Liveness Preserving Composition of Behaviour Protocols for Petri Net Agents

Michael Köhler   Daniel Moldt   Heiko Rölke
University of Hamburg, Department of Computer Science
Vogt-Kölln-Straße 30, D-22527 Hamburg

{koehler, moldt, roelke}@informatik.uni-hamburg.de

Abstract

The question of modelling and verification of agent-oriented systems is our research area. In multi agent systems the interplay of three parts of the system – called Δ-pattern – is central: first, local elements (e.g. agents), second, global elements (e.g. the multi agent system), and, third, the composition structure (e.g. the mobility structure in the agent system). The dynamic nature of agent systems makes composition a concept on its own rather than an implicit aspect of the two other concepts.

Multi agent systems are based on three major concepts: mobility, adaptivity, and cooperation, which are structured by the Δ-pattern. In this presentation we focus on agent conversations to present a model, which describes the interplay of local and global parts as well as the dynamic configuration structure of a conversation.

This model is based on the multi agent architecture Mulan – designed at our department. It is used to model mobility, adaptivity, and cooperation in a unified way. Mulan is a Petri net based architecture, which is both an implementation of a multi agent platform and also a framework for the modelling of agent applications.

Here we deal with the analysis of liveness preserving composition of Petri nets. The two areas of agent-oriented modelling and compositional verification – both in the setting of the Petri net theory – are combined by dealing with a class of models which have a restricted structure: For the special case of communication closed protocols it is shown that liveness of conversations is a structural property. This result is of crucial importance to compose agent protocols at run-time.

1 Introduction

Modelling and structuring of multi agent systems with respect to both software engineering and verification requirements is a major task for current efforts. Currently, agents are generally programmed using high-level languages such as Java (namely in agent frameworks as Jackal [CFL+98]) 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

[1]UML stands for Unified Modelling Language. See for example [RJB99].

neither proposed nor in general use.[2] A unifying framework based on the visual programming concept of Petri nets – as proposed by the Sam-architecture – thus bridges the gap between modelling and programming on the one side and modelling and verification on the other side.

global view

↑ ↓

local aspects

Figure 1: Central view on traditional software systems

The design of "traditional" software system deals with the interdependencies of global and local system views (cf. Fig. 1). For multi agent systems the treatment of *three* concepts – arranged in the Δ-pattern – has to be taken into account: The local elements (the agent), the global view (the agent system), and the composition structure (cf. Fig. 2).

Contrary to object-oriented approaches, all these three elements are concepts in their own right in the context of agent systems. This is due to the distributed and dynamic nature of agent systems.

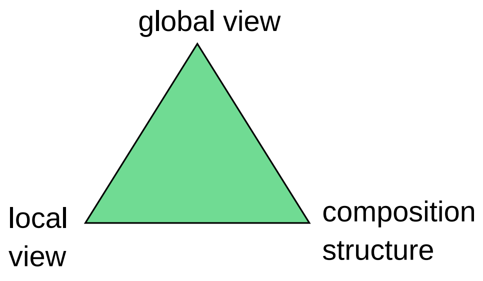

Figure 2: Central view on a multi agent system as a Δ-pattern

In this contribution we propose a modelling approach for agent conversations. Conversations are one central part of multi agent systems. We have chosen a special form of high-level Petri nets, called reference nets, as the basis for our multi agent architecture Sam (cf. [KMR01]). Sam is both an implementation of a multi agent platform and also a framework for the modelling of agent applications.

Petri nets in general and "nets within nets" in special offer several practical as well as theoretical properties for the needs of multi agent systems. As we have discussed in [KR01b], the major aspects of multi agent systems – mobility, adaptation, and cooperation – have to be addressed by the following formal concepts: encapsulation, concurrency, dynamic environment, assumption based modelling, and structural properties.

[2]The authors are aware of the upcoming proposals that base on UML i.e. the ones from Odell et al. [BOP00] (AUML). To our opinion these proposals capture only parts of the agent modelling tasks and leave out important areas such as agent mobility.

Due to these requirements we have chosen the paradigm of "nets within nets" of Valk [Val87, Val96, Val98, Val00] for the basis of the SAM-architecture. SAM is specified in the reference nets formalism (cf. [Kum98, Kum02]) and is implemented with the tool RENEW (cf. [KW98]).

The three central concepts together with their relationships have been addressed in the current research: The relations sketched in figure 2 are picked up as concepts in other approaches as well. The relation between local elements and the global view is the central concept of all bottom-up approaches for software-design, especially in the context of object-orientation (cf. [Mey97]) or the pattern driven approach (cf. [Fow97]). The relation between the global view and the composition structure is central in the component-based approach (cf. [Gri98]). Components are composed by special entities, which implement the "glue" between them in terms of a coordination language.

Our work is structured as follows: section 2 introduces the MAS-architecture SAM. In section 3 we describe the general requirements towards a verification system for MAS and to what extend they are met by SAM. Then the interplay of the three key concepts are presented in the context of agent conversations: Section 4 describes the global elements of conversation, described by a survey net. Section 5 deals with the global elements of conversation described by so called A/C-protocol Petri nets. Section 6 presents the composition structure, which describes the dynamic interaction of protocols. The composition structure for conversations is called conversation order. Since conversation orders are an explicit element in SAM, dynamic reconfiguration at run-time, e.g. for load balancing purposes or adaptation of agents' behaviour, is possible. The work closes with a conclusion and an outlook in section 7.

2 The SAM Architecture

This section gives a short introduction to a multi agent system modelled in terms of "nets within nets". This survey is given to make the general ideas visible that are prerequisite for 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 are discussed here. For a broad introduction see for example [Wei99], the special view taken in our work is a standard proposal of the "Foundation for Intelligent Physical Agents" (FIPA) [FIP98a]. The latest publications of the FIPA can be found in [FIP].

2.1 Reference Nets

SAM is based on the special Petri net formalism of reference nets [Kum98, Kum02].[3] As for other net formalisms there exist tools for the modelling and simulation of reference nets [KW98]. Reference nets show some expansions

[3]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.

compared 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 [Jen92]. The differences are now shortly introduced.

Nets as tokens Reference nets implement the ”nets within nets” paradigm of Valk [Val96, Val98]. This paper follows his nomenclature and denominates the surrounding net *“system net”* and the token net *“object net”*. Hierarchies of “nets within nets” relationships are permitted, so the denominators depend on the beholder’s viewpoint.

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 A synchronous channel (cf. also [CH94]) permits a fusion of transitions (two at a time) for the duration of one occurrence. In reference nets 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 bidirectional and is also possible within one net instance.

2.2 Goals and Design of SAM

The Sam architecture for multi agent systems has three goals: it should support software engineering needs, should be based on a formal model, and should be in accordance with real social processes (cf. figure 3).

