Diploma Thesis

A Tool for Editing Agent Organization Models and Their Deployment in MULAN

submitted by
Endri Deliu

supervised by
Dr. Michael Köhler
Prof. Bernd Neumann, Ph.D.

University of Hamburg
January 18, 2008

Declaration

I declare that this work has been prepared by myself, all literal or content-based quotations are clearly pointed out, and no other sources or aids than those declared have been used.

Endri Deliu, January 18, 2008 ______________

Acknowledgements

My sincere thanks go to all who contributed to the success of this work:

Dr. Michael Köhler for the exciting topic and the outstanding supervision and support of this work;

Prof. Dr. Bernd Neumann for his consent to supervise this diploma thesis;

Dr. Daniel Moldt, Matthias Wester-Ebbinghaus, Lawrence Cabac, Michael Duvigneau, Christine Reese and Benjamin Schleinzer for their motivational hints and support;

A special thank to my brother Orges and my friend Lidia, for their continuous support and their help in proof-reading this work.

Contents

Chapter 1.

Introduction

The agent concept has been around for several decades in computer science. Since their infancy in artificial intelligence where they were first used, agents have successfully crept from academic research into many application areas such as expert systems, search, logistics, financial markets, trading, etc. Yet, in software development, despite the fact that the agent concept is seen as a natural development of the concept of objects, agent technology still struggles to be accepted and adopted as a mainstream solution. One of the main reasons that objects have the upper hand on agents is that systems which are build on objects mostly deliver reliability and predictability, despite their level of complication. Agents have instead retained their initial AI[1] flair of building systems whose behavior *emerges* from the autonomous actions of the agents composing them. While emergent behavior of systems is desirable in some areas, in many other areas system properties such as predictability and reliability are among the top priorities in system design. Agents also represent too small components to be viewed as modules in terms of software development [WEM06]. In modern software applications, modularity is a key requirement that any approach to developing software can not afford to neglect.

1.1. Motivation and Scope

Recently, important influxes from sociology and organization theory have begun delineating what may dissolve the trade-off between agent autonomy and multi-agent system reliability and predictability. Between the system and the agents composing it, other levels of control have been introduced which are mainly derived from sociological concepts. The concept of organization is used as an umbrella term for groups of agents and their dependencies, interaction channels or relationships. As a result, an organizational perspective on multi-agent systems has emerged which focuses on organizational concepts such as groups, communities, organizations, etc., in contrast to the former focus of multi-agent systems on the agent's state and its relationship to the agent's behavior. Agent organizations may also provide the lacking concepts for making multi-agent systems more modular.

[1]Artificial Intelligence.

Since the rise of the organizational perspective, modeling agent organizations has become important for the construction of multi-agent systems. This process requires at least a modeling language that is able to express most (possibly all) of the notions that the concept of organization encompasses in an intuitive and easily understandable way. Several languages for modeling multi-agent organizations have been proposed including AGR [FGM03], MOISE$^+$ [HSB02], AUML [PO01] as an extension of UML [OMG99] as well as other languages. While these languages emphasize the most important modeling dimensions of organizations they still do not offer a complete formal body which allows a formal approach on modeling.

Petri nets are well suited for use in modeling systems and simultaneously offer a complete formal frame. Petri nets can also be used for programming despite being still unpopular in this area. Nevertheless, petri nets have advantages for development of concurrent and distributed systems. Also, in the face of the current direction of the computer industry to deliver multi-core processors instead of increasing the clock speed of a single processor, the importance of tools which can handle parallelism in an easy way will rise. These developments might usher the revival of petri nets in the programming field.

In this context, a framework for the development of concurrent and distributed software systems has been built as a multi-agent system basing on reference nets[2] [Kum02]. *Mulan* (**M**ulti **A**gent **N**ets) [KMR01] provides the framework's reference architecture used for the the multi-agent system. *Mulan* is built on Java and reference nets and can be executed in *Renew* [KWD01]. The framework has already been successfully used in several software application projects at the Department of Informatics of the University of Hamburg [MKR01, MDOR03].

In the modeling area, a formal model of agent organizations based on P/T petri nets with inscriptions named SONAR, is introduced in [Köh06]. SONAR is developed to be used for modeling formal organizations. It represents a model of agent organizations that is built on top of concepts like organizational positions, roles, tasks, services and delegation relationships. Being a petri net model, SONAR has the advantage of enabling formal analysis of the SONAR organization models over important properties. If used in conjunction with *Renew* and *Mulan*, SONAR can help close the gap between modeling agent organizations and their implementation for both the modeling process and the implementing process are based on petri nets. Such a close affinity between the model and its implementation can ensure shorter development cycles and can support the overall software development process.

However, no tool still exists which supports the creation of SONAR organization models. The scope of this thesis is to build a prototype graphical tool that supports users in the building process of SONAR organization models. The tool should offer an intuitive interface which can effectively shield users as much as possible from the formal background underlying the organization models that they create. It should also provide some evaluation features with the help of which the models constructed can be evaluated on formal properties. In addition, the SONAR formal organization models constructed with

[2]Reference nets are high level petri nets.

the aforementioned tool should be deployed in Mulan as agent organizations. The deployment process should generate agent organizations in *Mulan* out of SONAR formal organization models.

1.2. Structure of the Work

This thesis consists of eight chapters. The following chapter makes an introduction to the concept of agents and multi-agent systems. Chapter 3 presents the most important aspects of organizations from the point of view of computer science. Besides, it presents the organizational perspective of multi-agent systems. Chapter 4 gives a brief introduction to petri nets. Additionally, reference nets (a high level petri net type) and a reference net editor and simulator, *Renew*, are also presented. *Mulan* is also discussed as a reference architecture for multi-agent systems based on reference nets. In Chapter 5, several modeling languages for the design of multi-agent organizations are presented. The SONAR model, which is the model of reference for this work, is described in detail. The specification and implementation of OREDI, a tool for building SONAR organization and R/D nets, is introduced in Chapter 6. Chapter 7 describes the process of deploying SONAR organization nets into *Mulan*. Finally, a summary of the thesis and its main results are presented in Chapter 8.

Chapter 2.

Agents and Multi-Agent Systems

Agent based technology has called much attention because of its promise to become a new paradigm for the design and implementation of software systems in recent years. This technology is already used in financial markets, trading, logistics, industrial robotics etc. It is also continuously taking ground in mainstream commercial applications. Multi-agent systems rely on the concept of agents and are an important section of the agent based technology. These systems are adequate for designing and implementing complex distributed applications in which information and resources are shared among the parties. This chapter is dedicated to agents and multi-agent systems and focuses on communication and interaction aspects. The agent concept and some of the basic agent types are first introduced. Then, an introduction to multi-agent systems follows.

2.1. Agents

The concept of agents lies at the basis of the agent technology. Agents are known as computer systems which can make decisions for themselves and react autonomously to changes. Computer systems are generally perceived as systems for which their creators have to foresee every possible situation and accordingly program the reactions. Unforeseen situations are generally known to be dangerous to computer systems and lead to system crashes or to false results. An agent instead, can deal with unexpected or unknown situations and it can decide for himself what actions it should take.

An example for an agent would be an intelligent car that drives itself through the traffic and wants to move from point A to point B. The environment relevant to our intelligent car would be in its simplest case the streets and the other cars (neglecting pedestrians, nearby buildings, etc.). Our intelligent car can move, stop, accelerate in every direction on the streets and can sense the cars in its vicinity. The intelligent car has to make several decisions. For example, it should decide if it is too close to another car and what direction it should take in order to avoid a crash. It also should decide how much to accelerate if it is too close to another car.

The concept of agents was first introduced and applied in Artificial Intelligence (AI). Now, agent technology is creeping into a mainstream with applications in many domains ranging from expert systems to Internet search applications.

2.1.1. Definitions

Similar to many other abstract concepts with widespread use in many areas agents have no exact definition. Rather, many definitions can be found depending on the application domains. Indeed, for several years there has been an ongoing debate on a universally accepted definition of the term “agent”. While there is widespread consensus on *autonomy* as a central property of agency, opinions about other properties of agency differ quite remarkably. The ability of agents to learn is, for example, required in some domains while it is not desired in other domains. Agents are mostly referred to as entities embedded in an environment which can make decisions and act autonomously through sensing a part of their environment. Most definitions ascribe a goal to agents which they should strive to achieve.

One of the classic definitions on agents is the one made in [RN95] describing an agent as anything that can perceive its environment through *sensors* and act upon that environment through *effectors*. In [RN95] *rationality* and *autonomy* are introduced as properties of agents. Rational agents act so that they maximize the expected success given what they have perceived.

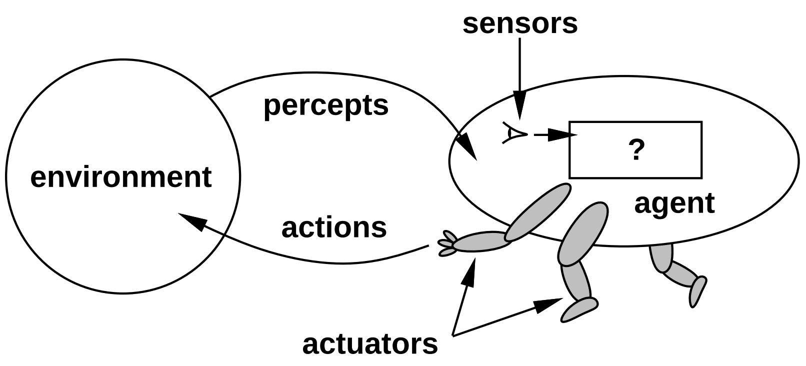


Figure 2.1.: Agents ([RN95]).

The degree of success is defined in [RN95] by the performance measure which is the criteria that determines how successful an agent is. Rationality depends on the performance measure, on the knowledge the agent has about its environment, on everything perceived by the agent and on the actions that the agent can perform. Autonomy is introduced in [RN95] as a property of rational agents. The degree of autonomy of an agent is presented as the extent to which an agents behavior is determined by its own experience. So, if an agent acts only according to its built-in knowledge without consulting its current and past perceptions this agent lacks autonomy as it is defined in [RN95]. An example would be a dog which has the built-in knowledge that it will receive its meal every day in a given place. Even if the meal is not at the right place anymore the dog would continue to go to the meal place and would try to eat the inexistent meal regardless of the meal being there or not.

Another classic and comprehensive definition of agents can be found in [WJ95]. Agents are split there in two categories : agents complying to the *weak* definition and those which comply to the *strong* definition. The weak definition ascribes agents the following properties:

- *autonomy*, agents are not influenced directly from people or other systems;
- *social ability*, agents can communicate with other agents or people through an agent communication language;
- *reactive*, agents can perceive part of their environment and can react to changes of the environment;
- *proactive*, agents do not only start actions as a reaction to changes of their environment but they can also take the initiative and act independently to achieve their goals.

This weak definition of agents seems to comply to a wide range of definitions from other researchers in mainstream computer science. The strong definition of an agent is relevant in AI and includes some additional properties besides those included in the weak definition that are usually seen as human-like properties. In many cases in AI agents are described using *mentalistic* notions such as *belief*, *intention* or *knowledge*. The strong definition in [WJ95] includes:

- *mobility*, the ability of agents to move in their environment;
- *veracity*, the assumption that agents will not knowingly communicate false information;
- *benevolence*, the assumption that agents do not have conflicting goals and therefore every agent will always try to do what it is asked;
- *rationality*, the assumption that an agent will always act in order to achieve its goals.

Another aspect of agents that is not mentioned in the definitions above is the *authorizing* and the *delegating* properties of agents. A user can delegate tasks to an agent and authorize this agent to execute these tasks in his name [Che96].

An interesting definition of agents is introduced in [Ode02]. Agents are introduced in [Ode02] as objects (as known in the object oriented paradigm) with a greater autonomy. This perception of agents is close to the notion of agent which is currently relevant for mainstream applications. In [Ode02] two aspects of autonomy are introduced:

- *dynamic autonomy*, the ability of the agent to start an action on its own, without being called from something else;

- *unpredictable autonomy*, describes the ability of the agent to deny the execution of a service that it offers;

While objects are strictly passive as they only react when their methods are called from methods of other objects, agents as described in [Ode02] can start actions on their own initiative. They do not need to wait for an external call from the other agents to become active. Agents in [Ode02] have the choice to refuse the execution of a service when they are called while objects are predictable and always execute their methods on call. The decision to execute a service depends only on the internal state of the agent. Another important aspect of agents mentioned in [Ode02] is the way how they interact and communicate with each other. Agents follow interaction templates, so called *protocols*, during communication. Agent interaction is based on the theory of communicative acts. The messages that the agents exchange with one another represent different communicative acts like request, query, refusal, etc.

It is interesting that the definitions of agents above as well as other well-known definitions of agents refer to an environment which exists independently of the agents and in which the agents are embedded. The environment is an important concept linked to the very definition of agents.

2.1.2. Environment Types and Agent Architectures

Independently of how diverse and complicated the properties attributed to agents in the existing definitions may be, all of these properties have to somehow face the challenges of implementation. The key issue an agent is concerned about is to make the right decision. That is, to choose the next action so that it can satisfy its goals or objectives best. Agent architectures are software architectures that enable agents situated in a particular environment to make decisions. The properties that the environment displays affects decision making in the agents situated in this environment. Environments can dramatically increase the complexity of the decision making process in agents. In the example of the intelligent car given at the beginning of this section the environment relevant for our agent-car was deliberately simplified. Imagine how complex the system would be if pedestrians, nearby buildings, sets of traffic lights and so on are included into the environment. In fact the inclusion of the other cars in our environment alone is enough to produce a complexity of high order. In [RN95], Russel and Norvig suggest the following classification of environment properties:

- *Accesible vs inaccesible:* in an accessible environment the agent can take anytime complete up-to-date information about the state of the environment.

- *Deterministic vs non-deterministic:* in a deterministic environment every action has a single guaranteed effect. There is certainty about the state of the environment resulting after performing the action.

- *Episodic vs non-episodic:* an episodic environment consists of discrete episodes. The performance of an agent is dependent on a number of discrete episodes, with no link between the performance of an agent in different scenarios. The agent can make decisions what action to perform next based on the knowledge of only the current episode.

- *Static vs dynamic:* a static environment remains unchanged except by the performance of the agents actions. A dynamic environment contains changing processes that take place independently of the actions of the agents. The environment can change in ways beyond the agents control.

- *Discrete vs continuous:* an environment is discrete if there is only a finite number of actions and percepts in it. Chess is a good example that Russel and Norvig give for a discrete environment.

The most difficult and complex environments to handle are those that are inaccessible, non-deterministic, non-episodic and continuous. However, as Russell and Norvig observe in [RN95] even sufficiently complex deterministic environments are as difficult to handle as non-deterministic ones. From the point of view of the complexity of decision making for an agent there is no difference and these kinds of environments may as well be non-deterministic.

So far only properties that agents should display have been presented. Agents were viewed as some kind of black boxes that mysteriously display a certain behavior that allows them to deal with the complexity of their environment. Building a practical agent requires that the model of the agent needs to be refined by breaking it into sub-systems. In [RN95] four categories of agents are introduced:

- *Simple reactive agents.* The next action of the agent is a function of its immediate perceptions and static rules. These agents do not refer to their past experience to make decisions and take the next actions. Also, they only respond to changes in their environment and do not take the initiative.

- *Agents with internal state.* The next action of an agent is a function of its internal state, the behavior of the environment, the action sequence of the agent and static rules. These agents have some kind of internal data structures to save information about the state, the environment and history.

- *Goal based agents.* The next action of an agent is a function of its internal state, the behavior of the environment, the action sequence of the agent and the goals of the agent.

- *Utility based agents.* The next action of an agent is a function of its internal state, the behavior of the environment, the action sequence of the agent and a utility function.

These categories can be viewed in Figure 2.2.

In [Woo95], a less abstract classification of agents is made. It defines:

- *Logic based agents:* decision making is realized through logical deduction;
- *Reactive agents:* decision making is implemented as a direct mapping between situation and action;
- *Belief-desire-intensions agents*, decision making is derived from manipulation of data structures representing the believes desires and intentions of the agents;
- *layered architecture.* Decision making is realized via various software layers, each of which explicitly reasons about the environment at different levels of abstraction.

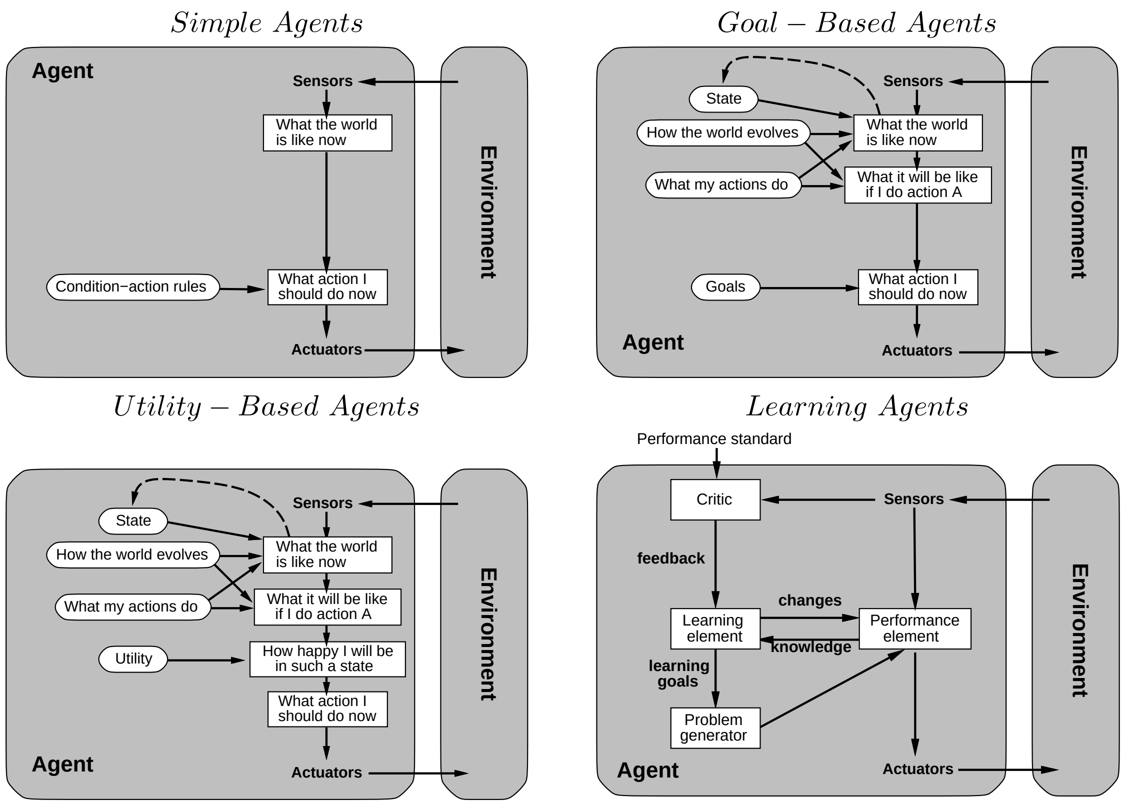


Figure 2.2.: Agent Categories ([RN95]).

Logic based agents have their roots in the traditional approach of building intelligent systems called symbolic AI. The basic assumption of symbolic AI is that intelligent behavior can be generated by a system by giving a symbolic representation of the environment and of its desired behavior to this system, and by manipulating the representation syntactically. Typically, symbolic representations are logic formula and the syntactic manipulation means theorem proving by deduction. Agents here are mainly theorem provers.

The reactive architecture is the result of an alternative approach to symbolic AI. This approach relies on the idea that intelligent behavior is seen as innately linked to the environment and is a product of the continuous interaction of the agent with its environment. Another basic assumption is that intelligent behavior emerges from the interaction of many simpler behaviors. There are several architectures relying on these ideas. The prominent architecture of the reactive approach is the *subsumption architecture* which has two main characteristics. The first is that the decision-making of an agent is implemented through a set of task accomplishing behaviors. Each of these behaviors is designed to achieve a particular task and is mostly implemented as a module in which situations are directly mapped to actions. So the agent here does not think or reason about its environment, it simply reacts to it. The second characteristic of the subsumption architecture is that many behaviors can be produced simultaneously resulting in conflicts which are solved through a *subsumption hierarchy* of the behavior modules. The behaviors are arranged in layers where lower layers have the priority over the higher and where higher layers represent more abstract behavior.

The belief-desire-intention (BDI) architecture has its roots in the understanding of practical reasoning which includes deciding moment after moment which action to take to achieve the goals. Agents are ascribed with *intentions* which lead to actions to achieve these intentions, and affect believes/assumptions of the agents on the future. The challenge in BDI agents lies in the question when an agent has to abandon its intentions. It seems obvious that a BDI agent should stop and periodically analyze its situation to decide if the achievement of its intention is still realistic. An agent that only analyzes or an agent that never analyzes results in inefficient behavior.

In layered architectures, proactive and reactive behavior are dealt with by being implemented separately in at least one sub-system for each behavior. These subsystems are arranged in a hierarchy of interacting layers. There are two types of control flow between these layers. The first is the horizontal layering where the software layers are each directly connected to the sensory input and action output and each of these layers produces suggestions on what action to perform next. Horizontal layers compete with each other and need a central mediator for deciding which of the layers has control over the agent at the given time. The second type is vertically layering where sensory input and action output are each dealt with by at most one layer each.

2.2. Multi-agent systems

Agents are always embedded in some kind of environment. Nearly every definition of agents refers explicitly or implicitly to an environment in which the agents are situated.

Agents are also seldom stand-alone systems. From an agent's point of view, other agents can be part of its own environment and it has to deal with them in some way or another. If this is the case, agents can choose between two possible options. One option would be to ignore the other agents. This option would be a sound choice for the agent only if the other agents are completely irrelevant to its design goals which is seldom the case. The very fact that many agents are situated in the same environment means mostly that they share some of the resources needed for the completion of their tasks. In this case, the only possible option for the agent would be to interact with the other agents.

In the previous subsection, a social ability was mentioned as one of the main properties of agency. Such an ability which includes the ability of agents to communicate with each other in an agent communication language is the basis for building multi-agent systems. The study of *multi-agent systems* (MAS) focuses on systems in which many agents interact with each other. Depending on their goals their interactions can be either cooperative or selfish. If the agents share a common goal they cooperate with each other. If they have conflicting goals they can pursue their own interests and mostly compete with each other. The use of multi-agent systems is generally seen as appropriate in systems where information and resources are shared between many autonomous participants. In this case, no participant has a complete view and knowledge of the overall system and the solution is reached through interaction with the other participants.

2.2.1. Agent Communication

Multi-agent systems rely on the ability of agents to communicate with each other and exchange their knowledge. Communication is seen as an ability of agents which is a part of perception (receiving messages) and part of action (sending messages). In order to communicate agents have to perceive, act and reason. Communication can be used to coordinate the actions of the agents. Another way to coordinate is precomputing the actions and behavior of other agents. However, this is an approach which requires knowledge on the overall state of the system and knowledge on the models of the other agents. Communication is appropriate in cases where agents have incomplete knowledge on their environment and on the other agents. Cooperation is a possible type of of coordination. Agents cooperate in a MAS when they do not have conflicting goals or when they are not antagonistic. Selfish agents typically compete with each other and have to coordinate their actions by negotiations.

Agent communication includes two types of messages: *assertions* and *queries*. Depending on the capabilities of agents to send or receive these types of messages, following categories of agents can be distinguished:

- *Basic Agents* are only able receive assertions.
- *Passive Agents* are able to send and receive assertions. Additionally, they can receive queries.

- *Active Agents* are able to send and receive assertions. Additionally, they can send queries.

- *Peer Agents* are able to send and receive assertions and queries.

Agent communication relies on concepts used for human communication and is founded on the theory of speech-acts and on ontologies. The speech act theory was introduced in [Aus62] and models communication as actions that are able to change the mental state of the participants. This theory views a certain class of utterances in the human language as actions such as requests, suggestions, commitments and replies. Under actions it is meant that this certain class of utterances can change the world similar to physical actions. Austin gives examples such as declaring war or declaring a pair as husband and wife. The speech act theory distinguishes three aspects:

- *Locution*, the act of making an utterance.

- *Illocution*, the meaning of the utterance intended by the speaker.

- *Perlocution*, the effect that was generated as a consequence of the locution.

Austin uses the so called *performative verbs* to identify different types of speech acts by their illocutionary degree. Some example performatives are *inform, request, promise, demand, propose*, etc. Speech act theory helps determine the type of messages by using the illocutionary force. Thus, the intention of the sender is always defined and the receiver understands the type of the message.

Communication protocols used in agent communication serve to enclose messages in an "envelope" that every agent understands. There are binary (one sender-one receiver) and n-ary (one sender-multiple receivers) communication protocols. A protocol is defined by a data structure containing the following fields:

- *sender*,

- *receiver(s)*,

- *language in the protocol*,

- *encoding functions*,

- *actions that the receiver(s) should take*,

Protocols are generally specified at several levels. The top and middle levels respectively define the semantics where the type of the message is described together with the substance of the message, and the syntax of the information. The lowest level specifies the method of interconnection. The semantics of the communication protocol is separated from the semantics of the enclosed message which may be domain specific. This separation allows the protocol to be universally shared by all agents independently of the domain specific languages they speak. The **K**nowledge **Q**uery and **M**anipulation **L**anguage (KQML) [Gro] is such a protocol. KQML defines a common format for messages. KQML protocol has a *performative* and parameters as key/value pairs. The basic structure of KQML messages looks like:

```
(KQML-performative
  :sender value
  :receiver value
  :language value
  :ontology value
  :content
)
```

The parameters ":content", ":language" and ":ontology" characterize the semantics of the message while other parameters including :sender, :receiver etc. are used for message passing. Any agent communication language can be used as content language. This includes KIF [Gin91], LISP, PROLOG and KQML itself. Indeed, it is possible to send embedded KQML messages inside other KQML messages.

