Modelling the Structure and Behaviour of Petri Net Agents

Michael Köhler, Daniel Moldt, and Heiko Rölke

University of Hamburg – Computer Science Department
Theoretical Foundations of Computer Science
Vogt-Kölln-Straße 30 – 22527 Hamburg
{koehler,moldt,roelke}@informatik.uni-hamburg.de

Abstract. This work proposes a way to model the structure and behaviour of agents in terms of executable coloured Petri net protocols. Structure and behaviour are not all aspects of agent based computing: agents need a world to live in (mostly divided into platforms), they need a general structure (e.g. including a standard interface for communication) and their own special behaviour. Our approach tackles all three parts in terms of Petri nets. This paper skips the topic of agent platforms and handles the agent structure briefly to introduce a key concept of our work: the graphical modelling of the behaviour of autonomous and adaptive agents.
A special kind of coloured Petri nets is being used throughout the work: reference nets. Complex agent behaviour is achieved via dynamic composition of simpler sub-protocols, a task that reference nets are especially well suited for. The inherent concurrency of Petri nets is another point that makes it easy to model agents: multiple threads of control are (nearly) automatically implied in Petri nets.

Keywords: agent, behaviour, concurrency, modelling, multi agent system, nets within nets, Petri net, reference net, structure

1 Motivation

To date agents are generally programmed using high-level languages such as Java (namely in agent frameworks as Jackal [8]) or they are defined by simple scripts. A graphical modelling technique that captures all parts of agents and their systems – as UML[1] in the context of object-orientation – is neither proposed nor in general use.[2]

The proper treatment of encapsulation, structuring, and flexibility in the scope of modelling is at the same time a major challenge in software engineering as well as in theoretical computer science.

[1] UML stands for Unified Modeling Language. See for example [16].
[2] The authors are aware of the upcoming proposals that base on UML i.e. from Odell et al. [2,30] (AUML). To our opinion these proposals capture only parts of the agent modelling tasks and leave out important areas such as agent mobility.

J.-M. Colom and M. Koutny (Eds.): ICATPN 2001, LNCS 2075, pp. 224–241, 2001.

Our department has expertise in examining to what extent Petri nets are suitable in the mastering of these questions. Starting from fundamental Petri net concepts especially Petri nets as active tokens (see [36]) and the basic construct of object oriented Petri nets (see [27]) have been investigated. The latter have led to a first approach to agent oriented Petri nets (see [29]). A complete redesign of agent oriented Petri nets (see [32] for the motivation and starting point) is now partly presented in this work.[3] Our approach aims at modelling a complete multi agent system in terms of Petri nets. This work decomposes into three parts: The system (e.g. the set of platforms) that hosts the agents, the agent itself, and its behaviour. A complete presentation of all three parts is out of the scope of this document, as the title implies, only the modelling of the structure and behaviour of a single agent is focused in this contribution.

Agent orientation is marked by intelligence, autonomy and mobility [4]. Whilst mobility requires an interplay of one agent and the agent system intelligence and autonomy are to be found in the overall architecture of agents (autonomy) and in their behaviour (intelligence). Mobility raises some hard questions e.g. concerning safety and is therefore not handled in this paper. Nevertheless our approach is easily expandable to allow forms of mobility as described in [28].

Our work uses the formalism of reference nets as presented by Kummer [23]. Reference nets are based on the "nets within nets"-paradigm that generalises tokens to arbitrary data types and even nets. The general idea behind our work is that an agent controls (sub-)nets as tokens which implement special kinds of behaviour. To (re-)act, the agent simply has to select (and instantiate) such a net. Additional concepts are dynamic resolution and binding of these nets.

The remainder of the document is organised as follows: in the following section 2, the formalism of reference nets is briefly introduced. Section 3 gives an overview of how the "nets within nets" paradigm can be used to model agents, their behaviour, and their environment. Section 4 describes the structure of the agents whose behaviour is determined by Petri net protocols. The net protocols are introduced in section 5 on the basis of an example. Section 6 cites some related material while the outlook in the closing section 7 names further points that are yet being discussed or are out of the scope of this paper.

2 Reference nets

It is assumed throughout this text that the reader is familiar with Petri nets in general as well as coloured Petri nets. Reisig [31] gives a general introduction, Jensen [20] describes coloured Petri nets. Generally speaking coloured Petri nets permit a more compact representation while offering the same computational power compared to for example P/T-nets.

Reference nets [23] are so-called higher (coloured) Petri nets, a graphical notation that is especially well suited for the description and execution of complex,

[3] The redesign was necessary for several reasons. The most important one is that the original work did not use the nets within nets paradigm and therefore lacked of architectural clearness when introducing new layers in the multi agent system.

concurrent processes. As for other net formalisms there exist tools for the simulation of reference nets [24]. Reference nets show some expansions related to "ordinary" coloured Petri nets: nets as token objects, different arc types, net instances, and communication via synchronous channels. Beside this they are very similar to coloured Petri nets as defined by Jensen. The differences now are shortly introduced.

Nets as tokens Reference nets implement the "nets within nets" paradigm of Valk [36]. 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 beholder's viewpoint.