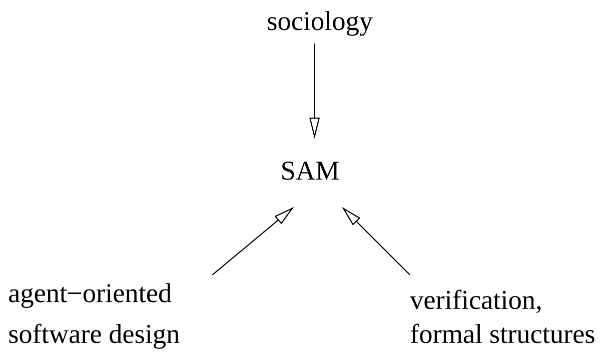
sociology
SAM
agent−oriented software design
verification, formal structures

Figure 3: The basic goals of Sam

The Sam architecture consists of four layers – all based on the “nets within nets”-paradigm (cf. figure 4): the Agnes layer, the Mulan layer, the Sonar layer and on top of these the Öri layer. The Agnes layer consist of the formal aspects of reference nets and object nets systems. The Mulan-layer describes the architecture of a multi agent system. Here, an unified framework for mobility, adaptation, and coordination is given. The next layer, called Sonar, enriches the technical aspects by social aspects, like group processes, roles, social norms, social structures etc.[4] The fourth layer, called Öri, specialises social agents for the context of public administration.

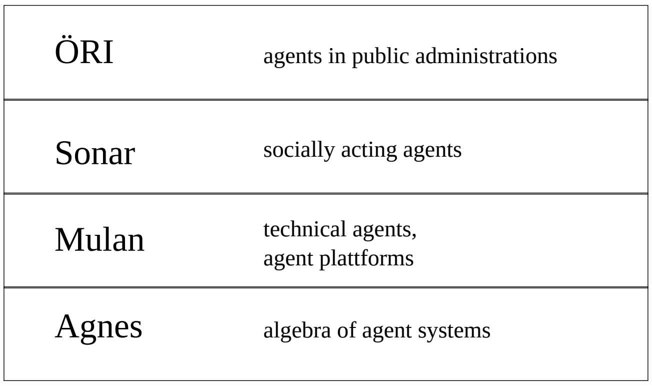

Figure 4: The Sam architecture

In this presentation we concentrate on the technical aspects of multi agent system, as they are treated by the Mulan layer. Take a look at figure 5 for a simplified model of Mulan: The gray rounded boxes enclose nets (net instances) of their own right. The ZOOM lines enlarge object nets that are tokens in the respective system net – following the concept of "nets within nets".[5] 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.[6]

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 migrate 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 calls 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

[4]The design of the Sonar and the Öri layer is founded in the DFG project "Agieren in sozialen Kontexten" (engl. "Acting in Social Contexts") (s. [vLMV01], and [KLMR00, HKL$^+$01a]) for some actual work.

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

[6]This is just an illustrating example, the number of places and the form of interconnection has no further meaning.

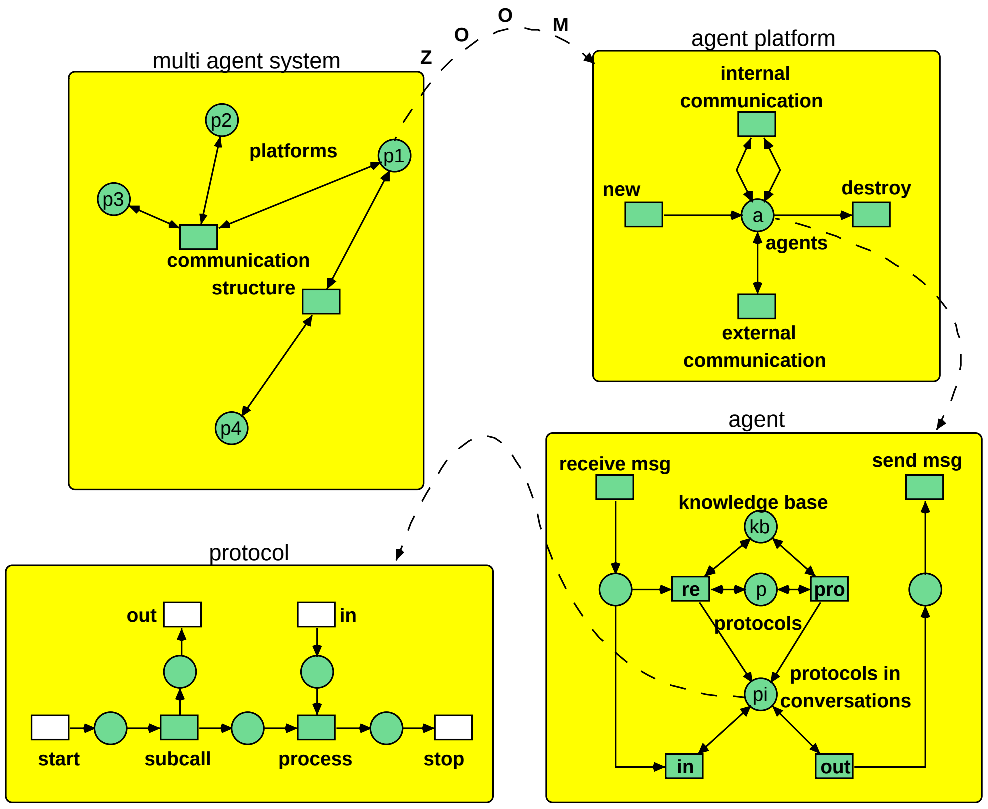

Figure 5: MAS as nets within nets

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.

The next zoom shows the structure of an agent. Each agent owns several net tokens, which describe its behaviour in terms of protocols. An example protocol is shown as the fourth net. Agents and their (dynamic) behaviour in form of protocols (protocol nets) are explained in more detail in the following.

Each zoom describes one central concept of multi agent systems. The relationship of the agent system to the platforms raises needs for the concept of mobility, whereas the relationship between the platform and the agent has to be treated by the concept of cooperation. The relationship of agents and their protocols is captured by the concept of adaptation of intelligent agents.

2.3 Agents and Protocols

In the following agents and their (dynamic) behaviour in form of protocol nets are introduced. 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 has to be noted that a completely synchronous message exchange mechanism as it is used in most object oriented programming systems, frequently violates the idea of autonomy among agents.[7]

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 Fig. 6 this interface is represented by the transitions receive message and send message. In the figure, the realization 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.

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 the user.

The processing of messages is realized by 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 Petri 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 instance), that represents its interface to the agent system and therewith its identity. As mentioned before all messages that an agent sends or receives have to pass this net.

The central point of activity of a protocol-driven agent is the selection of protocols and therewith the commencement of conversations [CCF$^+$99, KMR01]. 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 incoming 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

[7]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 [Jen00].

[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 [Woo99].

