A Colored Petri Net for a Multi-Agent Application

Danny Weyns, Tom Holvoet

Computer Science Department, Katholieke Universiteit Leuven

Celestijnenlaan 200A, B-3001 Leuven, Belgium

http://www.cs.kuleuven.ac.be/~danny/home.html

{danny,tom}@cs.kuleuven.ac.be

Abstract

In this paper we present a Colored Petri Net (CPN) for a multi-agent application. In particular we modeled the Packet-World. In our research we use the packet-world as a case to study the fundamentals of agents' social behavior. Our approach is to combine experiments with conceptual modeling. We start from a very basic model and then add social skills in a modular way. Integrating new social skills by means of adding new modules offers us a clear conceptual view on the evolution of agents and the environment. With a conceptual view we mean: (i) which concepts does an agent need in order to acquire a new kind of social ability, (ii) which infrastructure is necessary in the environment to support these abilities, (iii) how do these concepts relate to each other? With the insights we learn from the case study, we gradually develop a generic conceptual model for social agents situated in a MAS. In this paper we first present a CPN for a basic model of the packet-world. This model consists of agents that can only interact through passive objects in the environment. Because interaction is the central issue of multi-agent systems, we have incorporated basic infrastructure for agent coordination straight away into our basic model. Then we extend the model, making it possible for the agents to communicate information with each other. Communication is the basis for social organization. Besides the concrete realization of a CPN for a multi-agent application, the model we present in this paper has the potential to support our future research of agents' social behavior. Our major motives for using CPNs as modeling tool are (i) CPNs gives a clear conceptual view on agents and the environment wherein they live, and (ii) CPNs support neat verification and formalization.

Keywords

Colored Petri Nets, Multi-Agent Systems

1. Introduction

In this introduction we first situate our research goals in the domain of Multi-Agent Systems (MAS). Next we explain how we tackle the problems we want to solve and motivate our choice for Colored Petri Nets (CPN) as a tool for modeling MASs. Then we situate the subject of this paper in our research. We conclude with an overview of the paper.

1.1 Fundamentals of Sociality in MASs

The importance of MASs as a design concept for today's software is beyond dispute. A MAS models a part of the world as a community of autonomous agents that interact in an environment. In a MAS the activity is distributed over the agents of the community. The intelligence of the systems comes from the interaction between the agents, rather then from their individual capabilities. This contrasts to the approach of the classical artificial intelligence where an agent acts as an independent "cognitive reasoning machine".

A MAS is a society of agents that live and work together. Living in a community requires a number of social skills. Until now the agent research community has paid little attention to the fundamentals of sociality in MASs. Many unanswered questions remain. The lack of insight in agents' sociality limits the potential of the agent-based approach. We quote N. Jennings in [6]:

> *"To realize the full potential [of MASs], a better understanding is needed of the impact of sociality [..] on an individuals behavior and of the link between the behavior of the individual agents and that of the overall system."*

Until now, research about the fundamentals of agents' sociality can be divided into two approaches. In the first approach research is mainly experiment-oriented. Some references are [8][9][10][11][12]. These projects explore new kind of interactions and rules for setting up social structures. What we can see is that from the interaction of the agents new functionality's emerge that go beyond the sum of the capabilities of the individuals. The second approach intends to conceptualize the social aspects of agents in a MAS. Some examples are [6][13][14][15]. One group of researchers has integrated certain aspects of agents' sociality in a formal model of an agent (e.g. the BDI-model). Another group has set out some thoughts how agents' sociality and the organization of a MAS might be structured.

We conclude that most of the work that has been done so far, has one or more of the following characteristics:

- the research focuses on one particular aspect of sociality in MAS
- the research starts from a particular point of view
- most of the research is done separately from one another
- the focus is mainly directed on *how* social behavior could emerge in a MAS, much less attention is devoted to the questions *why* and *when* social behavior arises

1.2 The goals of our research

The goal of our research is to get a better understanding of sociality in MASs. Therefore we intend to build a *generic conceptual model* of social agents situated in a MAS. Our approach is to combine experiments with conceptual modeling, the two approaches we mentioned in the previous section. We use a case application to explore different kind of social behaviors. Parallel with experiments we build a conceptual model. We start from a very basic model and then add social skills in a modular way. Integrating new social skills by means of adding new modules offers us a clear conceptual view on the evolution of agents and the environment. With a conceptual view we mean: (i) which concepts does an agent need in order to acquire a new kind of social ability, (ii) which infrastructure is

necessary in the environment to support these abilities, (iii) how do these concepts relate to each other? With the insights we learn from the case study, we gradually will develop a generic conceptual model for social agents situated in a MAS. Therefore we have to generalize the insights we learned from the case application in order to build abstract models for different classes of social skills.

1.3 Modeling MASs with Colored Petri Nets

When we set out our approach the question arises how we should model the case application. Since Petri nets [1] have a long tradition to describe and analyze concurrent processes, they where excellent candidates. Colored Petri Nets (CPN) [2] combine the best of classical Petri nets and high level programming languages, and are for that very popular. CPNs have an intuitive graphical representation that paves the way for clear conceptual modeling of complex systems. The behavior of a system modeled with a CPN can be analyzed, not only by means of a simulation but also on a formal base. It is remarkable that CPNs, which offer most of the ingredients to tackle the complexity of multi-agent systems, are little used to model and study them. Some interesting references are [16][17][18][19][20][21][22]. The most far-reaching use of CPNs for modeling MASs is from Ferber. Ferber developed a formalism called "Basic Representation of Interactive Components" (BRIC). BRIC is based on a component approach, each of the primitive components ("bricks") described with a CPN. In his standard work "Multi-Agent Systems" [3], Ferber proposes an extensive set of BRIC components, each of them representing a generic model for a specific part of a MAS. Inspired by his ideas we decided to use CPNs in our research. In contrast with Ferber, who uses CPNs for an *operative representation of the functioning of a MAS* we use CPNs for a *conceptual modeling of sociality in MASs*.

1.4 Situating the paper in our research, overview of the paper

The multi-agent application we use in our research is that of the *Packet-World*. Originally, Huhns and Stephens proposed this application in [7] as a research topic to investigate sociality in MASs. The packet-world consists of a number of different colored packets that are scattered over a rectangular grid. Agents that live in this virtual world have to collect those packets and bring them to their corresponding colored destination. The agents have only a limited view on the world. The packet-world offers a rich set of fundamental characteristics for a broad range of multi-agent systems. E.g., agents may perform better their job when they share their information or when they set up a form of cooperation. In this paper we describe two models of the packet-world. These two models form a solid basis for our future research of agents' social behavior. After an intuitive description of the packet-world, in section 3 we present a CPN for a basic model. This model consists of agents that can only interact through passive objects in the environment. Next in section 4, we extend the model, making it possible for the agents to communicate information with each other. Communication is the basis of social organization. In section 5 we give results of our first experiments with the two models. Finally, we conclude and look to future work in section 6.

The CPNs that we present in this paper are designed with the Design/CPN tool [4][5]. In order to keep a clear view on the models, we limit the number of agents to two.

2. The packet-world

2.1 Introduction

Consider a rectangular grid of size S. The grid contains a number of colored packets and agents. It is the agents' job to collect the packets and bring them to their corresponding colored destinations. The grid contains one destination for each color. Figure 1 shows an example of a packet-world of size 8 with 3 agents.

In the packet-world agents can interact with the environment in a number of ways. We allow agents to perform a number of basic interactions with the passive objects of the environment. First, an agent

can make a step to one of the free neighbor fields around him. Second, if an agent is not carrying any packet, he can pick up one from one of his neighbor fields. Third, an agent can put down the packet he carries on one of the free neighbor fields around him or of course on the destination field of that particular packet. Finally an agent may wait for a while and do nothing.

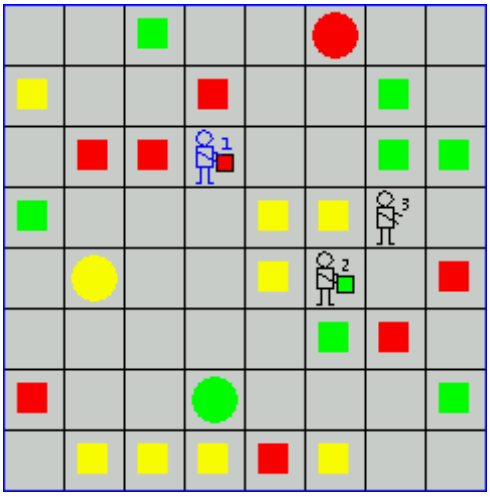

Figure 1. The packet-world (squares are packets, circles are delivering points)

It is important to notice that each agent has only a limited view on the world. This view covers only a small part of the environment around the agent. This property of limited knowledge is typical for the agents of a multi-agent system. In our model, the *view-size* of the world expresses how far (i.e. how many squares) an agent can "see" around him.