FIPA (Foundation for Intelligent Physical Agents) ACL [FIP00] is another protocol for message exchange whose syntax is quite similar to that of KQML. The structure of a FIPA ACL message is the same as that of a KQML message. They differ literally only in the collection of performatives. Unlike KQML which lacks formal semantics FIPA ACL has semantics which is described using a formal language called **S**emantic **L**anguage (SL) [FIP02]. This language allows to represent believes, desires, uncertain believes and actions. FIPA ACL also accepts every possible agent communication language as a content language.

Ontologies are another important aspect of agent communication. During communication, agents mostly exchange domain specific information. However, in order to really understand each other they have to share a common terminology. That is, both agents should understand the same under each word they use. It is generally known that many words have different meanings in different domains. This kind of ambiguity has to be solved in communication between agents. Thus, during its communication an agent has to know about the domain in which he is talking and about the domain's related terminology with all the relations between the specific words/terms. Ontologies are the right answer to this problem. An ontology is a shared vocabulary of agents. It is a formal definition of a knowledge body which typically involves a structural component. Basically, an ontology represents a taxonomy of classes where the relationships are also included. Thus, an ontology has to include classes and relationships in its representation. However, an instance of a class does not need to be included in an ontology as its possible relationships are defined through its class. Several languages exist for defining ontologies. KIF (also used as a content language in KQML) is based on first order logic. Other languages, such as XML [XML07] or DAML [DAR], are also used for defining ontologies.

2.2.2. Agent Interaction Protocols

In this subsection some of the main principles and techniques of agent interaction in multi-agent systems will be described. Interaction protocols build on top of communication protocols. They are used in the case when agents intend to exchange more than one message with each other. Interaction protocols manage conversations between agents. While the objective in conventional distributed systems is to avoid deadlocks and livelocks, multi-agent systems have additional concerns that have to be taken into account. If a multi-agent system consists of selfish agents which set the priority on their own interest the main objective of their interaction protocols is to maximize their utility [RZ98]. If agents have similar goals their interaction protocols are designed to maintain globally coherent performance of the agents without violation of autonomy or explicit global control [Dur88].

Coherence refers to the ability of the multi-agent systems to behave as a unit. The Distributed Problem Solving (DPS) is such a case where the agents have similar goals. Here is generally assumed that agents are benevolent which implies that they share a common goal and act so that system objectives have priority over personal objectives. Typically, in multi-agent systems such an assumption can not be made. Multi-agent systems are more general than DPS. They are considered rather *societies* of self interested agents which can be designed by and thus can represent the interests of many different persons or organizations. Research of multi-agent systems is focused on why and how agents cooperate [Woo94], how agents resolve conflicts [KB91] and on negotiation or compromise mechanisms when agents are in conflict [RZ94]. In multi-agent systems coordination has to occur during runtime. The agents have to recognize situations where coordination is required and choose the appropriate actions. Several methods for dynamic coordination have been developed which involve [Woo01]:

- Coordination through partial global planning.
- Coordination through joint intentions.
- Coordination by mutual modeling.
- Coordination by norms and social laws.

The main idea behind partial global planning [DL87] is the exchange of information between agents in order to reach common conclusions about the solution of the problem. Agents create non-local plans by exchanging their local plans and cooperating in order to achieve a non-local view of the solution. Three steps can be distinguished in partial global planning which every agent repeats:

- Agents determine their goals and create plans to achieve their goals.
- Agents exchange information to resolve if and where their plans and goals interact.

- Agents change their local plans to coordinate their activities.

Coordination through joint intentions was inspired from the human team model. Being part of a team with collective intentions toward some goal differs from individual intentions of agents by the fact that agents in a team have responsibility toward the other members of the team [LCN90]. Key agent structures in coordination through joint intentions are *commitments* and *conventions* [Jen93].

Commitments are a sort of promise of the agents to hold somehow to a course of actions. The other agents can then predict with some sort of accuracy the future behavior of this agent and can build their future actions basing on the expected behavior of this agent.

Conventions are means of managing commitments when circumstances change. They specify under what conditions the commitments can be dropped or changed.

Coordination by mutual modeling is based on the principle that agents have information on the internal models of other agents. Agents can make predictions on the behavior of other agents relying on the internal models that they have of other agents. Communication in such a coordination approach is generally unnecessary.

Coordination by social norms and laws, as the name suggests, is inspired by the coordination in human societies. A norm refers to an established pattern of behavior that the majority of the members of a society follow. A law is similar in meaning to a norm, however, a law is always associated with an enforcing authority which ensures its implementation. Norms and laws can be designed during implementation of the system by the engineers. Designing them during implementation is simple and allows a better system control. However, this approach is not suitable for open environments or for environments where the goals of the agents change constantly. In this kind of environments agents that organize themselves have more advantages.

A basic strategy for coordination among cooperative agents is to divide tasks into many smaller subtasks which require less resources. These subtasks can be distributed between the agents. The agents solve their subtasks and reassemble their small solutions in the overall solution. While eventually dividing tasks into subtasks alternative division possibilities and dependencies between subtasks have to be considered. The process of the decomposition of tasks and of their distribution to the agents is also known as *task sharing*. The main issue in task sharing is how to make the right distribution of subtasks to the right agents as the agents in many cases have entirely different capabilities. Distribution of tasks is mainly achieved through [HS99]:

- market system, tasks are distributed to agents through general agreements or mutual selection;

- contract net, announce bid and award;

- multi-agent planning, specific agents are responsible for task distribution;

- organizational structure, agents have fixed responsibilities for specific tasks;

The contract net protocol [Smi88] is a protocol for cooperative problem solving. It provides a solution for the correct assignment of tasks to appropriate agents. The contract net involves some basic steps where cycles of announcements, bids and awards are executed by agents. An agent that needs to have a task solved, *the manager*, makes an announcement which is a multicast to the agents that might be able to solve the task. The agents that received the managers announcement, the potential *contractors*, evaluate their possibility to solve the task and depending on their capabilities either make bids or decline. The manager receives the bids and awards a suitable contractor. The contractor performs the task and sends the result to the manager. Any agent can be a manager or it can assume the contractor's role. This allows even contractors to outsource their tasks to other agents.

Another important issue in coordination is how agents reach agreements. Reaching agreements is a capability required for cooperating as well as for competing agents. The negotiation and argumentation capabilities of agents play a key role in reaching agreements. Negotiating agents follow a negotiation protocol. It is important that any history of the negotiation has some desirable properties which include [San99].

- Guaranteed success, a protocol guarantees that at the end an agreement is always reached.

- Social Welfare, a protocol guarantees that the resulting agreement maximizes the sum of the utilities of the negotiators.

- Pareto efficiency, a negotiation result is pareto efficient if there is no other possible negotiation result that will make at least one agent happier without making at least another agent worse off.

- Individual rationality, following individual rational protocols is in the best interest of the negotiators.

- Stability, all agents in a stable protocol have an incentive to behave in a particular way.

- Distribution, no single point of failure exists in the protocol.

The negotiation protocols mostly used are:

- voting;

- auctions;

- negotiations;

- arguments.

Voting refers to the process in which all agents give an input to a mechanism. The result that this mechanism computes based on the inputs from the agents is the solution for all the agents. This result is generally obligatory for all agents and there is some way the solution computed is enforced for all agents. There are several alternatives for voting protocols which involve the functions that compute the results out the input from the agents. One of them is the *plurality protocol* where all alternatives voted are compared and the alternative with the highest number of votes is chosen. In such a majority protocol, just like in politics, introducing an alternative similar to the majority alternative splits the majority and may cause another alternative that was formerly a minority to become the majority alternative. Another voting protocol is the *binary protocol* where each time only two alternatives are voted for. The winning alternative remains to challenge another alternative and the loosing alternative is eliminated. In such a protocol the ordering of the pairs of alternatives presented to be voted each time can change the outcome of the final winning alternative. Also, the introduction of similar alternatives causes here a splitting effect as in the plurality protocol.

Auctions specify an auctioneer and many bidders which interact with each other. Auctions are used when the auctioneer wants to allocate some goods to one bidder. Mostly the auctioneer tries to maximize the price of the good to be allocated to a bidder by choosing the most appropriate auction protocol. Bidders, instead, want to minimize the price at which the goods are allocated to them by choosing strategies inside the framework of the auction protocol.

One of the most important auction properties that affects the bidders' strategy is how the bidders' value of the goods being allocated is defined. If the value of the goods depends only on the preferences of the agent the goods have *private value*. If an agent's value of a good depends completely on the values of the other agents it is said that the good has *public value*. Indeed, the value of the goods in such a case is the same to all the agents. If the value of the goods depends partly on an agent's own preferences and partly on the values of the other agents it is said that the goods have *correlated values*. The second important property that affects auctions is if an agent sees the bids of the other agents. If it is the case the auction is called *open cry* otherwise it is called a *sealed-bid* auction. The third important property of auctions is how the process of bidding proceeds. One possibility is that the price starts low and it increases with every additional bid. These auctions are called *ascending*. If the price starts high and is decreased in successive rounds by the auctioneer the auction is called *descending*. Several auction protocols exist:

- English auctions;
- Dutch auctions;
- first-price sealed-bid auction;
- Vickrey auction.

The English auction is an *open cry* and *ascending* auction protocol. The auctioneer starts with a low price. After this, bids are expected from the agents with the condition that the bids must be higher than the current highest bid. If agents do not raise the bid anymore than the good is allocated to the current highest bidder.

Dutch auctions are *open cry, descending* auctions. The auctioneer starts with an unrealistic high price for the good. The auctioneer continues then to lower the price by small amounts in several rounds until an agent makes a bid for the current amount. The good is then allocated to the agent that made the offer.

In first-price *sealed-bid* auctions every agent makes a bid during a single round. The winner is the agent that made the highest bid. The good is allocated to the price of the highest bid. As the name suggests, in these kind of auctions agents can not see each other's bids.

Vickrey auctions are also *sealed-bid* auctions. Here agents can make hidden bids in a single round. The winner is the agent that made the highest bid. However, the price paid by this agent for the allocation of the good is equal to the second highest bid made during the bidding round. In this kind of unintuitive auction telling the truth about how much the good is worth to them is the most efficient strategy for the bidders.

Negotiation is a mechanism where agreements are reached on the basis of mutual interest. It is more general than auctions which are appropriate only for the allocation of goods. Auctions are in fact specific negotiation examples. Typically any negotiation involve four important components:

- The negotiation set, the set with the possible proposals that the agents can make
- The protocol, defines the proposals that agents can make as a function of the negotiation history
- The collection of private strategies of the agents, which more or less defines the future proposals of the agents
- The rule that specifies when a deal can be called as made and the content of the deal

[RZ94] Negotiations are executed in series of rounds where every agent makes a legal proposal from the negotiation set following its own strategy. The negotiation terminates if an agreement is reached as defined in the agreement rule. Negotiations can become complicated if agents negotiate over many issues which are interrelated. A more detailed view on Negotiations is given in [RZ94].

Argumentation in multi-agent systems is the process when agents attempt to convince each other that one proposal is true or false by putting forward several arguments. The concept and use of argumentation emerged as several setbacks exist in agent negotiations. One of these setbacks is that the positions of agents do not change on a specified issue or task during negotiation [JFL+01]. The utility function of the agent does not change during the negotiation. This behavior does not correspond to real life situations where argumentation is a technique that is used abundantly during negotiations. Agents are typically involved in logic based argumentation among the several possible types of argumentations [Gil04] that exist as logic based argumentation is easier to formalize and implement. A system of logic based argumentation is introduced in [Woo01] where agents argument with each other on the basis of a common set of formulas, the so called database. Arguments are here logical conclusions inferred from formulas drawn from the database. Arguments are given weights or are classified on the basis of their structure to determine which arguments are stronger than others.

2.3. Summary

Agents and multi-agent systems are increasingly finding their way into mainstream software development and are already used in a wide range of domains. Agents are autonomous entities that can sense part of their environment and have the ability to make decisions. They can be reactive or proactive meaning that they can only react to changes of their environment or act on their own initiative as well. Several architecture types exist for building agents including, logical, reactive, belief-desire-intention (BDI) and layered architectures. Agents have also social abilities and can communicate or interact with each other. Multi-agent systems are systems where many agents interact with each other. Agents in multi-agent systems can cooperate or compete depending on their goals being similar or conflicting. The ability of the agents to communicate in an agent communication language is central to multi-agent systems and to coordination between agents. Agent communication relies on the theory of speech acts [Aus62]. Agents can coordinate their actions and interact by cooperating or by competing with each other. Several coordination techniques exist which include coordination through joint intentions, coordination by mutual modeling which does not require communication capabilities, coordination through social laws or norms and coordination through partial global planning. Cooperative agents can coordinate their actions through subdivision of tasks into smaller tasks which require less resources. The main issue here becomes the efficient allocation of the subtasks to the cooperating agents. The most important methods for task allocation are market systems, contract nets, multi-agent planning and organizational structures. Reaching agreements is important for coordination between cooperating as well as competing agents. Negotiation and argumentation capabilities are central for reaching agreements. Negotiation requires agents to follow a specific negotiation protocol. Among the available negotiating protocols are voting, auctions, negotiations and arguments.

Chapter 3.

Organizations and Multi-Agent Systems

Until recently, multi-agent systems were viewed as systems which did not need any explicitly defined structural level to handle the coordination of their autonomous agents. The general idea behind such a view was that the interactions of the autonomous agents and the limitation in resources would provide the basis for a self-regulatory and highly adaptive system. This view dictates an approach in the design of multi-agent systems where desired properties of the system are achieved through tweaking in the interaction and coordination rules of the autonomous agents. The structure of the multi-agent systems was implicitly *incorporated* in the agents themselves in form of their interaction or coordination rules. Thus, the structure of the system *emerged* during the interaction of the agents. Such a bottom-up approach in designing multi-agent systems has advantages if the system is required to have emergent properties or has to be adaptive. However, it has several drawbacks [Jen00] and it becomes a problem if it comes to build robust and predictable systems with specific properties. While self-regulation and adaptiveness are certainly desirable properties for most systems, other properties such as emergent behavior or unpredictability of the system are not. This may be one of the reasons why agent-oriented software development have not replaced object-orientation despite agents being seen as a natural development of objects[1].

Recently, important impulses from sociology and especially concepts from the organization theory have flowed in the research on multi-agent systems. The introduction in multi-agent systems of organizational concepts defining a structure independent from the agents provided what the multi-agent systems lacked. An explicit structural level of the system which exists independently of the agents and which can be easily modified. In this chapter some key concepts from organizational theory as well as an organizational perspective on multi-agent systems will be introduced. Additionally, the main organization types used in multi-agent systems will be listed and briefly explained.

[1]Objects in the object-oriented paradigm.

3.1. Organizations

Similar to the concept of agency the concept of organizations is still widely discussed and lacks a broadly accepted definition. Due to the complexity of organizations in real life, most of the books on organizational theory concentrate on specific aspects of organizations and provide different definitions for organizations which emphasize one or some of its overall aspects. While there is no broadly accepted definition of organizations there are some basic principles that are generally accepted as valid for organizations. Organizations are characterized as [CG99]:

- large-scale systems for problem solving;
- systems that include multiple human, artificial or both types of agents;
- organizations are systems of activity, they are involved in one or more tasks;
- organizations are goal directed;
- organizations can affect or be influenced by their environment;
- they have knowledge, culture, memory, history, and capabilities different from their member agents;
- have legal standing different from that of individual agents.

Organizational structures exist in most of the biological species (including humans) in various forms (families, tribes, classes, societies, etc.) as a way of coordination between individuals and as a means of ensuring better survival chances in certain environments. Presumably, such social and organizational structures exist for a very long time. On the contrary, organizations such as modern bureaucratic states, Coca Cola, IBM, Greenpeace, etc., which is generally what we mean when we speak of organizations in everyday life are surprisingly recent. Organizations of this type began to emerge only during or after the Industrial Revolution. It is useless to stress the importance of these organizations in our lives as it is generally known that they are the pillar of the economic and technological development in many countries. One of the causes of the existence of such organizations is to overcome the limitations of individual members. In [CG99] four types of basic limitations are discerned:

- *Cognitive*, individuals have cognitive limitations. They can not percept all of their environment and have to cooperate to achieve better understanding of their surrounding environment and a better performance.
- *Physical*, individuals are physically bounded in terms of the resources available to them as well as in terms of the actions they are capable of. They need to cooperate to achieve better productivity levels.

- *Temporal*, individuals are not eternal, they are limited in time. They have to join with younger individuals to secure the achievement of the goals or the circulation of their results beyond their lifetime.

- *Institutional*, legal or political limitations of individuals are an incentive for the creation of organizations that are able to remove such limitations.

3.1.1. Organizations as Rational, Natural and Open Systems

Research and study on organizations has produced many theories what organizations are, how they function and what principles and factors drive their development. These theories can be summed up in three different perspectives [Sco02] which will be briefly introduced below:

- organizations as rational systems;

- organizations as natural systems;

- organizations as open systems.

The perspective of *organizations as rational systems* focuses on the characteristic of organizations to accomplish specific goals and on the formalization of rules and roles. Under this perspective organizations are created for the explicit purpose of achieving specific goals. The goals of the organization determine the organization's structure, its tasks and the allocation of resources. The behavior of the organization is specified through a set of formal rules, roles and role relationships. The main characteristics of rational organizations are thus the specification of goals and the high degree of formalization of its structures where roles are independent of the personal attributes of the members occupying these roles.

In [Sco02] four rational schools are identified:

- Scientific Management which uses time and motion studies to optimize work processes and increase productivity.

- Administrative Theory which presents guidelines of how to formalize organizational structures.

- Bureaucratic Theory which focuses on the analysis of bureaucracy as a specific type of administrative structure.

- Rational Decision-Making which analyzes how goal specification and structure formalization contributes to rational behavior in organizations by focusing on decision-making.

Scientific Management and Rational Decision-Making stressed the individual actions and decisions of the members of organizations where the formal structure is handled as a framework which affects the member's behavior. Administrative Theory and Bureaucratic Theory attempt to conceptualize and analyze organizational forms [Sco02].

While considering *organizations as natural systems* they are viewed as social groups attempting to adapt and survive in their particular environment. The natural perspective developed mainly as a critical reaction to the rational perspective. The natural approach suggests that an organization consists not only of a formal structure but also of an informal structure which may play an important role in determining the organization's behavior. Organizations can also have multiple goals which can be complex and conflicting. In [Sco02] three natural system approaches are reviewed.

- *Barnard's Conception of Cooperation* handles organizations are handled as cooperative systems integrating the contribution of their individuals.
- *Selznick's Institutional Approach* stresses that individuals do not act purely based on their formal roles and the organization structure includes not only the formal structures but also the informal systems that link individuals within and external to the organization.
- *Parson's Social System Model*, specified what an organization must meet to survive in terms of the AGIL system (Adaptation, Goal attainment, Integration, Lattency).

The *open system* perspective focuses on the ties that link the organization to its environment. The environment is handled as the source of information, materials and energy. Organizations are treated here both as hierarchical systems and as loosely coupled systems. Systems are composed of multiple subsystems and systems themselves may be part of larger systems. Interconnections within subsystems are generally tighter than the connections between subsystems. In [Sco02] following open system approaches are described:

- *Contingency Theory* that evolved from the general assumption that organizations that best fit the demands of their environments will have the best adaptation abilities. So the best way to organize is closely related to the nature of environment of the organization.
- *Systems Design* that uses simulation, probabilistic and statistical techniques to study complex systems focusing on modeling flows of information materials and energy.
- *Weick's Theory of Organizing* that is a social psychological approach which shifts the focus from structure to process. The organization is enacted through the interpreted meaning of the individual interactions.

Several attempts have been made to combine the three perspectives. In [Sco02] a cross classification of the three perspectives is made based on open/closed and rational/natural perspectives.

3.1.2. Computational Organization Theory (COT)

Another approach on understanding and analyzing organizations that has emerged in the last years is the Computational Organization Theory (COT) which uses computational and mathematical models to study organizations. COT focuses on both the study of human organizations which have the ability to continually gain, transform and produce information, and on the study of computational organizations consisting of multiple agents. Theories developed from COT are usually founded on the cognitive and information-processing views of individual behavior and have contributed to the evolution of these views by adding an organizational perspective. In the information processing view on organizations the basic characteristics are [CG99]:

- Bounded rationality. Agents in organizations have rational bounds. In addition to previous specifications of bounded rationality in older information processing theories agents have now two types of bounds: limits in capabilities and limits in knowledge. Capabilities limits depend on the agent's cognitive and computational limits. Knowledge limits depend on the ability limits of the agent to learn and on the agent's intellectual history.

- Information ubiquity. Information is distributed in the organization among its agents.

- Task orientation. Organizations and its member agents are continuously involved in executing tasks.

- Distributional constraints. The performance of organizations depends on how the information is distributed among its member agents and how can be searched for informations. The organization's culture is the distribution of information and processes across the member agents of the organization.

- Uncertainty. Uncertainty about environmental conditions, task outcome etc. affects organizational activity.

- Organizational intelligence. Organizational intelligence resides in the distribution of knowledge and processes among the agents of the organization. As part of their learning capabilities organizations can redesign themselves and their view on their environment basing on the informations available. Thus, they can change their intelligence and/or their organizational performance.

- Irrevocable change. Organizations and agents that learn alter their intelligence structure irrevocably. This puts an emphasis to the order in which agents and organizations learn things.

- Necessity of communication. The ability of organizations to handle as a unit requires the agents of the organizations to have the ability to communicate with each other.

Modeling organizations requires modeling factors like the agents of the organization and their capabilities, the organizational structure, the tasks that the organization has to execute, the environment, the technology that the organization uses to process information and stressors like deadlines, legislation changes, etc. The capabilities and knowledge of agents in an organization as well as the situation they are in and the tasks that agents are performing determine the actions and decisions that agents make. The organizational roles in which agents are situated limit the range of the actions that the agents can make. Positions in an organization specify the tasks that agents that occupy those positions can carry out, their communicating partners in terms of roles or positions etc.

The *organization structure* includes procedures and rules that represent organizational knowledge (such as procedures for hiring, firing, moving and promoting agents) and the relations between agents and tasks. In the overall organization structure along with the communication and authority structures other structures are included. Informal structures (friend networks, advices, etc.) are for example a subset of these additional structures along structures that define how subtasks have to be executed (task structures), what resources tasks need to be executed (task-resource structures), and what skills are needed by the agents to execute tasks successfully (task-skill structures).

Organizations and its agents are always involved in the execution of one or many tasks. Tasks can contain subtasks. Such subtasks can be distributed to the other agents for execution. However, several dependencies may exist between subtasks such as *reciprocal* when subtasks depend on each other, *pooled* when subtasks are needed from another task and *sequential* when subtasks have to be executed in a specified sequential order. The set of tasks that an organization has to carry out is called the task environment. The task environment can vary along the degree of repetition (agents keep doing the same thing), volatility (how fast the task environment changes), bias (the degree that all tasks, regardless of differences in their features, have the same solutions) and complexity (the information that has to be processed to execute the task). The performance of an organization can be measured relying on the tasks that it performs. The basic performance measures are three: effectiveness (how well tasks are being executed), efficiency (is the ratio output/input optimal?) and perceived effectiveness (how is the effectiveness of the organization perceived by the interest groups). Effectiveness itself has three aspects which include the relative performance of the organization compared with other organizations, accuracy (how many correct decisions are made) and timeliness (how much time takes to make a decision).

The organization technology has also been introduced in the organization models to be able to view how it affects the actions of the agents within the organization. One way of modeling technology is to handle it as a tool and distinguishing tools by their attributes. Another approach is handling/modeling technology as an agent where agents that have a technology have access to the technology agent.

As large differences can exist in the relationships between environments and organizations, tasks required to be executed by organizations, goals of organizations, cognitive abilities of the individuals/agents in the organizations and legal framework on organizations, it is generally accepted that there is no organizational structure that is optimal in all domains, circumstances, legal frameworks, etc. The key toward the right organizational structure is understanding and evaluating the specific relevant factors that play a role in the performance of the organizations.

3.2. Organizational Perspective of Multi-Agent Systems

Intelligence is increasingly perceived as closely related to interaction. Intelligence and interaction are linked to each other and multi-agent systems seem to be the approach that naturally embodies this view. Multi-agent systems handles interactions between its agents as central to its overall design goals. However, most of the time interactions between agents in MAS were treated on the coordination level without any structural level on top of it. Agent-based systems were generally viewed as systems which did not require any real structure. Such a view on MAS has several drawbacks [Jen00]. One of them is that in such systems the outcome of the interactions becomes unpredictable. Predicting the overall behavior of the system becomes literally impossible due to the emergent behavior that evolves out of uncontrolled interactions. In software engineering the development of modular software is also a central issue. Multi agent systems did not seem to support modularity as for large systems agents represent too small entities to be viewed as subsystems in terms of the modular approach.