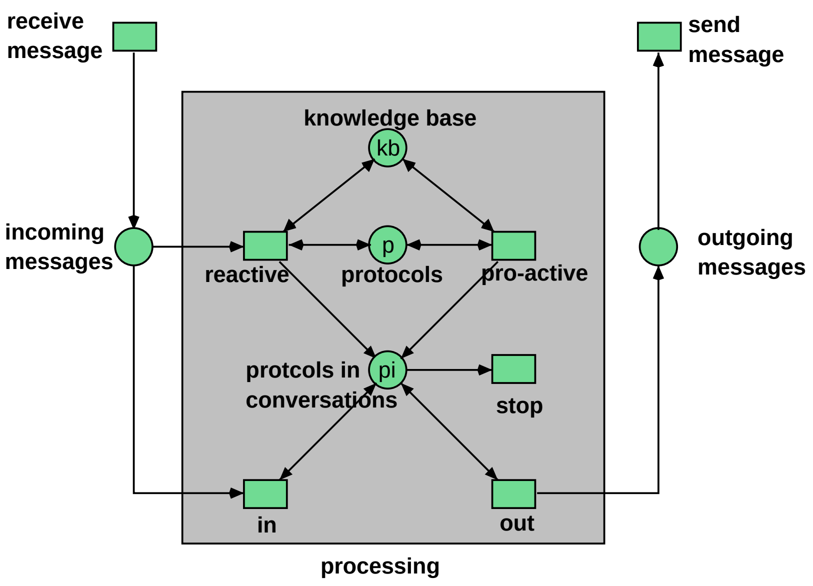

Figure 6: A protocol-driven agent

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).

A selected and activated protocol[9] participates in 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 represented in figure 6 for reasons of simplicity. The transition in passes incoming messages to the corresponding protocol in execution. Examples for this process follow in the remaining sections.

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 send message.[10] The communication between a protocol net and the agent's main net takes place via synchronous channels.

An interesting feature of any agent derived from the (template) agent in figure 6 is that they cannot be blocked, neither by incoming messages nor by

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

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

their protocols[11] and therefore cannot loose their autonomy.

Examples for concrete conversation protocols are to be found in the following chapter, where a producer-consumer process is modelled exemplarily. This scenario acts also as the case study for our verification approach for Sam.

3 Agent Oriented Verification

The analysis of agent systems raises the need for a special style of verification systems. This need is due to the dynamics and openness of agent systems. As stated before, modelling approaches lack a uniform basis. The same is true also for verification approaches.

Most approaches handle multi agent systems (MAS) the same way as one would describe a single intelligent agent. This style is based on the traditional artificial intelligence (AI) view, disregarding the needs of MAS. It is mostly based on (modal) logical specification and model checking techniques (cf. [HR00] as an example). In general, approaches addressing the specialities of MAS focus on one single aspect of description. For mobility several algebraic approaches describe the change of environment (cf. [MPW92], [VC98] or [CGG99]). Adaptation (cf. [GPdFC98]) and cooperation (cf. [SHM99] and [CCF$^+$99]) are addressed, too.

In our point of view such an approach is insufficient for the analysis of an agent system, since only parts of the system are specified. It is unclear whether the combination of the isolated formalisations leads to a correct description of what the system does. Since the integration of the models has do be done manually by the developer this leads to an error prone style of construction.

So, we conclude that not only the construction of an agent system should be done on a uniform basis but also the formal part of reasoning about it should be based on one single calculus. In the following, we describe which characteristics such a formal basis should own. So, we have to identify central characteristics and suited verification techniques.

3.1 Formal Key Concepts in Multi Agent Systems

In our Sam-approach, the central parts of a dynamic, open agent system – mobility, adaptation, and cooperation – are parts of the concept of compositionality (cf. [Zwi89]), which is based on several fundamental concepts, like locality, concurrency etc. In the agent context, the major issues – mobility, adaptation, and cooperation – must be handled by formal concepts. These aspects might be of interest in the context of object-orientation, too, but in the context of agents they cannot be ignored. These concepts are:

1. *Encapsulation and modularity.* An agent system must be able to express the concept of information hiding, so the system cannot be formulated in a "flat" and global way.

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

2. *Concurrency.* Agent systems are highly independent and run concurrently. So specification, implementation, and verification must not rely on totally ordered action sequences. Partial order semantics (true concurrency) must be used instead.

3. *Dynamic environment.* Agent systems are conceptually based on distribution and mobility. Therefore, a proof system must be able to describe environments and their dynamic change in an explicit way.

4. *Assumption based modelling.* An approach for MAS must allow to specify assumptions and commitments towards the environment This can be done in the assumption/commitment (A/C) concept introduced by [MC81] and [Jon83]. We additional postulate, that these assumptions and commitments should not only be visible for the verifier but also for the developer. To avoid a gap between verification and modelling, assumption and commitments should be an integral part of the model.

5. *Structural properties.* The systems should allow to easily describe structural restriction in order to guarantee e.g. security properties. These restrictions should be easily adaptable in the modelling approach.

These formal concepts are arranged the same way as in the general Δ-pattern of figure 2: The global view can formally be regarded as a distributed, concurrent system. The local view is described by modular subclasses, whereas the dynamic environment plays the role of the composition structure. Assumption/commitment relates the local and the global view. Figure 7 shows the key concepts within the AGNES layer of SAM.[12]

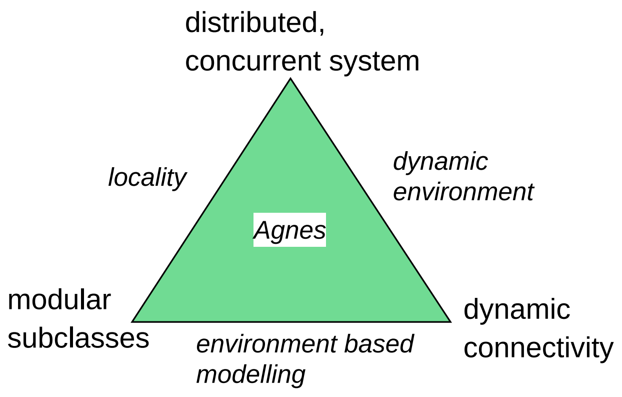

Figure 7: Central view on the formal basic of SAM

[12]The SONAR layer is also based on the triangle concept, now in the setting of social science. The SONAR layer is based on the interplay of actors, social acting, and social structures. The relationships are described by actor modes, theories based on the concept of social structures, and theories based on the concept of actor interaction (cf. [HKMM00, HKL$^+$00, KLMR00, HKL$^+$01b, HKL$^+$02]).

3.2 Classification of Verification Approaches

In the following we have to analyse, which techniques are suited to allow specification and verification of the above describes concepts.The great number of publications in the very general field of verification raises the need for a classification scheme. We propose a classification scheme mainly oriented on two categories: first the development style and second the verification time. The first category discriminates top-down and bottom-up construction:

1. *Top-down development.* This class incorporates the aspect of information hiding. Programs are composed of encapsulated modules. This style of development and verification is known as the "top-down" approach. The most common proof system has been developed by Apt, Francez, and de Roever [AFdR80].

2. *Bottom-up development.* This class considers the verification of modules on their own. The context where modules are embedded in is not known a priori. This style of reasoning is known as the "bottom-up" approach. A central step in the development of bottom-up proofs is the assumption/commitment formulation by Chandy and Misra [MC81].

Verification can also be classified by its place in the development:

1. Verification is done *after* programming (a posteriori verification). First the system is specified, then coded and afterwards the implementation is checked against the specification. Central approach is the model checking technique by Clarke and Emerson [EC82].