In our model we use a simple measure to indicate how efficient the agents perform their job. Each time an agent makes a step or moves a packet (by picking it up, putting it down or step with it) a counter is incremented. At each point in time the value of this counter indicates how much energy the agents have invested in their work so far.

In the basic model for the packet-world we limit the agents possible interactions with the environment to the basic set we mentioned above. We modeled the basic agents without any social skills. Their *goal* is to collect the packets of the world and bring them to their destinations. This general goal can be divided into a set of *primary goals*. In short, those agents act in a repeating cycle driven by two primary goals: look for a packet and pick it up, look for the destination and deliver the packet.

In the extended model, agents can interact with each other. This interaction is the foundation of cooperation between the agents. For the packet-world one can imagine different kinds of cooperation. Agents can for example agree on a plan to form a chain and pass on packets to each other. We modeled another form of cooperation. In the extended model we present in this paper, we integrated facilities into the basic model to let agents communicate with each other. In particular, agents are extended with functionality to request information from each other. Instead of exploring the world to find a target an agent does not see, the agent now can ask a visible colleague for the desired information. If the requested agent knows or sees the asked information he can respond the query with the information. This allows the requesting agent to act more efficient.

2.2 Actions, influences and reactions

Agents of a multi-agent system are endowed with autonomy. They are driven by a set of goals. In order to achieve those goals agents undertake *actions*. When an agent acts in the environment, e.g. he picks up a packet next to him he has no full guarantee that this action will succeed. Another agent might be trying to pick up the same packet at about the same time. As a consequence only one of

them will get the packet leaving the other with empty hands. Therefore we say that an action of an agent results in an *influence* in the environment. Influences result in *reactions* from the environment. Each influence can succeed or fail. For example if an agent performs the action "pick" he invokes the influence "perform pick" on the environment. If the action succeeds, the environment reacts with the reaction "do pick", if the action fails the reaction will be "can't pick". So it is only after the reaction of the environment to all the performed influences (at about the same point of time) that the agents actually experience the result of their intended actions. When an agent is notified about the result of the action he undertook, we say that the agent *consumes* the result of his produced influence.

2.3 Agent state

An agent can only decide to perform an action if he is endowed with some attitudes and has some information at his deposal. In section 2.1, we mentioned an agent is driven by a set of primary goals. Agents act to achieve those goals. Therefore they perform influences into the environment, as described in section 2.2.

When an agent selects an action, he has to take the state of the world into account. If an agent for example decides to pick up a packet, first of all, he must be aware of the fact he actually does not carry a packet. In general this means the agent must possess some state of his own. In our model of the packet-world an agent maintains the state of his position and whether he actually carries a packet or not. Further, the agent must "see" the packet near to him. He needs some information about the environment around him in order to act. We call this information the *view* of the agent. In our model, regularly each agent gets an update of its own view on the world. As mentioned in section 2.1, this view covers only a small part of the environment around the agent. A special synchronization module is responsible for the timing of the updates of the agents' views. We explain this synchronization process in detail later.

As an alternative, the agent might "know" something about the world in order to take a decision what to do. It is for example not necessary that an agent "sees" the destination for his packet if he "knows" the location of the destination. In our model we therefore endowed an agent with a *belief base*. This belief base contains records with information that the agent has collected in the past. It is clear that some of this information is volatile. A destination for a particular color of packets will never change, but a packet located at a certain field might have disappeared after a while. In our model agents "know" which beliefs unconditionally can be trusted and which are not trustable. Agents revise suspicious beliefs as soon as they get information about them from their percept update, i.e. as soon as the subject of the belief comes inside a certain range of their vision.

2.4 A job and the states of the world

At start time the packet-world is in an initial state. Packets are scattered over the grid, and the agents are located between them. The counter that measures the agents' performance is initialized to zero. We define a *job* as the task of the agents to collect all the packets and deliver them at the right destinations. A job starts when a synchronization module triggers the environment to send the agents their view. Driven by their goals, agents select an action to perform. These actions result in a transformation of the state of the packet-world. This cycle repeats until the whole grid is cleared. Each time the action of an agent modifies the state of the world the performance counter is increased. As soon as all packets are delivered at their destination the synchronization module stops the process of updating the view of the agents. This informs the agents that the job has come to an end. The packet-world is then in the end state. The transformation of the world can be described as a dynamic process that transforms the initial state of the world along a sequence of discrete states into the final state by means of performing synchronized actions of the agents.

2.5 Conflict resolution and synchronization

When the agents act, the environment reacts. Thereby it takes the influences of the agents into account and produces a new consistent world. In our model we distinguish between two levels of

synchronization. First we have synchronization at the level of concurrent actions. We call this *system synchronization.* This level of synchronization guarantees that the "laws of the world" are respected. For example, only one agent at a time can step to a particular free field. System synchronization is implicitly integrated into a CPN. A second level of synchronization is situated at a higher level of interaction between agents. We call this the level of *functional synchronization*. Functional synchronization offers support for coordination of actions between agents. All actions of the agents are synchronized in *action cycles* as shown in Figure 2.

A cycle starts with updating the perception of each agent. Based on its state and the new percept he receives, each agent then can reason about what he wants to do. The agent selects an action and produces an influence invoked on the environment. The environment calculates the reactions of all performed influences and notifies the agents by means of a consumption for each of them. As soon as all reactions are completed the environment will be triggered to calculate a new percept for each agent, and that starts a new action cycle. In our model functional synchronization is realized by means of the synchronization module.

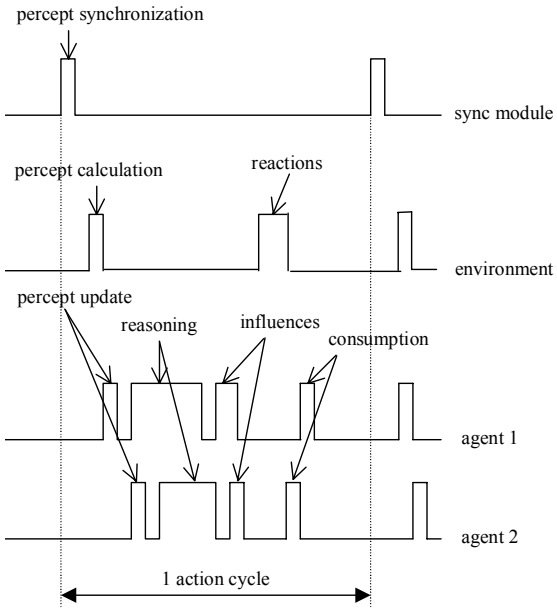

Figure 2. Functional synchronization.

One might wonder why we decided to introduce functional synchronization. After all it limits the freedom of action of the agents. Agents are no longer allowed to handle on their own rhythm. But this is just the point. The problem solving power of a multi-agent system arises from the interaction between the agents of the system. In order to cooperate, agents have to coordinate their actions. Coordinating actions between two (or more) autonomously running agents is hard to achieve. Therefore we introduced functional synchronization. For the price of some individual freedom we offer the agents a clean framework to coordinate their interactions. We fit in communication into this model. This means that sending a question or responding to it by means of sending an answer, are both integrated into the extended model as first class actions.

3. Modeling the basic components of the packet-world

In this section we discuss how we modeled the basic model of the packet-world by means of a CPN. First we give a high level overview of the model. This identifies the different modules of the multi-agent system. Next we discuss the CPN for each separated module.

We bring the separated modules together in a global net after discussing the integration of communication into the basic model in section 4.

3.1 High level model

We have divided the basic model for the packet-world into three separated modules, each representing one fundamental component of the packet-world. We distinguish between the environment (box), the agents (rounded boxes) and the synchronization module (diamond).

As shown in Figure 3 agents in the basic model only interact with the environment. The white arrows represent the influences performed by the agents. The gray arrows represent the consumptions and percepts for the agents. The synchronization module regulates the updates of the latter.

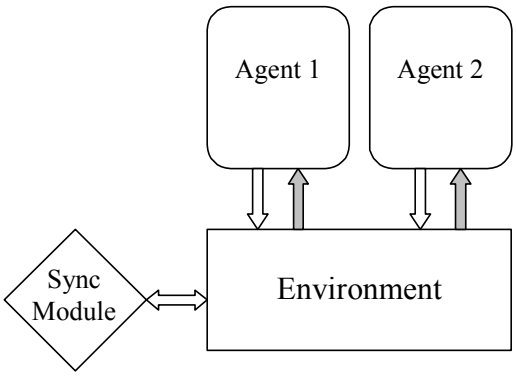

Figure 3. High level basic model with 2 agents.