Arc types In addition to the usual arc types reference nets offer *reservation arcs*, that carry an arrow tip at both endings and reserve a token solely for one occurrence of a transition, *test arcs*, and *inhibitor arcs*. 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). Inhibitor arcs prevent occurrences of transitions as long as the connected places are marked.

Net instances Net instances are similar to the 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.

Synchronous channels Synchronous channels [6] permit a fusion of transitions (two at a time) for the duration of one occurrence. In reference nets (see [23]) 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.

3 Multi agent system

This section gives a short introduction to a multi agent system modelled in terms of "nets within nets" (see figure 1). This survey is given to make the general ideas visible that are prerequisite to the understanding of the concepts that follow in later sections of this paper. It is neither an introduction to multi agent systems nor the assets and drawbacks of dividing the system into platforms is discussed here. For a broad introduction see for example [37], the special view taken in our work is a standard proposal of the "Foundation for Intelligent Physical Agents" (FIPA) [14]. The latest publications of the FIPA can be found in [13].

Take a look at figure 1: The grey rounded boxes enclose nets (net instances) of their own right. The ZOOM lines enlarge object nets that are tokens in the

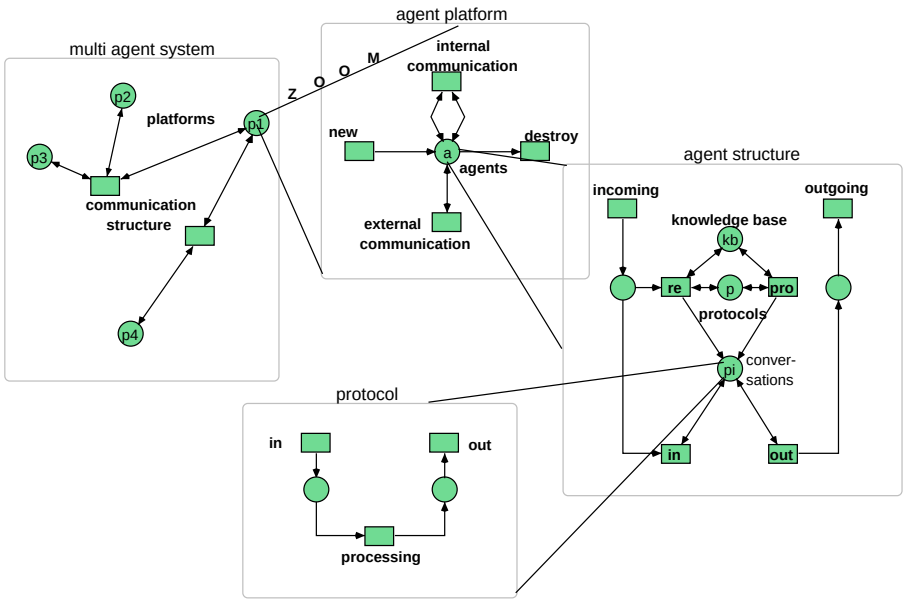

Fig. 1. MAS as nets within nets

respective system net.[4] The upper left net of the figure is an arbitrary agent system with places containing agent platforms and transitions modelling communication channels between the platforms. This is just an illustrating example, the number of places and the form of interconnection has no further meaning.

The first zoom leads to a closer view of a simplified agent platform. The central place agents contains all agents that are currently hosted on the platform. New agents can be generated (or moved from other platforms) by transition new, agents can be destroyed or migrate to another platform (transition destroy). Internal message passing differs from the external case so it is conceptually separated: The internal communication transition binds two agents (the sender and the receiver of a message) and allows them to hand over a message via call of synchronous channels. External communication involves only one agent of the platform. For external communication as well as for agent migration the communication transitions of the top level agent system net are needed. The interaction of the multi agent system and the agent platform is made possible by inscribing the transitions with synchronous channels connecting for example the transition external communication of an agent platform with that of another one via the communication structure transition of the multi agent system. These inscriptions are not visible in the figure.

[4] Beware not to confuse this net-to-token relationship with place refinement.

The remaining nets that show the structure of an agent and an example of its (dynamic) behaviour in form of protocols (protocol nets) are explained in more detail in section 4 and 5, respectively.

4 Agent structure

An agent is a message processing entity, that is, it must be able to receive messages, possibly process them and generate messages of its own. In this context it is to be noted that a completely synchronous messages exchange mechanism as it is used in most object oriented programming systems, frequently violates the idea of autonomy among agents.[5] The fundamental concepts that characterise agents are (in our opinion) *autonomy*, *intelligence*, and *mobility*.

4.1 Abstract agent model

As a Petri net model, figure 2 shows the most abstract view on an agent. The input and output transitions are a speciality of the reference nets that are used in the figure. They can communicate with other (input and output) transitions in other net copies through the use of synchronous channels (Because they are agents this is done via message passing). The basic agent model takes advantage of the ability of a transition to occur concurrently with itself.[6] So the agent is able to receive, process, and send several messages at the same time, it does not block. The transition processing can be refined for concrete agents as desired. In this and all following net figures all not unconditionally necessary inscriptions have been omitted. This leads to simpler models but may sometimes bring in the danger of confusing the reference nets (special coloured Petri nets) with more basic net formalisms (e.g. P/T-nets). So the reader is kindly asked to keep in mind the power of the used net formalism.