The above concerns led to the gradual development of an additional perspective on MAS, *the organizational perspective*, where the concept of organization plays a central role. In fact, organizations and organizational structures have been present since the early phases of research on multi-agent systems. However, they have been mostly implicitly or informally presented as they were often regarded as a marginal issue, more or less a technique of coordination [Gas01]. An important emphasis in MAS lied earlier on the agent, the agent's state and the relation between these states and the agent's behavior. Only recently, the concept of organization is becoming increasingly important in MAS and the attention is shifting from the agent and its internal states to organizational concepts such as organizations, groups, communities, etc. [FGM03]. Almost all multi-agent systems form organizations [HL04]. Lacking a standard definition on organizations in multi-agent systems the basic characteristics of the organizational perspective in multi-agent systems will be presented as proposed in [Jen00]. The nomenclature of the characterization made in [Jen00] follows that of [New82]. In [Jen00], several dimensions are distinguished:

- *The system* to be characterized is an agent organization.

- *The components* are the primitive entities of which the system is composed. For agent organizations, *agents*, *interaction channels*, *dependencies* and *organizational relationships* are distinguished as their components. The interaction channels enable agents to communicate and interact with each other. Dependencies between agents drive agents to interact with each other [Cas98]. Dependencies can exist in form of shared resources or they can be found in the agents' objectives. Besides, various organizational relationships can exist between agents such as authority, competition or cooperative relationships.

- *The compositional laws* define how the components are assembled to build the system. For agent organizations, such laws are specified in roles and organizational rules. The concept of roles is key to the organizational perspective in multi-agent systems. The role's purpose is to specify functionalities and to accomplish specific objectives without referring to agent architectures or agent technologies. Thus, roles isolate the organization from the agents. Roles enable modeling the organizational structure separated from the agents. The agents are not the organization anymore. They simply act in the roles specified in the organization. The concept of roles is therefore a powerful tool to model agent behavior within an organization on an abstract level. Role specifications include objectives, responsibilities of the agents that act in these roles, the specifications of the rights and the specification of the system resources that the agents need to accomplish the objectives [Gas91].

  Organizational rules accompany role definitions. Among other things they specify under what terms and conditions agents can adopt specific roles, how to react if roles are modified and how to resolve conflicts between roles.

- *The behavioral laws*, specify how the composition of the components affects the overall system. In the organizational context it refers to the organizational rationality, the dependency of the behavior of the organization on the ways in which the roles are enacted and the degree to which the organizations rules are adhered to. For agents, the principle of rationality refers to their ability to ponder on their objectives and actions and to choose the actions that they believe will help them most to accomplish their goals. Organizational rationality represents the degree to which agents follow their organizational obligations and the circumstances under which agents follow these obligations. Organizational rationality defines the cases in which agents can make social commitments, possibilities to violate these commitments, and the compensating actions that should be demanded in such situations.

- *The medium* refers to the elements that are processed to achieve the desired system behavior. Agents process knowledge to secure their goal. In an organizational context there are three types of elements included in the medium. The organizational and social obligations that agents commit to either through the enactment of roles or through social interaction is one type of these elements. The means and mechanisms through which the components influence the behavior of one another constitute the second type of elements included in the medium. Such mechanisms are, for example, negotiation techniques, cooperation protocols, and coordination models. The final type of elements included in the medium are the various means available for modifying the organizational structure. These means include, for example, the elements processed for the creation of new roles or for the modification of the organizational rules and rationality.

An interesting contribution to the organizational perspective of multi-agent systems in software engineering has been made in [WEM06]. In this contribution the concept of *organization units* is introduced. The concept of organization units presented here is more general than the concept of agents as individual agents are specified as the simplest kind of organization units. All other kinds of organization units have a more complex nature. Such complex organization units are presented in [WEM06] as entities with two main requirements. The first being that an organization unit has to serve as a platform for its members and place them within an organizational context through the concepts and mechanisms described in [Jen00]. The second basic requirement of complex organization units is that they should be able to appear as holistic and autonomous from outside by showing agency properties. This second characteristic enables agents as well as complex organization units to enact roles within a parent organization unit. The internal structures of the child organization unit are irrelevant for the parent unit. Organization units allow the creation of multi-level organizations by this method.

In [WEM06], a reference model is developed around the concept of organization units. This model relies on some basic hypotheses on organization units. One of the hypotheses refers to the important role that the *autarchy* of organization entities will play in organization oriented software development. Autarchy of an organization unit represents the degree with which this unit uses its own resources for everything it needs. The aggregation of organization units results in a greater degree of autarchy for both units taken as a whole. Another basic hypothesis is that complex organization units should offer *physical* as well as *logical platform* functionalities. Platforms in multi-agent systems are typically physical as they often represent execution environments where agents are created or destroyed and where agents can enter or leave. In the context of organization units it means that if an organization unit serves as a physical platform for another then the embedded one can affect the wider environment only through the parent unit. This implies a strong coupling between these organization units with the consequence that an organization unit can be physically embedded only within one platform unit. In a logical platform relationship, the embedded unit and the platform unit are loosely coupled and an organization unit can be logically embedded within many platform units.

The reference model proposed in [WEM06] contains four abstraction levels on organization units:

- *Internals* is the level that represents the internals of the organizations. Within this level the composing organization units are strongly coupled. The notions and methods proposed in [Jen00] can serve to model this level.

- *Organization*. In this level organizations are modeled as holonic entities. They can act, use resources, interact with other organizations and manage specific relationships to other organizations. Such relationships can include nested relationships in the form of enacting a role of a parent organization.

- *Markets/Communities* are modeled in this level as collections of organizations. Markets and communities have their own rules and their main function is to organize the way that the organizations within them interact with each other. Organizations within markets/communities are loosely coupled with one another.

- *Societies*, refer to the networks containing the markets and communities described above. Societies have rules and laws which are valid for all communities and markets within them. Such overall rules and laws specify, for example, how migrations between communities or markets can take place or how markets/communities can communicate with each other.

Between these four levels there are three types of relationships. For each of them, the involved levels can have a different perspective. Thus, there are six relationship perspectives. Each of the above abstract levels can be modeled as an organization unit. Additionally, depending on the context organization units can be modeled to be in the role of each of the four abstract levels. In the example given in [WEM06] universities were once viewed as societies (fourth level) and on another context as organizations (second level).

The reference model presented in [WEM06] remains however still mainly a theoretical work for multi-agent systems that are designed basing on it have yet to be deployed. Basing on the characterization in [Jen00], organizations are a collection of roles among which several relations exist. This point of view has been submitted to some criticism in [FGM03] for the lack of partitions in organizations which would help to represent sub-organizations, departments, etc. In [FGM03], the possibility of partitioning the organizations has been additionally introduced. The characterization from Jennings abstracts however, from implementation details and his proposal has been the basis for many of the organizational multi-agent systems developed after its publication.

In [HL04], a survey of organization types used in multi-agent systems is made identifying hierarchies, holarchies, coalitions, teams, congregations, societies, federations, markets, matrix organizations and compound organizations as the main organization forms used. All these notions are shortly introduced below leaning mostly on the work in [HL04].

In *hierarchies*, agents are arranged in a treelike structure where agents higher in the tree have a more global view than those below them. Interactions do not take place across the tree, but only between connected entities. The data coming from the agents in the low levels travels in a hierarchy upwards to provide a broader view, while control flows downward [BG88] as the agents in the higher levels provide direction to those below them. In a simple hierarchy a single member on the top hierarchical level has the decision making authority over the system. Decision requests travels here up to the top agent and when the decision is finally made it travels downward. In uniform hierarchies the authority is distributed among many agents in different areas. Decision requests do not necessarily need to travel to the top but can be processed from agents in lower levels which have the needed information and the required authority to take the decisions. Such an approach has efficiency advantages through locality. In multi-divisional hierarchies the organization is divided along division lines. These lines are normally specialized for specific products, goals, etc. and have complete control over their product. The division lines may be situated under a higher level hierarchy which evaluates their performance. However such higher entities do not have direct influence in the decision process inside the division line.

Holarchies are an organization form where the defining characteristic is the partially-autonomous *holon*. Each holon is composed of one or more subordinate entities, and can be a member of one or more holons. Holons have thus a dual nature and can exist as both entities composed of a collection of other sub-entites and as part of larger entities. Holons have generally neither a complete nor a completely absent form of autonomy as it would let the holarchy degrade in a strict hierarchy or an unorganized grouping. Subordinate holons relinquish some of their autonomy to the superordinate holon. Holons can display their capabilities and actions as an entity and hide their internal structure from their parent holons. Thus they can create an abstraction over their internal structure.

Coalitions are generally goal directed and short-term groups that are formed to achieve an objective. As soon as the objective no longer exists coalitions dissolve. Coalitions can form among both cooperative as well as self-interested agents. Within a coalition, the organizational structure is typically flat. However, in coalitions there may be a leading agent who acts as the representative of the coalition. If coalitions are created they may be handled as atomic entities.

Teams, are groups of cooperative agents that work together to achieve a common goal. Unlike coalitions, teams attempt to maximize the utility of the team and not that of the members. Generally, each team member accepts one or more roles with which it addresses subtasks from the general team goal. Such roles may change over time as a reaction to planned or unplanned events.

Congregations, are groups of individuals with a flat organizational form. Unlike coalitions or teams congregations are long lived and do not form with a specific goal. They are created among individuals with similar or complementary characteristics to facilitate finding collaborators.

Societies are long lived social constructs. Agents may come and go, live and die but the society remains there acting as a platform where agents can interact. Agents in a society may have different goals, different levels of rationality, and different capabilities. Societies provide a system through which they can act and communicate with each other. Societies may have a specific structure but they allow a flexible arrangement of interactions. Agents in societies may be organized in sub-organizations. Societies have a set of constraints on the agents' behavior which are generally referred to as social laws or norms. Such constraints are rules by which agents have to act. They provide some level of consistency of behavior and are aimed to facilitate the coexistence of the society's members.

Federations are groups of agents which have ceded some of their autonomy by choosing an agent which represents the group. Group members interact only with this agent that acts as an intermediary between the group and the environment.

Markets consist of buying agents, sellers, or sometimes designated third parties called auctioneers. Buyers may request or place bids for a common set of items, such as shared resources, tasks, services or goods. Sellers and auctioneers are responsible for processing bids and choosing the winner. Unlike in federations, market members are competitive. However, they still trust the entities managing the market and abide by decisions they make. Any agent can take part in a market system as long as it follows the systems rules.

In *matrix organizations*, many managers or peers influence the actions of an agent.

Compound organizations are based on more than one organizational structure. In a system, control can be deployed according one form of organization and the data can travel according another organization form. Compound organizations can be overlapped, or be nested, so that subsets of agents in a group are organized in a potentially different way within the larger context.

The organizational perspective of multi-agent systems has already changed the academic and research landscape of multi-agent systems through a paradigm shift from agents to organizations of agents. It has the potential to dissolve some serious drawbacks of multi-agent systems and to make agents an alternative to objects in software development. However, being a relatively new approach, effective modeling and implementation tools that support organizational thinking in the development of multi-agent systems or general software have yet to be developed. The development of such tools may as a consequence encourage the acceptance from the industry of multi-agent systems and design.

3.3. Summary

Recently, research on multi-agent systems is especially being influenced by the organization theory from sociology. Organizations lack an exact definition, but there are some properties of organizations that are generally agreed upon. Organizations are viewed as goal-directed large scale systems for problem solving which can be involved in many tasks. Organizations are also linked to their environment. They can affect it or be influenced by it. Organizations can have different capabilities, knowledge, memory, culture, history as well as limitations from their member agents. Organization theory delivers three main perspectives on organizations including the rational perspective (organizations are viewed as systems which have specific goals and function on formal rules and roles), the natural perspective (organizations are viewed as social groups attempting to adapt and survive in their environment) and the open perspective (focuses on the ties of the organizations to its environment). Recently, computational and mathematical methods in the study of organizations have given birth to Computational Organization Theory (COT). COT studies organizations from a cognitive and information processing view. Important characteristics ascribed to organizations in this view are bounded rationality, information ubiquity, task orientation, distributional constraints, uncertainty, organizational intelligence, irrevocable change and necessity of communication. Organization theory has flowed in the research on multi-agent systems and has had a deep impact on it. The focus in multi-agent systems has shifted from the internal agent states toward organizational concepts such as groups, communities, organizations, roles, etc. Since concepts from the organization theory have been adopted for multi-agent systems the structure of multi-agent systems is no longer incorporated in the agents in form of interaction or coordination rules. Rather, the structure is viewed and implemented explicitly and independently of agents. This ensures that the multi-agent system becomes more manageable and predictable. The main dimensions of multi-agent systems in the organizational perspective include [Jen00] the system, the components (entities of which the system is composed), the compositional laws (define how the components are assembled to build the system), the behavioral laws (specify how the composition of components affect the overall system) and the medium (the elements processed to achieve the desired behavior of the system). The organization types mostly used in multi-agent systems have been shown in [HL04] hierarchies, holarchies, coalitions, teams, federations, markets, matrix and compound organizations.

Chapter 4.

Petri Nets, Renew and Mulan

Petri nets have been further developed and adopted as a modeling and specification formalism since their proposal in 1962. In this chapter after a short introduction to petri nets in general, P/T petri nets and reference nets will be presented. Further, a brief introduction to "Renew", a reference net editor will follow. Finally, a multi-agent reference architecture based on reference nets and Renew will be explained.

4.1. Petri Nets

Petri nets represent a mathematical formalism for the description and/or specification of concurrent processes or systems. Petri nets were first introduced in 1962 by Carl Adam Petri in his dissertation thesis "Kommunikation mit Automaten", submitted to the faculty of Mathematics and Physics at the technical university in Darmstadt, Germany. Since then, the Petri net formalism has been widely adopted and further developed as a modeling and specification tool for concurrent systems. The main reasons of the success of the formalism of Petri nets are the rather intuitive graphical representation of the formalism which is very forthcoming for humans and its mathematical robustness which makes it adequate for research on formal properties and facilitates automated machine processing.

Petri nets consist of passive elements called *places* and active elements called *transitions*. A place is typically drawn as a circle while a transition is drawn as a square. A place contains a set of markers called *tokens*. Single places with their tokens represent local states of the system. The distribution of tokens in all the places of the net is called a marking. A marking represents the overall state of the system. Transitions represent events or actions whose execution requires specific states and produces other states. The change of the system state through the execution of events or actions (transitions) is described through the *flow relation* represented through directed arcs. The directed arcs link places with transitions or vice versa (the arcs do not link places with places or transitions with transitions). The places from which an arc links to a transition are referred to as the input places of the transition while the places to which arcs link from a transition are referred to as the *output places* of the transition.

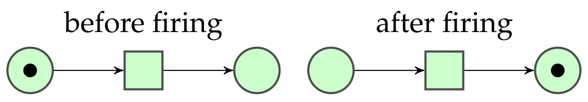


Figure 4.1.: Petri Nets

A transition is said to be *enabled* if a specific distribution of tokens exist in its *input places*. If a transition is enabled it can *fire* which means that it can remove a specific number of tokens from all its input places, process some tasks and add specific number of tokens to all its output places. After a transition fires it changes the system state.

In [JV87] petri nets are defined as a triple of sets. They represent a bipartite and directed graph which consists of *places*, *transitions* and *directed arcs* which link places and transitions as described above. The formal definition is presented here as given in [JV87]:

$N = (P, T, F)$ is a net if

- P is a finite set of places and T a finite set of transitions
- $P \cap T = \emptyset$
- $F \subseteq (P \times T \cup T \times P)$ is the *flow relation*.

A marking of N is a multiset of places $m \in \mathbb{N}^P$. The firing of a transition is represented by $m \xrightarrow{t} m'$. Firing sequences are represented by $m \xrightarrow{t_1 \cdots t_n} m'$. The set of the *reachable markings* is defined as $RS(m) = \{m' \mid \exists t_1 \cdots t_n : m \xrightarrow{t_1 \cdots t_n} m'\}$

The *preset* of a node (a place or a transition) y is ${}^\bullet y := \{x \mid xFy\}$. The *postset* of a node y is $y^\bullet := \{x \mid yFx\}$.

4.2. P/T Nets

P/T Nets are a subclass of low level petri nets. The main characteristic of low level petri nets is that their tokens are undistinguishable which means that one token is identical to every other token in the net.

P/T nets extend the above definition of petri nets by adding capacities to places, an initial marking function and weights to arcs. The formal definition is given below as in [BDC92]:

A P/T net is a tuple (P, T, F, K, W, M_0) where

- (P, T, F) is a net as described above.

- $K : P \rightarrow N^+ \cup \{\infty\}$ is a capacity function which describes the maximal number of tokens that a place can contain.
- $W : F \rightarrow N^+$ is a weight function for the arcs. It defines how many tokens have to flow through them.
- $M : P \rightarrow N$ is an initial marking function that satisfies $\forall p \in P : M_0(p) \leq K(p)$.

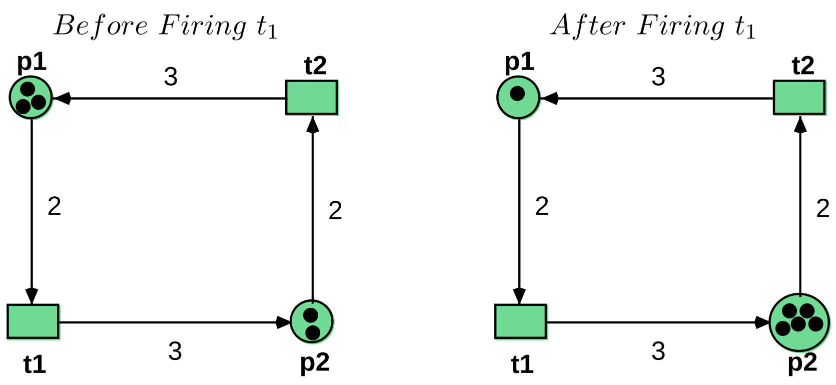


Figure 4.2.: P/T net before and after firing.

While the formal definition may seem a little complicated the graphical representation of P/T nets as that of most other Petri net types facilitates understanding and is much more readable. A transition in a P/T net complying to the above definition is activated and can fire only if all it's input places have at least as many tokens as specified in the input arcs that link them to the transition. The weigh function of the arcs has the same effect as multiple arcs between the same place and transition. In Figure 4.2 a P/T net is shown before and after one of its transitions fires.

4.3. Reference Nets

Reference nets [Kum02] are high level petri nets that can contain objects or other nets within their places as tokens. In high level petri nets, tokens are distinguishable. Reference nets use reference semantics as their tokens can have a data type and can be references to other nets. The net where the referencing token is contained is called *the system net* while the net referenced from the token is called *the object net*. The object net can itself contain tokens that refer to other nets. This "nets within nets" [Val95] paradigm makes reference nets very interesting as it allows hierarchical structures and modular design for large and complex systems. Reference nets are closely related to Renew [KWD01]. Renew is a reference net editor. It combines the "net within net" paradigm with the Java language and its object-oriented approach. Reference nets are instantiated like classes are instantiated in object-oriented languages. The structure, initial marking and behavior of instances of reference nets is specified in net templates.

4.3.1. Reference Net Elements in Renew

Compared to P/T nets, reference nets not only have other types of tokens but also additional inscriptions, the possibility to instantiate nets and a close affiliation to the Java programming language.

The tokens in reference nets can be of any type available in the Java programming language. Thus, tokens can be any kind of objects. Additionally, anonymous tokens (equal to tokens in P/T nets) are also available. They are represented in Renew through square brackets "[]". Besides, in reference nets, a list of data values of different types, objects as well as primitives, can be specified as a token. Such tokens are represented by a tuple of the form "[object,int,..]".

Reference nets have several inscription types including arc, place and transition inscriptions. Arc inscriptions can be variables or constants. Additionally, output arcs can have expressions as inscriptions. Place inscriptions can represent tokens as well as define the place type. If a place is typed it means that only tokens that have the type ascribed to the place can be contained within the place. Transitions can have a range of inscription types including *guard*, *action*, *expression*, *creation* or *synchronous channel* inscriptions. Guards are boolean expressions that have to be **true** so that the transition can fire. Beyond guards resulting to **true**, in order to fire a transition all its input places have to contain a sufficient number of tokens from the type specified in the arc inscriptions. Expressions are evaluated before the firing of the transition while actions are expressions that are evaluated during the firing.

Reference nets are object oriented nets. They are objects of the class *NetInstance*. In fact, creation inscriptions are used to create new *net instances*. Net instances are instances of *net templates*. The relation between net instances and net templates is similar to the relation between objects and their classes in object-oriented programming languages. Net templates are similar to classes. They define the initial marking and the behavior of their net instances. Net instances have the same behavior as defined in their net template, but they have different identities and, at a specified time, their states, represented through their markings, can be different. In a reference net, tokens that represent nets reference in fact net instances. A net instance can also create other net instances.

Synchronous channel inscriptions represent synchronous communication channels which enable communication between net instances. Synchronous channel inscriptions consist of two types of inscriptions, *up-links* and *down-links*. Up-links are used in object nets while down-links in system nets. Synchronous channels between reference net instances are specified in [CH92]. They consist of at least two transitions where one of the transitions is seen as the initiator of the communication having a down-link inscription. Transitions can have only one down-link but they can have many up-links. The transitions involved in a synchronous channel fire simultaneously as soon as all of them are activated. Communication with synchronous channels is bidirectional and is achieved through passing objects as channel parameters between the involved transitions.

Reference nets have some additional arc types as shown in Figure 4.3. They are test arcs, reserve arcs and flexible input and output arcs. Test arcs and reserve arcs are very similar. Reserve arcs behave equally to two opposite unidirectional arcs connecting the same place and transition. Reserve arcs take a token from the place and put it back again during the firing process of the transition. Test arcs do not take the token away so other test arcs can have simultaneous access to the same token. Simultaneous access to tokens is not possible with reserve arcs. Flexible input arcs are used to remove all the objects of a *Collection* object within a place. A flexible output arc puts all the elements of a *Collection* object in a place.

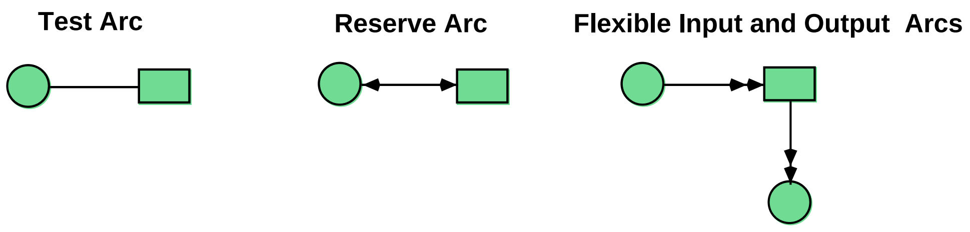


Figure 4.3.: Additional arcs in reference nets.

Virtual places are also elements specific to reference nets. A virtual place represents one original place in the net. To have an arc to a virtual place would have the same effect as having an arc to its original place. If an arc removed a token from a virtual place than this token would not be even in the original place anymore. Virtual places are elements intended to further simplify modeling reference nets. Graphically, they are represented as a place surrounded by a double line as shown in Figure 4.4.

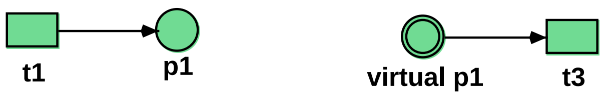


Figure 4.4.: Virtual place.

Declaration nodes are a special kind of inscription used in reference nets. They are used to declare Java variables used in the reference nets or to import needed Java classes. A reference net is allowed to have mostly one declaration node.

4.4. Renew

Renew [KWD01] is a graphical tool for creating, editing and simulating reference nets. Renew is implemented in Java. It unites the advantages of the object oriented paradigm and the “nets within nets” approach in reference nets. In Renew, the behavior and the initial marking of reference nets is specified through net templates. Net templates correspond to classes in object oriented languages. All net instances of a net template have the same behavior. Nevertheless, net instances can have different states at a specified time. Net templates are created (drawn) in graphical files with the Renew Editor. Net instances are created during the simulation process.

4.4.1. Renew Editor

Designers work with Renew mainly through editing net templates. This is similar to programmers editing classes (not objects) in object oriented languages. Net templates are created with the Renew editor in graphical files (*.rnw*). The editor offers features that facilitate the creation of net templates. The editor's graphical user interface as shown in Figure 4.5 is composed of a range of icons and a status bar. The first row of icons represent visual features that can be added to the net template to make the nets more readable to designers/programmers that work with them. These features have no formal meaning for the nets and are ignored during simulation. The second row of icons represent features with which formal net elements (places, transitions, arcs, tokens, inscriptions) can be created or edited. Net elements can also be dragged, selected or edited only with mouse actions on them, without clicking on the feature icons.