Before we go into the separate CPNs for the different modules we first have to tell something about our approach for modeling modularity. In each module there are two kinds of places. There are circles that represent *internal places* and ovals that represent *interface places*. Interface places are similar to the notion of *fusion places* as defined in [4].

Different modules can be combined with each other by merging overlapping interface places. Note that we distinguished only between internal and interface places to indicate that some places of a module will overlap with similar places of other modules when a global CPN is composed. The graphical distinction has no particular semantic meaning related to places of a CPN in general.

3.2 Model of the environment

The environment models the world in which the agents live. For our packet-world we modeled the environment as one centralized entity. The agents can interact with the environment by means of a set of actions. The concurrent actions of the agents lead to the modification of the world. Agents are notified of those transformations by means of (i) consumptions (i.e. what they get from their invoked influences) and (ii) percepts (i.e. a partial view of the state of the world around the agent). The environment keeps track of how efficient the agents perform their job. We have modeled this efficiency tracker as a simple counter that is incremented each time an agent invests a relevant portion of energy, i.e. makes a step or moves a packet. Figure 4 shows the CPN for the environment.

The data of the environment is modeled as a token of the colorset *World*, located in the place ENVIRONMENT. This token is a list of *Item*, each item being a record with two components:

```
color Item = record name:Name * coord:Coordinate;
color World = list Item with 1..(worldsize*worldsize);
```

The performance efficiency of the agents at a certain point in time is modeled as a set of anonymous tokens collected in the place COUNT.

The reactions of the environment are modeled as transitions. A reaction takes an influence and the state of the world as input. In case the reaction produces a successful action the involved part of the world is modified. Otherwise the world is left untouched.

Furthermore a reaction produces a consumption for the agent that is sent to the corresponding interface place and a synchronization token that is sent to the synchronization module.

Whether an action ends successfully or not depends on the actual state of the world. This is tested by means of the guards of the transitions. Let us look at one example.

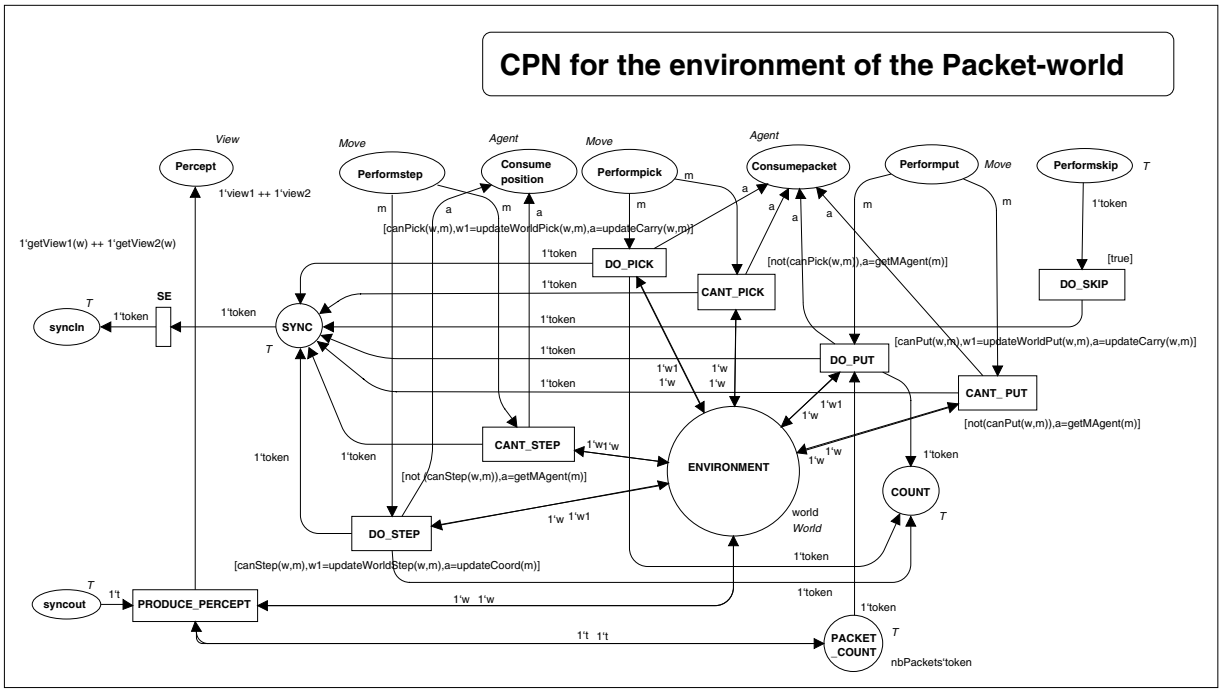

Figure 4. CPN for the environment of the packet-world.

Suppose an agent intends to make a step. Therefore he puts a token m of the colorset *Move* into the interface place "Performstep". The token m contains information about the identity of the agent and the coordinate of the square that he wants to step to. The reaction of the world will be one of the two following possibilities:

1. If the action succeeds the transition DO_STEP with the following guard will fire:

```
(*w models the world, m the invoked influence and a the consumption *)
[canStep(w,m),w1=updateWorldStep(w,m),a=updateCoord(m)]
```

2. If the action fails the transition CANT_STEP will fire:

```
[not(canStep(w,m)),a=getMAgent(m)]
```

In the first case the condition `canStep(w,m)` is fulfilled and the state of the world as well as the position for the agent are updated. In the second case `canStep(w,m)` fails. In this case the original location of the agent is copied into the consumption for the agent.

The last part of the environment concerns the production of the agents' percepts, modeled as the transition PRODUCE_PERCEPT. As soon as a token arrives at the "syncout" place of the synchronization module, this transition fires. It reads the world, produces the agents' updated views and puts them in the interface place "Percept". There the agents can pick them up. Note that the environment only produces percepts as long as the produce percept transition can read a token from the PACKET_COUNT place. Initially this place contains one anonymous token for every packet on the grid. Later on, each time an agent delivers a packet on its destination, one token is consumed from the packet count place. Finally when the latest packet is delivered there remain no longer tokens in the packet count place and that ends the production of new percepts.

3.3 Model of a basic agent

Agents are the active entities of the packet-world. Each agent is endowed with a number of operations in order to act in the environment. He can perceive information and use it instantly or

register it for later use. He can act in the environment and manipulate things. The CPN model for a basic agent is shown in Figure 5.