The introduced basic agent model implies an encapsulation of the agents: regardless of their internal structure, access is only possible over a clearly defined communication interface. In figure 2, this interface is represented by the transitions incoming and outgoing. In the figure, the realisation of the interface (through connection of both transitions to a messages transmission network via synchronous channels) is not represented. Obviously several (then virtual) communication channels can be mapped to both transitions.

Providing a static interface is the key to interoperability amongst agents. The agents presented in this paper speak and understand FIPA messages [13]. Neither the content of the messages nor the way that they are used is limited, only their syntactical structure is fixed. Some advantages of the use of a standardised communication mechanism can be found in [9].

[5] To our understanding agents are not exclusively (artificial) intelligent agents, but rather a general software structuring paradigm on top of the ideas of object orientation [19].

[6] Please note that this is not a special feature of reference nets but of all proper net formalisms.

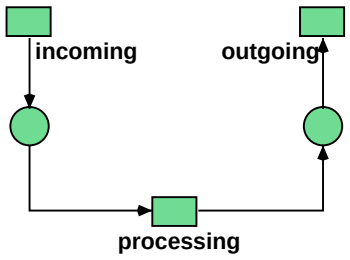

Fig. 2. Most abstract view on an agent

The presented agent model corresponds to the fundamental assumptions about agents: Because agents should show autonomy, they must be able to exercise an independent control over their actions. Autonomy implies the ability to monitor (and, if necessary, filter) incoming messages before an appropriate service (procedure, method...) is called. The agent must be able to handle messages of the same type (e.g. asking for the same service) differently just because of knowing about the message's sender. This is one of the major differences between objects and agents: A *public* object method can be executed by any other object, *protected* methods offer a static access control that is very often inconvenient to the programmer and user. Without regard to the fundamental autonomy, an agent can obviously be sketched (or take the obligation for itself) to appear like an object to the outside world, therefore to be perfect cooperative. Another reason to prefer messages over for example method calls is that methods are fixed both in respect to their arguments (number and type) and the number of methods that are offered by one object. Using method calls makes it tricky to adapt to new situations.

Autonomy is the major difference between agents and *active objects*: The latter may show some of the properties that characterise agents (an often used example is mobility) but they do not have the ability to control who is calling their (public) methods. Agents may have arbitrary fine grained access control.

The model does not affect nor restrict the intelligence nor the mobility of the agents: Intelligent behaviour is achieved through refinements of the transition processing, mobility requires interaction of the agent and the agent system.[7] Therefore, mobility is not a topic of this work, intelligence is raised again in the chapter on agent behaviour protocols (chapter 5).

[7] Note that the proper handling of mobility is a research area of its own rights. There is already some work done [28] in this direction, that will soon be brought into our agent definitions.

4.2 Refined agent model

Beside the fundamental agent concepts the agent model has to meet additional requirements concerning their ease of use: On the one hand it has to offer a high degree of flexibility in particular during execution and on the other hand the modelling process has to be manageable and adaptable. In addition for a broader acceptance, intuitive intelligibility of the processes within the agents is necessary. These considerations have played an important role in the development of the protocol-driven agents sketched here.

The abstract agent net of figure 2 is refined in the following manner (see figure 3): The agent net as shown in the figure is further used as the interface of the agent to the outside world. The transition processing is refined to a selection mechanism for specialised subnets, that implement the functionality of the agent, therefore (beside the selection process) its behaviour. These subnets are named *protocol nets* (or short *protocols*) in the following.

Each agent can control an arbitrary number of such protocols, possesses however only one net (in reference net nomenclature: one net page), that represents its interface to the agent system and therewith its identity. This main net (page) is the visible interface of an agent in the multi agent system. As mentioned before all messages that an agent sends or receives have to pass this net.

The main net of the (protocol-driven) agents introduced here is given in figure 3. It is a refinement of the abstract agent net given in figure 2.

The central point of activity of a protocol-driven agent is the selection of protocols and therewith the commencement of conversations [7,33]. The protocol selection can basically be performed pro-actively (the agent itself starts a conversation) or reactively (protocol selection based on a conversation activated by another agent).[8] This distinction corresponds to the bilateral access to the place holding the protocols (protocols). The only difference in enabling and occurrence of the transitions reactive and pro-active is the arc from the place input messages to the transition reactive. So the latter transition has an additional input place: the incoming messages buffer. It may only be enabled by incoming messages. Both the reaction to arriving messages and the kick-off of a (new) conversation is influenced by the knowledge of an agent. In the case of the pro-active protocol selection, the place knowledge base is the only proper enabling condition, the protocols are a side condition. In simple cases the knowledge base can be implemented for example as a subnet, advanced implementations as the connection to an inference engine are also possible (and have been put into practise). Unfortunately this topic can not be deepened here any further.

An agent has several possibilities to react to dynamically changing environments: It may not react at all (if it decides that no changing of its behaviour is needed or possible), it may alter its protocol selection strategy (choose different

[8] The fundamental difference between pro-active and reactive actions is of great importance when dealing with agents. An introduction to this topic is e.g. given by Wooldridge in [38] (in: [37]).