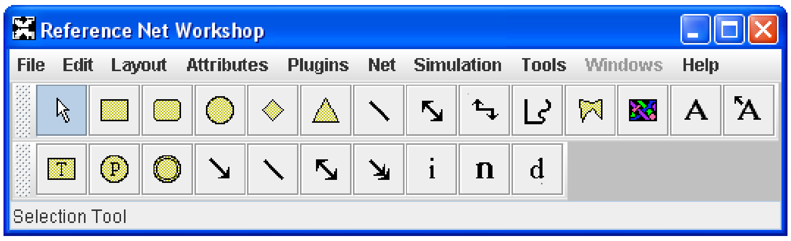


Figure 4.5.: Renew editor GUI.

4.4.2. Renew Simulator

In Renew, reference nets can be created as "shadow net system" files (*.sns*) or as graphical files (*.rnw*). Shadow nets are used for simulation and are basically an abstraction of the graphical nets. Shadow nets strip all unnecessary information such as color, position of net elements and leave only the needed information for use in the simulation. A graphical file can be compiled in Java. After compilation of the graphical file a simulation can be started. During the simulation, a net instance is created and can be viewed in a separate window as its active transitions fire. Simulation is used in Renew to view firing sequences of active transitions in reference nets. Simulation can run in a one step modus where users can progress in steps where only one transition fires. Renew also offers the possibility to set breakpoints to hold the simulation process. Breakpoints can be set to places as well as to transitions. By changing the compiler, Renew can also simulate P/T nets, timed petri nets, boolean nets, etc.

4.4.3. Renew Plug-In Architecture

Renew is built on a plug-in architecture since version 1.7. The Renew plug-in architecture, which was developed and introduced in [Sch03], allows the extension of Renew with additional functionality through the use of interfaces from Renew components without changing the core of Renew. Additional functionality can be added to Renew through providing the classes of the new plug-in. This means to literally add the Java archive of the classes to the "plugins" folder inside the Renew application folder structure. Plug-ins can not be included after Renew has started. The Java archive has to be present in the "plugins" folder at the start of Renew.

In this work, the plug-in architecture of Renew is used to deliver an editor for organization and R/D nets. It is also used for the development of a tool which is able to deploy SONAR formal organizations to Mulan agent organizations. The organization editor is described in detail in Chapter 6, while the deployment tool is described in Chapter 7.

4.5. Mulan

Mulan [KMR01] is a reference architecture for multi-agent systems. Mulan is built on reference nets complying to the "nets within nets" paradigm and includes four layers. Each Mulan layer is a reference net embedded within the upper layer (reference net) as tokens. The layers communicate with each other through synchronous channels. In cite-Duvigneau02, an agent platform called CAPA has been developed which extends Mulan so that it complies to the open specifications of FIPA (Foundation for Intelligent Physical Agents) [FIP03].

In Figure 4.6, a simplified model of the four layers is displayed. The first layer is the multi-agent network where the places represent locations in which multi-agent platforms can reside as tokens. The transitions represent communication paths between the locations. The multi-agent system net layer shown in Figure 4.6 is only a simple example of how the system net layer can look like. Depending on the locations and the number of platforms the reference net representing this layer can be different.

The second layer specifies the agent platforms. Platforms represent a physical environment where agents are embedded. They can hold many agents and can manage their lifecycles through creation of new agents or destruction of existing ones. Platforms offer internal or external communication paths for their agents. Internal paths are used when agents residing on the same platform need to communicate. External communication paths are used for cross-platform communication between agents. The platform contains several components including the agent management system (AMS) and the directory facilitator (DF). An AMS is a mandatory component of the platform. A platform has to contain only one AMS. The AMS is responsible for managing the creation of agents, the deletion of agents and for overseeing the migration of agents. Additionally, an AMS can

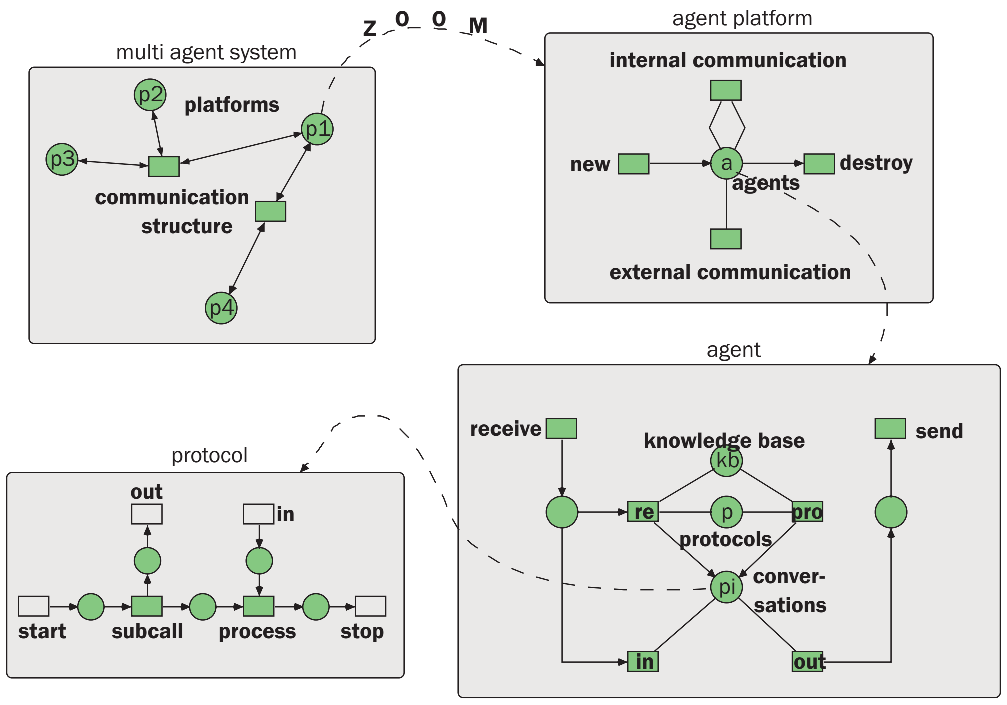


Figure 4.6.: Mulan architecture ([KMR01]).

be queried to obtain a description of its platform. The DF is also a mandatory component. However, a platform can have many different DF-s. A DF provides a yellow pages directory service to agents. Every agent within the platform that needs to make its services public to other agents, should find an appropriate DF of the platform and request the registration of its agent description. Agents that need to find other agents which offer specific services can search the DF. Both AMS and DF are part of the FIPA open specifications. In Mulan/CAPA, they are implemented as agents existing within the platform that they serve.

4.5.1. Mulan Agents

The third layer showed in Figure 4.6 is a simplified version of the agent layer. All Mulan Agents have the same structure as shown in Figure 4.6. Mulan agents reside on platforms. One place of a platform holds all the agents of that platform. Mulan agents can receive or send messages to other agents over the platforms in which they are situated. The incoming and outgoing synchronous channels of the agent provide this functionality. Mulan agents communicate in ACL (Agent Communication Language) messages as specified in FIPA. The content of the messages is encoded in SL0(Semantic Language, Level 0) [FIP02]. All the information in ACL messages and in the SL content is contained in two basic types of data structures, key-value tuples (KVT) and value tuples (VT). A KVT can have many key-value pairs. A VT is a tuple containing many values. KVT-s and VT-s also have a special name which represents the type of the tuple. KVT-s an VT-s are specified in FIPA. For Mulan, they are implemented in CAPA. Below, an example is given showing the string representation of a KVT. The type of the tuple is specified by the special name "inform". The keys are preceded by a colon ("sender", "receiver", etc.) and are followed by the corresponding value.

```
(inform
    :sender (agent-identifier ...)
    :receiver (agent-identifier ...)
    :content "((lasttaskfinished))"
    :language FIPA-SL0)
```

Agents handle their communication with other agents through Mulan protocol nets. Mulan protocol nets, which are also reference nets, define the behavior of the agent during communication. The protocol nets are all contained within a place in the agent reference net. Agents manage the lifecycles of their protocol nets by creating or destroying them.

Mulan agents also have to make decisions on how to act or how to reach their goals. Storing and searching knowledge is important to achieve this. The knowledge of Mulan agents is stored in their *knowledge base*. The knowledge base of Mulan agents is also a reference net and is shown (simplified) in Figure 4.7. The knowledge base reference net provides functionality for its Mulan agent to store or search information as key value tuples. The structure of the knowledge base reference net is simple. The white elements in Figure 4.7 initialize the knowledge base before it can be used. Initialization of the knowledge base means that a file (with the ending *.wis*) where the initial information is stored, is read and a hash table with key value tuples is created. A reference to the hash table is placed then to the *Knowledge* place in the reference net. After initialization the knowledge base can be searched or modified through the synchronous channels such as ':exists(..,..)', ':new(..,..)', ':modify(..,..)', etc. The knowledge base is typically used to store property values or match received messages from other agents to specific Mulan protocols. Thus, the knowledge base plays a crucial role in determining the behavior of the Mulan agent.

Mulan agents can be reactive or proactive. When they receive a message from the other agents they act reactively. First it is checked if the message belongs to an active conversation. If this is not the case the protocol factory, which is responsible for initializing protocol Mulan instances, initializes a new Mulan protocol instance after it consults the knowledge base. The new protocol instance is put then in the *conversations* place where it can begin the conversation with the sender agent. Responses from the Mulan protocol instances used in an agent conversation are then passed through the *out* transition to the outgoing synchronous channel where they are finally sent to the sender agent. Mulan agents can also be proactive by initializing protocol instances and starting conversations by themselves.

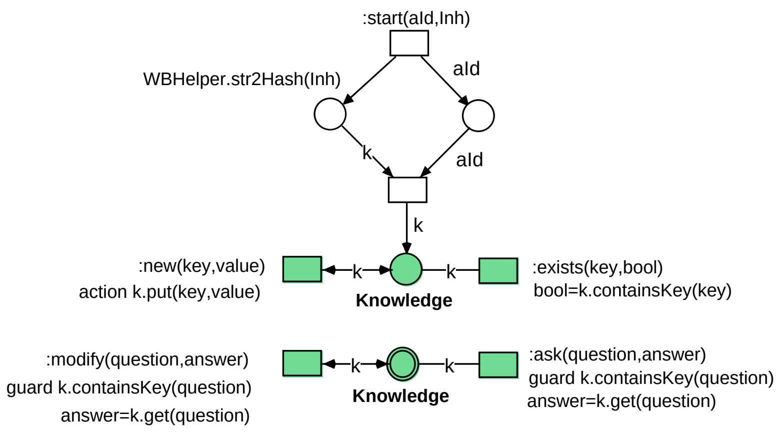
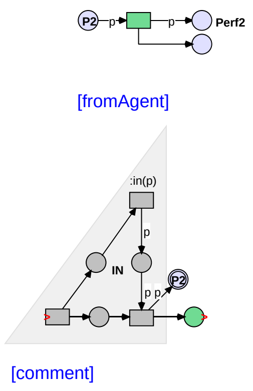


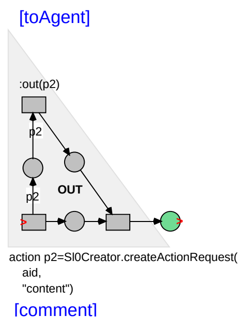
Figure 4.7.: Knowledge base of agents.

4.5.2. Mulan Protocols

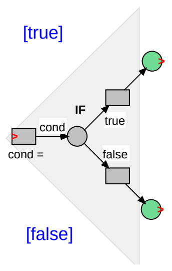
The fourth layer of Mulan represents Mulan protocols. Mulan protocols are also reference nets and define the behavior of the agents during the communication with other agents. In Figure 4.6, only a very simple example of the structure of a protocol is shown. Mulan protocols are typically more complex. During the communication of Mulan agents messages are actually exchanged between Mulan protocols. Mulan agents put some additional infrastructure information and pass the messages of their Mulan protocols to their platforms. Mulan agents control their behavior through choosing the “right” Mulan protocols for given situations by consulting their knowledge base. Mulan protocols control the conversation during their processing until they pass the control to other protocols or they finish processing. Mulan protocols determine how a message it receives is processed, what messages to send to other Mulan agents and how to change the knowledge of the Mulan agent on behalf of which it processes the message. Mulan protocols do not belong to specific Mulan agents. Rather, they can be initialized and started by any agent that can estimate their use as appropriate. Building agent applications in Mulan means that programmers most of the time create Mulan protocols and knowledge base files. In [Cab03], standard components for Mulan protocols were developed. The use of standard components in Mulan protocols for tasks occurring repeatedly makes the protocols more readable and understandable. In Figure 4.8, the standard components for sending and receiving messages as well as the components for branching and iteration over a collection are exemplary shown. Such standard components are also used in this work for the development of Mulan protocols.



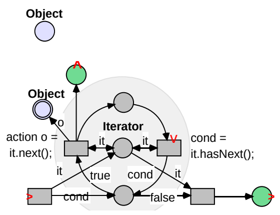
(a) Receive a message.



(b) Send a message.



(c) Branching.



(d) Iteration over a collection.

Figure 4.8.: Some standard components of Mulan protocols.

4.6. Summary

Petri nets represent a mathematical formalism for the description and specification of concurrent processes or systems. The petri net formalism has been widely adopted and further developed as a modeling and specification tool for concurrent systems since it was first proposed. A subclass of petri nets are P/T Nets. P/T nets are low level petri nets. The main characteristic of low level petri nets is that their tokens are undistinguishable which means that one token is identical to every other token in the net. In P/T nets, places can have capacities and arcs can have weights. The firing prerequisites of a transition in P/T nets depend on the weights of its input arcs and on the distribution of the tokens in the transition's input places. Reference nets are high level petri nets that can contain objects or other nets within their places as tokens. In high level petri nets tokens are distinguishable. Reference nets can particularly contain other reference nets as their tokens. This ”nets within nets“ paradigm allows the development of modular systems. Reference nets can also be instantiated similar to how classes can be instantiated with objects in object oriented programming. Reference nets are closely related to Renew. Renew is a reference net editor. It combines the "net within net" paradigm with the Java language and Java's object-oriented approach. Reference nets have some additional net elements when compared to low level petri nets such as P/T nets. These additional elements include arc inscriptions, transition inscriptions, virtual places, test arcs, reserve arcs and flexible arcs.

Mulan [KMR01] is a reference architecture for multi-agent systems. Mulan is build on reference nets complying to the "nets within nets" paradigm and includes four layers. The multi-agent network layer, the platform layer, the agent layer and the protocol layer. In Mulan, agents interact with each other through their protocols. They decide which protocols to involve during conversation with other agents. A protocol message does not go directly to the receiving protocol but is first passed to the agent on behalf of which the protocol is used. The agent passes the message to its platform (second layer) which passes the message to the receiving agent if it is situated on the same platform. The receiving agent passes the message to the receiving protocol. If the communicating agents are situated on different platforms the messages pass additionally through the first layer before getting to the receiving agent/protocol.

Chapter 5.

Models of Multi-Agent Organizations

One of the main requirements on multi-agent systems is the adaptability of its structure. Such a requirement becomes central if the multi-agent system is situated in a dynamic or unstable environment. Modeling and designing the structure of a multi-agent organization through powerful formalisms is a central issue. In this chapter some of the models will be presented. Finally, SONAR, a formal model of Multi-agent organizations introduced in [Köh06] will be described in detail. SONAR is the reference model used in this work.

5.1. Some Organization Models

In this section four organization models will be introduced. The models chosen to be presented here are not representative of all existing multi-agent organization models. Nevertheless, they delineate important aspects and dimensions in modeling multi-agent organizations [LJO05]. The models covered here are AGR, MOISE$^+$, AUML, and ISLAND.

5.1.1. AGR

In AGR (**A**gent, **G**roup, **R**ole) [FGM03], the basic concepts around which an organization is structured, as its name suggests, are *roles*, *groups*, and *agents*. An agent in AGR is an active, communicating entity that can play multiple roles within one or more groups. Agents in AGR have no constraints on their architecture or on their mental capabilities. Groups serve to partition the organization. A group is a set of agents that share common characteristics. Agents can communicate with each other only if they belong to the same group. Roles are requested by agents and are local to groups. Many agents can play the same role. Roles are an abstract representation of the functional position of an agent in a group. The *organizational structure* in AGR is what persists when agents enter or leave the organization. The organizational structure involves two aspects: a *structural aspect* and a *dynamic aspect*. The structural aspect consists of the *partitioning structure* and of the *role structure*. The partitioning structure defines how agents are assembled to groups and how groups are related to each other. The role structure defines the set of roles for each group and relationships between roles in the group. The role structure also defines the constraints that the agents should satisfy to play a specific role. The *dynamic aspect* is related to the interaction patterns defined within roles and specifies the modalities of creating, killing, entering a group and playing roles. In AGR there are two structural constraints between roles: *correspondence* and *dependence*. A correspondence constraint expresses that agents that play one role automatically play another role. Dependence constraints specify dependencies between group membership and role playing.

In AGR, the organizational structure is represented graphically by group structures (the white boxes in Figure 5.1) which contain roles represented by hexagons and interaction diagrams represented by rounded rectangles. In Figure 5.1 an example of the graphical representation of the organizational structure in AGR is shown. The interaction rectangles *I3* and *I4* represent communication (on an abstract level) between the agents that play the roles *R3*, *R4* and *R5*. The dependency *d2* expresses that all agents that play *R4* must play *R5*.

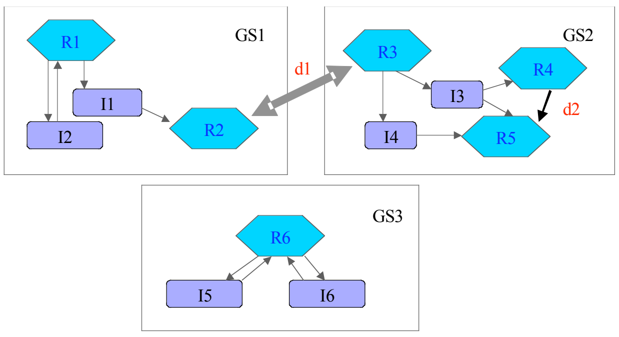


Figure 5.1.: AGR organizational structure representation ([FGM03]).

Another type of diagram, called *organizational sequence diagram*, is used in AGR to describe the dynamic organizational activities such as creating, entering or leaving a group, the acquisition of a role in a relation, etc. The organizational sequence diagram is a variant of AUML (Agent UML) [PO01].

5.1.2. MOISE$^+$

MOISE$^+$ (**M**odel of **O**rganization for mult**I**-agent **S**yst**E**ms) [HSB02] is an approach where the structural (role, links, groups etc.), the functional (goal, plans, etc.) and the deontic (norms, laws, etc.) aspects of the organization are clearly distinguished and integrated in a single model. The *structural aspect* in MOISE$^+$ defines the agents' relations through the notions of roles and links. It also includes concepts such as role inheritance, recursive groups, role compatibility and role cardinality. The former two notions have a specification purpose while the latter two constrain the role adoption by an agent according to its current roles. The *functional aspect* describes how a multi-agent system achieves its

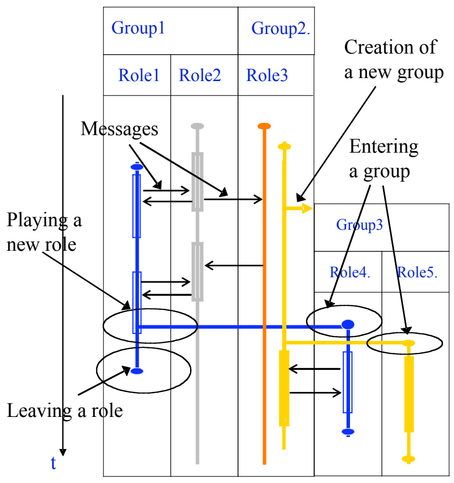


Figure 5.2.: AGR organizational sequence diagram ([FGM03]).

global goals, how these goals are decomposed (by plans) and distributed to the agents (by missions). In the functional aspect in MOISE$^+$ the concept of a global plan, called *social scheme*, and the definition of preferences between missions are included. A *social scheme* is a goal decomposition tree where its root is the *social scheme* goal and where the responsibilities for the sub-goals are distributed along missions. The *deontic aspect* describes the roles' permissions and obligations for missions. A $permission(p, m, tc)$ states that an agent playing the role p is allowed to commit to the mission m and tc is a time constraint on the permission. The time constraint tc specifies a set of time periods during which the permission is valid. An $obligation(p, m, tc)$ states that an agent playing p ought to commit to m during the time periods in tc. Permissions and obligations assign roles to missions (collection of goals). By doing so, they can be viewed as a sort of interaction protocols.

In Figure 5.3, a structural model on an example of a soccer team is displayed. Compared to the structural aspect of AGR in MOISE$^+$, additional concepts can be found such as abstract roles, inheritance and composition between roles, as well as communication and authority links. In Figure 5.4, the social scheme of the soccer team example is depicted.

5.1.3. AUML

AUML (**A**gent **U**nified **M**odeling **L**anguage) [PO01] is an extension of UML [OMG99] developed to model agents and agent based systems. In AUML, the structural aspect of organizations within a multi-agent system builds on the concepts of *agents*, *roles* and *groups* and is based on AALAADIN [FG98] with extensions from dependency theory and holonics. Roles are here not ontologically primitive but are defined as recurring patterns of dependencies and actions. The definition of a group includes not only a set of agents but also the environment through which they interact. Groups can interact with

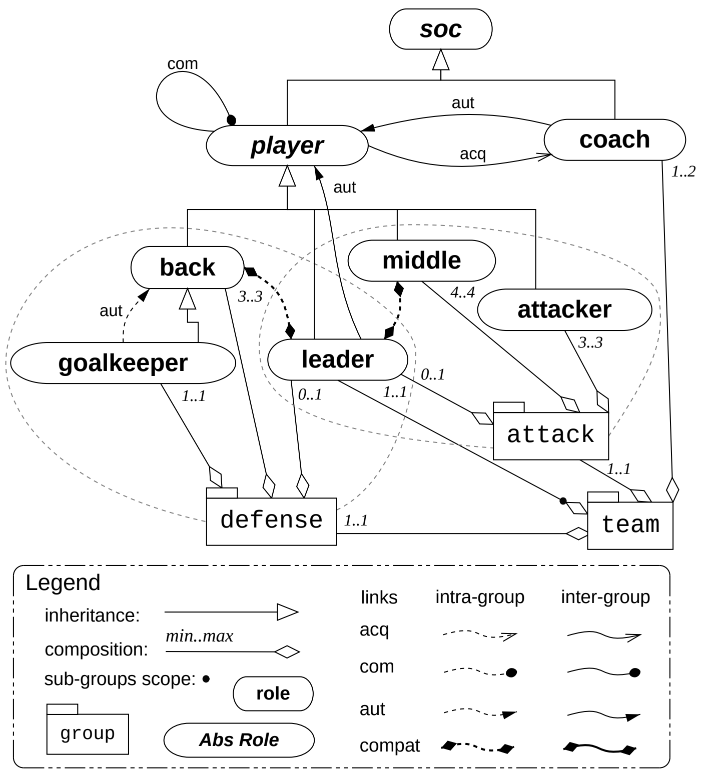
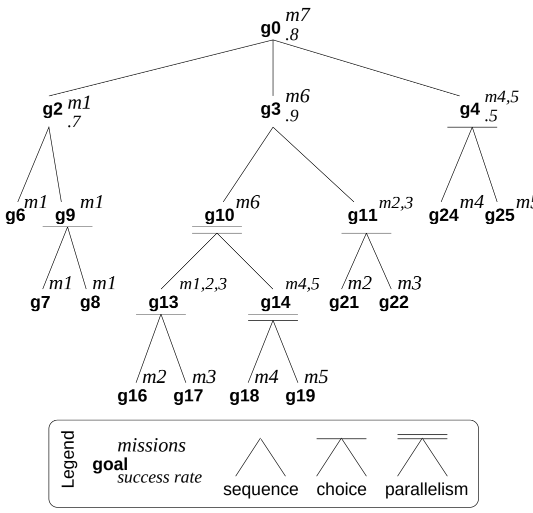


Figure 5.3.: MOISE$^+$ structural specification of a soccer team ([HSB02]).



goal	*description*
g_0	score a soccer-goal
g_2	the ball is in the middle field
g_3	the ball is in the attack field
g_4	the ball was kicked to the opponent's goal
g_6	a teammate has the ball in the defense field
g_7	the ball was passed to a left middle
g_8	the ball was passed to a right middle
g_9	the ball was passed to a middle
g_{10}	—
g_{11}	a middle passed the ball to an attacker
g_{13}	a middle has the ball
g_{14}	the attacker is in good position
g_{16}	a left middle has the ball
g_{17}	a right middle has the ball
g_{18}	a left attacker is in a good position
g_{19}	a right attacker is in a good position
g_{21}	a left middle passed the ball to a left attacker
g_{22}	a right middle passed the ball to a right attacker
g_{24}	a left attacker kicked the ball to the opponent's goal
g_{25}	a right attacker kicked the ball to the opponent's goal

Figure 5.4.: MOISE$^+$ social scheme of soccer team ([HSB02]).

each other through identified members. In the case of unanalyzed groups, groups are also permitted to occupy roles in higher-level groups by building holonic structures. In Figure 5.5, the class diagram represents the AUML meta-model. It depicts a consolidated view on the relationship between agents, roles and groups.