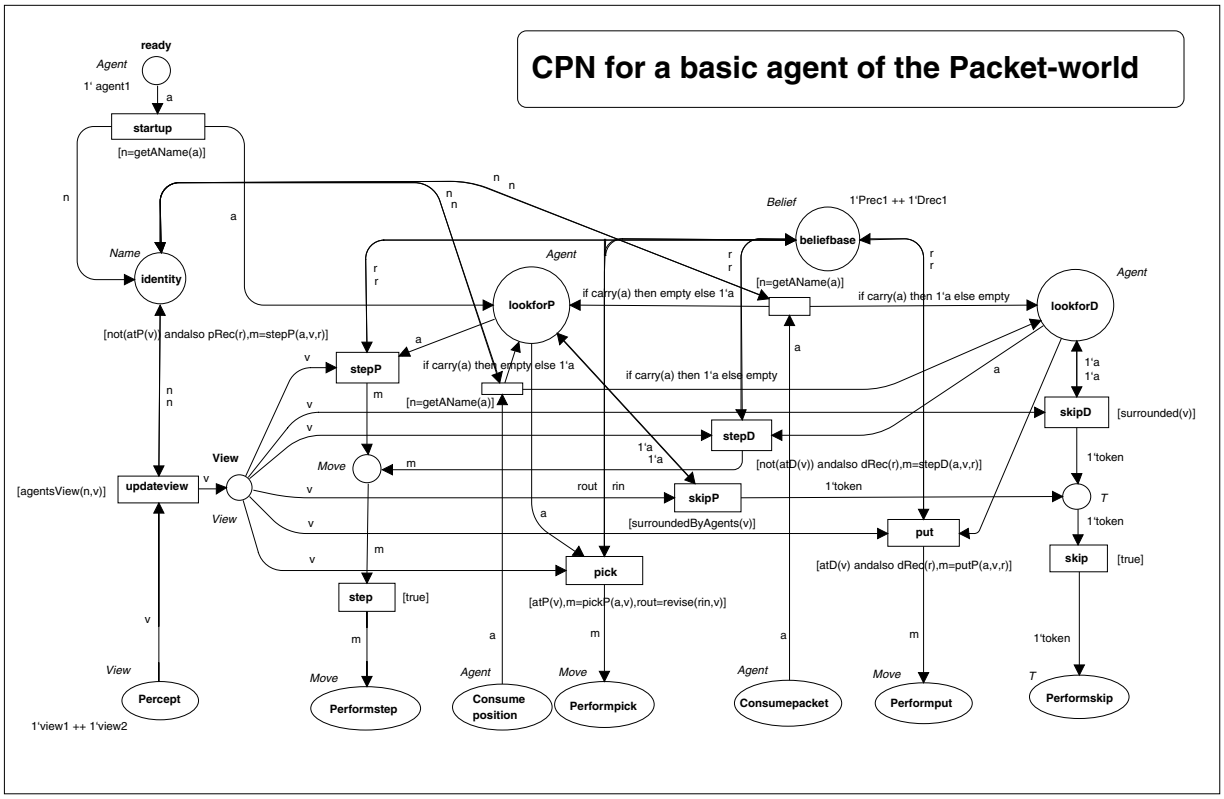

Figure 5. CPN for a basic agent of the packet-world.

All actions an agent undertakes are driven by a set of goals. The goals of our basic agent are quite limited. In case the agent hands are free he will look for a packet and pick it up. As soon as the agent holds a packet he will look for the destination and deliver it there. All actions available for an agent to fulfill its first goal (go for a packet) can only be started when a token of the colorset Agent is located in the place "lookforP". Performing one of the actions available to fulfill the second goal (deliver a packet) requires an Agent token in the "lookforD" place. An Agent token contains the state of the agent he maintains about himself. Such a token consists of three parts:

```
color Agent = record name:Name * coord:Coordinate * carry:Name;
```

Initially the Agent token is located in the place “ready”. When the execution starts, the “startup” transition fires. This passes the Agent-token to the “lookforP” place. At the same time the name of the agent is placed in the “identity” place. The agents’ identity will be used later on to dispatch the percepts of the environment to the various agents.

The state an agent maintains about the world around him is modeled as tokens of the colorset *Belief* stored in the place “beliefbase”:

```
color BeliefSubj = with pRec | dRec;
color Belief = record subj:BeliefSubj * item:Item;
```

In our basic model a belief contains information about an item of the world (for the definition of an Item, see section 3.2). We have provided two kind of beliefs, one for a packet (subj = pRec) and one for a destination (subj = dRec). Our basic agents actually use their belief base only passive and in a limited way. In fact they will only look for a packet or the destination of a packet in the base when they do not see it. So only when the programmer has given the agents some initial information (by

means of an initial marking of the belief base) they take profit of their belief base, otherwise it is of no help.

An agent trusts the beliefs about a destination but he revises beliefs about a packet as soon as he approaches the subject of the belief. This is done by means of the `revise(Belief,View)` function in the guard of the transition "pick".

The agents view on the world is modeled as tokens of the colorset *View*:

```
color View = list of Item with 1..(worldsize*worldsize);
```

In practice this list never contains all items of the world. The head of the list is always the item that corresponds to the agent himself. Thereafter the environment copies only the items around the agent in a range defined by the variable "view-size". Figure 6 illustrates the limited view an agent has on the world. In this example the size of the view is 2.

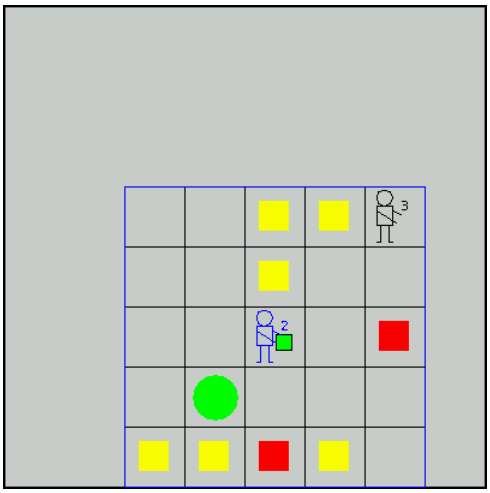

Figure 6. View on the world of Agent 2.

Now we discuss the action set of the agent. Each action is modeled as a transition. Such a transition consumes at least one token of the colorset Agent and one of the View set. Optionally the belief base is consulted. If the Agent token is located in the "lookforP" place the agent can make a step, pick up a packet or skip. If the token is in the "lookforD" place the agent can make a step, put its packet down or skip. In each case, the selection of the action is based on the criteria described in the guards of the transitions. We illustrate this for the action "put":

```
(* v models the view of the agent, a the intern data of the agent, r
one of the beliefs of the agent and m the invoked influence *)
[atD(v) andalso dRec(r),m=putP(a,v,r)]
```

The action "put" will be selected only if the agent is next to the destination of the packet he carries, i.e. the `atD(v)` condition. The record with possible information about the destination is selected from the belief base with the `dRec(r)` condition. If this record contains the coordinate of the destination, the agent creates an influence m at once, delivering its packet. If the destination is unknown he searches it from its actual view and creates a similar influence. The influence m is sent to the "Performput" interface place, where the environment takes it up for handling. An influence is modeled as a token of the colorset *Move*:

```
color Move = product Agent * Coordinate;
```

This tuple contains the Agents' identification (see section 3.3) and the coordinate (i.e. a tuple (x,y)) of the square where he intends to perform some influence. A Move token together with the place where it lands offers the environment enough information to determine the action an agent intends to

perform. The percepts from the environment come from the interface place "Percept". As soon as a new percept arrives the agent must identify himself in order to obtain the new information. Therefore the "updateview" transition reads the agents' identity. If there is a match, the view will be accepted and broadcast over the possible actions of which one is selected for execution during the next action cycle. The reactions to the influences are consumed from "Consumeposition" and "Consumepacket". Here a similar identification scenario is used. After accepting a consume the Agent token is directed to one of the main places "lookforP" or "lookforD" according to the fact the agent carries a packet or not.

3.4 Model for the synchronization module

The synchronization module models the notion of *functional synchronization* as we already described in section 2.5. It offers the agents an implicit framework for coordinating their interactions. Figure 7 shows the module with its connections to the environment.

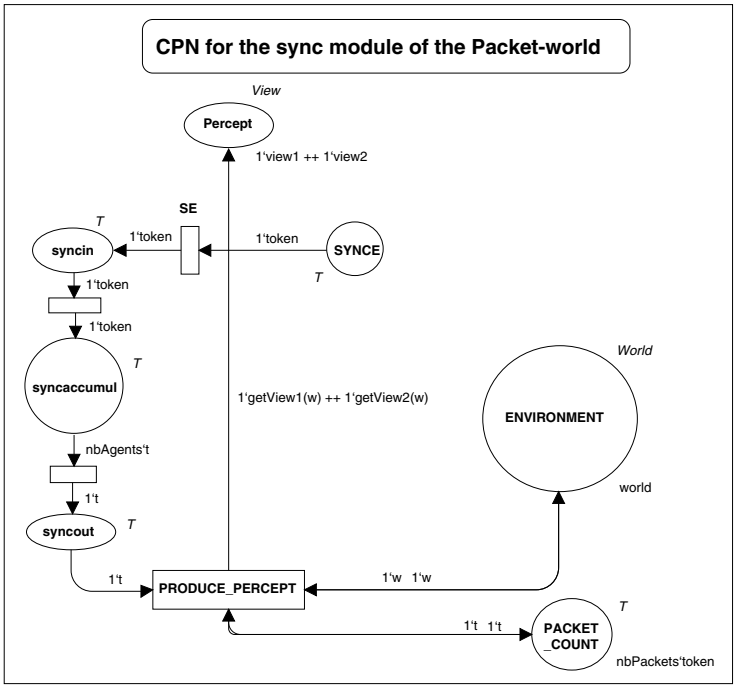

Figure 7. Synchronization module for the packet-world.

The behavior of the module is straightforward. Each time the agents performs their actions, the environment will react on them. For each reaction in particular, an anonymous synchronization token is produced and placed in the "SYNCE" place. These tokens are sent to the "syncin" interface place of the synchronization module. From there on, the tokens are collected in the "syncaccumul" place. When the actions for all agents are handled this place contains a number of tokens equal to the number of agents living in the packet-world. This triggers the output transition to fire, placing an anonymous token in the "syncout" place. On his turn this enables the PRODUCE_PERCEPT transition of the environment, such that new percepts can be calculated and sent to the agents. This starts a new *action cycle* (see section 2.5).