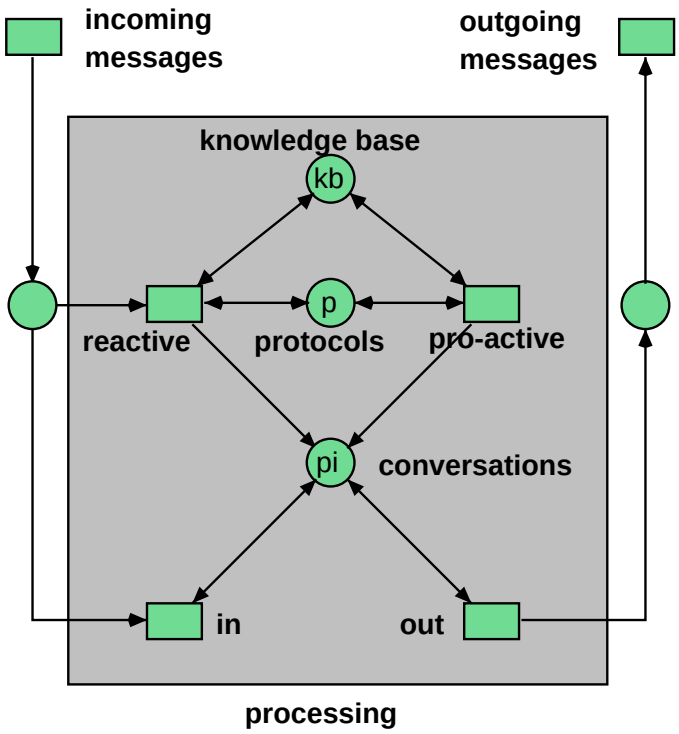

Fig. 3. Refined protocol-driven agent

protocols for the same message type), adapt one or more of its protocols[9] or ask other agents for protocols that suit to the new situation (protocols may be communicated as well).

A selected and activated protocol[10] is also called a conversation because it usually includes the exchange of messages with other agents. A conversation can however also run agent internal, therefore without message traffic. A freshly invoked conversation holds an unambiguous identification that is not visible in the figure. All messages belonging to a conversation carry this identification as a parameter to assign them properly. If an agent receives a messages carrying such a reference to an existing conversation, transition in is enabled instead of transition reactive. The net inscriptions that guarantee this enabling are not

[9] Protocol adaptation is done in a way similar to the "'reconfigurable nets"' formalism [1], i.e. restricted self-modifying nets [35]. The adaptation of protocols together with the agent's knowledge base unfortunately has to be skipped in this paper.

[10] Following the object oriented nomenclature one speaks of an instantiated net or protocol (that is represented in form of a net).

represented in figure 3 for reasons of simplicity. The transition in passes incoming messages to the corresponding conversation protocol in execution. Examples for this process follow in section 5.

If the sending of messages to other agents is required during the run of a conversation, these messages are passed from the protocol net over the transition out to the agent's main page and are handed over to the message transport mechanism by the transition output.[11] The communication between protocol net (conversation) and the agent's main net takes place via synchronous channels.

An interesting feature of any agents derived from the (template) agent in figure 3 is that they cannot be blocked, neither by incoming messages nor by their protocols[12] and therefore cannot loose their autonomy.

Examples for concrete conversation protocols are to be found in the following chapter 5 where a producer-consumer process is modelled exemplary.

5 Agent protocols

An important field of application of Petri nets is the specification of processes as that in figure 4, that shows a simple producer-consumer process. In order to give no room to conceptual confusion, such nets that spread over several agents and/or distributed functional units will be called "survey nets".

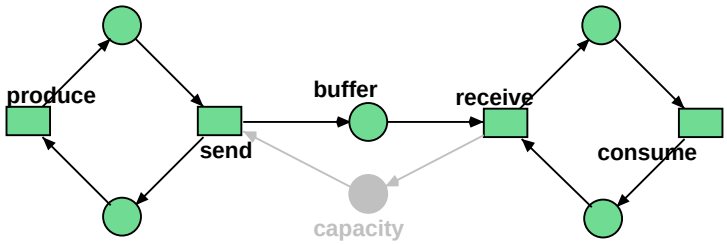

Fig. 4. Producer-consumer (survey net)

The place buffer in the middle of the figure represents an asynchronous coupling between the process of producing and that of consuming. This coupling is however to that extent independent that it for example blocks the consumer if it is empty or, given the case that it is inscribed with a capacity, blocks the producer when this maximal filling is reached. In the following, producer and

[11] The message transport mechanism is part of the agent system (or platform) and is therefore only sketched in section 3.

[12] Unless it is strictly necessary for a protocol to block the entire agent and this is explicitly modelled.

consumer are introduced as autonomous agents and are modelled according to figure 3 by means of a reference net. The buffer is not modelled as an independent agent, nevertheless this would both syntactically (this will be explained in the following) and semantically (in consideration of the level of autonomy the buffer owns) be no problem.

An interesting point is the re-usability of the protocols: Consider a refined model in which the buffer should play an active role and should therefore be modelled as an agent of its own. The protocols of the producer and the consumer remain structurally unchanged, only the addressees of their messages have to be adapted. But these should be modelled dynamically in any case.