2. Verification is done *while* programming. This could be done if the programmer has a central idea how to proof the implementation while he programs. This kind of style is known as "structured programming". All axiomatic approaches – like [Hoa69] – are examples for this approach..

3. Proof *by* construction. Programming and verification are essentially the same. These approaches are based on constructive logic (like the COQ system [BBC$^+$99]) or on property preserving transformation (like in [Ber87]) or on structural restrictions resulting in subclasses (like in [BT87]). For top-down modelling the notion of property preserving refinement ([BGV91]) plays a major role.

3.3 From Compositional Verification to Agent Oriented Verification

Which approaches are suited for the development and verification of agent systems? In the following the central requirements of MAS are contrasted with the existing verification styles, as classified before. Several approaches try to lift verification styles designed for object oriented programs up to the agent context – neglecting the special needs of multi agent systems.

The first speciality is due to the nature of agents: agents are encapsulated, autonomous entities which are loosely coupled. They are developed with only with few knowledge of the whole agent system. Developers and agents cannot know the whole system or the state of the whole system. There is no such thing as global knowledge for agents.

Due to encapsulation and isolated development only bottom-up verification can apply to the agent context. All approaches considering agent systems as one unit must reject the openness of agent systems and rely on some kind of "closed agent world assumptions". Only the assumption based style of reasoning can apply to open systems.

Since agents are considered as active entities (in contrast to passive objects), we additionally take into account who is reasoning about the agent system. If it is the developer, we are mainly confronted with "static" problems, if it is the agent, we are confronted with more "dynamic" aspects. In the second case reasoning must be done automatically. This seems to be harder than the first case, where it can be done semi-automatically.

Reasoning at run-time is the consequence when dealing with open systems, where no one can know in advance, what might enter the system. Security in mobile agent systems is a central aspect (cf. [Vig98]) which can only be achieved by structural restrictions on agents or on agent behaviour. Structural restrictions reduce the proof burden of an agent system.

If one compares these requirements with the classification scheme, one can recognise, that we can restrict our investigation to the "bottom-up" style and to "verification while and by development", since the assumptions of a global systems or a top-down view is unreasonable.

Due to these requirements we have chosen the paradigm of "nets within nets" [Val87, Val96, Val98] for the developmental and formal basis of the SAM-architecture. This approach meets the above mentioned requirements directly or is adjusted currently by the authors. It has also the benefit that the "nets within nets" paradigm has it native "machine" implemented in the RENEW tool [KW98].[13] This approach could be seen as the formal implementation of the Δ-pattern of figure 7:

- *Concurrency/distribution:* Like every Petri net formalism it incorporates the concept of concurrency. The requirement of encapsulation and modularity is fulfilled by the concept that nets could be regarded as tokens again – a view that meets well with modularity and locality.
- *Dynamic environment:* The requirement to handle dynamic environments, A/C modelling is captured by the formalism of assumption/commitment Petri nets, short: A/C Petri nets. The subclass of A/C Petri nets deals with protocols and is therefore called A/C protocols.
- *Structural subclasses:* The formal and algebraic notion of object nets allows one to define subclasses with structural guaranteed properties. In the following, we present the subclass of well formed conversations.

[13]The work of [BDM$^+$99] favours an approach based on linear logic instead. This work is a good argument for our approach, since Petri nets and linear logic are strongly related (cf. [Far00]). Our approach has the additional advantage that our models have a graphical representation.

Nets within nets build the basic formalism for the agent system architecture Sam. In the further presentation we will focus our attention mainly on the aspects of the modelling of agents protocols in terms of A/C-Petri nets.

4 Conversations

An important field of application of Petri nets is the specification of processes, like the producer-consumer process shown in figure 8. 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". Survey nets are used the same way like those Petri nets used to model distributed algorithms in [KRVW97, Rei98], since they provide the global view on a – in principle – distributed system.

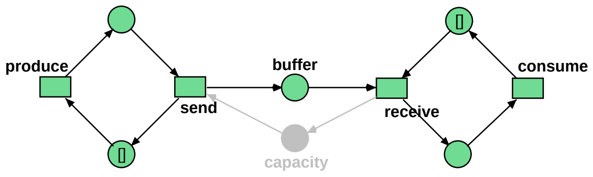

Figure 8: Producer-consumer (survey net)

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 an acting entity 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. 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 8 by the gray place capacity under the buffer place.

The producer-consumer scenario acts as our demonstration conversation. To emphasise the conversation characteristic of our scenario we refined the model (cf. Fig. 9). In the following, producer and consumer are introduced as autonomous agents and are modelled according to figure 6 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.

5 Conversation-Protocols

Conversations are built in terms of protocols. Each agent has its own set of protocols used for participating in a conversation. To link the local view of a protocol with the global one of the conversation, a new formalism is proposed: A/C Petri nets. A/C Petri nets and the protocols based on them include a