3.5 Complete CPN for the basic version of the packet-world

With de separated modules we now can compose the complete CPN for the packet-word. This model, depicted in Figure 8, gives a detailed picture of the high level model we presented in Figure 3. All interface places are combined according to the techniques we mentioned earlier in the paper. To keep a clear overview we limited the number of agents to two. In general however, a MAS may be composed of much more agents. In our model each agent has its own CPN module.

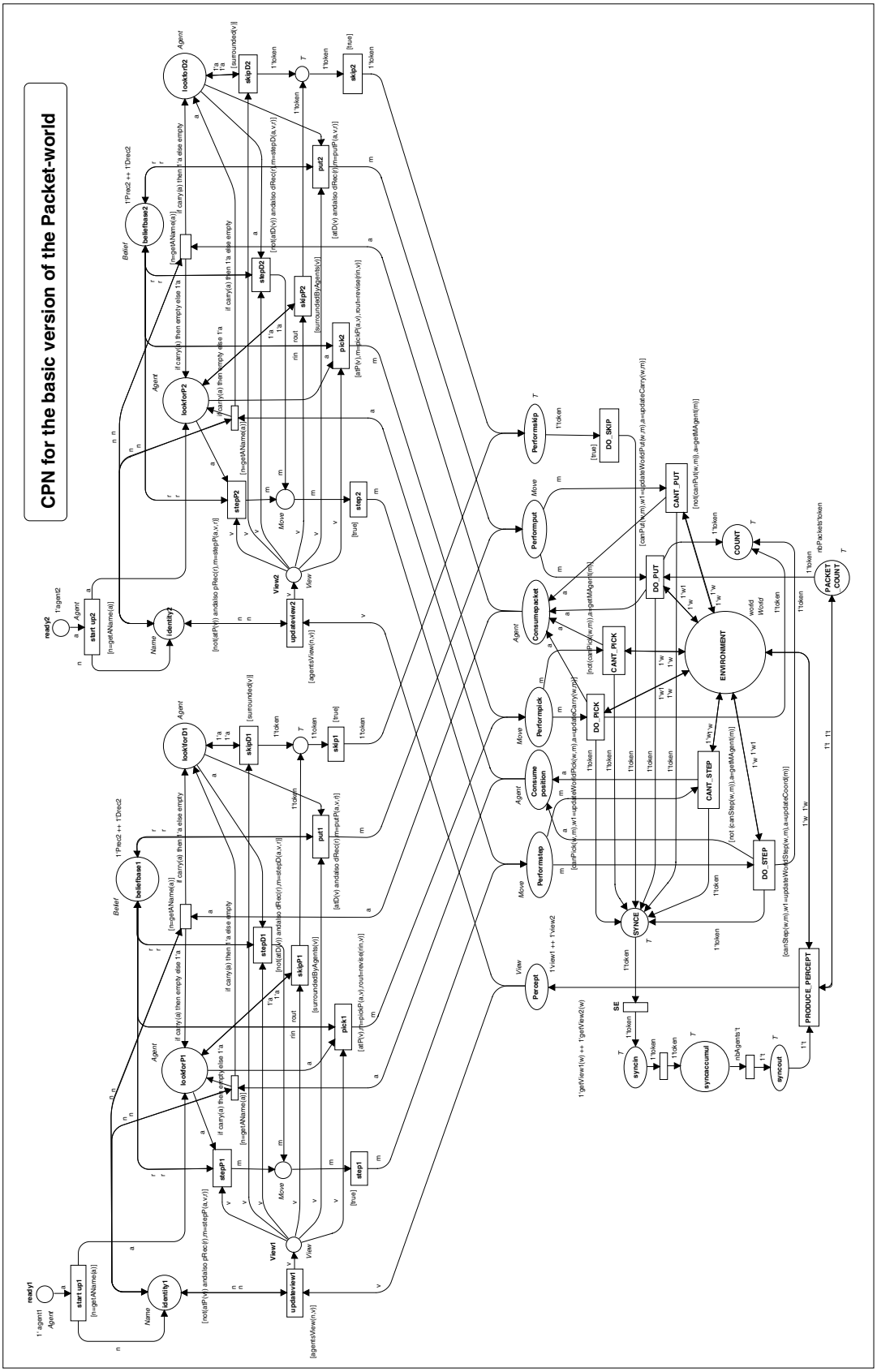

Figure 8. CPN for the basic model of the packet-world.

If we want to model a MAS with more agents we can combine the modules in a hierarchical CPN. However we do not discuss this further in this paper. To illustrate the purposes of this paper a model with two agents is sufficient.

4. A CPN for communicating agents

In this section we extend the basic model of the packet-world in a way the agents can communicate information with each other. Communication enables agents to coordinate their actions and behavior, resulting in a multi-agent system that is more coherent. Agents can use their abilities to communicate to better achieve the goals they are driven by.

Communication is part perception (the receiving of messages) and part action (the sending of messages). The conversation between two agents follows a protocol. A protocol enables agents to exchange and understand messages. To extend a basic agent with functionality for communication we build a "communication module" that can be plugged into the basic model of that agent. Furthermore the environment must be equipped with infrastructure to handle messages (mail). Therefore, we build a "postal service module" that can be plugged into the environment.

In this section we first introduce the communication protocol for our agents. Then we give a high level overview of our extended model for the packet-world, including communication. Next, we present the new modules necessary for communication and conclude with the complete CPN for the extended model.

4.1 Communication protocol

Basically the agents in our packet-world all have the same capabilities. This is reflected in the roles they play in a dialogue. As long as an agent "sees" another agent he is capable of sending a message to that peer colleague. For the moment we only model question/answer types of messages. In particular we limit the subject of the messages to requests for information. Figure 9 shows the different steps in a dialogue. The syntax of the protocol is described in section 4.3.

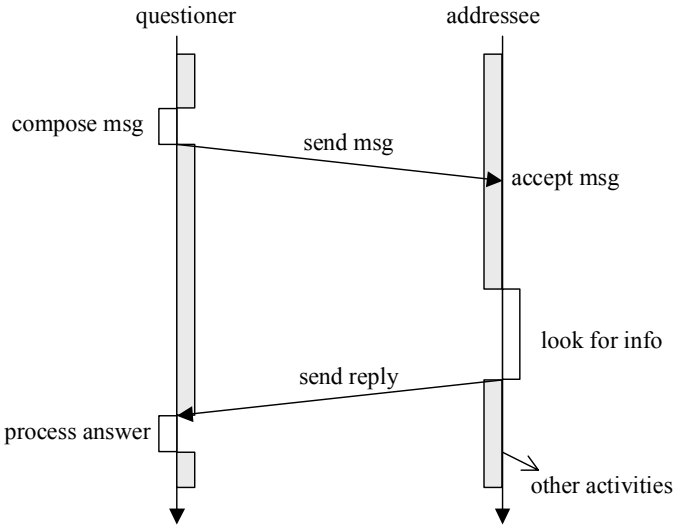

Figure 9. Communication among the agents.

An agent delivers a composed message at the "inbox" of the postal service. This service has knowledge of the mailboxes of the agents and routes the message to the mailbox of the addressee. As soon as the message arrives, the addressee can pick it up from his mailbox. In our model an agent is not obliged to handle an incoming message at once.

When the addressee decides to read the message, he will look for the requested information. If he knows the information he sends an answer, otherwise he informs the requester he can't help him for

the moment. When the reply arrives the information will be processed and possibly update the belief base of the requester.

4.2 High level model of the packet-world with communication extension

Figure 10 shows a high level model for the packet-world in which agents are equipped with functionality to communicate. The basic agents of our basic model are now extended with a communication module. This module permits an agent to interact with a colleague.

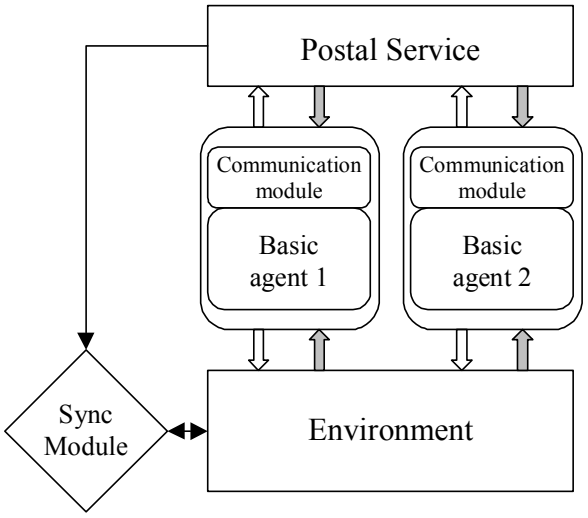

Figure 10. High level model of with functionality for inter-agent communication.