Note that this is not the first work that uses the consumer-producer process as an example to illustrate new ideas of how to model and structure software systems by means of Petri nets. It is for example used by Reisig to introduce Petri nets in general [31] and by Valk to show different models of synchronisation [21].

The following example assumes that the buffer is restricted by a capacity of one item. This restriction is for simplification purposes only and may be lifted easily. The restriction is indicated in figure 4 by the grey place capacity under the buffer place.

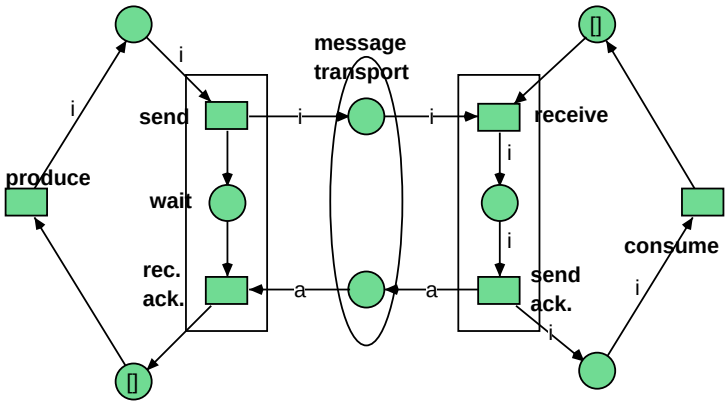

Fig. 5. Synchronous producer-consumer

The producer-consumer survey net is refined to the net in figure 5 which uses an explicitly modelled synchronous message exchange. The buffer and its capacity get carried away to form the message transport system. The message transport system is also the borderline of the two agents producer and consumer that will be introduced in the following subsections. The producer produces one item (denoted as i in the figure) and sends it to the consumer. The producer has to wait for an acknowledge (a) from the consumer to fire the transition rec.ack.

in order to reach its initial state. When the consumer receives an item it sends the acknowledge (send ack.) and consumes the item received. After consuming it is ready to receive the next item.

The marking of the net in figure 5 indicates the starting points of the protocol nets telling the agents how to produce or consume: The production protocol has to be selected pro-actively and starts with the production of an item. The consume protocol is selected in a reactive manner to process an incoming message from the producer containing an item.

5.1 Producer

The protocol of the producer agent is represented in figure 6. The upper transitions with the channels :start, :out, :in, and :stop are typical for all types of protocol nets. The :start channel serves as a means to pass possibly necessary parameters to the protocol. It is called on the agent main page (see figure 3) either by transition reactive or pro-active. The channels :in and :out are responsible for the communication of an operating protocol with the environment. They connect to the transitions of the same denominators on the agent's main page. When a protocol has finished its task, the transition inscribed with channel :stop is enabled. By calling of this channel the agent may delete the protocol or, more correctly, the protocol instance.

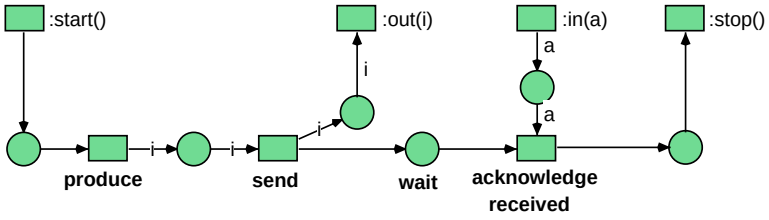

Fig. 6. Produce protocol

After the start of the protocol the transition produce produces a performative[13] (here i) containing an item, that is directed to the consumer. Note that in the example the performative is the only thing that is produced. The performative will be sent over the :out channel; subsequently the protocol is blocked waiting for an answer message. The blocking behaviour is necessary to simulate

[13] Some of the ideas that led to the agent model introduced here are partially originated in the area of the KQML- ([12]) or FIPA-agents ([15]). Roughly speaking a performative is a message. KQML stands for "Knowledge Query and Manipulation Language", FIPA is the abbreviation of "Foundation for Intelligent Physical Agents".

a synchronous communication between producer and buffer. Without waiting for an answer the producer would be able to "inundate" the buffer with messages, what requires an infinite buffer capacity. An arriving confirmation enables the transition acknowledge received. After occurrence of that transition the protocol is not blocked any further and terminates (by enabling the stop transition). The producer agent is now able to select and instantiate the produce protocol again.

5.2 Consumer

The protocol net that models the consume behaviour of the consumer agent (see figure 7) is selected (*reactively*) by the agent's main page to process an incoming performative from the producer agent. It is instantiated and the :start channel is used to pass the performative to the protocol. Beside others the performative is needed to send a acknowledge performative to the originator of the conversation (the producer). Note that the consumer agent does not know the producer or if there is one or several of them. The protocol works in either case.