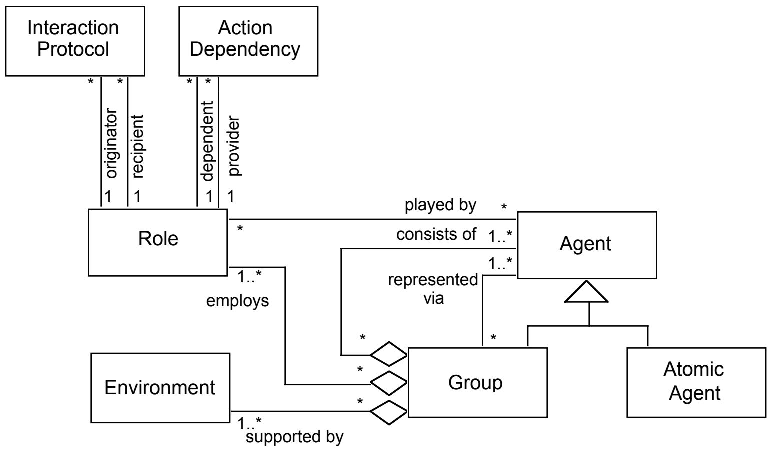


Figure 5.5.: AUML meta-model ([PO01]).

In [PO01], AUML is explained by an example of a terrorist organization engaged in arm deals with the weapon cartel and in terrorist activities in the western societies. The terrorist organization, the weapon cartel and the western societies are modeled as groups. The example includes: the *operative* and the *ringleader* as roles within the terrorist organization; the customer, the supplier and the negotiator as roles within the weapon cartel; the citizen and student as roles in the western society. In Figure 5.6, a class diagram depicts the roles from the weapon cartel group and their relationships in the example.

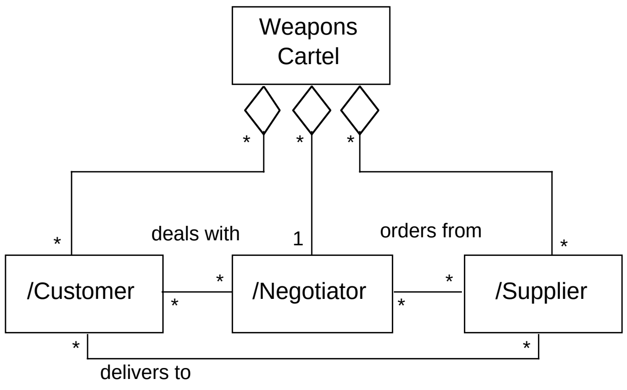


Figure 5.6.: A class diagram depicting the role relationships in the weapon cartel group ([PO01]).

Class diagrams model entities in the system and their relationships. Modeling the interactions that occur between these entities is represented using an extended UML sequence diagram as shown in Figure 5.7.

The box surrounding the sequence diagram indicates that the interaction can be viewed as a unit called *package*. The diamond shaped symbol is an extension of the sequence

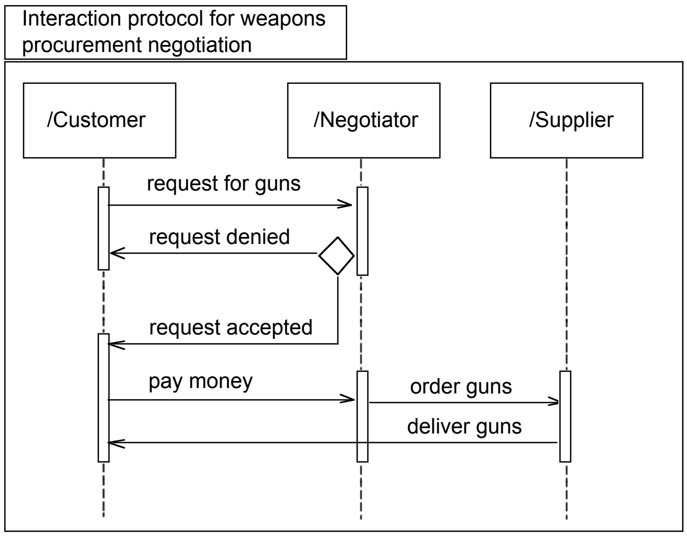


Figure 5.7.: A sequence diagram for weapons procurement ([PO01]).

diagram and represents a decision. For more AUML extensions to the sequence diagram see [OPB00].

5.1.4. ISLANDER

ISLANDER is a declarative language for specifying electronic institutions [EPS01]. In ISLANDER, an electronic institution consists of: a *dialogical framework*, *scenes*, the *performative structure* and *norms*. The dialogic framework specifies the *ontology*, the *content-language* which has to be PROLOG, KIF, or LISP, and the valid illocutions that agents can exchange with each other. The intention is to enable knowledge exchange between agents using the vocabulary of the ontology. The statements composed by the agents in the content-language are embedded in *illocutionaly particles* in accordance with the speech act theory. The dialogic framework also specifies the roles involved as well as the role relationships. Roles define patterns of behavior that the agents are required to adopt if they need to communicate. There are two types of roles: *internal roles* and *external roles*. The internal roles represent roles that can be played only by staff agents of the institution. External roles can be played only by agents external to the institution. Finally, the dialogic framework specifies the relationships between roles through its *social structure*.

The activities in electronic institutions are the composition of multiple, eventually concurrent, dialogic actions which involve different groups of agents playing different roles. For each activity, interaction between agents is specified by *scenes* following well-defined *communication protocols*. A scene models a dialogic activity. It is a directed graph where nodes represent the different states of conversation and the directed arcs are labeled with illocution schemes or with timeouts. A communication protocol of a scene describes the possible dialogic interactions between roles. The number of agents that play a specific role in a communication protocol may vary dynamically so communication protocols define a set of access and exit states for each of the roles.

The *performative structure* specifies the relationships among scenes. While a scene models a dialogic activity, more complex activities involving many dialogic activities that are related to each other can be modeled by performative structures. In a performative structure agents can navigate from scene to scene constrained by the rules specifying the relationship between the scenes. Moreover, the same agent can participate in different scenes at the same time. The connections between the scenes define which agents can move from one scene to another depending on their role. The *norms* of an electronic institution define the commitments, obligations and rights of participating agents. The activation of a norm depends on the values of variables in the uttered illocutions of an agent. For instance, in an auction scenario, a norm is specified that states that if a buyer submits a bid which exceeds his credit possibilities, the auctioneer is obliged to sanction him.

5.2. SONAR, a Formal Model of Organizations

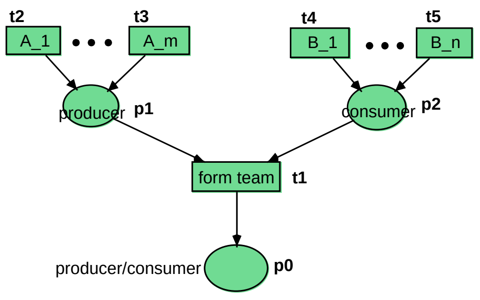
SONAR [Köh06] is a petri net model which is used to define *formal organizations*. A formal organization is the name for the combination of organization structure and organizational services in a multi-agent system [Köh06]. Members of the multi-agent systems are not included in the formal organization. To introduce the petri net model of formal organizations, an example of a market place is given in [Köh06]. A market scenario is modeled with petri nets (Figures 5.8a, 5.8b). However, to explain the example, some additional definitions are needed.

A *causal net* is a finitely branching petri net $N = (P, T, F)$ where the transitive closure F^+ is acyclic and $|{}^\bullet p| \leq 1$, $|p^\bullet| \leq 1$ are true for all $p \in P$.

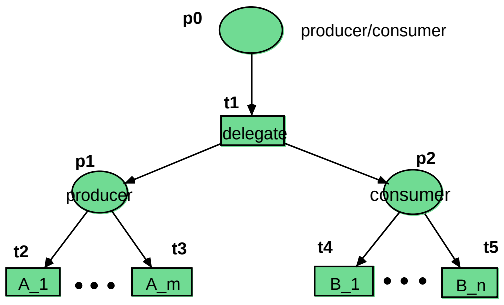
For a set $A \subseteq P \cup T$, ${}^\bullet A$ is defined as ${}^\bullet A := \bigcup_{a \in A} {}^\bullet a$ and $A^\bullet := \bigcup_{a \in A} a^\bullet$. The minimal nodes are defined as ${}^\circ N := \{x \in P \cup T \,|\, {}^\bullet x = \emptyset\}$, the maximal nodes as $N^\circ := \{x \in P \cup T \,|\, x^\bullet = \emptyset\}$

The market place example involves the roles *producer* and *consumer*. In the two Figures 5.8a and 5.8b two aspects are shown. The first figure represents the formation process of the team producer-consumer. The second represents the task delegation process within the team producer-consumer. During team formation, m agents $A_1, \ldots, A_m$ that are able to produce and n Agents $B_1, \ldots, B_n$ that are able to consume are involved. In Figure 5.8a the agents are modeled as transitions and roles as places. Each role can have several possible applications represented by multiple tokens that p_1 and p_2 can have. The multiple tokens in a role place, are put there from the firing processes of the (transitions) agents that want to participate as producers or consumers. When the transition *form team* fires it assigns one of the applying agents to the producer role and one to the consumer role. The team formation process is represented in the petri net model of the marketplace example. There are $m \cdot n$ possible firings from the agents A_i to p_1 and from the agents B_i

to p_2 which represent the $m \cdot n$ possible combinations of producer and consumer. A (t, p) arc describes the possible application for a role and a (p, t) arc describes the assignment of roles to tasks. While all roles in the preset of a task are needed for the team formation, the agents in the preset of a role are all alternative applications of the role.



(a) Team formation.



(b) Task execution within the team.

Figure 5.8.: Team formation and task execution ([Köh06]).

After the team formation process, the execution of tasks within the team has to be considered. This scenario involves task division and the delegation of subtasks within the team. In Figure 5.8b, this case is modeled again as a petri net. Here, the structure of the net is the same as the structure of the team formation net except for the arcs, which are reversed. The agents that are chosen to participate as producers or consumers are represented by filled transitions. The places in the postset of a task represent the roles needed to execute the task. All roles in the postset of tasks participate in the execution of the tasks. The tasks in the postset of a role represent the assignment of the role to alternative tasks.

Both perspectives, the team formation as well as the task execution require the same team structure. Thus, it is enough to consider only one as it is done in [Köh06] where only the delegation perspective is viewed. Using petri nets the formalization of the agent interaction services, the agent teams and the organization structure will follow.

5.2.1. Service Nets

Organizational services are represented in [Köh06] through *service nets*. Service nets are petri net models that specify how agents interact with each other. Nevertheless, service nets abstract from the agents and describe interaction between roles. Roles can be viewed as similar to types in object orienting languages. They describe the agents behavior in a multi-agent system. All agents that acquire a specific role have to comply to the role's constraints and fulfill the requirements set by the role. Defining interaction protocols between roles makes such protocols independent of the agents that are assigned to the role. In the example service net given in [Köh06] (Figure 5.9a) the roles *producer* and *consumer* interact with each other. After firing the transition called *produce* the producer sends the item to the consumer. The consumer fires the *receive* transition and then the *acknowledge* transition. Finally, it consumes the item and stops while the producer receives the acknowledgment and stops.

In [Köh06], $\mathcal{R} := 2^{Rol}$ is defined as the role universe where Rol is a set of roles. Every $R \in \mathcal{R}$ is called a role profile. The set operation $\subseteq$ defines for $\mathcal{R}$ a partial ordering. If $R_1, R_2 \in \mathcal{R}$ and $R_1 \subseteq R_2$ then R_1 is said to be more specialized than R_2. Besides, roles in a service net are assumed to be different from roles in other service nets.

Formally, service nets are defined as a tuple $D = (N, r)$ where $N = (P, T, F)$ is a petri net and r is a function $r : T \to Rol$. The function r assigns a role $r(t) \in Rol$ to every transition t of the net D. This means that the task t is executed only by the agents that implement the role $r(t)$. In Figure 5.9a, all transitions assigned to a role are drawn below the role, creating a vertical line of places and transitions. As it can be seen from Figure 5.9a service nets are very similar to agent UML (AUML). A role function for the whole service net is defined as $R(D) := r(T)$. If $\mathcal{D}$ is a set of service nets and if for each different pairs $D_1, D_2 \in \mathcal{D}$ it is true that $R(D_1) \cap R(D_2) = \emptyset$, than $\mathcal{D}$ is called a *service set*.

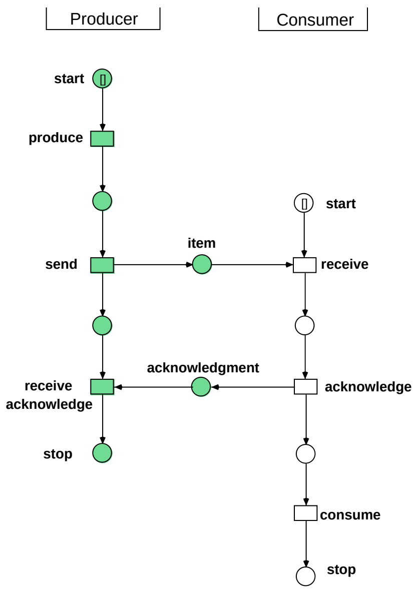
For a service net D and a role profile $R \subseteq R(D)$, D can be restricted to $D[R]$ which is a subnet of D defined as a tuple $D[R] = (P_R, T_R, F_R)$. The subnet is determined by the nodes $T_R := r^{-1}(R)$ and $P_R := ({}^\bullet T_R \cup T_R^\bullet)$ and the arcs linking these nodes. In Figure 5.9a, the subnet $PC[producer]$ is shown by the filled nodes.

Because every service net in *service sets* has different roles (from the condition $R(D_1) \cap R(D_2) = \emptyset$) every role can be assigned to a service net. Thus, for a *service set* $\mathcal{D}$ each role R can be assigned to a *reference service* $D(R, \mathcal{D}) \in \mathcal{D}$.

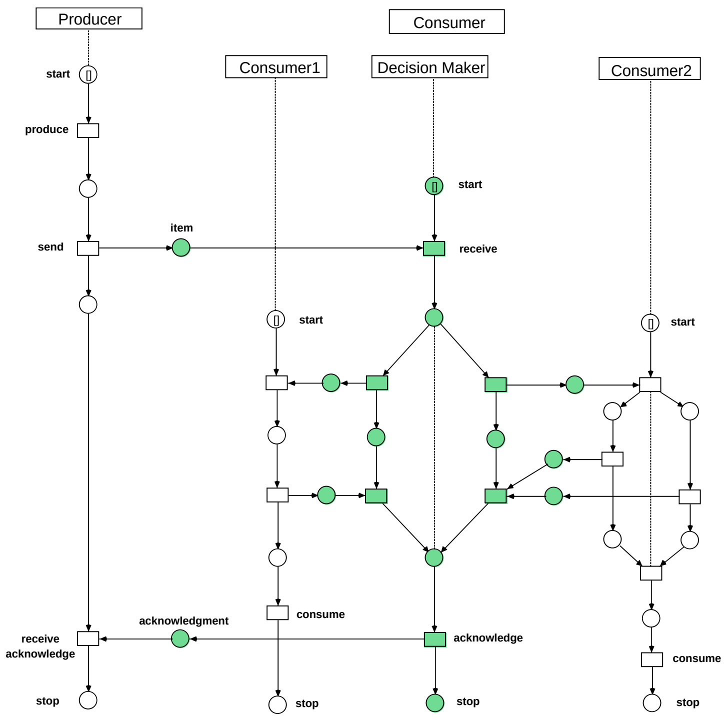
For the service nets D_1, D_2 and the role profiles $R_1 \subseteq R(D_1)$ and $R_2 \subseteq R(D_2)$ the relation

$$\langle\langle D_1; R_1, D_2; R_2 \rangle\rangle$$

is defined if $D_1[R_1]$ and $D_2[R_2]$ can replace each other in D_1 or D_2 without changing the behavior. That means that $D_1[R_1]$ and $D_2[R_2]$ are the same in terms of the input and output. In [Köh06], an example net named PC_2 is given where the role *consumer* is refined (Figure 5.9b) when compared to the *consumer* of the net PC (Figure 5.9a). The refined



(a) Simple service net, PC.



(b) Refined service net, PC2.

Figure 5.9.: Simple and refined service nets of producer-consumer ([Köh06]).

consumer in Figure 5.9b involves the role *decision maker*, *consumer1* and *consumer2*. The decision maker decides which consumer receives the item.

5.2.2. R/D Nets and Teams

R/D nets are petri nets (P, T, F) in which the role profiles are modeled by places and the tasks by transitions. The preset of a transition should contain exactly one place ($|^\bullet t| = 1$). A transition $t \in T$ is said to be *executive* if $t^\bullet = \emptyset$ and *delegative* if $t^\bullet \neq \emptyset$. Every place is labeled by a role profile and the transitions are labeled by service nets.

Formally, a R/D net is a tuple (R, N, D) where:

- $N = (P, T, F)$ is a petri net with $|^\bullet t| = 1$ for all $t \in T$ and $p^\bullet \neq \emptyset$ for all $p \in P$.
- $R : P \to \mathcal{R} \backslash \{\emptyset\}$ where $\forall t \in T \ : \ \forall p, p' \in t^\bullet \ : \ p \neq p' \Rightarrow R(p) \cap R(p') = \emptyset$.
- $D : T \to \mathcal{D}$.

The second condition represents the assignment of role profiles to places where the places in the postset of a transition should not have intersecting role profiles. The third condition represents the assignment of service nets to transitions.

A *team* is an R/D net where the N in the R/D net is a causal net and it has exactly one place as a minimal node: $^\circ N = \{p_0\}, p_0 \in P$.

In [Köh06], an example of a R/D net is given (Figure 5.10). For the service net assignments to the transitions the two service nets PC and PC_2 specified in the previous examples are used as a set of services $\mathcal{D} = \{PC, PC_2\}$. The roles used in the example are $Rol = \{Prod, Cons, DM, Cons1, Cons2\}$.

In [Köh06], well-formed R/D nets are defined. Well-formed R/D nets have additional properties which allow some kind of type checking. They are used for the formal analysis of the structure and behavior of the nets. In well-formed R/D nets, the service net labeled to a transition as well as the role profiles labeled to places are related to the structure of the net. In the following, because in R/D nets $|^\bullet t| = 1$ and there is exactly one place p in the preset of every transition t, the preset of t will be represented by $p(t) = {}^\bullet t$. Also, the role profile of $p(t)$ will be represented by $R_p = R(p(t))$.

Now, it can be continued with the definition of well-formed R/D nets as given in [Köh06]. Formally, a well-formed R/D net is an R/D net where for all $t \in T$ with $t^\bullet \neq \emptyset$:

- Static role compatibility: $R(t^\bullet) \subseteq R(D(t))$ and $R(D(R_p, \mathcal{D})) \setminus R_p = R(D(t)) \setminus R(t^\bullet)$
- Dynamic role compatibility: $\langle\!\langle D(R_p, \mathcal{D}); R_p, R(t^\bullet); D(t) \rangle\!\rangle$.

and for all $t \in T$ with $t^\bullet = \emptyset$ the static role compatibility $R(p(t)) \subseteq R(D(t))$.

Thus, for all delegative transitions t the roles $R(t^\bullet)$ have to be a subset of $R(D(t))$ and all the roles of the service net $D(t)$ different from $R(t^\bullet)$ have to be equal to the roles of the reference service $D(R_p, \mathcal{D})$ that are different from $R(p(t))$. Moreover, the behavior of $R(p(t))$ in the reference service $D(R(p(t)), \mathcal{D})$ has to be equal to the behavior of $R(t^\bullet)$ in $D(t)$. For the executive transitions t only the static condition should be fulfilled that $R(p(t))$ should be a subset of $R(D(t))$.

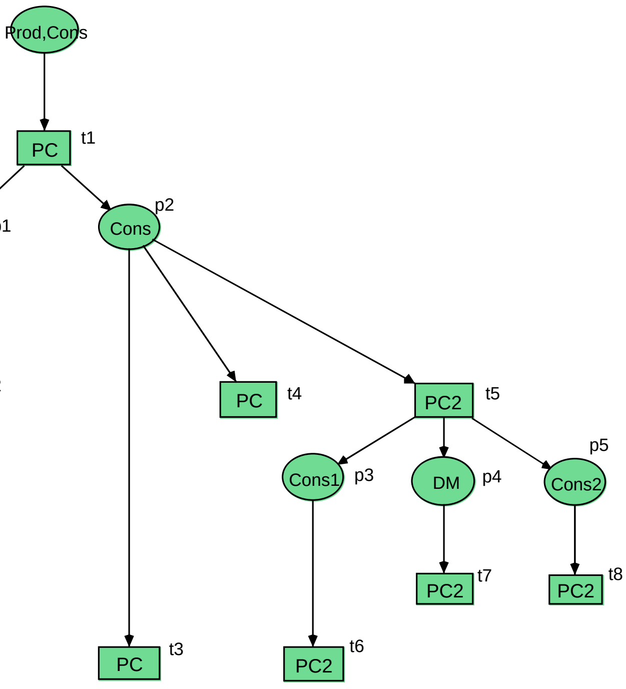


Figure 5.10.: A well-formed R/D net of producer-consumer.

In the example in Figure 5.10 the displayed net is actually a well-formed R/D net. All of the net's transitions comply to the conditions described in the above definition. For the transitions t_1, t_2 and t_5 the compliance to the conditions of well-formedness will be shown. These transitions were chosen as they respectively represent a simple delegative transition, an executive transition, and a somewhat more complicated delegative transition. The *reference service* for the roles $Prod$ and $Cons$ is $D(Prod, \mathcal{D}) = PC = D(Cons, \mathcal{D})$. For t_1 $R(t_1^\bullet) = \{Prod, Cons\}$ and $R(D(t_1)) = \{Prod, Cons\}$. Besides, $R(D(R(p_0), \mathcal{D})) \setminus R(p_0) = \{Prod, Cons\} \setminus \{Prod, Cons\} = \emptyset = R(D(t_1)) \setminus R(t_1^\bullet)$. The dynamic role condition for t_1:

$$\langle\langle D(R(p_0), \mathcal{D}); R(p_0), R(t_1^\bullet); PC \rangle\rangle = \langle\langle PC; \{Prod, Cons\}, \{Prod, Cons\}; PC \rangle\rangle$$

is obviously true. For t_2 only the static condition has to be considered as t_2 is an executive transition ($t_2^\bullet = \emptyset$).

$$R(p(t_2)) = R(p_1) = \{Prod\} \subseteq R(D(t_2)) = \{Prod, Cons\}$$

For the delegative transition t_5, to test the static condition, the expressions $R(D(t_5))\backslash R(t_5^\bullet)$ and $R(D(R(p(t_5)),\mathcal{D}))\backslash R(p(t_5))$ have to be evaluated:

$$\begin{aligned} R(D(R(p(t_5)),\mathcal{D}))\backslash R(p(t_5)) &= \{Prod, Cons\}\backslash\{Cons\} = \{Prod\} \\ R(D(t_5))\backslash R(t_5^\bullet) &= \{Prod, DM, Cons_1, Cons_2\}\backslash\{DM, Cons_1, Cons_2\} \\ &= \{Prod\} \end{aligned}$$

Thus, it is clear that

$$R(D(R(p(t_5)),\mathcal{D}))\backslash R(p(t_5)) = R(D(t_5))\backslash R(t_5^\bullet)$$

For the dynamic condition of t_5:

$$\begin{aligned} \langle\langle D(R(p(t_5)),\mathcal{D}); R(p(t_5)), R(t_5^\bullet); D(t_5)\rangle\rangle &= \langle\langle D(R(p_2),\mathcal{D}); R(p_2), R(t_5^\bullet); PC_2\rangle\rangle \\ \langle\langle D(R(p_2),\mathcal{D}); R(p_2), R(t_5^\bullet); PC_2\rangle\rangle &= \langle\langle PC; R(p_2), R(t_5^\bullet); PC_2\rangle\rangle \end{aligned}$$

The dynamic condition holds for t_5 as $R(t_5^\bullet) = \{DM, Cons_1, Cons_2\}$ in the service net PC_2 is a refinement of $R(p(t_5)) = R(p_2) = \{Cons\}$ in PC.

5.2.3. Organization Nets

Organization nets [Köh06] is defined as a formalism to represent the organizational structure. Central to this formalism is the new notion of *organizational positions* which represent positions in real organizations. Organization nets involve *organizational positions*. Organizational positions are responsible for several *tasks* and can also delegate tasks to other organizational positions.

Formally, an *organization net* is a tuple $(N, \mathcal{O})$ where N is a petri net $N = (P, T, F)$ and $\mathcal{O}$ is a partition on the set $P \cup T$ where for all $O \in \mathcal{O}$ the following is true:

$$\begin{aligned} &\forall p \in O \cap P : {}^\bullet p \subseteq O \wedge p^\bullet \subseteq \overline{O} \quad (with\ \overline{O} = (P \cup T) \setminus O) \\ &and\ also \\ &\forall t \in O \cap T : {}^\bullet t \subseteq \overline{O} \wedge t^\bullet \subseteq O. \end{aligned}$$

The elements $O \in \mathcal{O}$ describe the *organizational positions* previously mentioned. If a task belongs to an organizational position then all the role profiles that are used by the task also belong to the organizational position. Additionally, if a role profile represented by a place belongs to an organizational position then all tasks that use that role profile also belong to the position. Organizational positions that have no tasks cannot be used by other organizational positions. Also, organizational positions that have no role profiles cannot use other organizational positions. These two properties follow from the basic properties of the organizational positions as specified in the definition of the organization nets (the proofs can be found in [Köh06]).