The arrows above the agents model the communication channels with the postal service module. This latter is responsible for delivering posted messages in the mailboxes of the addressees. Note that the postal service module too can produce functional synchronization pulses.

4.3 Model of the communication module for the agents

The communication module assembles functionality for an agent to send requests, respond to questions and process answers. The CPN for such a module is depicted in Figure 11.

In our model, agents can gather information from a colleague about the location of a packet or a particular destination according to its actual state. Asking for information is modeled as a transition, respectively "askforP" and "askforD". To fire one of these transitions (i.e. compose a message) a number of conditions must be fulfilled.

These conditions are described in the guards of the transitions. Let us look to one example, the "askforD" transition:

```
(* a models the internal state of the agent, v its current view *)
[canCallD(a,v),question=askForD(a,v)]
```

The function `canCallD(a,v)` returns true only if (i) the agent actually does not see the destination of its packet and (ii) he sees a colleague on which he can ask the information. If `canCallD(a,v)` succeeds, the function `askForD(a,v)` produces a question that as a token of the colorset Message is delivered in the inbox of the communication module.

A Message has the following structure:

```
color Performative =
 with questP | answP | noanswP | questD | answD | noanswerD;
color Message =
 record from:Name * to:Name * perform:Performative * content: Item;
```

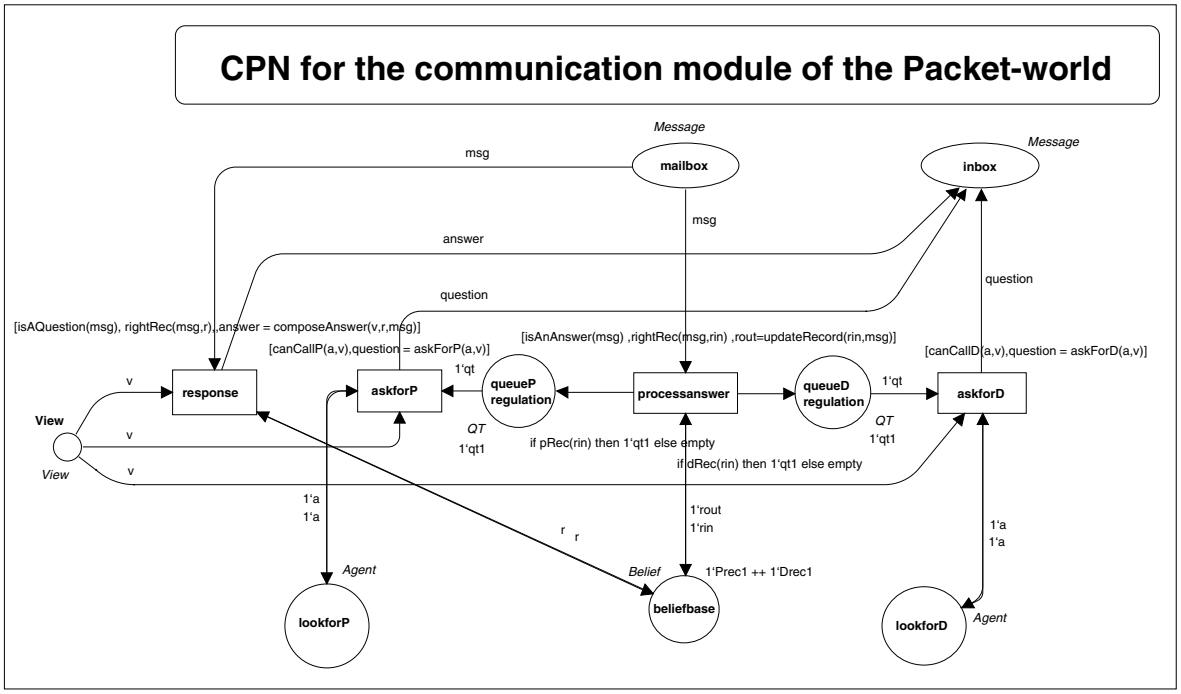

Figure 11. Communication module for an agent.

The performative informs the addressee about the type of message that is been sent. The content of a question is an item structure that has to be completed by the addressee. If for example an agent "a1" asks an agent "a2" for the location of the destination for yellow packets he will compose the following message:

```
{from="a1",to="a2",perform=questD,item={name="yellowDest",coord=null}};
```

When agent "a2" receives this message he knows exactly what "a1" is asking for. He will uses his belief base and actual view to find the coordinate of the "yellowDest". If he finds e.g. at coordinate (4,3) the yellow destination, he replies the following message :

```
{from="a2",to="a1",perform=answD,item={name="yellowDest",coord=(4,3)"}}
```

Replying to a message is performed in the transition "response". To fire this transition a view is consumed together with the message from the mailbox. The information in the believe base is only consulted as an extra information source.

The model prevents an agent to send messages for information over and over again. The "queueregulation" places contain a limited number of tokens that are consumed each time a question is sent and only restored when the answer is processed. Processing an answer comes down to update the belief base for the case the answer contains new information, otherwise the answer is thrown away.

4.4 Model of the posting service module

The postal service is responsible for delivering the mail of the agents at the right mailbox. Figure 12 shows the CPN for this module. The postal service has one "inbox" place where agents can leave their messages. Each message is accepted in the transition "acceptmsg". This transition puts the message in the "msgbuffer" place and produces three other tokens. The first token goes to the "msgcount" place where the user can read the total number of messages the postal service has handled so far. The second token goes to the "msglog" place, where a log of all messages is saved. The third token is sent to the "syncP" place from where it is directed to the "syncin" place of the synchronization module. This means that in our model, sending messages is coordinated with the

other actions agents can perform. This fits in our concept of functional synchronization, offering a solid base for the agents to coordinate their activities.

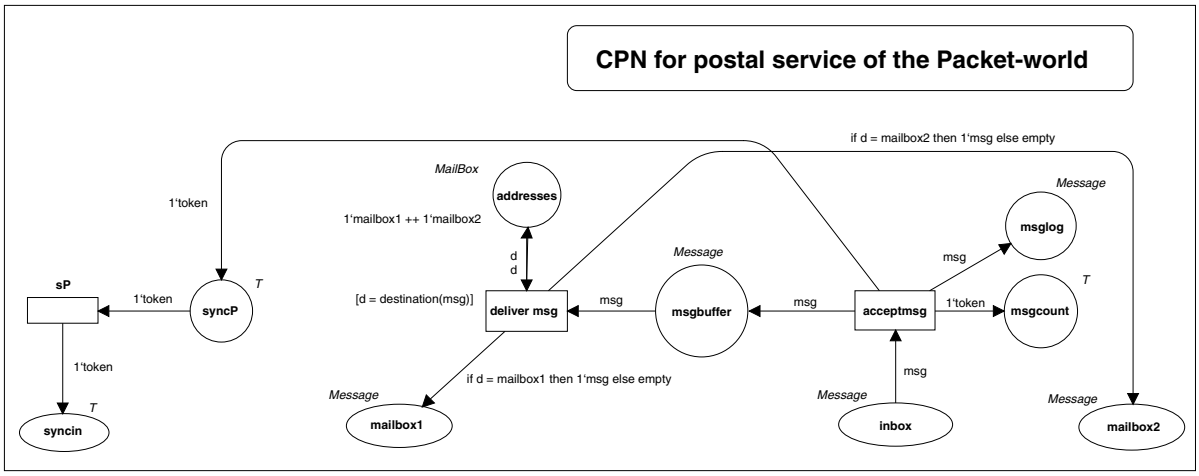

Figure 12. Postal service module.

When a message resides in the "msgbuffer" it is delivered to the addressee by firing the "delivermsg" transition. This transition consults the "addresses" place where the references to the different mailboxes are stored. Based on the mapping between the addressee indicated in the message and the information of the mailbox references, the message is delivered in the mailbox of the addressee where he can pick it up later on.

4.5 Complete CPN for the packet-world with communicating agents

With de separated modules we have proposed in the previous sections, we now can compose the complete CPN for the packet-word with communicating agents. This model, depicted in Figure 13, gives a detailed picture of the high level model we presented in Figure 10.

5. First experiments

In this section we briefly give an overview of the results of our first experiments with the CPNs for the basic version of the packet-world and the extended version with functionality for communication. We first discuss results of simulations; next we look to a number of verifications.