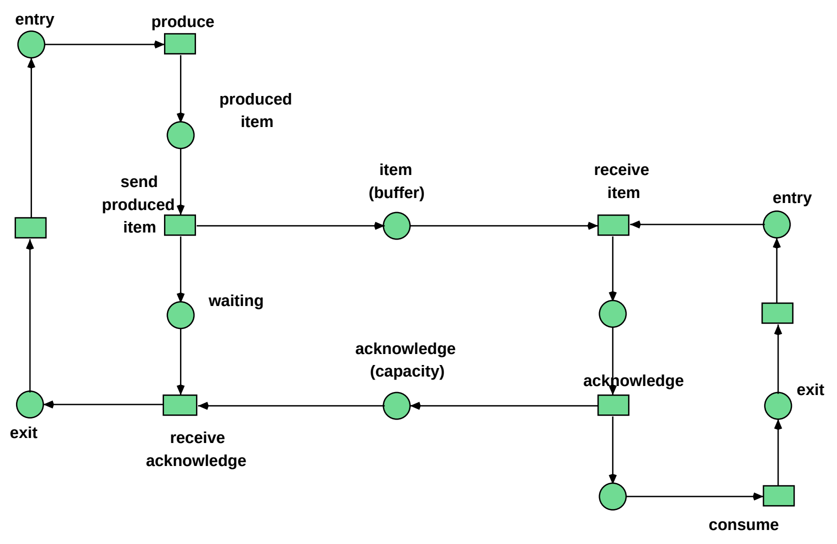

Figure 9: Refined producer-consumer scenario

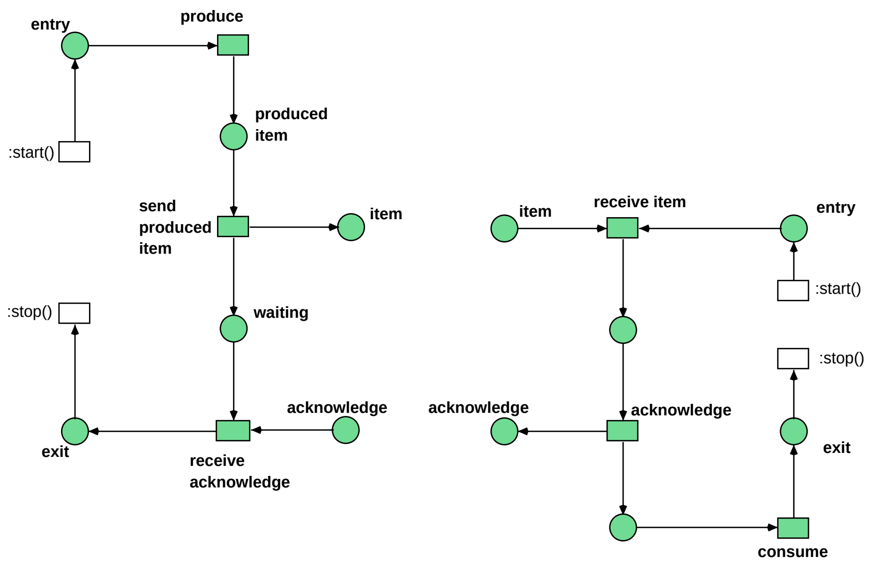

Figure 10: Splitting the producer-consumer scenario into two protocols

model of the assumptions made towards the environment. In this section we describe how to relate the producer-consumer-scenario to protocols and which environmental assumptions should made explicit, both for the design and also for verification.

5.1 Distributing a Conversation

The producer-consumer conversation can easily be distributed: the left part of figure 10 builds the protocol of the producing agent while the right part implements the protocol for the consuming agent. The places in the middle describe message pools shared by the protocols.

Since agent protocols describe the instantiation of one produce-consume-cycle the "looping" transition is split into two transitions inscribed by the channels :start, and :stop. So, a protocol is started by the agent, processed, and stopped afterwards by the agent. For the next produce-consume-cycle a fresh instance of the producer protocol is generated by the agent, the same for the consumer side.[14]

Protocols can be formally described and analysed in terms of the A/C notion: A/C notions should not only establish the possibility for modular correctness proofs – it should further be a integral part of the Petri net model. Assumptions towards the environment are integrated in the model of the protocol P. This approach enables the modeller to describe directly the scenario assumed. Petri net protocols allowing this style of modelling are called A/C protocols (cf. [KR01a]).

In the Sam framework, A/C protocols are based on A/C Petri nets. A/C-Petri nets are described in terms of synchronous channels, more precisely by uplinks: The notion of an up-link describes the "passive" part of two synchronising transitions. This is exactly what we want for a model of the environment: an external action matching an assumption about an external action.

The models have the same meaning as the protocols nets in Fig. 10, but they are more expressive by denoting assumptions about the environment, without the disadvantage of including directly the environment. This is done by the assumption part of the protocol, depicted by unfilled net elements: the transitions with the inscription (: in() and : out() as well as : start() and : stop()). Additionally, the places info, $\text{ticket}_{\text{in}}$, and $\text{ticket}_{\text{out}}$ are declared to be part of the assumptions.[15] The place info contains data that is needed to match each request and reply. How this assumption part could be exploited for verification is shown in [KR01a] where the correctness proof of the producer-consumer scenario is given.

Producer The protocol of the producer agent is represented in figure 11. 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

[14]Protocols are related to protocol instances similar as classes to objects.

[15]Note, that places in A/C Petri nets are not subdivided like transitions, so this fact is only a point of presentation.

page (see figure 6) 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 this channel the agent may delete the protocol or, more precisely, the protocol instance.

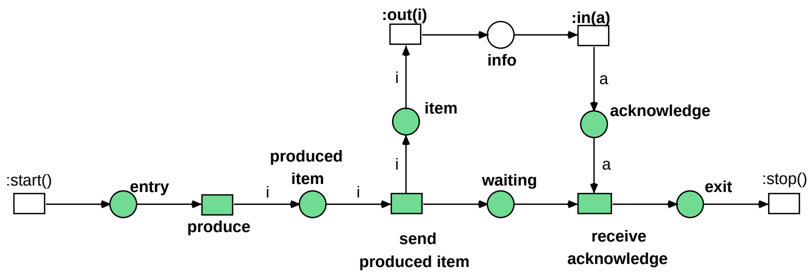

Figure 11: The producer protocol

After the start of the protocol the transition produce produces a performative[16] (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 a synchronous communication between producer and buffer. Without waiting for an answer the producer would be able to "flood" the buffer with messages, what requires an infinite buffer capacity. An arriving confirmation enables the transition acknowledge received. After occurrence of this 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.

Consumer The protocol net that models the consume behaviour of the consumer agent (see figure 12) is selected (*reactively*) by the agent's main page to process an incoming performative from the producer agent. It is instantiated, started via the channel :start and the channel :in 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.

The consumer can block an arriving message as long as it wants, until it is ready to "consume" the carried item. In figure 12, 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

[16]Some of the ideas that led to the agent model introduced here are partially originated in the area of the KQML- ([FL97]) or FIPA-agents ([FIP98b]). 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".

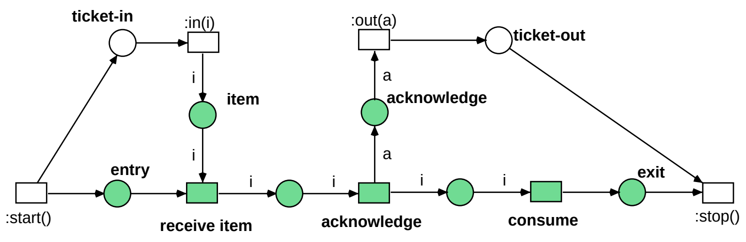

Figure 12: The consumer protocol

be deleted. The place $\mathsf{ticket}_{\mathsf{out}}$ controls that the protocol is only deleted after sending the acknowledge.

Figures 11 and 12 show the protocols that model a conversation between producer and consumer. They are executed within agents of the type of figure 6. The nets form a simple example that illustrates how to model a producer-consumer process by means of agent oriented Petri nets.

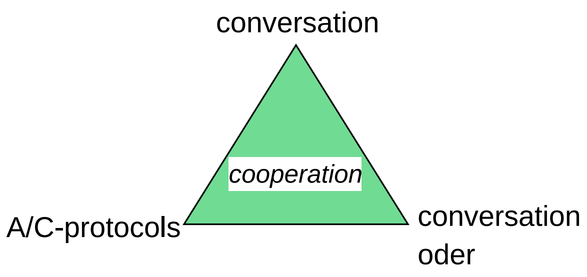

Figure 13: Central view on conversations

Agent conversations represent all the three aspects of the Δ-pattern (cf. Fig. 13): The global element is the agent conversation modelled by the survey net. Conversations consists of several interacting, agent-local protocol nets, so, protocols are the local element. The relation of the protocols and the global conversation is represented by the unfilled nodes in the protocol nets. Conversations are structured by the causal ordering of the messages. This order builds the configuration part. It should be considered as valuable like the two others, since only the explicit expressed order makes it possible to change the conversation dynamically. This order is considered in the following.

6 Well Formed Conversations

Agents in Sam cooperate in terms of conversations. A conversation consists of a set of protocol P_i, which participate at this conversation. Each protocol P is identified with an infinite set $P\langle i\rangle, i \in \mathbb{N}$, where $P\langle i\rangle$ denotes the ith instantiation of the protocol P. Since each instance of P is a net on its own, P

could be identified with the parallel composition of all instances:

$$P = P\langle 1\rangle || P\langle 2\rangle || \ldots$$

In the cyclic producer-consumer scenario in Fig. 8 infinite many instances exist: Producer⟨1⟩, Producer⟨2⟩, ... as well as Consumer⟨1⟩, Consumer ⟨2⟩, ...