Organization nets combined with R/D nets form *formal organizations*. Formal organizations are defined as tuples $Org = (N, \mathcal{O}, R, D)$ where (N, R, D) is a R/D net and $(N, \mathcal{O})$

is an organization net. An example of a formal organization is shown in Figure 5.11. In Figure 5.11, the same R/D net is presented as in Figure 5.10. The gray boxes represent the organizational positions. As can be seen from Figure 5.11 an organizational position can implement more than one role/role profiles (O_4). Also, an organizational position can use different service nets to implement a role behavior (O_3).

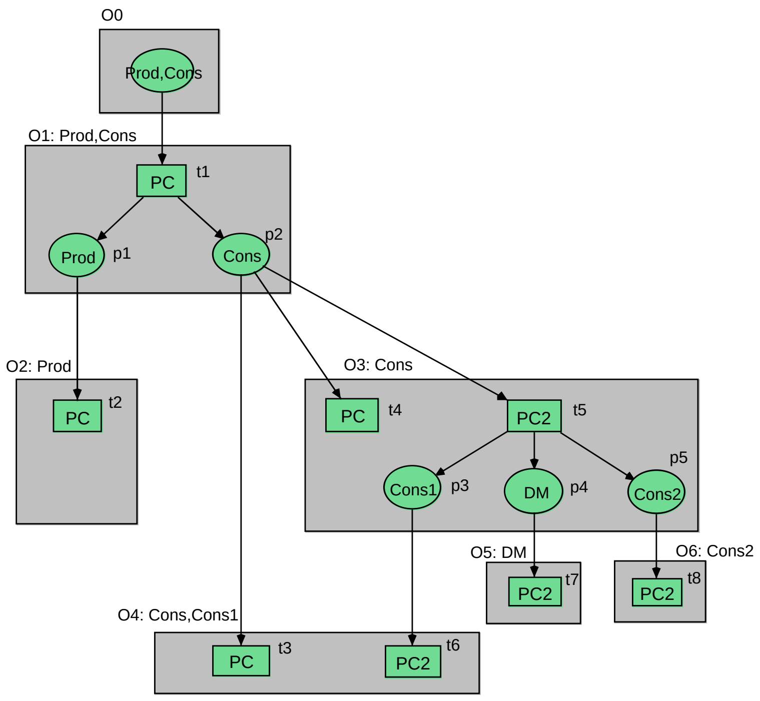


Figure 5.11.: The organization net of producer-consumer ([Köh06]).

Organization nets are similar to organizational charts. As mentioned in [Köh06], organizational charts can be viewed as a special case of organization nets. If all nodes of each organizational positions in an organization net are merged into one single node, then the obtained model represents an organization chart. While organization charts only display delegation structures, organization nets can additionally display information on execution of the tasks. Information on the execution of the tasks make organization nets interesting for performance analysis of the organizations that the organization nets represent.

5.2.4. Task Sequences in Organization Nets

Task sequences [Köh06] in organization nets are called those firing sequence which can fire a marking m to $\mathbf{0}$, the empty marking. Firing sequences are represented by the concatenation of the transitions that fire, $w \in T^+$. Task sequences are dependent on the marking of the net to be enabled. In the following some basic definitions of specific types of markings are given according to [Köh06].

For an R/D net (N, R, D):

- A marking m is called *processable* if **0** $\in RS(m)$.
- A marking m is called *strongly processable* if all $m' \in RS(m)$ are processable.
- (N, R, D) is called *(strongly) processable* if all of its possible markings are (strongly) processable.

An important property of R/D nets is that they have linear *reachable marking sets*. Formally, for a R/D net (N, R, D), it is true that:

$$RS(m_1 + m_2) = RS(m_1) + RS(m_2).$$

This property (s. [Köh06] for proof) allows to consider only markings of the type $m = \{p\}$ for some $p \in P$ during the analysis of the net.

R/D nets can be converted into context free grammars due to their property $|{}^\bullet t| = 1$. In this context, the different markings generated from the R/D net can be interpreted as words generated from the grammar. With the set of places of the R/D net $P = \{p_1, \ldots, p_n\}$, a marking m has a string representation:

$$\alpha(m) = A_{p_1}^{m(p_1)} \cdots A_{p_n}^{m(p_n)}.$$

In [Köh06], the context free grammars corresponding to R/D nets are defined. For a R/D net (N, R, D) with $N = (P, T, F)$ a context free grammar $G(N, m) = (X_G, V_G, R_G, S)$ is defined where $X_G = \{a_t \,|\, t \in T\}$ are terminals, $V_G = \{A_p \,|\, p \in P\} \uplus \{S\}$ are variables, $R_G = \{A_{p(t)} \to a_t A_{p_1} \cdots A_{p_n} : t \in T, t^\bullet = \{p_1, \ldots, p_n\}\} \cup \{S \to \alpha(m)\}$ are the productions and S is the start variable.

Task sequences can also be expressed through the context free grammars defined above. *Productive variables* in such grammars correspond to task sequences. Formally, a variable A is productive if there exists a terminal string $w \in X^*$ so that $A \Rightarrow *w$. A variable A is *reachable* if there exist strings $\alpha, \beta \in (X \cup V)^*$ such that $S \Rightarrow *\alpha A \beta$.

Now, the processability of markings can be expressed using the context free grammars above and productive variables. The following propositions are true (the proofs can be found in [Köh06]):

- A marking m is processable iff all $A(p)$ of $G(N, m)$ with $m(p) > 0$ are productive.
- A marking m is strongly processable iff all reachable variables $A(p)$ of $G(N, m)$ are productive.

The transformation of R/D nets to context free grammars facilitates the analysis of formal properties of R/D nets that concern task sequences. Other properties that R/D nets display are:

1. A marking m is processable if the markings $\{p\}$ with $m(p) > 0$ are processable.
2. The processability of a marking m is decidable in $O(|N|)$.
3. The strong processability of a marking m is decidable in $O(|N|)$.
4. If all markings $\{p\}$ with $p \in P$ are processable, then N is strongly processable.

Crucial to the proof of the second and third properties of R/D nets, which can be seen at [Köh06], are the set of productive variables and the set of reachable variables for a grammar. Both these sets are constructed inductively. The set of productive variables $PV(G) \subseteq V$ is constructed through:

$$PV_0(G) = X \tag{5.1}$$
$$PV_{n+1}(G) = PV_n(G) \cup \{A \in V \mid \exists (A \to w) \in P \,:\, w \in PV_n(G)^*\} \tag{5.2}$$

The set of reachable variables is given as

$$RV_0(G) = \{S\} \tag{5.3}$$
$$RV_{n+1}(G) = RV_n(G) \cup \{B \in V \mid \exists (A \to \alpha B \beta) \in P \,:\, A \in RV_n(G)\} \tag{5.4}$$

These sets are used for the implementation of the evaluation of processability or strong processability of markings which is described in detail in Chapter 6.

5.3. Summary

Recently, an organizational perspective on multi-agent systems has arisen in a number of research works. In this perspective, the focus of multi-agent systems is shifted from agents to agent organizations. As a consequence, modeling the structure of multi-agent organizations is a central issue. Many models of multi-agent organizations exist. In AGR (Agent, Group, Role), the organizational structure involves two aspects: a *structural aspect* and a *dynamic aspect*. The structural aspect consists of the *partitioning structure* and of the *role structure*. The partitioning structure defines how agents are assembled to groups and how groups are related to each other. The role structure defines the set of roles for each group and their relationships in the group. The *dynamic aspect* is related to the interaction patterns defined within roles and specifies the modalities of creating, killing, entering a group and playing roles.

MOISE$^+$ is a model built around the structural (role, links, groups, etc.), as well as the functional (goal, plans, etc.) and the deontic (norms, laws, etc.) aspects of an organization. The functional aspect in MOISE$^+$ includes a global decomposition tree, called the social scheme, where the responsibilities of the sub-goals are distributed along *missions*. The *deontic aspect* describes the roles' permissions and obligations for missions.

AUML is an extension of UML for modeling agent based systems. The structural aspect of organizations in AUML is based on AALAADIN and also combines elements from the dependency theory and holonics. Roles are defined in AUML as recurring patterns of dependencies and actions. Groups in AUML can interact with each other through identified members. In the case of unanalyzed groups, they can occupy roles in higher-level groups by building holonic structures.

ISLANDER is a declarative language for specifying electronic institutions. Electronic institutions correspond to human institutions which structure human interactions and enforce norms for all individuals. In ISLANDER an electronic institution consists of: a *dialogical framework*, *scenes*, the *performative structure* and *norms*. The activities in electronic institutions are the composition of multiple, eventually concurrent, dialogic actions which involve different groups of agents playing different roles. For each activity, interaction between agents is specified by *scenes* following well-defined *communication protocols*.

Petri nets are used in [Köh06] to build SONAR, a formal model on the organization structure and on the organizational services in a multi-agent system. SONAR is the model used in this work. *Organization nets* are defined in this model as a formalism to represent the organizational structure. Organization nets are a combination of *R/D nets* and *organizational positions*. Organizational positions represent positions in real organizations. Positions are responsible for several *tasks* and can also delegate tasks to other organizational positions. R/D nets are used for task delegation and display most of the formal properties. Organizational services are represented in this model by *service nets*. Service nets are very similar to AUML interaction diagrams. They are petri net models that specify how agents interact with each other. The formal properties of R/D nets make organization nets adequate for performance analysis of task sequences within the organization net.

Chapter 6.

OREDI, a Tool for Modeling SONAR Organizations

Building a prototype that can be used to easily create or edit SONAR organization nets or R/D nets is one of the main goals of this thesis. In this chapter, OREDI (*OR*ganization *EDI*or), a tool for building SONAR organization and R/D nets, is presented and an overview of the design and development process of OREDI is provided. In Section 6.1, the design process which includes the general requirements for the tool and the conceptual solutions adopted is described in detail. The technical realization of OREDI and the different aspects of its implementation are presented in the section that follows. Finally, the process of translating OREDI nets into a cross platform standard language like XML is discussed.

6.1. Specifications and Design

In this section the main requirements for OREDI are first specified. The presentation of the main solutions follows. The solutions include the specification of service and refinement definition nets (Subsection 6.1.3) which support in the development process of R/D nets, the organization editor (Subsection 6.1.2), the context based suggestion system (Subsection 6.1.4) and the evaluation and validation of organization and R/D nets (Subsection 6.1.5).

6.1.1. General Requirements

In the specification of requirements for OREDI there are some basic aspects that need to be considered first. The first aspect to be discussed is the future usage area and the eventual user base of OREDI Basing on assumptions about these topics the specification of adequate requirements for the tool can follow. The usage area of OREDI will mainly depend on the usage and popularity of SONAR as an organization model. Being a model that is visually similar to the widespread organizational charts and additionally covering, among other things, functional, dynamic as well as process evaluation aspects of organizational structures, SONAR can have usage areas that varying from multi-agent system design and programming to process development and optimization in real world organizations. Thus, one can assume the wide user base, ranging from groups made of people from academia, graduate and undergraduate students to multi-agent system developers and various types of managers in real world organizations. Also, OREDI will most certainly be used as a support tool in the various undergraduate agent-oriented projects of the Department of Theoretical Foundations of Informatics at the Hamburg University.

As a modeling software OREDI should enable users to use standard operations of graphical and modeling software such as moving, dragging, selecting, resizing, connecting, cutting and pasting different graphical components of the model as well as restrict interactions that bring the model into a formally not valid state.

Because of the mentioned reasons it is clear that OREDI, apart from offering a graphical tool with which SONAR organization and R/D nets can be built, should in the first place support its users into building such nets without explicitly knowing the formal rules of SONAR. Thus, one of the main requirements for OREDI is supporting users to develop correct organization and well-formed R/D nets without requiring previous knowledge of the formal rules of the type of nets they are creating. Users should be able to build correct organization and well-formed R/D nets at most with the support of few, easy to remember and intuitive thumb rules.

OREDI should also support some basic evaluation of the nets it can create and should be able to formally validate the created nets for correctness. Especially, the validation of the correctness of organization nets and well-formed R/D nets and the evaluation of task sequences and markings on processability or strong processability as defined in Subsection 5.2.4 should be supported.

Exchanging and sharing of the nets created with OREDI across platforms or applications is also a basic requirement that can contribute to the popularity and adoption of SONAR as an organization model. The availability of the nets created with OREDI in a standard format is an important step toward interoperability across platforms and applications. A standard format for the nets would allow other applications reading and parsing the nets.

Being a prototype implementation, OREDI has at first no specific optimization or performance requirements. The main issue here is to build an application that can deliver acceptable response times and a normal user experience. Thus, optimization or performance will not be in focus of the following discussion.

In the following subsections the main solutions to fulfill the above requirements are presented and discussed in detail.

6.1.2. Modeling Organization and R/D Nets

Building valid SONAR formal organizations is a process which involves the creation of correct organization nets and the creation of valid and well-formed R/D nets. Following the policy of restricting user interactions in such a way that we always have a valid formal state of the whole model can be successful especially in the case of creating or editing organization nets. A simple set of static user interaction restrictions can ensure that we always have correct organization nets.

Users of OREDI should be able to create and edit transitions, places, arcs and positions as the main graphical components of OREDI (s. Figure 6.1). Positions should contain transitions and arcs formally representing a member of the partition $O \in \mathcal{O}$ that hold a subset of the places and transitions. Additionally, the users should have the possibility to add inscriptions to places, transitions and positions. Initially, to comply to the formal rules of P/T petri nets, arc connections should take place only between places and transitions and vice versa. A set of additional restrictions should be added to comply with the formal rules of organization nets. Transitions and places should exist only within position components. This rule ensures that there is a partition $\mathcal{O}$ on all $P \cup T$. Transitions should connect with unidirectional arcs only to places inside their own position. The direction of the arcs here should be from the transition to the place. This rule ensures the compliance to the rule $\forall t \in O \cap T : {}^{\bullet}t \subseteq \overline{O} \wedge t^{\bullet} \subseteq O$ for each $O \in \mathcal{O}$. Additionally, places inside a position should connect with unidirectional arcs only to transitions in other positions. This rule ensures that the other formal rule of organization nets $\forall p \in O \cap P : {}^{\bullet}p \subseteq O \wedge p^{\bullet} \subseteq \overline{O} \quad (with\ \overline{O} = (P \cup T) \setminus O)$ also holds. Consistently maintaining these restriction rules throughout the whole interaction possibilities (creation of components, dragging, dropping, moving, deleting, copying and pasting) given to the users should ensure correct organization nets during their whole development process. Most importantly, these static restrictions enable users to build correct organization nets without requiring from them to know or be aware of the formal rules underlying organization nets.

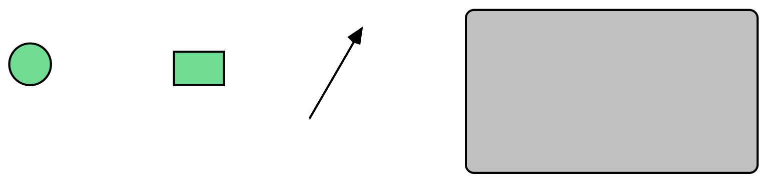


Figure 6.1.: Main graphical components for organization nets (from left to right:a place component, a transition component, an arc, a position component).

Valid R/D nets include rules involving role and service net assignments to places and transitions as well as a rule that specifies that every transition should have exactly one place in its preset and that every place should have more than one transition in its postset.

The rules specifying the number of elements in the preset of transitions or in the postset of places can be approached in a similar manner as in the organization nets: by restricting interaction possibilities. The role and service net assignment rules for correct R/D nets, although trivial, can not be entirely enforced by interaction restrictions. The choice for the preset or postset of a petri net element lies between a place or a transition, thus it lies between two elements. The number of choices for what inscription to assign to a petri net element is potentially much bigger [1]. Thus, users have to actively input inscription assignments, either through typing the roles or service net names themselves or through selecting the roles/service net names from a predefined list.

The inscription assignment process should be the same for the creation of R/D nets as for well-formed R/D nets, which put additional restrictions to the inscriptions allowed. Thus, the solution should be a shared one. Assisting users in the inscription assignment process is important for R/D and well-formed R/D nets as both types of nets have formal rules that require set operations which are difficult to remember let alone to be calculated by the users. Assistance in the inscription assignment process is however only possible if the set of service nets $\mathcal{D}$ (s. 5.2.2) and the roles involved in these service nets are known to OREDI. These sets have to be predefined before the inscription assignment process can be supported by OREDI. Moreover, in the case of well-formed R/D nets the set of all known refinement relationships between different sets of roles have to be known or predefined as refinement relationships are required in the dynamic condition for well-formed R/D nets (s. 5.2.2). If the set of the service nets $\mathcal{D}$ and the set of existing refinement is known to OREDI, then, it can offer a complete precomputed list of the inscriptions for a place or transition which would lead to a valid R/D or well-formed R/D net to users. The precomputed list of inscriptions suggested to the users can be dynamically generated based on the place or transition where the users need to assign inscriptions. The only thing that users need to remember is that every place and transition has to have at least one inscription in order to obtain a correct R/D or well-formed R/D net. However, they are completely relieved from the burden of remembering or calculating which inscriptions lead to correct nets. Context based suggestions (Figure 6.2) can provide the support that makes the creation of correct and well-formed R/D nets easy and intuitive.

The definitions of the set of service nets and their involved roles as well as the existing refinement relationships are described in detail in the next subsection. In Subsection 6.1.4, a detailed description of how suggestions can be generated is given.

As users have the choice of filling all their nets with inscriptions or neglecting to do it, OREDI has to offer a validation mechanism of the created R/D nets. Additionally, the evaluation of the *processability* and *strong processability* (s. 5.2.4) of markings is offered as an extra feature for evaluating the performance of the organizations designed with OREDI. Both the validation of nets and the evaluation of markings are described in detail in Subsection 6.1.5.

[1] In fact, $|\mathcal{R}|$ or $|\mathcal{D}|$, see Subsection 5.2.2.

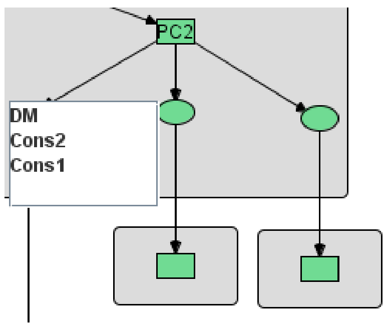


Figure 6.2.: Context based suggestions.

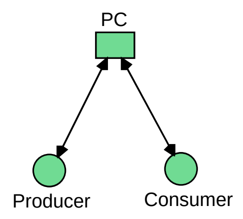
6.1.3. Service and Refinement Definition Nets

Service Definition Nets and *Refinement Definition Nets* are introduced as simple petri nets with inscriptions that support the creation of correct R/D and well-formed R/D nets. Service definition nets are used as a replacement for service nets in OREDI as the latter do not have an adequate tool that supports their creation or editing. Moreover, a single service definition net can provide the basic information that OREDI may need from multiple service nets.

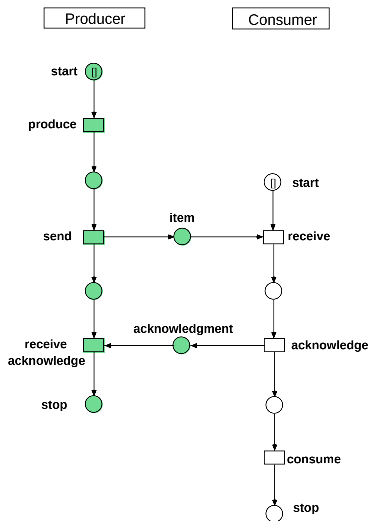
Service definition nets were first introduced in [Köh07]. They can specify one or more service definitions. Service definitions displayed in service definition nets are basically an abstraction over service nets (s. 5.2.1). Hence, service definition nets can provide an overview of many service nets. Service definition nets specify service definitions which contain services and the roles involved in these services. The service is represented by a transition and the roles involved in the service are represented by places connected to the transition. The transition of a service definition corresponds to a service net. The places of a service definition correspond to the roles involved in the service net.

In Figures 6.3a and 6.3b, a service definition net with one service definition and the corresponding service net producer-consumer (PC) are shown. The correspondence between the service definition and the service net is obvious except for the bidirectional arcs. The arcs represent the input and output messages from a role. If a role involved in a service sends one or more messages to other roles, the corresponding place in the service definition net connects to the transition (its service) with an outgoing arc. If the role receives messages it additionally connects to its transition with an incoming arc. If a role sends and receives messages in a service the corresponding place in the service definition connects with a bidirectional arc to its transition.

A service definition net can specify multiple service definitions. It can contain multiple services and their involved roles. Such a case is shown in Figure 6.4, where the equal roles shared between the services are modeled as shared places between the transitions.



(a) Service definition net with one service.



(b) Service net, PC.

Figure 6.3.: Service definition net and service net.

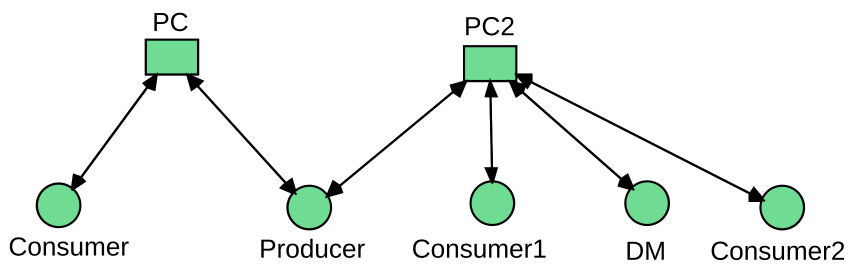


Figure 6.4.: Service definitions which share one role.

In Figure 6.4, the service definitions displayed correspond to the service nets producer-consumer (PC, s. Figure 6.3b) and refined producer-consumer (PC2, s. Figure 5.9b). A service definition net can thus contain basic information about multiple service nets and serve as an overview of service nets containing basic information. Up to this date there is no implementation of service nets. Additionally, a single service definition net can contain the information from multiple service nets that is needed for R/D nets. The relationship between service definitions and service nets can be seen as the relationship between interfaces and classes in the Java programming language. Thus, service definition nets are used as a more general replacement for service nets in OREDI. An extra tool for building and editing service definition nets is also developed as part of the OREDI development.

Refinement definition nets represent another type of petri nets with inscriptions. Refinement definition nets model the refinement relationships (s. Subsection 5.2.1) between sets of roles involved in service nets. Refinement relationships are used in well-formed R/D nets [2]. They describe the equality of input and output behavior of sets of roles involved in different service nets. Thus, refinement relationships can theoretically be automatically parsed from existing service nets. This is however a non-trivial task which goes beyond the scope of this thesis. Instead, the problem of providing refinement relationships for OREDI is approached in a similar fashion as it is done with the service nets. Refinement definition nets are put forward as a specification of refinement relationships. This way, we can have some kind of interfaces specifying the basic information for both the yet not implemented service nets as well as for the difficult to parse refinement relationships.

Formally, a refinement definition net over the refinement $\langle\langle D_1; R_1, R_2; D_2 \rangle\rangle$ is a tuple (N, r) where N is a petri net $N = ({}^\bullet t \cup t^\bullet, \{t\}, F)$ and additionally r is a function which assigns places to role inscriptions where $\forall p \in {}^\bullet t \cup t^\bullet$ it is true that $r(p) \in R_1 \cup R_2$ and $r({}^\bullet t) = R_1, r(t^\bullet) = R_2$. In Figure 6.5, a refinement net is displayed showing the refinement relationship between the sets of roles:

$$\langle\langle PC; \{Cons\}, \{Cons1, DM, Cons2\}; PC_2 \rangle\rangle$$

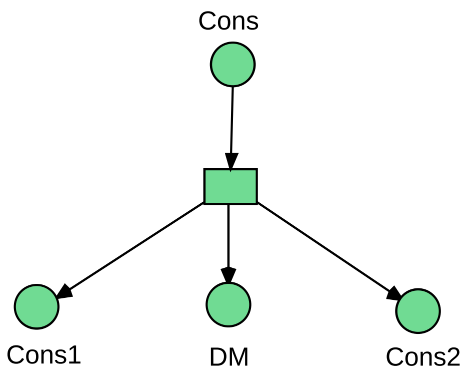


Figure 6.5.: Refinement definition net.

The refinement relationship described above is the same refinement relationship previously encountered in the example in Subsection 5.2.2. The combination of service definition nets with refinement definition nets provide all the information that may be needed for the creation of correct organization nets and also well-formed R/D nets. Therefore, the service and refinement definition nets create the conceptual basis upon which OREDI can generate the context based suggestions discussed in the next subsection.

[2]See the dynamic condition for well-formed R/D nets in Subsection 5.2.2.

6.1.4. Context Based Suggestions for R/D Net Inscriptions

The context based suggestion system is one of the core features that OREDI should provide to support users and lead them to the construction of correct R/D and well-formed R/D nets. A list of suggestion inscriptions should be offered to users that need to assign inscriptions to a specific place or transition. The suggestion list items certainly depend on the place or transition to which they have to be assigned. The main idea behind the generation of suggestions is that starting on the role or service net assignments of the neighbors or siblings of the elements, all the possibilities for the right assignments for the elements themselves can be derived.