5.1 Simulations

With the Design/CPN tool a CPN can be executed, automatically or interactive. This allows us to follow the successive actions of the agents. We did tests on a world with size 5 and one with size 8. For both we changed the view-size for the agents. Table 1 gives an overview of the results. The numbers are rounded averages for 5 jobs.

Table 1. Simulation results.

World	view-size	Kind of model	COUNT	% gain	msgcount
world-size = 5 nbAgents = 2 nbPackets = 5	2	basic	26	31	--
		communication	18		4
	3	basic	15	7	--
		communication	14		2
world-size = 8 nbAgents = 2 nbPackets = 16	3	basic	167	23	--
		communication	129		16
	4	basic	110	2	--
		communication	108		2

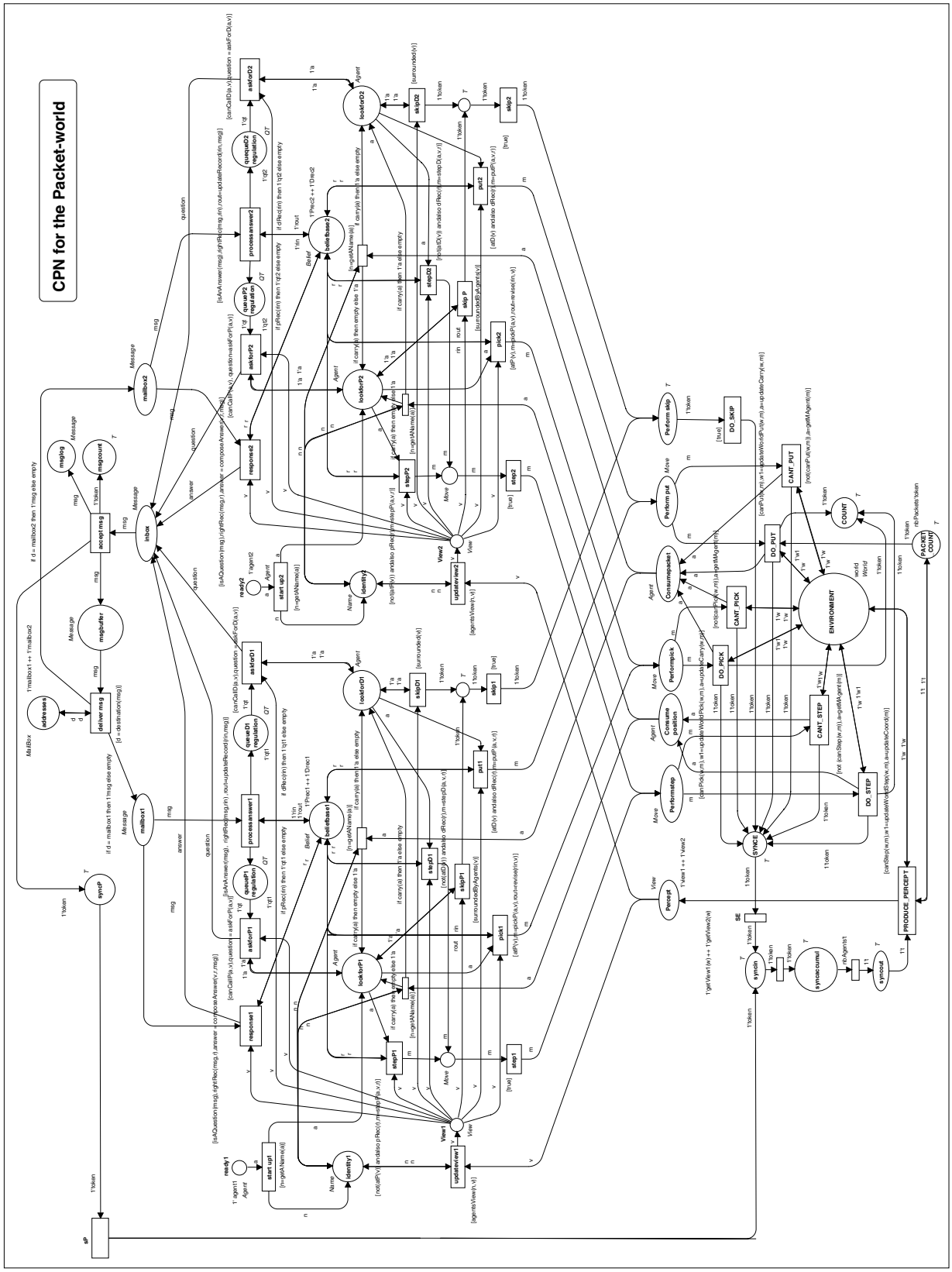

Figure 13. Complete CPN for the packet-world with two communicating agents.

If we compare the results for one kind of model (basic or communication), we see that increasing the view-size significantly reduces the number of steps the agents need to complete a job. The obvious explanation is that a greater view-size increases the information for the agents, so they can act more efficiently. When we compare the results between the two kinds of models for one view-size we notice that for small view-sizes the communicating model scores significantly better then the basic version. For greater view-sizes the gain is only marginal. In the first case agents communicate information to each other, so they can act more efficiently. In the latter case the view of the agents covers a great part of the world, so they mostly "see" what they are looking for, and there is no need to request information from each other. Our first tests confirm the value of information interchange between agents, but for better-founded conclusions we need to do more tests, especially with greater worlds and more agents.

5.2 Verifications

Besides simulation, the Design/CPN tool offers support for formal verifications of CPNs by means of the Occurrence Graph Tool [4]. An occurrence graph is a directed graph with a node for each reachable marking and an arc for each occurring binding element. With such occurrence graphs we did a number of formal verifications for the CPNs of the packet-world. We discuss here some results for the basic version of a world of size 5 with 2 agents that have to collect 5 packets.

The tool generates a standard report (for more information see [5]) that already gives a lot of information. E.g., the "Liveness Properties" gives the "Dead Transitions Instances" for the occurrence graph. If e.g. CANT_PICK is such a dead transition instance this means that for the given packet-world there where no conflicts between the agents with picking up packets. Contrary if e.g. CANT_STEP is not a dead transition instance we are sure both agents must have stepped at least once to the same square. To investigate the CPN in more detail the occurrence graph tool offers a lot of standard query functions. Besides, users can formulate their own customized queries too. To do a number of formal verifications, we extended the CPN for the packet-world with an extra "test-module", depicted in Figure 14.

We added four more places to the Petri Net, PACKETS_ON_GRID, CARRIED_PACKETS, DELIVERED_PACKETS and FINISCH_JOB, as well as two more transitions FINISH_JOB and TEST_END_JOB. Initially PACKETS_ON_GRID contains *nbPackets* anonymous tokens, while CARRIED_PACKETS and DELIVERED_PACKETS are empty.

When an agent picks up a packet (i.e. DO_PICK fires) one token from PACKETS_ON_GRID is passed to CARRIED_PACKETS. When an agent delivers a packet at its destination (i.e. DO_PUT fires) the token is further passed from CARRIED_PACKETS to DELIVERED_PACKETS. At the end of the job all packets are delivered, so PACKETS_ON_GRID and CARRIED_PACKETS are empty, while DELIVERED_PACKETS contains *nbPackets* anonymous tokens. When for each agent (during the final action cycle) the Agent token reaches the "lookforP" place and a new synchronization token reaches the "syncout" place, the FINISH_JOB transition is enabled and will fire. This clears the Petri Net and an anonymous token arrives in the END_JOB place. This enables the TEST_END_JOB transition that from then on will fire forever.

The packet-world is free of deadlocks. To prove that no deadlock appears we have to prove that there exists a path from each node in the occurrence to the node that represents the final marking, representing the state in the END_JOB place. That particular node, the leaf node of the occurrence graph, is shown with its predecessors in Figure 14. The proof is straightforward. The SearchNodes function "PROOF 1" in Figure 14 searches the number of nodes that have no path to the leaf node. Since this number is zero we have proven that the packet-world is deadlock free.