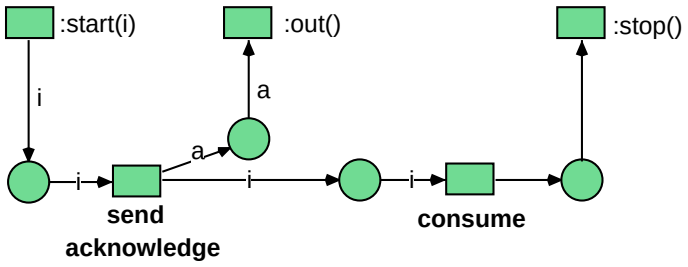

Fig. 7. Consume protocol

The consumer can block an arriving message as long as it wants, until it is ready to "consume" the carried item. In figure 7, this is represented by the transition send acknowledge. After acknowledging the receipt of the item the transition consume may occur. After that the protocol terminates and can be deleted.

Figures 6 and 7 show the protocols that model a conversation between producer and consumer. They are executed within agents of the type of figure 3. The figures form a simple example that illustrates how to model a producer-consumer process by means of agent oriented Petri nets. The proposed methodology to implement protocol nets in a top-down manner starting with so-called survey nets is not the only possibility to develop protocols. One can easily think of a bottom-up style or mixed cases especially for hierarchical protocol relationships as in the following subsection. Unfortunately this topic can not be deepened here.

5.3 Refined producer

Consider a producer capable of producing several different items. The naive approach to model this more powerful producer is to enlarge the producer protocol of figure 6 and to use this new protocol afterwards. For several reasons this is not a good idea:[14]

- Redefinition of existing nets is tiresome and error-prone.
- Large nets tend to be complex and thus difficult to understand and maintain (see above).
- Further enhancements become more and more difficult.

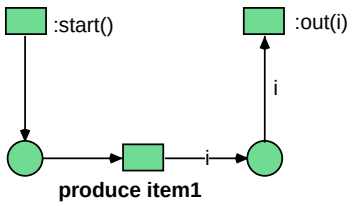

Fig. 8. Internal produce protocol

For these reasons the new produce process is split into pieces. First protocols for the different produce procedures like that in figure 8 have to be modelled. Now the production of the items can be driven from other protocols inside the producer agent. An example of such a higher-order protocol is a description of the following specification: imagine a consumer that does not care if it receives items of type 1 or type 2. In that case the producer can decide to produce items maybe on reasons of availability or price.

This decision is independent of the production process[15] and should therefore be carried out in an independent protocol. This protocol is shown in figure 9. The protocol is an extension to the producer protocol of figure 6. At first a decision is made about the type of item to produce (transitions i1 and i2). After that the protocol has to send a message to the selected production protocol, e.g. to that of figure 8. The protocol waits for the item to arrive, sends it to the consumer and so on. It may appear that the protocol net in figure 9 shows a conflict introduced through several transitions carrying the same synchronous channel. This possible conflict is resolved through additional inscriptions concerning the type of message that is the argument of the synchronous channels.

[14] This enumeration could easily be continued.

[15] Certainly only to that extent that an informed decision needs to know something about the needs of the production.

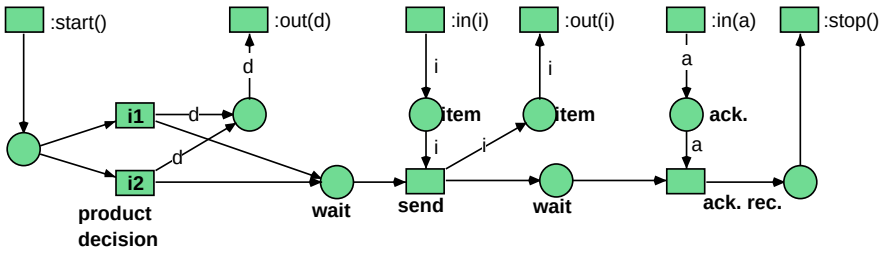

Fig. 9. Production alternatives

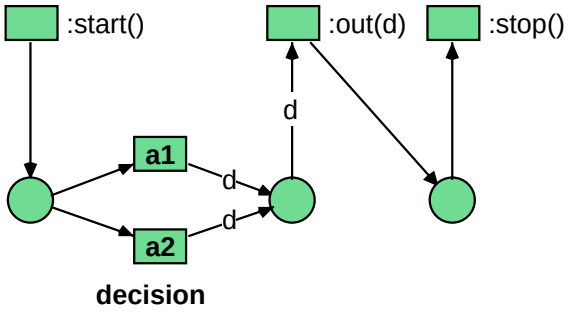

Fig. 10. Decision protocol

Note that this solution is not optimal considering the reasons for not enlarging existent nets. As a consequence the process of decision making is sourced out to a protocol of its own; it may be used by other protocols, too. To illustrate this, figure 10 shows the new decision protocol. The new produce protocol is quite similar to the original one, just the decision protocol is called instead of producing.

6 Related work

Note that this is not the first approach to use Petri nets to model or implement agents. This section will give an idea of similarities and differences between our work and that of other researchers.

A major difference between related work of other authors and our approach is the use of reference nets and therefore the nets within nets idea. To our knowledge there are no other implementations of this net paradigm beside reference nets. Despite of this difference there are certainly similarities to the results of other researchers. This work aims at modelling multi agent systems as a whole.

Therefore it is not possible to relate it to work dealing with smaller parts of agents (e.g. the planning process or reasoning in uncertainty). This will be made up when discussing these parts of our work in future publications.