There are two types of suggestions that should be generated.

- Suggestions for the service net assignment to a specific transition.
- Suggestions for the role assignment to a specific place.

Each of these cases will be handled separately. First some basic definitions are needed. The concept of the *local context* of a petri net element $t \in T$ or $p \in P$ can be defined as:

$$L(t) = {}^{\bullet}t \cup t^{\bullet} \cup \{t\}$$
$$L(p) = (\bigcup_{\forall t \in {}^{\bullet}p} L(t)) \cup \{p\}$$

We say that $L(t)$ with $\forall p_i, p_j \in t^{\bullet} : p_i \neq p_j \Rightarrow R(p_i) \cap R(p_j) = \emptyset \wedge D(t) \neq \emptyset$ is well-formed if t has $t^{\bullet} \neq \emptyset$ and it complies to the static and dynamic conditions specified in Subsection 5.2.2 or if t has $t^{\bullet} = \emptyset$ and it complies to the condition $R(p(t)) \subseteq R(D(t))$ (s. Subsection 5.2.2 for the definitions of R and D).

Similarly, $L(p)$ is well-formed if ${}^{\bullet}p \neq \emptyset$ and $\forall t \in {}^{\bullet}p$, $L(t)$ is well-formed, or if ${}^{\bullet}p = \emptyset$ and $R(p) \neq \emptyset \wedge R(p) \subseteq R(D_n)$ for any $D_n \in \mathcal{D}$.

Clearly, if in a R/D net all transitions have well-formed local contexts, the R/D net itself is well-formed. Suggestions should lead users to well-formed local contexts.

Suggestions for Transitions

We define that the suggestions $sugg(t)$ with $sugg(t) \subseteq \mathcal{D}$ ($\mathcal{D}$ is the set of all known service nets) for the service net assignment for a transition t are *correct* if for a known $R(D(R(p(t)), \mathcal{D})) \setminus R(p(t))$ any service net $D_s \in sugg(t)$ can lead to a well-formed local context $L(t)$ of t with an arbitrary $R(t^{\bullet})$ for which $R(t^{\bullet}) \subseteq R(D_s)$. Suggestions $sugg(t)$ are *complete* if there is no other service net assignment $D_x \in \mathcal{D} \wedge D_x \notin sugg(t)$ for t which can lead to a well-formed local context $L(t)$. A service net assignment D_s for a transition t leading to a well-formed local context $L(t)$ means that:

- for $t^\bullet \neq \emptyset$, there is at least one role assignment for each element $p \in t^\bullet$ so that $L(t)$ is well-formed with $D(t) = D_s$
- for $t^\bullet = \emptyset$, $L(t)$ is well-formed with $D(t) = D_s$.

The suggestions for the service net assignment include only service net inscriptions. These suggestions are made only for the inscription assignment of transitions. The symbol $\mathcal{D}$ represents here the set of all defined service nets which we can take for granted because of the service definition net(see previous subsection).

For the suggestions for transitions for well-formed R/D nets there are two cases: $t^\bullet = \emptyset$ and $t^\bullet \neq \emptyset$. In the case where $t^\bullet = \emptyset$, the only thing to be taken into account is the static condition: $R(p(t)) \subseteq R(D(t))$ where $p(t) = {}^\bullet t$ and $|{}^\bullet t| = 1$ as defined for well-formed $L(t)$. The correct suggestions in this case would be all $D_x \in \mathcal{D}$ for which additionally it is true that $R(p(t)) \subseteq R(D_x)$. Assuming that $R(p(t))$ is known, the suggestion set can be generated from a simple search of the set of service nets that contain the roles of $R(p(t))$. So,

$$sugg(t) = \{D_x | D_x \in \mathcal{D} \wedge R(p(t)) \subseteq R(D_x)\}. \tag{6.1}$$

For the second case, ($t^\bullet \neq \emptyset$) , both the static and dynamic conditions for a well-formed local context $L(t)$ have to be considered. For a well-formed $L(t)$ the suggestion set should contain all service nets that satisfy both of these conditions. This means that the suggestion set is an intersection of the two sets which contain respectively, all service nets that satisfy the static condition and all service nets that satisfy the dynamic condition. Thus, the problem of specifying the set of suggestions that contains all service nets complying to both conditions has been divided into two smaller problems.

For the static condition two basic assumptions can be made. The first condition being that $R(D(R(p(t)), \mathcal{D})) \setminus R(p(t))$ is known. This means that we assume that the role assignment for the preset of our transition t has already been made. The other assumption is that the assignment $R(t^\bullet)$ has not been made yet. These two assumptions imply that the role or service net assignment process proceeds in a top-down fashion. Thus, the assignment for the preset of any element is made before the assignment of the element itself and the assignment of the postset of any element is made after the assignment of the element itself. Again, $R(t^\bullet)$ is not known. However, it is assumed that $R(t^\bullet) \subseteq R(D(t))$ will in any case be true. From this abstraction, some statements can be made about $D(t)$. If the static condition is to hold than, $R(D(t))$ has to contain all the roles from $R(D(R(p(t)), \mathcal{D})) \setminus R(p(t))$. Also, $|R(D(t))| \geq |t^\bullet|$ should be true as otherwise there would be at least a place p_i with $R(p_i) = \emptyset$. This means that:

$$sugg_{static}(t) = \{D_x | D_x \in \mathcal{D} \wedge R(D(R(p(t)), \mathcal{D})) \setminus R(p(t)) \subseteq R(D_x) \wedge |R(D(t))| \geq |t^\bullet|\}.$$

For the dynamic condition the same assumptions can be made as for the static case. We can also assume that the set of all relationships between role sets is known as all refinement relationships are defined in the refinement definition nets (see previous subsection). The set of suggestions would be $sugg_{dynamic}(t)$ which specifies the set of all service nets D_x contained in the refinement relationships $\langle\langle D(R(p(t)),\mathcal{D}); R_p, R(t^\bullet); D_x\rangle\rangle$ with $D(R(p(t)),\mathcal{D})$ and R_p known and $R(t^\bullet)$ arbitrary.

The resulting suggestion set $sugg(t)$ is the intersection of the two specific suggestion sets:

$$sugg(t) = sugg_{static}(t) \cap sugg_{dynamic}(t) \tag{6.2}$$

The proofs for correctness and completeness of $sugg(t)$ are simple. Let us assume there is a service net D_y which is not in $sugg(t)$, and which can lead $L(t)$ to a well-formed state. That means that if $t^\bullet \neq \emptyset$ there is at least one role assignment for each element $p \in t^\bullet$ so that $L(t)$ is well-formed with $D(t) = D_y$. From this can be followed that D_y is contained in a refinement relationship $\langle\langle D(R(p(t)),\mathcal{D}); R(p(t)), R(t^\bullet); D_y\rangle\rangle$, it contains all the roles $R(D(R(p(t)),\mathcal{D})) \setminus R(p(t))$ and all $p \in t^\bullet$ have at least one role as role assignments. However, this was how $sugg(t)$ was defined for the case $t^\bullet \neq \emptyset$. This is a contradiction. Therefore, the assumption cannot be true. The same way it can be continued for the case $t^\bullet = \emptyset$. Thus, $sugg(t)$ is complete. The proof that $sugg(t)$ is correct is trivial as it follows directly from its construction.

Suggestions for Places

After considering how the suggestions for the service net assignments can be generated for transitions, the analysis for the generation of the suggestions for any $p \in P$ can follow. For the suggestions for a place p we specify $sugg(p) \subseteq Rol$ (Rol is the set of all known roles). The main requirement for $sugg(p)$ is that it should contain roles so that, for every possible role assignment $R(p)$ that can lead to well-formed local contexts $L(p)$, it is true that $R(p) \subseteq sugg(p)$. Also, $sugg(p)$ should leave out as many roles as possible which, if assigned to a place, lead definitely to a not well-formed $L(p)$. For a place p there are three relevant cases:

- $|{}^\bullet p| = 1$
- $|{}^\bullet p| > 1$
- ${}^\bullet p = \emptyset$

For each of these cases we will specify the terms for correct suggestions. For the first case ($|{}^\bullet p| = 1$) the suggestions for the place p have to include every role so that for every role assignment $R(p)$ leading to a well-formed local context $L({}^\bullet p)$, it should be true that $R(p) \subseteq sugg(p)$. Here, the fact that roles may or may not have been assigned to some $p_n \neq p \wedge p_n \in ({}^\bullet p)^\bullet$ has to be taken into account. We will denote $R(t^\bullet)$ as the union of the role assignments already made. An assignment $R(p)$ leading to a well-formed $L(t)$ with $t = {}^\bullet p$ means that there is an assignment for $t^\bullet$ with $R(p)$ as the assignment for p which makes $L(t)$ well-formed.

So, for a place p with $|^\bullet p| = 1$ where for $t = {}^\bullet p$ and given $R(D(R(p(t)), \mathcal{D})) \setminus R(p(t))$, $R(D(t))$ and $R(t^\bullet)$, we define that the suggestions $sugg(p)$ are *correct* if it is true that:

1. $(R(t^\bullet) \cup sugg(p)) \subseteq R(D(t)) \wedge R(t^\bullet) \cap sugg(p) = \emptyset$.

2. $R(D(R(p(t)), \mathcal{D})) \setminus R(p(t)) = R(D(t)) \setminus (R(t^\bullet) \cup sugg(p))$.

3. $\langle\!\langle D(R(p(t)), \mathcal{D}); R(p(t)), R(t^\bullet) \cup sugg(p); D(t) \rangle\!\rangle$.

If $R(D(R(p(t)), \mathcal{D})) \setminus R(p(t))$ and $R(D(t))$ are given, the term $R(t^\bullet) \cup sugg(p)$ with $R(t^\bullet) \cup sugg(p) = R(D(t)) \setminus (R(D(R(p(t)), \mathcal{D})) \setminus R(p(t)))$ represents the set to which the union of the role assignments for every $p_i \in t^\bullet$ should result in any case if $L(t)$ has to be well-formed. This means that correct suggestions $sugg(p)$, as defined above, include every role contained in the possible role assignments $R(p)$ that can lead to well-formed $L(t)$. The calculation of suggestions complying to the first and second conditions for correctness can be made:

$sugg_{static}(p) = (R(D(t)) \setminus (R(D(R(p(t)), \mathcal{D})) \setminus R(p(t)))) \setminus R(t^\bullet)$.

If $sugg_{static}(p)$ complies to the third condition for correctness, meaning that there is a refinement relationship with

$\langle\!\langle D(R(p(t)), \mathcal{D}); R(p(t)), R(t^\bullet) \cup sugg_{static}(p); D(t) \rangle\!\rangle$,

then our suggestion set can be generated with

$$sugg(p) = sugg_{static}(p) = (R(D(t)) \setminus (R(D(R(p(t)), \mathcal{D})) \setminus R(p(t)))) \setminus R(t^\bullet). \qquad (6.3)$$

With $|^\bullet p| = 1$ and known $R(D(R(p(t)), \mathcal{D})) \setminus R(p(t))$, $R(D(t))$and $R(t^\bullet)$ (the set of already made assignments), for every possible assignment $R(p)$ of p which leads to a well-formed $L(t)$ it is true that $R(p) \subseteq sugg(p)$. The proof for this proposition is again simple. Let us assume that for the already made assignments $R(t^\bullet)$ there is an assignment $R(p)$ on p which is not a subset of $sugg(p)$ and can lead to well-formed $L(t)$. This means that there is an assignment for all elements of $t^\bullet$ so that the union of already made assignments $R(t^\bullet)$ with $R(p)$ and the assignments of the rest of places R_{rest} makes $L(t)$ well-formed. From this follows that the static condition for well-formed $L(t)$ holds and $R(p) \cup R_{rest} = (R(D(t)) \setminus (R(D(R(p(t)), \mathcal{D})) \setminus R(p(t)))) \setminus R(t^\bullet)$ is also true. It also follows that the dynamic condition $\langle\!\langle D(R(p(t)), \mathcal{D}); R(p(t)), R(t^\bullet) \cup R(p) \cup R_{rest}; D(t) \rangle\!\rangle$ holds. However, $sugg(p)$ was defined in the same way. So, $sugg(p) = R(p) \cup R_{rest}$. This leads to $R(p) \subseteq sugg(p)$ which is a contradiction to the initial assumption. Thus, the assumption cannot be true.

When $|^\bullet p| > 1$, then suggestions $sugg(p)$ should contain the set of roles so that for the any assignment $R(p)$ which can lead all $L(t)$ with $t \in {}^\bullet p$ into a well-formed state, it should be true that $R(p) \subseteq sugg(p)$. Obviously, this set is more limited than the set of possibilities that would lead only one single $L(t)$ with $t \in {}^\bullet p$ into a well-formed state. Especially, the second and third condition defined for correct suggestions when $|^\bullet p| = 1$ cannot hold for every $t \in {}^\bullet p$ in the case when $|{}^\bullet p| > 1$. Hence, these conditions have to be relaxed in this case.

Suggestions $sugg(p)$ for a place p with $|{}^\bullet p| > 1$ where $\forall t \in {}^\bullet p$, $R(D(R(p(t)), \mathcal{D})) \setminus R(p(t))$ and $R(D(t))$ are known, are *correct* if $\forall t_i \in {}^\bullet p$ it is true that:

1. $(R(t_i^\bullet) \cup sugg(p)) \subseteq R(D(t_i)) \wedge R(t_i^\bullet) \cap sugg(p) = \emptyset$.

2. $R(D(R(p(t_i)), \mathcal{D})) \setminus R(p(t_i)) = R(D(t_i)) \setminus (R(t_i^\bullet) \cup sugg(p) \cup R_{s_i})$.

3. $\langle\langle D(R(p(t_i)), \mathcal{D}); R(p(t_i)), R(t_i^\bullet) \cup sugg(p) \cup R_{s_i}; D(t_i)\rangle\rangle$.

with $sugg(p)$ equal for all t_i and R_{s_i} a variable and arbitrary set of roles for each t_i where $R_{s_i} \subseteq R(D(t_i))$. The correct suggestion set $sugg_{static}(p)$ and the arbitrary R_{s_i} can be calculated from the two first conditions for correctness. If both calculated sets comply to the third condition as well, then they qualify as the final solution:

$$sugg(p) = sugg_{static}(p) = \bigcap_{\forall t_i \in {}^\bullet p} (R(D(t_i)) \setminus (R(D(R(p(t_i)), \mathcal{D})) \setminus R(p(t_i)))) \setminus R(t_i^\bullet) \quad (6.4)$$

$$R_{s_i} = (R(D(t_i)) \setminus (R(D(R(p(t_i)), \mathcal{D})) \setminus R(p(t_i)))) \setminus (R(t_i^\bullet) \cup sugg(p)) \quad (6.5)$$

The proof that for every $R(p)$ which leads to a well-formed $L(p)$ it is true that $R(p) \subseteq sugg(p)$ can be achieved in the same way as with the case $|{}^\bullet p| = 1$.

The special case ${}^\bullet p = \emptyset$ means that the place p is a top element with no preset. The requirement for the suggestion set $sugg(p)$ is that it should include every role so that every possible assignment $R(p)$ that can lead to well-formed local contexts $L(t)$ with $t \in p^\bullet$ should be contained in the suggestion set $R(p) \subseteq sugg(p)$. Let R_{top} be a set of roles with $R(p) = R_{top} = R(D_i)$ for a $D_i \in \mathcal{D}$. R_{top} can lead to a well-formed $L(t)$ for $t \in p^\bullet$ if $D(t) = D_i$ and $R(t^\bullet) = R(D_i)$. This assignment fulfills the static condition and the dynamic condition. The dynamic condition is fulfilled as subsets of the set of roles involved in a service net are always a refinement of themselves (it is always true that $\langle\langle D; R_s, R_s; D\rangle\rangle$ with $R_s \subseteq R(D)$). So

$$sugg(p) = \bigcup_{\forall D \in \mathcal{D}} R(D). \quad (6.6)$$

At this point, it must be noted that role sets $R_n \subseteq sugg(p)$ which are not a subset of any $R(D_i)$ with $D_i \in \mathcal{D}$ should not be assigned to p as they would not lead to well-formed local contexts $L(t)$ (the dynamic condition would not hold). This constraint can however be imposed on users as an interaction constraint. Users can be forced to assign only $R(p) \subseteq R(D_i)$ with $D_i \in \mathcal{D}$ to a top place p.

Suggestions for Transitions and Places if the Reference Service is Ambiguous

For the calculation of the suggestion sets in all the cases described above, it was assumed that $R(D(R(p(t)), \mathcal{D})) \setminus R(p(t))$ is known. In special cases though, having a set of roles assigned to $p(t)$ does not necessarily mean that the reference service $D(R(p(t))$ is unambiguous. In fact, $D(R(p(t))$ can have different values $D(R(p(t)) \in \{D_r | D_r \in \mathcal{D} \wedge R(p(t)) \subseteq D_r\}$. Thus, for this case, a separate analysis for the calculation of correct suggestions needs to be made. First, the set of optional values for $D(R(p(t)))$ is named

$$D_{ref}(t) = \{D_r | D_r \in \mathcal{D} \wedge R(p(t)) \subseteq D_r\}.$$

The set of suggestions $sugg_{allref}(t)$ for a transition t, where $t^{\bullet} \neq \emptyset$ and $D(R(p(t))$ has optional values, is the union of the suggestion sets $sugg(t)$ specified in (6.2) for each $D(R(p(t)) \in D_{ref}(t)$. The correctness of $sugg_{allref}(t)$ results from its construction. The completeness of $sugg_{allref}(t)$ can be proved by contradiction similarly as for $sugg(t)$. For a transition t with $t^{\bullet} = \emptyset$, the set of suggestions $sugg_{allref}(t)$ remains the same as in the case when $D(R(p(t))$ has a determined value (s. (6.1)).

For the generation of suggestions for a place p, only the two cases $|{}^{\bullet}p| = 1$ and $|{}^{\bullet}p| > 1$ need to be considered[3]. The set of suggestions $sugg_{allref}(p)$ for a place p, where $|{}^{\bullet}p| = 1$ and $D(R(p(t))$ has optional values with $t \in \bullet p$, is the union of the suggestion sets $sugg(p)$ specified in (6.3) for each $D(R(p(t)) \in D_{ref}(t)$. For the case $|{}^{\bullet}p| > 1$ for a $t_i \in {}^{\bullet}p$ the set $R_{i_{allref}}(p)$ is specified as the union of the suggestion sets $sugg(p)$ specified in (6.3) for each $D(R(p(t_i)) \in D_{ref}(t_i)$. The set of suggestions $sugg_{allref}(p)$ is then the intersection of all $R_{i_{allref}}(p)$ with $t_i \in {}^{\bullet}p$. The proof for the completeness of $sugg_{allref}(p)$ for both cases ($|{}^{\bullet}p| = 1$ and $|{}^{\bullet}p| > 1$) can be achieved again by contradiction.

Obviously, users can select role sets from the set $sugg_{allref}(p)$, the assignment of which can not lead to well-formed local contexts. Such cases occur when the selected role set contains roles generated by making calculations with different reference services (mixed selections). However, mixed selections can be avoided if the selection of mixed roles is not allowed in the first place and if the roles in $sugg_{allref}(p)$ that are calculated from the same reference service are grouped together visually. Grouping roles visually makes users understand that roles in different groups can not be mixed together in the same selection.

Conclusions

In this subsection a mathematical analysis of possible suggestions as user aids during construction of well-formed R/D nets resulted in several conclusions.

- A set of service nets $sugg(t) \subseteq \mathcal{D}$ can be offered to the users for the service net assignment to any transition t. The offered set $sugg(t)$ contains all possible assignment possibilities that lead to well-formed local contexts $L(t)$. The set $sugg(t)$ contains only service nets the assignment of which to the transition t lead to well-formed local contexts.

[3]The case $|{}^{\bullet}p| = 0$ does not involve the term $D(R(p(t))$ at all for the generation of suggestions. Thus, it is the same as for an unambiguous $D(R(p(t))$ (s. 6.6).

- A set of roles $sugg(p) \subseteq Rol$ (or $sugg_{allref}(p) \subseteq Rol$) can be offered to the users for the role assignments to any place p. Users can choose any subset of roles from $sugg(p)$ or $sugg_{allref}(p)$ to assign to p. The set $sugg(p)$ (and $sugg_{allref}(p)$) was limited so that it complies to several formal conditions and it was proved that all possible assignments $R(p)$ that lead to well-formed local contexts $L(p)$ are subsets of $sugg(p)$ or $sugg_{allref}(p)$ ($R(p) \subseteq sugg(p)$). Especially for the case $|^\bullet p| > 1$ it is difficult to conclude that every role included in $sugg(p)$ or $sugg_{allref}(p)$ is contained in a role assignment $R(p)$ which leads to a well-formed local context. Presumably, the set of suggestions $sugg(p)$ (or $sugg_{allref}(p)$) for this case can be further limited. However, this is a non trivial problem which well be left here for future research.

- The role assignments chosen from the suggestion list can not always lead to well-formed R/D nets. This leads to the construction of a validation feature for well-formed local contexts and R/D nets.

These results can be practically used if users hold to some simple rules which incorporate some of the most important assumptions made for the generation of the context based suggestions. These rules well be discussed in Subsection 6.2.4.

Validating and evaluating R/D and organization nets will be left to the next subsection to discuss.

6.1.5. Organization and R/D Net Validation and Evaluation

The validation of organization and R/D nets is an important feature that OREDI has to support. Especially, the validation of R/D nets is a must as the context based suggestions do not always guarantee the construction of well-formed R/D nets. Validating an organization net means literally to check the net for compliance of the respective conditions. The conditions for organization nets are that every place or transition is a position $O \in \mathcal{O}$. Also, it should be checked if the preset of every place is in the same position and if the postset of every place is in a different position than that of the place itself. These are static conditions that can be checked easily for each of the places or transitions of the net. If all elements comply to the conditions it can be concluded that the net is an organization net.

Validating the R/D nets includes checking nets if they fulfill the basic conditions for R/D nets and checking nets if conditions for R/D nets and well-formed R/D nets are fulfilled. The validation for basic R/D nets means that every place has to have a postset and a role set as a role assignment which does not intersect with the role sets assigned to its siblings. Also, every transition has to have a preset with $|^\bullet t| = 1$ and a service net assignment. These conditions can be checked for every place or transition element of the net. If all elements comply to the condition then the net is an R/D net.

Checking if a net is a well-formed R/D net can also be done in the same fashion. Every transition of an R/D net can be checked if they comply to the static and dynamic conditions for well-formed R/D nets. Here, the set of all available service nets and refinement relationships is needed. This is provided by the service definition nets and the refinement definition net. If all transitions comply to the conditions than the R/D net is well-formed.

As SONAR has a clear formal definition for each of its net types based on petri nets, it has the advantage that organizations designed with SONAR can be evaluated for several properties. The evaluation of *processable*, *strongly processable* markings and as a consequence the evaluation of the whole R/D net for strong processability can be implemented in OREDI as the theoretical basis for performing such a task has already been laid in [Köh06]. Evaluating R/D nets for processable and strongly processable markings means that all possible processable markings need to be found. Processable markings can be found with the help of the context free grammars corresponding to R/D nets which are described in Subsection 5.2.4. More specifically, finding processable markings for a R/D net, is an issue of finding productive variables (s. Subsection 5.2.4) in the corresponding context free grammars. Since a marking m is processable iff all $A(p)$ of $G(N, m)$ with $m(p) > 0$ are productive, after asserting all productive variables A_p with $p \in P$, the processable markings are all markings with $m(p) \geq 0$ for each p with a corresponding productive variable A_p and with $m(p) = 0$ for each p with A_p not a productive variable. The algorithm to assert which variable is productive is then pretty much the implementation of the construction formula for productive variables specified in (5.1).

To specify all possible strongly processable markings, the linearity of R/D nets and the formula for the construction of reachable variables (s. (5.3)) are additionally needed. Initially, for a marking $m = \{p\}$ the set of reachable variables has to be constructed. Then, it can be checked if the set of the resulting reachable variables is contained in the set of the productive variables. If this is the case, the marking $m = \{p\}$ is strongly processable. This process can be repeated for each marking $m = \{p\}$ where $p \in P$. Let $P_{strong} \subseteq P$ be the set of places with $p \in P_{strong}$ if for a $m = \{p\}$ then m is strongly processable. Because of the linearity of R/D nets, strongly processable markings are all the markings with $m(p) \geq 0$ where $p \in P_{strong}$ and with $m(p) = 0$ where $p \notin P_{strong}$. If for all $p \in P$ it is also true that $p \in P_{strong}$, then the R/D net is strongly processable.

The most important implementation specifics for both the validation of organization and R/D nets as well as for the evaluation of R/D nets are given in the next section.

6.1.6. Exporting Organization and R/D Nets in a Standard Format

Exporting the organization and R/D nets in a standard format makes sense only if the format is widely adopted. XML [XML07] is a broadly accepted standard in the industry as well as in the academic world. The hierarchical structure of XML is also well-suited for describing organization positions, their tasks and role profiles. This makes XML the format of choice for exporting the nets. The main requirement while designing the XML schema which will be used for the nets, is to ensure that it offers description possibilities for all important net elements and their relationships. In this way, all the information from the nets that have to be exported can be described by the format and the nets can be recreated from XML without loss of relevant information.