In general, each agent owns a set of protocol instances, which participate at the conversation. The set of protocol instances of one agent belonging to one conversation is

$$Conv_i = P_{i,1} || \ldots || P_{i,n_i}$$

A conversation is formed by the behaviour of all protocol instances:

$$\text{Conversation } = Conv_1 || \ldots || Conv_n$$

In general, the set $Conv_i = P_{i,1} || \ldots || P_{i,n_i}$ contains more than one instance of one protocol.

6.1 Phase Ordered Conversations

If conversations are finite, i.e. in fact only a finite number of instances are involved, correctness can be described in terms of a well funded order on instances. In the case of infinite conversations such an order cannot exist, so a different criterion for correctness has to be defined.

A infinite conversation is described by the composition of infinite many instances:

$$\text{Conversation } = (P_1\langle 1\rangle || P_2\langle 2\rangle || \ldots) || \ldots || (P_n\langle 1\rangle || P_n\langle 2\rangle || \ldots)$$

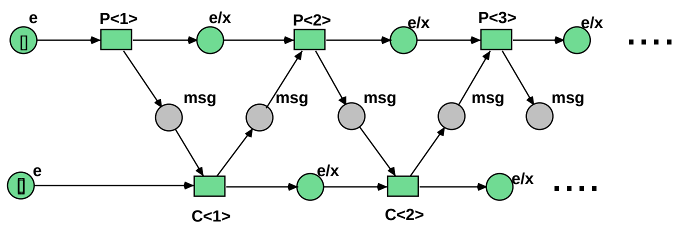

Figure 14: An infinite conversation between two agents

Correctness of infinite conversation intuitively describes some progress. The infinite conversation depicted in figure 14 has some kind of progress. There, the interaction between two partners is described. One partner uses the producer protocols P, the other the consumer protocols C. Each activated instance is described by one transition, named $P\langle i\rangle$ resp. $C\langle i\rangle$. Places named e or e/x describe states between two activated instances, places named msg describe messages in transit.

As one can observe, there is no S-cut in the process, which consists only of e/x-places, since in each S-cut also one message is included. So, no global state exists, that could naturally describe the "next" state of the conversation.

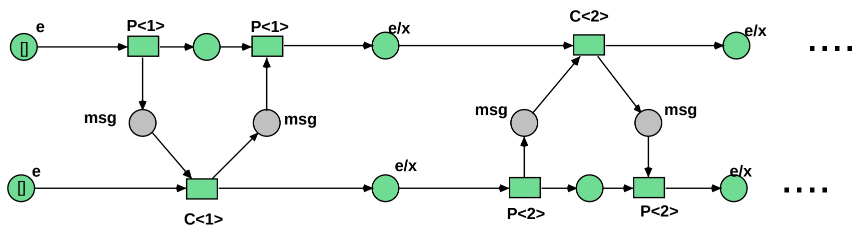

Figure 15: A phase structured infinite conversation

To obtain a definition for progress of infinite conversations, we postulate that in each process there are infinite many S-cuts, that consist only of e/x-places. All states between two adjoining e/x-cuts naturally form one single entity, which is called a *phase*. Thus, correctness is the infinite sequence of phases. This characterisation of communication is inspected in the field of communication-closed-layers (cf. [EF82], [FPZ93] und [SdR94]):

$$(P_{1,1}; \ldots; P_{1,m})||\ldots||(P_{n,1}; \ldots; P_{n,m}) = (P_{1,1}||\ldots||P_{n,1}); \ldots; (P_{1,m}||\ldots||P_{n,m})$$

In the case of reactive agent protocols, this notion could be extended onto the infinite case, thus describing the reactive protocols:

$$\begin{aligned} & (P_{1,1}; \ldots; P_{1,m}; \ldots)||\ldots||(P_{n,1}; \ldots; P_{n,m}; \ldots) \\ = \; & (P_{1,1}||\ldots||P_{n,1}); \ldots; (P_{1,m}||\ldots||P_{n,m}); \ldots \end{aligned}$$

Figure 15 shows an example for an infinite process of a conversation, which is structured in phases. In each phase there are two active protocol instances: the i-th phase consists of the interaction of $P\langle i\rangle$ and $C\langle i\rangle$. In this conversation the assignment of protocols to agents is not fixed, as one can see in Fig. 15: the relation is switched by a phase change. Each phase has one producing and one consuming agent. The roles, in which they act, are switched from phase to phase.

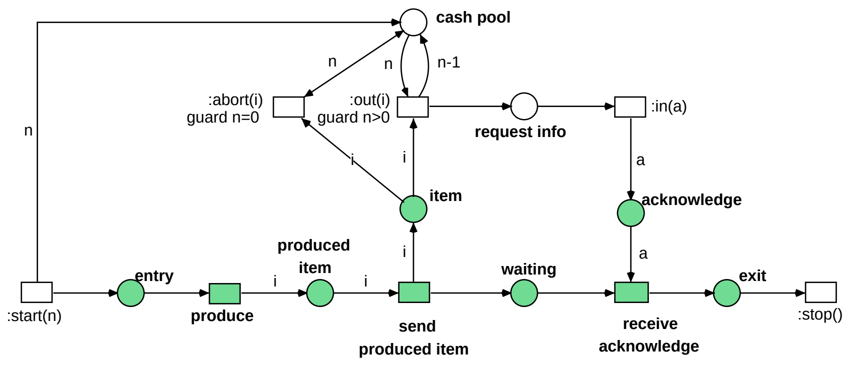

Figure 16: The produce protocol with phase order

Phase structuring of conversation gives us a progress criterion for infinite conversation, since it reduces the progress criterion for the whole conversation towards an phase-local criterion. If the initial phase terminates and each terminating phase is followed by an terminating one, the correctness of the whole conversation follows inductively. The infinite reproduction of phases could be formally described in terms of liveness analogous to the correctness definition for work-flow nets [Aal97].

In general, termination of one single phase is described in terms of an order, which we call *phase order*. Each agent owns a fix amount of "message cash" for paying the sending and receiving of messages. Phase and conversation orders are integrated in the formalism of SAM (cf. figure 16 and 17). Each A/C-protocol has an additional place cash pool, which contains the actual cash units. Cash units are used to restrict the message transfer: Sending of messages subtracts – if $n > 0$ – an amount of one cash unit from the actual cash amount $n \in \mathbb{N}$. This is expressed by the guard: (guard $n > 0$). If not enough cash units are present (guard $n = 0$) the conversation is aborted via the channel : abort.