Sibertin-Blanc et.al. [5] implement agent behaviour in form of *Cooperative Nets* [34]. Cooperative nets model the behaviour of objects, this approach is enlarged to agents. While the basic idea to model the behaviour is somehow similar to our work there are also differences: The behaviour of a (cooperative) object is statically defined by one net, it can not be altered or adapted at runtime, which is a major drawback for agents. Furthermore, the work of Sibertin-Blanc et.al. contains no notion of an multi agent system.

Holvoet [18] proposes Petri net agents that shall communicate in a manner similar to synchronous channels. Like Sibertin-Blanc (see above) he does not explicitly handle multi agent systems. The proposed agents are not autonomous in the strict sense presented in section 4 of this paper, because their interaction (via transition synchronisation) is not filtered by an interface but directly concerns the transitions modelling the agent's behaviour. So a proper encapsulation of the agents can not be assured, the agents are rather active objects. The agents Holvoet proposes are "special agents": They are specialised and restricted to do tasks that are exchangeable protocols to the agents introduced in this paper.

Fernandes and Belo [10] introduce a modelling case study that uses coloured Petri nets to model multi agent systems activities. They do not model single agents (that are token in their approach) but the overall system behaviour for one example system.

Miyamoto and Kumagai [25,26] enhance the Cooperative nets (by Sibertin-Blanc, mentioned above) in several ways to multi agent nets. Their work is closer to our approach because it uses quite similar protocols to define the agents' behaviour. Their protocols offer public interfaces, therefore they are not autonomous in our strict sense.

Xu and Shatz [39] build agents on top of the G-Net formalism of Figueiredo and Perkusich [11]. Despite of the different formalisms their work has some similarities to ours, namely the planning process (how to react to incoming messages). The main difference between the approaches is that their multi agent system is completely unstructured. Therefore mobility is not taken into account. Furthermore their agents have a fixed set of methods and may only adapt by changing their goals, plans and knowledge, not by reconfiguring, adapting or exchanging their actions (methods).

7 Outlook

In consideration of the topic of this paper some interesting aspects of the agent introduced here could not be mentioned: the protocol nets an agent possesses are interchangeable at run time (so the agent allows this). A mobile agent (mobility is possible in the multi agent framework but not introduced here) that arrives at an agent platform can adopt the protocols valid there as his own.

The modelling process is a topic of its own rights. The transformation of a survey net that describes an entire conversation (as given in figure 4) to the protocol nets is exemplarily carried out in [17] (in German).

A further point concerns the protocols themselves: only very simple procedures were shown. More complicated protocols include a hierarchical nesting that is constructed at run time by mutual calls of protocols (in contrast to protocols forming a conversation this is done within one agent). In this way a dynamic adaptation of the agents via self modification becomes possible.

Besides intuitive and compact modelling of concurrent processes Petri nets are particularly well-known for their provability. It should not be concealed that there are no standard proving techniques nor frameworks for reference nets. This is a major drawback because exhaustive testing via simulation is not always wanted nor possible. There is an ongoing PhD. thesis in our department that deals with the question of property conserving composition in agent Petri nets, first results of this work can be found in [22]. This is a very promising approach because it fits to the compositionality of the protocols presented in this work: Once a property like liveness has been proven for a protocol net, it may be used in any conversation without loosing this property (subject to the condition that all other protocol offer this property, too).

At present the knowledge base is in an internal discussion. There exists a Prolog interpreter that is implemented in Java and therefore can be integrated into the reference net simulation tool Renew. It is used as an inference engine inside the agents in more elaborated examples. Desirable is a graphically modelled knowledge model.

References

1. Eric Badouel and Javier Oliver. Reconfigurable Nets, a Class of High Level Petri Nets Supporting Dynamic Changes. Research report pi 1163, INRIA, 1998.
2. Bernhard Bauer, James Odell, and H. van Dyke Parunak. Extending UML for Agents. In *Proceeding of Agent-Oriented Information Systems Workshop*, pages 3 – 17, 2000.
3. Jeffrey M. Bradshaw, editor. *Software Agents*. AAAI Press, 1997.
4. J.M. Bradshaw. *An Introduction to Software Agents*, chapter 1. in: [Bra97a], 1997.
5. Walid Chainbi, Chihab Hanachi, and Christophe Sibertin-Blanc. The Multi-agent Prey/Predator problem: A Petri net solution. In P. Borne, J.C. Gentina, E. Craye, and S. El Khattabi, editors, *Proceedings of the Symposium on Discrete Events and Manufacturing systems*, Lille, France, July 9-12 1996. CESA'96 IMACS Multiconference on Computational Engineering in System Applications.
6. S. Christensen and N.D. 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.
7. R. Scott Cost, Ye Chen, T. Finin, Y. Labrou, and Y. Peng. Modeling agent conversation with colored Petri nets. In *Working notes on the workshop on specifying and implementing concersation policies (Autonomous agents '99)*, 1999.

