be directly integrated in the design of the Petri net based multi agent system architecture MULAN.

References

[AB96] Andrea Asperti and Nadia Busi. Mobile Petri nets. Technical report, Department of computer science, University of Bologna, TR UBLCS-96-10, 1996.

[CGG99] Luca Cardelli, Andrew D. Gordon, and Giorgio Ghelli. Mobility types for mobile ambients. In *Proceedings of the ICALP'99*, volume 1644 of *LNCS*, pages 230–239. Springer-Verlag, 1999.

[CH94] Søren Christensen and Niels Damgaard Hansen. Coloured Petri nets extended with channels for synchronous communication. In Rober Valette, editor, *Application and Theory of Petri Nets 1994, Proc. of 15th Intern. Conf. Zaragoza, Spain, June 1994*, LNCS, pages 159–178, June 1994.

[COZ99] Marco Cremonini, Andrea Omicini, and Franco Zambonelli. Modelling network topology and mobile agent interaction: An integrated framework. In *Proceedings of the 1999 ACM Symposium on Applied Computing (SAC'99)*, 1999.

[DMR02] Michael Duvigneau, Daniel Moldt, and Heiko Rölke. Concurrent architecture for a multi-agent platform. In *Proceedings of the 2002 Workshop on Agent Oriented Software Engineering (AOSE'02)*, 2002.

[Duv02] Michael Duvigneau. A fipa-compliant multi agent platform. Master's thesis, Universiät Hamburg, 2002.

[Fow97] Martin Fowler. *Analysis patterns: reusable object models*. Addison-Wesley series in object-oriented software engineering. Addison-Wesley, 1997.

[HB01] Jarle Hulaas and Didier Buchs. An experiment with coordinated algebraic petri nets as formalism for modeling mobile agents. In *Workshop on Modelling of Objects, Components, and Agents (MOCA'01) / Daniel Moldt (Ed.)*, pages 73–84. DAIMI PB-553, Aarhus University, August 2001.

[Jen92] Kurt Jensen. *Coloured Petri nets, Basic Methods, Analysis Methods and Practical Use*, volume 1 of *EATCS monographs on theoretical computer science*. Springer-Verlag, 1992.

[KMR01] Michael Köhler, Daniel Moldt, and Heiko Rölke. Modeling the behaviour of Petri net agents. In J. M. Colom and M. Koutny, editors, *Proceedings of the 22st Conference on Application and Theory of Petri Nets*, volume 2075 of *LNCS*, pages 224–241. Springer-Verlag, June 2001.

[Köh02] Michael Köhler. Mobile object net systems: Petri nets as active tokens. Technical report, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln Str. 30, 22527 Hamburg, Germany, 2002.

[Kum98] Olaf Kummer. Simulating synchronous channels and net instances. In J. Desel, P. Kemper, E. Kindler, and A. Oberweis, editors, *Forschungsbericht Nr. 694: 5. Workshop Algorithmen und Werkzeuge für Petrinetze*, pages 73–78. Universität Dortmund, Fachbereich Informatik, 1998.

[Kum02] Olaf Kummer. *Referenznetze*. Dissertation, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln Str. 30, 22527 Hamburg, Deutschland, 2002.

[KW98] Olaf Kummer and Frank Wienberg. *Reference net workshop (Renew)*. Universität Hamburg, http://www.renew.de, 1998.

[KWD01] Olaf Kummer, Frank Wienberg, and Michael Duvigneau. *Renew - User Guide*. Universität Hamburg, Fachbereich Informatik, Vogt-Kölln Str. 30, 22527 Hamburg, Deutschland, 1.5 edition, May 2001.

[MPW92] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, parts 1-2. *Information and computation*, 100(1):1–77, 1992.

[Rei85] Wolfgang Reisig. *Petri Nets: An Introduction*. Springer-Verlag, Heidelberg, 1985.

[Röl02] Heiko Rölke. The Multi Agent Framework Mulan. Technical report, Universität Hamburg, 2002.

[Sun] Sunsoft. Java Online Reference Manual. `http://www.javasoft.com`.

[UTT00] Adelinde M. Uhrmacher, Petra Tyschler, and Dirk Tyschler. Modeling and simulation of mobile agents. *Elsevier, Artificial Intelligence?*, 2000.

[Val87] Rüdiger Valk. Modelling of task flow in systems of functional units. Technical Report FBI-HH-B-124/87, Universität Hamburg, 1987.

[Val98] Rüdiger Valk. Petri nets as token objects: An introduction to elementary object nets. In Jörg Desel and Manuel Silva, editors, *Application and Theory of Petri Nets*, volume 1420 of *LNCS*, pages 1–25, June 1998.

[Val00] Rüdiger Valk. Concurrency in communicating object Petri nets. In G. Agha, F. De Cindio, and G. Rozenberg, editors, *Concurrent Object-Oriented Programming and Petri Nets*, Lecture Notes in Computer Science, Berlin, 2000. Springer-Verlag.

[VC98] Jan Vitek and Giuseppe Castagna. Seal: A framework for secure mobile computations. In *ICCL Workshop: Internet Programming Languages*, pages 47–77, 1998.