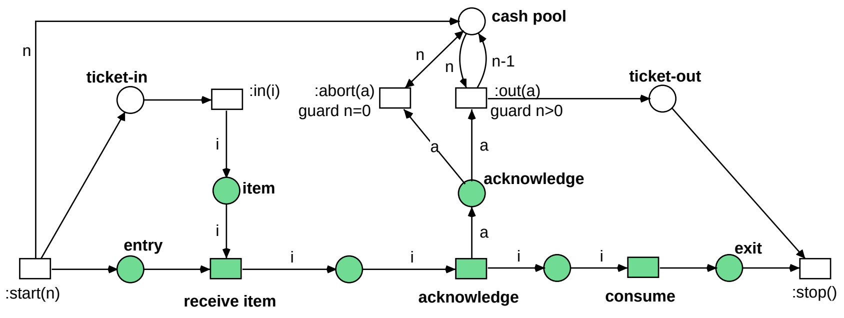

Figure 17: The consumer protocol with phase order

Since the number of protocols per phase is fixed in all processes the marking of the place cash pool is strictly monotone decreasing. So, we can conclude that if a protocol terminates under the assumption that all request messages will be replied by the environment, the whole conversation must terminate.

6.2 Dynamic Reconfiguration

Since agent protocols are aware of the phase order, they can make use of it: The phase ordering can be changed when the conversation steps forward to the next one. Agents could negotiate how this should be done. The process is controlled by a *meta-protocol* which controls the conversation and the dynamic reconfiguration between phases. Such a meta-protocol is shown in figure 18, where the process of instantiating, stopping, and reconfiguring of a protocol is specified.[17] The protocol net is a token ppn of the net. The transition instantiate creates a new instance, called ppn, by the expression ppn: new acppn. The protocol type acppn is passed by as an parameter. The instance is started (ppn:start(n)), where the phase order is described by the initial cash amount

[17]Figure 18 is only for demonstration purposes. In fact, this meta protocol is part of the MULAN-agent itself, which controls the conversation and the protocol instances as described before (cf. Fig. 6).

n. When the protocol instance has finished its job, it is stopped (ppn:stop()) and discarded. The actual phase is finished now. The transition reconfigure switches then to the next phase by setting the protocol type to new_acppn and assigning new_cash to the cash pool.[18] Then the cycle could be restarted again.

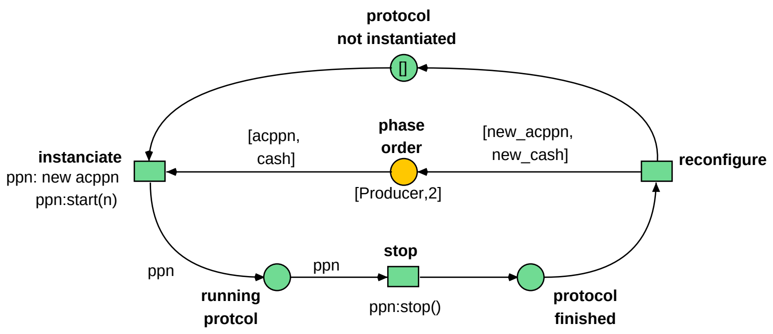

Figure 18: Conversation meta protocol

In general, a conversation is *well formed* iff each agent's conversation meta protocol is live, since this guarantees the progress of phases. The characterisation that there are an infinite number of e/x-cuts is correct, cause this is equivalent to the existence of an infinite number of reconfigure-cuts, so, each reconfigure transition fires exactly once per phase. The intention of the well formedness definition is similar to those for work-flow nets in [Aal97]. Our definition is more general, since we first have to cope with several nets and not only a single one, and second we have to deal with reactive nets, which communicate with their environment, which is not present in the context of [Aal97].

7 Conclusion and Outlook

In this presentation, three concepts are shown to be central for the modelling of mobility, adaptivity, and cooperation in multi agent systems, namely: the local elements, the global view, and the composition structure, together with their relationships.

Here, we concentrated on the field of coordination, which is modelled as an interplay of the concepts: conversation, A/C-protocols, and conversation orders. Due to the dynamic reconfiguration mechanism implemented in the agent system architecture MULAN the question of correctness is raised.

In this work we propose liveness preserving composition of agent protocol nets in terms of conversation orders. Here, the special case of phase-ordered communication in combination with protocol nets is presented. For this case, liveness turns out to be a structural property of the composition of A/C-protocol nets by a conversation order. This result is needed to combine agent protocols at run-time.

[18]In the concrete implementation the values of new_acppn and new_cash are chosen with respect to the knowledge base of the agent.

Our results are embedded in the more general context of the development of a Petri net based architecture for multi agent systems, called Sam. Further work has to be done to formally establish a notion for structures of multi agent platforms, communication behaviour etc. This work is supposed to lead into a complete calculus for multi agent systems, that hopefully allows more general composition than just phase-oriented composition.

References

[Aal97] Wil van der Aalst. Verification of workflow nets. In Pierre Azeme and Gianfranco Balbo, editors, *Application and theory of Petri nets*, volume 1248 of *Lecture Notes in Computer Science*, pages 407–426, Berlin Heidelberg New York, June 1997. Springer Verlag.

[AFdR80] K. R. Apt, N. Francez, and W.P. de Roever. A proof system for communicating sequential processes. *ACM Transactions on Programming Languages and Systems*, 2(3):359–385, 1980.

[BBC+99] B. Barras, S. Boutin, C. Cornes, J. Courant, Y. Coscoy, D. Delahaye, D. de Rauglaudre, J.-Ch. Fillitre, E. Giménez, H. Herbelin, G. Huet, H. Laulhère, C. Muñoz, C. Murthy, C. Parent-Vigouroux, P. Loiseleur, Ch. Paulin-Mohring, A. Sabi, and B. Werner. *The Coq Proof Assistent – Reference Manual, Version 6.3.1, Coq Project*, December 1999. http://pauillac.inria.fr/coq.

[BDM+99] M. Bozzano, G. Delzanno, M. Martelli, V. Mascardi, and F. Zini. Logic Programming & Multi-Agent Systems: a Synergic Combination for Applications and Semantics. In K.R. Apt, V.W. Marek, M. Truszczynski, and D.S. Warren, editors, *The Logic Programming Paradigm: a 25-Year Perspective*, pages 5–32. Springer Verlag, 1999.

[Ber87] G. Berthelot. Transformations and decompositions of nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri nets: Central models and their properties*, number 254/255 in LNCS. Springer-Verlag, 1987.

[BGV91] Wilfried Brauer, Robert Gold, and Walter Vogler. A survey of behaviour and equivalence preserving refinements of Petri nets. In G. Rozenberg, editor, *Advances in Petri nets 1990*, volume 483 of *Lecture Notes in Computer Science*, pages 1–46, Berlin, Germany, 1991. Springer-Verlag.

[BOP00] 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.

[BT87] Eike Best and P.S. Thiagarajan. Some classes of live and safe Petri nets. In Klaus Voss, editor, *Concurrency and nets*, pages 71–94, Berlin, Germany, 1987. Gesellschaft für Mathematik und Datenverarbeitung, Springer-Verlag.

[CCF+99] 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 conversation policies (Autonomous agents '99)*, 1999.

[CFL$^+$98] 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.

[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.

[EC82] E. A. Emerson and E. M. Clarke. Using branching time temporal logics to synthesize synchronisation sceletons. *Sci. Comput. Program.*, 2:241–266, 1982.