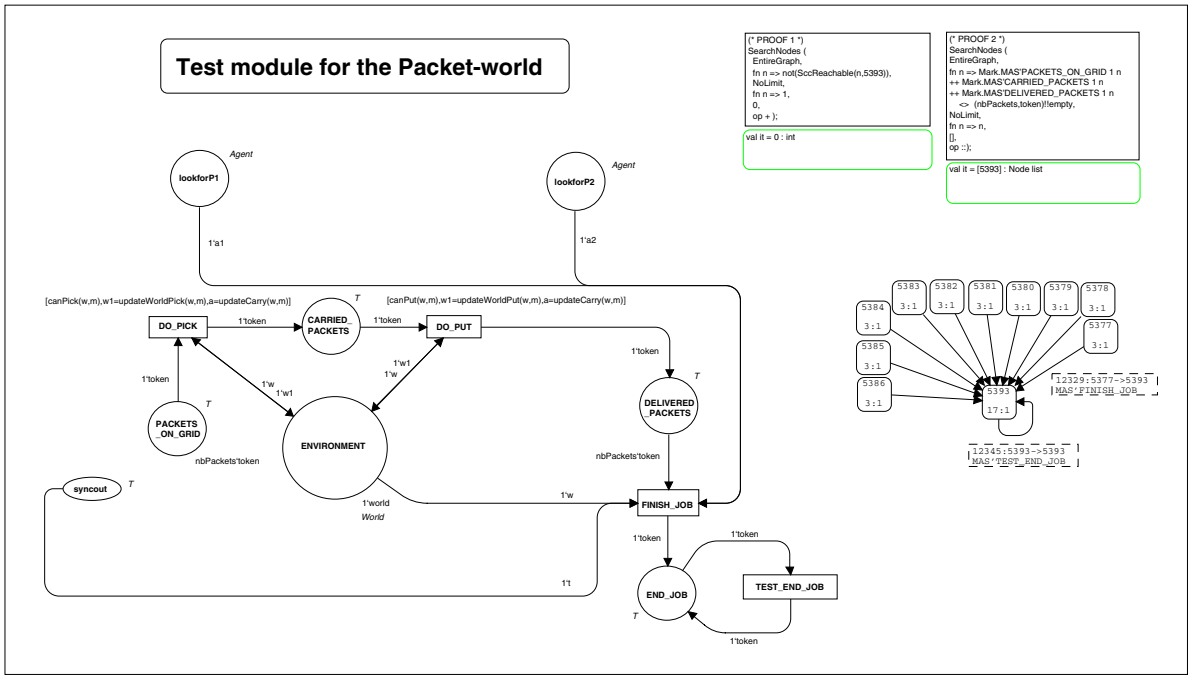

Figure 14. Test module for the packet-world.

A job is correctly solved in a limited number of steps. To prove that a job of the packet-world is correctly solved in a limited number of steps we have to take two steps. First we have to prove that the following place-invariant holds:

> *"the sum of anonymous tokens (each representing a packet) for the places PACKETS_ON_GRID, CARRIED_PACKERTS and DELIVERED_PACKETS is constant and equal to nbPackets in each node of the occurrence graph, except in the leaf node"*

This invariant tells us that neither strange packets enter the packet-world, nor any packet is lost during a job. The SearchNode function of "PROOF 2" in Figure 14 shows that the invariant holds. To complete the proof we must demonstrate that the number of steps to reach a solution (i.e. the TEST_END_JOB is enabled) is limited.

Since "PROOF 1" tells us that there exists a path from each node in the graph to the leaf node, we can conclude that execution always ends in a limited number of steps.

6. Conclusions and future work

In this paper we presented a CPN for the packet-world, a multi-agent application. In our research we use this application as a case to study the fundamentals of sociality in MASs.

Let us now reflect and verify that our expectations from using CPNs have been worked out. An important argument for using CPNs was its strong graphical expressiveness. We build up the packet-world by means of compositional modules. When we integrate communication infrastructure into the basic model we got a clear view on how this impacts the agents and the environment. Building an executable CPN leaves *no escape* for the designer. Every aspect must be modeled explicitly and unambiguously. Therefore we are forced to find concrete solutions for several problems. One typical example is the way we realized functional synchronization. One can talk al lot about such an aspect, but modeling it in a CPN brings the designer to the very *essence* of it. As such, we can state that we learned a lot about MASs, using CPNs to model them. Another argument why we have chosen CPNs was the possibility of simulation. Simulating a MAS like the packet-world can be done in different ways. Executing a CPN is not always the most attractive way to

simulate such a problem. But in fact, that is not the point. What is important is the fact that the execution of a CPN is a direct simulation of the *model itself*. So the simulation directly shows us the value of the model we have built. A last argument for CPNs we mention here is the possibility of formal verification. The MAS community has a strong tradition in formal description and verification of its ideas. CPNs join this approach. Formal verification lets the designer proof the correctness of (parts of) his model. Without the Design/CPN tool it would be very hard to prove that our packet-world has a correct solution in a limited number of steps. With the tool it is quite simple to proof this property.

This paper reflect our first experiences with CPNs as a tool to model agents' sociality. The model we have developed forms a solid basis for future research of agents' social behavior. We conclude with some thoughts about our future work. It is our intention to build modules for a number of other social skills for the agents of the packet-world. Examples are agents that cooperate by forming a chain and passing packets to each other, or agents that coordinate their actions avoiding future conflicts (e.g. 2 agents who both step a long way to the same packet). Building such models will gain us more in-depth knowledge about the fundamentals of sociality in MASs. To manage the complexity of extensive models we can use hierarchical CPNs. Later on we intend to generalize the insights we learned from the packet-world. We intend to build abstract models for different classes of social skills. The aggregate of these models can serve as a well defined and easy to communicate formal model for social agents in MASs.

7. References

[1] C.A.Petri, "Communication with Automata", Vol.1. Applied Data Research, Princeton, AF 30(602)-3324, 1966.

[2] K. Jensen, "Coloured Petri Nets", Lecture Notes Comp. Science, nr. 254, Advances in Petri Nets, Bad Honnef, 1986.

[3] J. Ferber. "Multi-Agent Systems, an introduction to distributed artificial intelligence", Addison-Wesley, ISBN 0-201-36048-9, 1999.

[4] Design/CPN, A Computer Tool for Coloured Petri Nets http://www.daimi.aau.dk/designCPN/ .

[5] K. Jensen, Coloured Petri Nets. Basic Concepts, Springer Verlag, 1992, ISBN: 3-540-60943-1.

[6] N. Jennings, On agent-based software engineering, Artificial Intelligence, 117 (2) 277-296, 2000.

[7] Huhns, Stephens, Multi-agent Systems and Societies of Agents, in G. Weiss, Multiagent Systems, MIT press, 1999.

[8] Y. Shoham, M. Tennenholtz, On Social Lows for Artificial Agent Societies: Off-Line Design. Agents 1998.

[9] R. Conte, C. Castelfranchi, Simulations understanding of norm functionality's in social groups, 1993.

[10] Goldman, Rosenschein, Emergent Coordination through the Use of Cooperative State-Changing Rules, DAI, 1994.

[11] J. Kittock, Emergent Conventions and Structure of MAS, Complex Systems Summer School, Santa Fe 1995.

[12] A. Walker, M. Wooldridge, Understanding the Emergence of Conventions in Muli-Agent Systems, ICMAS'95.

[13] P.R. Cohen, H.J. Levesque, Teamwork, Special Issue on Cognitive Science and Artificial Intelligence, 1991.

[14] P. Panzarasa, N.R. Jennings, T.J. Norman, Social Mental Shaping: Modeling the Impact of Sociality on the Mental States of Autonomous Agents, Computational Intelligence 17 (4) 738-782, 2000.

[15] P. Panzarasa, N. Jennings, The organization of sociality: a manifesto for a new science of multi-agent systems, Proc. 10th European Workshop on Multi-Agent Systems (MAAMAW-01), Annecy, France, 2001.

[16] T. Holvoet, An approach for open concurrent software development, PhD thesis, K.U.Leuven, Belgium 12/1997.

[17] J. Fernandes, O. Belo, Modeling Multi-Agent Systems through Colored Petri Nets, 16th IASTED International Conference on Applied Informatics (AI'98), Garmisch-Partenkirchen, Germany, pp. 17-20, Feb/1998.

[18] M. Costa Miranda, A. Perkusich, Modeling and Analyses of a Multi-Agent System using Colored Petri Nets, In Workshop on Applications of Petri Nets to Intelligent System Development, Williamsburg, USA, June 1999.

[19] D. Moldt, F. Wienberg, Multi-Agent Systems based on Coloured Petri Nets, Azéma und Balbo 1997, 1997.

[20] R. Cost et al., Modeling Agent Conversations with Colored Petri Nets, IJCAI '99, Stockholm, Sweden, 1999.

[21] M. Duvigneau, D. Moldt, H. Rolke, Concurrent Architecture for a Multi-Agent Platform, in Proceedings of AOSE'02, AAMAS, Bologna Italy, July 2002.

[22] Miyamoto et al. wrote a number of articles in Petri Nets Newsletter, http://www.informatik.uni-hamburg.de/TGI/pnbib/keywords/a/agent_net.html