8. R.S. Cost, T Finin, Y. Labrou, X. Luan, Y. Peng, L. Soboroff, J. Mayfield, and A. Voughannanm. Jackal: A Java-based Tool for Agent Development. In *Working Notes of the Workshop on Tools for Developing Agents, AAAI'98*, pages 73–82. AAAI Press, 1998.
9. Jonathan Dale and Ebrahim Mamdani. Open standards for interoperating agent-based systems. Software Focus, Wiley, 2001.
10. Joao M. Fernandes and Orlando Belo. Modeling Multi-Agent Systems Activities Through Colored Petri Nets. In *16th IASTED International Conference on Applied Infomatics (AI'98)*, pages 17–20, Garmisch-Partenkirchen, Germany, Feb. 1998.
11. Jorge C. A. Figueiredo and Angelo Perkusich. G-Nets: A Petri Net Based Approach for Logical and Timing Analysis of Complex Softwaree Systems. *Journal of Systems and Software*, 39 (1)(39-59), 1997.
12. Tim Finin and Yannis Labrou. A Proposal for a new KQML Specification. Technical report, University of Maryland, Februar 1997.
13. FIPA. Homepage. `http://www.fipa.org`.
14. FIPA. FIPA 97 Specification, Part 1 - Agent Management. Technical report, Foundation for Intelligent Physical Agents, `http://www.fipa.org`, October 1998.
15. FIPA. FIPA 97 Specification, Part 2 - Agent Communication Language. Technical report, Foundation for Intelligent Physical Agents, `http://www.fipa.org`, October 1998.
16. Martin Fowler. *UML Distilled*. Addison-Wesley Longman, Inc., 1. edition, 1997.
17. Daniela Hinck, Michael Köhler, Roman Langer, Daniel Moldt, and Heiko Rölke. Bourdieus Habitus-Konzept als prägendes Strukturelement für Multiagentensysteme. Mitteilung 298, University of Hamburg, Department for Computer Science, 2000.
18. Tom Holvoet. Agents and Petri Nets. *Petri net Newsletter*, 49:3–8, 1995.
19. Nicholas R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 177(2):277–296, 2000.
20. K. Jensen. *Coloured Petri nets, Basic Methods, Analysis Methods and Practical Use*, volume 1 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1992.
21. E. Jessen and R. Valk. *Rechensysteme – Grundlagen der Modellbildung*. Springer-Verlag, 1987.
22. Michael Köhler, Daniel Moldt, and Heiko Rölke. Liveness preserving composition of agent Petri nets. Technical report, University of Hamburg, Department for Computer Science, 2001.
23. 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.
24. Olaf Kummer, Frank Wienberg, and Michael Duvigneau. *Renew - User Guide*. University of Hamburg, Vogt-Kölln-Straße, Hamburg, 1.4 edition, 2001.
25. Toshiyuki Miyamoto and Sadatoshi Kumagai. A Multi Agent Net Model of Autonomous Distributed Systems. In *Proceedings of CESA'96, Symposium on Discrete Events and Manufacturing Systems*, pages 619–623, 1996.
26. Toshiyuki Miyamoto and Sadatoshi Kumagai. A Multi Agent Net Model and the Realization of Software Environment. In *20th International Conference on Application and Theory of Petri Nets, Proceedings of the Workshop: Applications of Petri nets to intelligent system development*, pages 83–92, June 1999.

27. Daniel Moldt. *Höhere Petrinetze als Grundlage der Systemspezifikation*. Dissertation, University of Hamburg, Department for Computer Science, Vogt-Kölln Str. 30, 22527 Hamburg, Germany, 1996.
28. Daniel Moldt and Ivana Tričkovič. The paradigm of nets in nets as a framework for mobility. Unpublished Technical Report, 2001.
29. Daniel Moldt and Frank Wienberg. Multi-agent-systems based on coloured Petri nets. In P. Azéma and G. Balbo, editors, *Lecture Notes in Computer Science: 18th International Conference on Application and Theory of Petri Nets, Toulouse, France*, volume 1248, pages 82–101, Berlin, Germany, June 1997. Springer-Verlag.
30. James Odell and H. van Dyke Parunak. Representing Social Structures in UML. In *Proceedings of Autonomous Agents'01*, Montreal, Canada, May/June 2001.
31. Wolfgang Reisig. *Petri nets: An introduction*. Springer, 1985.
32. Heiko Rölke. Modellierung und Implementation eines Multi-Agenten-Systems auf der Basis von Referenznetzen. Diplomarbeit, Universität Hamburg, 1999.
33. Heiko Rölke. Die Mulan Architektur. Technical report, Universität Hamburg, 2000.
34. Christophe Sibertin-Blanc. Cooperative Nets. In *Proceedings of the 15th International Conference on Application and Theory of Petri nets*, volume LNCS 815, Saragossa, June 1994.
35. Rüdiger Valk. Self-modifying nets. Technical Report Nr. 34, Universiät Hamburg, 1977. R 6033.
36. 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 *Lecture Notes in Computer Science*, pages 1–25, June 1998.
37. Gerhard Weiß, editor. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. The MIT Press, 1999.
38. Michael Wooldridge. *Intelligent Agents*, chapter 1. The MIT Press, 1999.
39. Haiping Xu and Sol M. Shatz. A Framework for Modeling Agent-Oriented Software. In *Proc. of the 21th International Conference on Distributed Computing Systems (ICDCS-21)*, Phoenix, Arizona, April 2001.