[EF82] T. Elrad and N. Francez. Decomposition of distributed programs into communication closed layers. *Science of computer programming*, 2:155–173, 1982.

[Far00] Berndt Farwer. *Linear Logic Based Calculi for Object Petri Nets.* Logos Verlag, Berlin, 2000.

[FIP] Foundation for Intelligent Physical Agents. `http://www.fipa.org`.

[FIP98a] FIPA. FIPA 97 Specification, Part 1 - Agent Management. Technical report, Foundation for Intelligent Physical Agents, `http://www.fipa.org`, Oktober 1998.

[FIP98b] FIPA. FIPA 97 Specification, Part 2 - Agent Communication Language. Technical report, Foundation for Intelligent Physical Agents, `http://www.fipa.org`, Oktober 1998.

[FL97] Tim Finin and Yannis Labrou. A Proposal for a new KQML Specification. Technical report, University of Maryland, Februar 1997.

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

[FPZ93] M. Fokkinga, M. Poel, and J. Zwiers. Modular completeness for communication closed layers. In E. Best, editor, *Proceedings CONCUR'93*, volume 715 of *LNCS*, pages 50–65, Hildesheim, Germany, 23–26August 1993. Springer.

[GPdFC98] Gustavo M. Gois, Angelo Perkusich, Jorge C. A. de Figueiredo, and Evandro B. Costa. Towards a multi-agent interactive learning environment oriented to the Petri net domain. In *Proc. IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC'98), 11-14 October 1998, San Diego, USA*, pages 250–255, October 1998.

[Gri98] Frank Griffel. *Componentware: Konzepte und Techniken eines Softwareparadigmas.* dpunkt Verlag, 1998.

[HKL$^+$00] Daniela Hinck, Michael Köhler, Roman Langer, Daniel Moldt, and Heiko Rölke. Akteurstheoretische Betrachtungen organisationaler Handlungen. Arbeitsberichte des Forschungsprogramms: Agieren in sozialen Kontexten, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln Str. 30, 22527 Hamburg, Germany, 2000. Beitrag auf dem Workshop Sozionik 2000.

[HKL$^+$01a] Daniela Hinck, Michael Köhler, Roman Langer, Daniel Moldt, and Heiko Rölke. Modelling a sociological case study. In *Modelling Artificial Societies and Hybrid Organisations (MASHO'01)*, 2001.

[HKL$^+$01b] Daniela Hinck, Michael Köhler, Roman Langer, Daniel Moldt, and Heiko Rölke. Organisation etablierter Machtzentren: Modellierungen und Reanalysen zu Norbert Elias. Arbeitsberichte des Forschungsprogramms: Agieren in sozialen Kontexten 306, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln Str. 30, 22527 Hamburg, Germany, 2001.

[HKL$^+$02] Daniela Hinck, Michael Köhler, Roman Langer, Daniel Moldt, and Heiko Rölke. Soziologische Grundlagen des Handelns modelliert mit Petrinetzen. Arbeitsberichte des Forschungsprogramms: Agieren in sozialen Kontexten, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln Str. 30, 22527 Hamburg, Germany, 2002.

[HKMM00] Sven Heitsch, Michael Köhler, Marcel Martens, and Daniel Moldt. High level Petri nets for a model of organisational decision making. In *Proceedings of the Workshop HLPN 2000*, 2000.

[Hoa69] C.A.R. Hoare. An axiomatic basis for computer programming. *Communication of the ACM*, 12:576–580, 1969.

[HR00] Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and reasoning about systems.* Cambridge University Press, 2000.

[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.

[Jen00] Nicholas R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 117:277–296, 2000.

[Jon83] Cliff B. Jones. Specification and design of (parallel) programs. In R. E. A. Mason, editor, *Information processing*, pages 321–332. Elsevier Science Publishers B.B., 1983.

[KLMR00] Michael Köhler, Roman Langer, Daniel Moldt, and Heiko Rölke. Combining the sociological theory of Bourdieu with multi agent systems. In C. Jonker, A. Letia, G. Lindemann, and T. Uthmann, editors, *Modelling Artificial Societies and Hybrid Organisations (MASHO'00), Workshop at the ECAI 2000*, 2000.

[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.

[KR01a] Michael Köhler and Heiko Rölke. A/c petri nets - assumption based modelling and reasoning. In Gabriel Juhas and Robert Lorenz, editors, *Proceedings des 6. Workshop Algorithmen und Werkzeuge für Petrinetze*, pages 43–48. Universität Eichstätt, 2001.

[KR01b] Michael Köhler and Heiko Rölke. Towards a unified approach for modeling and verification of multi agent systems. In D. Moldt, editor, *Proceedings of the Workshop on Modelling, object, componets, and agents (MOCA'01)*, pages 85–104. Computer Science Department, Aarhus University, 2001.

[KRVW97] E. Kindler, W. Reisig, H. Völzer, and R. Walter. Petri net based verification of distributed algorithms: an example. *Formal Aspects of computing*, 9:409–424, 1997.

[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, 2002.

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

[MC81] Jayadev Misra and K. Mani Chandy. Proofs of networks of processes. *IEEE Transactions on Software Engineering*, 7(4):417–426, 1981.

[Mey97] Bertrand Meyer. *Object-oriented software construction.* Prentice Hal, 1997.

[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.

[Rei98] Wolfgang Reisig. *Elements of distributed algorithms. Modelling and analysis with Petri nets.* Springer-Verlag, 1998.

[RJB99] J. Rumbaugh, I. Jacobson, and G. Booch. *The unified modeling language reference manual: The definitive reference to the UML from the original designers.* Addison-Wesley object technology series. Addison-Wesley, Reading, Mass., 1999.

[SdR94] F. Stomp and W.-P. de Roever. Designing distributed algorithms by means of formal sequentially phased reasoning. *Formal Aspects of Computing*, 6(6):716–737, 1994.

[SHM99] A. El Fallah Seghrouchni, S. Haddad, and H. Mazouzi. A formal study of interactions in multi-agent systems. In *14th ISCA-CATA*, Cancun, Mexique, April 1999.

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

[Val96] Rüdiger Valk. On processes of object Petri nets. Technical Report FBI-HH-B-185/96, Universität Hamburg, FB Informatik, 1996.

[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. pages 47–77, 1998.

[Vig98] Giovanni Vigna, editor. *Mobile Agents and Security*, volume 1419 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[vLMV01] Rolf von Lüde, Daniel Moldt, and Rüdiger Valk. Agieren in sozialen Kontexten. http://www.informatik.uni-hamburg.de/TGI/forschung/projekte/sozionik, 2001. Universität Hamburg: Institut für Soziologie und Fachbereich Informatik.

[Wei99] Gerhard Weiss, editor. *Multiagent systems*. MIT Press, 1999.

[Woo99] Michael Wooldridge. Intelligent agents. In Weiss [Wei99], chapter 1.

[Zwi89] Job Zwiers. *Compositionality, concurrency, and partial correctness: proof theories for networks of processes and their relationship*. LNCS 321. Springer-Verlag, 1989.