The main elements and relationships of the Organization and R/D nets are:

- Positions of the net;
- The tasks (representing transitions) within positions and their service net assignments;
- The role profiles (representing places) that tasks implement or delegate and the roles within them;
- All service nets;
- All roles;
- The refinement relationships.

Basing from the mentioned points, every other relationship between elements can be derived. However, pieces of information which could be often required and which can be derived from the above elements and relationships could also be added explicitly as it would facilitate the parsing from the XML documents.

Such redundant information which could be interesting is:

- All the role profiles that a specific position implements;
- All the role profile that a specific position delegates;
- All the neighbors to which a specific position delegates;
- All the neighbors from which a specific position implements;

For each of the points listed above, special XML elements are designed. The exact specification of the XML format can be found in Appendix A. However, some key elements will be described here. The whole net is represented through the root element *formalOrganization*. A *formalOrganization* element can contain multiple *organizationPosition* elements. It also contains a set of service net elements, a set of role elements and a set of refinement relationships. An *organizationPosition* element contains a list of the role profiles which it implements (element *positionRoleProfiles*), a list of the role profiles which it delegates (element *delegatedRoleProfiles*), a list of the tasks (element *implTasks*), a list of neighbor positions (element *neighborPositions*) including those from which the position implements (elements *incoming*) and those to which it delegates elements (elements *outgoing*). All other information is contained within tasks. Tasks can contain a service net, a list of role profiles which it delegates, and a role profile from which it implements. The repetition of the same elements is prevented by using unique id values and references to elements already defined. Below, an example of the XML format of a position with two tasks is displayed. The position is linked to another position from which it implements role profiles and to three positions to which it delegates role profiles.

```
<?xml version="1.0" encoding="UTF-8"?>
<formalOrganization>
    .
    .
    .
    <organizationPosition>
        <typeId>3</typeId>
        <delegatedRoleProfiles>
            <roleProfile>
                <typeId>4</typeId>
                <roleTypeId>1</roleTypeId>
                <roleTypeId32</roleTypeId>
            </roleProfile>
            <roleProfile>
                <typeId>6</typeId>
                <roleTypeId>3</roleTypeId>
                <roleTypeId>5</roleTypeId>
            </roleProfile>
            <roleProfile>
                <typeId>7</typeId>
                <roleTypeId>4</roleTypeId>
                <roleTypeId>7</roleTypeId>
            </roleProfile>
        </delegatedRoleProfiles>

        <positionRoleProfiles>
            <roleProfileTypeId>3</roleProfileTypeId>
        </positionRoleProfiles>

        <implTasks>
            <executiveTask>
                <typeId>5</typeId>
                <serviceNetTypeId>4</serviceNetTypeId>
                <implementingFromRoleProfile>
                    <roleProfileTypeId>3</roleProfileTypeId>
                </implementingFromRoleProfile>
            </executiveTask>

            <delegativeTask>
                <typeId>5</typeId>
                <serviceNetTypeId>6</serviceNetTypeId>
                <implementingFromRoleProfile>
```

```
                    <roleProfileTypeId>3</roleProfileTypeId>
                </implementingFromRoleProfile>
                <delegatingToRoleProfile>
                    <roleProfileTypeId>5</roleProfileTypeId>
                    <roleProfileTypeId>6</roleProfileTypeId>
                    <roleProfileTypeId>7</roleProfileTypeId>
                </delegatingToRoleProfile>
            </delegativeTask>
        </implTasks>

        <neighborPositions>
            <incoming>
                <positionTypeId>2</positionTypeId>
            </incoming>
            <outgoing>
                <positionTypeId>4</positionTypeId>
            </outgoing>
            <outgoing>
                <positionTypeId>5</positionTypeId>
            </outgoing>
            <outgoing>
                <positionTypeId>6</positionTypeId>
            </outgoing>
        </neighborPositions>
    </organizationPosition>
   .
   .
   .
   <allRoles>
       <role>
           <typeId>1</typeId>
           <name>Prod</name>
       </role>
       <role>
           <typeId>2</typeId>
           <name>Cons</name>
       </role>
       .
       .
       .
   </allRoles>
</formalOrganization>
```

6.2. Tool Development

Implementing OREDI means that tools should be made available for the creation, editing and evaluation of organization and R/D nets as well as service and refinement definition nets described in the previous section. The decision to implement OREDI as a set of Renew plug-ins was the first taken. Renew already offers a flexible plug-in architecture for building petri net editors so that the main work for implementing OREDI would be extending Renew to handle organization and R/D nets. Renew is also the main development tool for several agent-based and Mulan projects in the Department of Informatics at the University of Hamburg. Implementing OREDI as Renew plug-ins enables its use in undergraduate student projects which can provide immediate and direct feedback on additional features, bugs, etc. This feedback can help reduce eventual future development cycles and it can help OREDI adapt for use in academic or industry projects as a design tool. An editor for the manipulation of organization and R/D nets and an extra editor for the manipulation of service and refinement definition nets are provided as two separate Renew plug-ins. Both editors will be described in more detail in the following subsections.

6.2.1. Organization and R/D Net Editor

The organization and R/D net editor is a tool for the manipulation of organization and R/D nets. The editor is implemented as a Renew plug-in which extends the default Renew editor GUI with two palettes of tools. The first palette contains tools for the creation of the positions, places, transitions and for their assignment with inscriptions. The second palette contains buttons which represent features for the validation and evaluation of organization and R/D nets. In Figure 6.6, the Renew GUI including the palettes of the organization and R/D net is displayed.

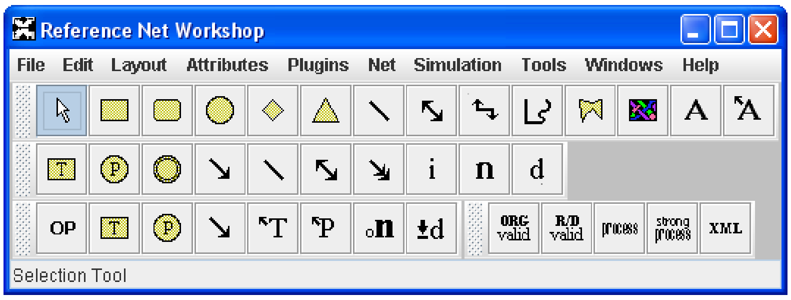


Figure 6.6.: Renew extended with the organization and R/D net plug-in.

The tools of the first palette of organization and R/D plug-in shown are (from left to right): a tool for the creation of positions, a tool for the creation of transitions, a tool for the creation of places, a tool for the creation of connections between places and transitions, a tool for the transition inscriptions, a tool for the place inscriptions, a naming tool for positions and a tool for loading service and refinement definition nets. The second palette contains (from left to right): a tool for the validation of organization nets, a tool for the validation of R/D and well-formed R/D nets, an evaluation tool for the processability of the net, an evaluation tool for the strong processability of the net and a tool for exporting the created net in XML format.

To build a net in Renew there are two possible approaches. A user can explicitly select the needed creation or connection tools in the palette and then click to the drawing. The places and transitions for example can be created in the drawing by selecting the place or transition creation tool and then clicking to an arbitrary point inside the drawing. Existing places and transitions inside the drawing can be connected by first selecting the connection tool and then connecting the two net elements.

However, Renew also offers a shortcut policy for the creation of nets where the explicit selection of the tools is reduced to the minimum. The user does not have to leave the drawing for the selection of tools to create the nets. One of the shortcuts is the *handle* figure which exists inside many Renew figures such as transitions and places. Clicking inside the handle of a place or a transition creates an arc the end of which can be dragged then to an arbitrary point in the drawing. After releasing the end of the arc respectively a new transition or place is created on the same point if on the point of release no figures existed that could be connected with the initial place or transition. If on the point of release there is a figure which can be connected to the initial figure, then, the two figures are simply connected. The first approach would require leaving the drawing three times for the same operation to select the respective tools. This way, the graphical shortcuts in Renew allow quick and easy construction of the nets.

Another important graphical shortcut of Renew is the mouse right click on figures. In Renew, the default action after the right click on a figure is the creation of inscription for the figure. This shortcut is used for the generation of the context based suggestions described in the previous section.

The actual creation and manipulation of all nets in Renew is done in diagram drawings which are separate from the Renew GUI. In the diagram drawings place and transition figures are created and manipulated. While SONAR nets are indeed petri nets, they also include elements like positions that have formal meaning but are not petri net elements. Therefore, a new diagram type was developed for the construction of organization and R/D nets.

To ensure the construction of organization nets and R/D nets the main work consisted in creating connection tools and specializing the handles so that the connectivity rules between places and transitions would follow the formal rules of organization nets and R/D nets. New creation tools were also developed so that the places or transitions could only be created inside a position figure representing the position inside the partition $\mathcal{O}$. In Figure 6.7, the interaction with the handle of a place in an organization or R/D net is displayed.

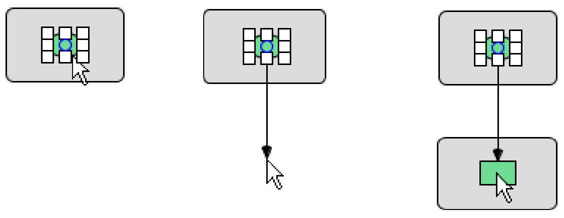


Figure 6.7.: A handle figure (small blue circle inside places) of a place in organization and R/D net.

Also, the actions of dragging, releasing and deleting figures were implemented so that the formal rules of organization and R/D net hold in any case. Dragging a position figure drags the places, transitions and arcs between them so that they retain their relative positions inside the position figure. After the release of a place or transition into another position figure, it is checked if the dragged place and its connections fulfill the formal condition of organization and R/D nets with the new position. In this case, the place or transition are added inside the new position figures, otherwise they return to their original place in the old position figure. By dragging the start or the end of an arc from one place or transition to another, after the release it is again checked if the formal properties of organization and R/D nets hold for the eventual new connection. Only then, the connection changes. Thus, dragging places, transitions or arcs is a context sensitive action. Deletion of positions had also to occur so that the formal rules of organization or R/D nets are not violated. Deleting a position figure deletes all places and transitions inside it together with all their connections.

Organization or R/D nets are saved as *.org* files. Saving organization and R/D nets and constructing them from *.org* files was also adapted so that the relationships between positions and their places and transitions can be reestablished after opening the saved net.

6.2.2. Role and Refinement Definition Editor

The role and refinement definition editor is an editor for the creation and manipulation of role and refinement definition nets. It is conceived as a support tool in OREDI for the organization and R/D net editor. The editor is implemented as a Renew plug-in and extends the default Renew GUI with two palettes. In Figure 6.8, Renew GUI is displayed extended with the service and refinement definition plug-in.

The first palette consists of (from left to right): a tool for the creation of transitions for the service definitions, a tool for the creation of places for the service definitions, an arc tool for the connection of places and transitions in the service definitions, an inscription tool for the transitions and an inscription tool for places. The transition inscription and place

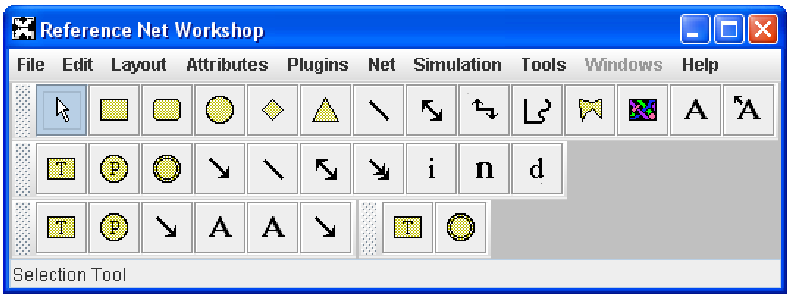

Reference Net Workshop
File Edit Layout Attributes Plugins Net Simulation Tools Windows Help
Selection Tool

Figure 6.8.: Renew extended with the service and refinement definition plug-in.

inscription tools serve to add names to the transitions and places representing respectively the name of the service nets and the name of the roles involved in the service net. Here, places can only have one name inscription meaning that one place can represent only one role. Similarly, transitions can only have one name inscription representing one service net. In Figure 6.9, a service definition net with two service definitions is displayed as it is created with the tool.

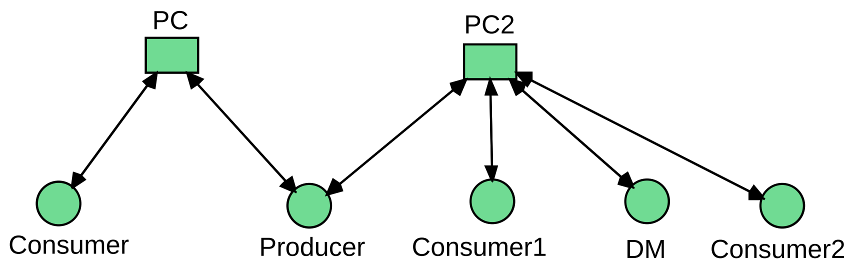
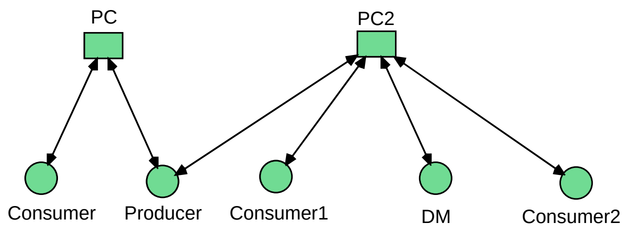


Figure 6.9.: Service definitions which share one role.

The second palette consists of a tool for creating transitions and a tool for creating virtual places for the refinement definition nets. In this editor, refinement definition nets consist of transition and virtual places which are a kind of representation (s. Subsection 4.3.1) of the places in the service definitions. Virtual places were chosen to model the specialized and specializing roles of refinement definition nets because they allow a definite correlation between the roles of the service definitions and the roles of the refinement relationship. By this way, in refinement definition nets it is clear which role belongs to which service net. The service definition net as well as refinement definition nets have to be created in the same diagram so that refinement definition nets can refer to the roles of the service definitions. The *handles* of the virtual places and the transitions of the refinement definition nets were specialized so that the refinement transition and the virtual places can only connect to each other. Also, the creation tool for the virtual places was specialized so that virtual places can only be created for service definition places which have name inscriptions. To discern the transitions of service definitions from the transitions of refinement relationships they have different colors, respectively, green and blue. In Figure 6.10, a service definition net and a refinement definition net are displayed.



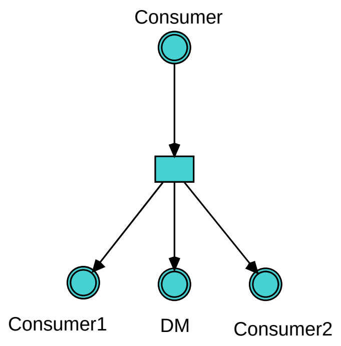


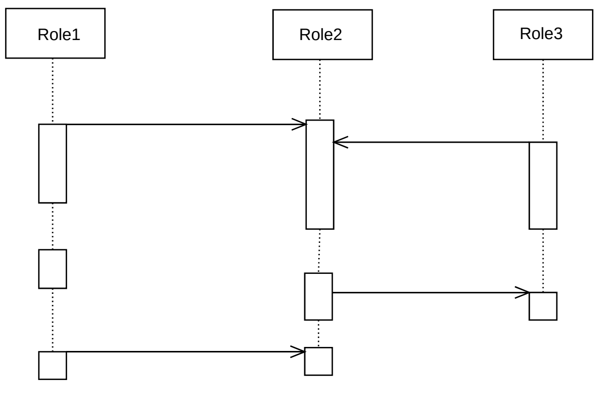
Figure 6.10.: Service definitions with a refinement definition net.

The service definitions and the refinements definition nets are saved in a single file with the extension *.rnd*. These kind of files can be read and parsed from the organization and R/D net plug-in. The service nets and relationships parsed from the service and refinement definition *.rnd* files are used then for the generation of the context based inscriptions in the R/D nets. This process will be described in more details in Subsection 6.2.4.

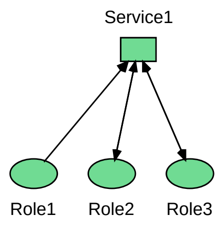
6.2.3. Generation of Service Definition Nets From AUML Diagrams

Another feature that was implemented in the plug-in for the service and refinement definition nets was the generation of service definition nets from AUML sequence diagrams. AUML sequence diagrams are already a documentation convention for agent interactions in the agent-oriented projects for undergraduate students at The Department for Theoretical Foundations of Informatics at the University of Hamburg. For this reason, the generation of service definition nets from AUML diagrams was implemented to automate and simplify the process of the creation of service definition nets from a pool of already present AUML diagrams. AUML diagrams are very similar to service nets. AUML sequence diagrams include all the information that a service definition net needs, namely the roles involved in the agent interactions and the messages that these roles send or receive. So, a service definition net can be generated from an AUML sequence diagram. The tool used in the aforementioned agent-oriented projects for the creation and manipulation of AUML diagrams is developed in [Cab03] and is also implemented as a Renew plug-in.

The AUML diagram tool saves AUML diagrams as files with the extension *.aip*. Such files can be chosen, read and parsed from the the plug-in for service and refinement definition nets. Choosing *.aip* files is realized with a simple Java file chooser and the selection of multiple files is also supported. These files are sequentially read with the help of the AUML diagram plug-in and parsed for the information needed by service definition nets. As service definition nets have no characterizing order function to be hold, extracting the information needed from the AUML diagrams and mapping this information into service definition nets is relatively straightforward. First, the roles in an AUML diagram have to be found and then it has to be asserted if those roles send or receive messages to/from other roles. After this information is extracted the generation of service definition nets can proceed. For each *.aip* file a transition is created with the name of the file as a a name inscription representing the service. For each role in the *.aip* files a place is created with the name of the role as the name inscription. The places in the service definition nets generated from the roles in an AUML diagram are connected with the respective generated transition with an arc pointing to the transition if the role sends messages, and with an arc pointing to themselves if they receive messages. In Figure 6.11, an example of such a generation process is displayed.



(a) AUML diagram, Service1.

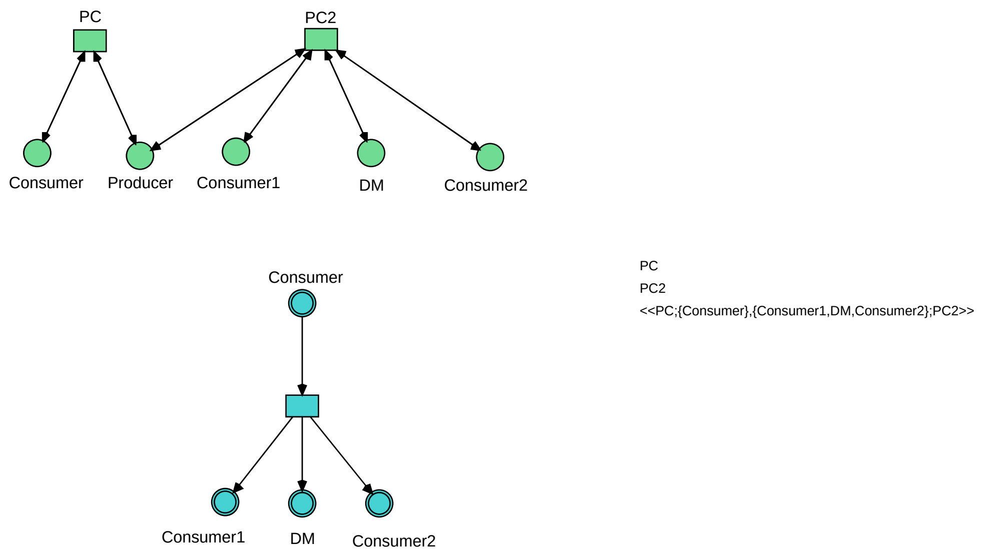


(b) Generated service definition.

Figure 6.11.: Generation of service definitions from AUML diagrams.

6.2.4. Generating Context Based Suggestions

After constructing the service and refinement definition nets and saving these nets as a file with the extention *.rnd*, the organization and R/D net editor can load the service and refinement definition file and start the generation of context based suggestions. A simple file chooser in the organization and R/D net editor selects which service and refinement definition file to read and parse. After selecting the file, all service definitions and refinement definition nets contained in the file are parsed and for each of them a special text figure is created for the R/D net. The text figures containing all these definitions are then added to the top of the diagram drawing of the organization and R/D net. These special text figures are consulted every time context based suggestions have to be generated as they represent the set of all service nets $\mathcal{D}$ and the set of refinement relationships. The refinement relationships of the form $\langle\!\langle D; R_s, R_s; D\rangle\!\rangle$ with $R_s \subseteq R(D)$ are always true, therefore, they do not need to be extra specified in the refinement definition nets or in the special text figures. In Figure 6.12, the process of loading a file containing service and refinement definition nets is shown.



PC
PC2
<<PC;{Consumer},{Consumer1,DM,Consumer2};PC2>>

(a) Service and refinement definitions contained in an *.rnd* file.

(b) Text figures in the organization editor representing the service and refinement definition nets.

Figure 6.12.: Loading service and refinement definitions in the organization net.

After the service and refinement definitions have been loaded and after the structure of the R/D net is created, the assignment process of roles and service nets to respectively places and transitions can begin. To start the action of generating suggestions for a place or transition a user can either select the suggestion tool for places or transitions in the Renew GUI and then click on the desired place or transition. A more practical way to start the generation of suggestions is to use the right click shortcut on places or transitions. The process of suggestion generation is different for places and for transitions. Suggestions generated for transitions consist of service nets. Only one service net can be selected for the assignment of the transition to a service net. If the user makes a request for the generation of suggestions for a transition (for example with a right click on the transition), the preset and the postset of the transition are first determined. Then, if the preset of the transition is assigned, the suggestions are calculated as specified in Subsection 6.1.4.

From the list of suggestions made one service net can be selected and assigned to the transition. However, the possibility that the list of suggestion is empty also exists. If the list of suggestions is empty it means that the postset of t has more elements than the number of roles involved in each of the possible service nets. In this case the user should start removing places from the postset of t and try to generate suggestion again. In Figure 6.13, the generation of suggestions for a transition is displayed.

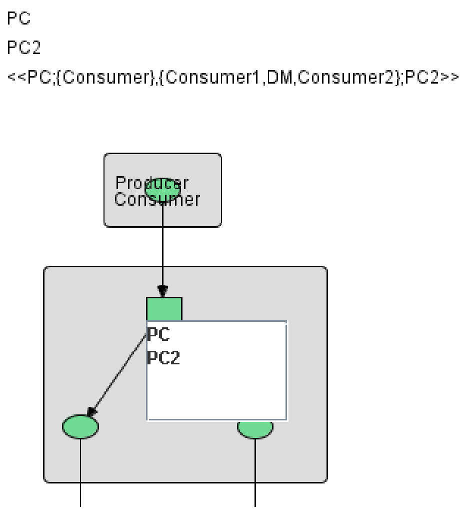

PC
PC2
<<PC;{Consumer},{Consumer1,DM,Consumer2};PC2>>

Figure 6.13.: Suggestion list for a transition.

Suggestions generated for places consist of roles. Many roles can be selected from the list of suggestions to be assigned to the places. If the user makes a request for the generation of suggestions for a place p, the preset of the place ${}^\bullet p$, the preset of the preset of the place $({}^\bullet p)$ and the siblings of the place $t^\bullet$ for each $t \in {}^\bullet p$ are determined. Then, the suggestions are generated as specified in Subsection 6.1.4. If the reference service of the preset of the preset of p is ambiguous, the list of suggestions includes suggestions calculated from each of the possible reference services. The suggestions from the same reference service are grouped together and have a small number denoting the group to which they belong. Suggestions belonging to more than one group, have multiple numbers, each number denoting a group. Only suggestions that belong to the same group can be selected. The notation of groups with numbers is made only to give users a visual feedback.

If the place has no preset then all roles are contained in the suggestion list. However, only selection of roles are possible so that for the selected roles R_s it is true that $R_s \subseteq R(D_x)$ with $D_x \in \mathcal{D}$. The possibility that the suggestion list is empty exists here too. In that case, there can be only two causes. If all places have roles assigned to them, then an empty suggestion list means that the assignment for the local context of the place p has ended and users can move on. If the suggestions list is empty and there are places which have no roles, then, there is a problem. For the case when the preset of p consists of only one transition t the problem is that the siblings of the place p in t have taken all possible roles as their assignments and have left the unassigned places with no roles. If the preset of p includes many transitions then the problem is either that the distribution of the roles to siblings is again faulty or that there are simply no roles that satisfy the condition for all $t \in {}^\bullet p$. To be sure where the problem lies users should generate first the suggestions for places shared between several transitions and then continue with those of the siblings. This way a faulty distribution of roles can be ruled out.

During the inscription assignment process the users can achieve well-formed R/D nets if they use the rules:

- The assignments have to be made in a *top-down* fashion. Service net suggestions can be generated for a transition only if the preset is assigned to roles. Role suggestions can be generated for a place only if its preset and the preset of its preset are assigned.

- If a transition t is assigned, users can continue with the assignment of $t^\bullet$. With an assigned t all places $p \in t^\bullet$ can be assigned. For the last unassigned place $p \in t^\bullet$ the whole suggestion list has to be selected.

- Errors:

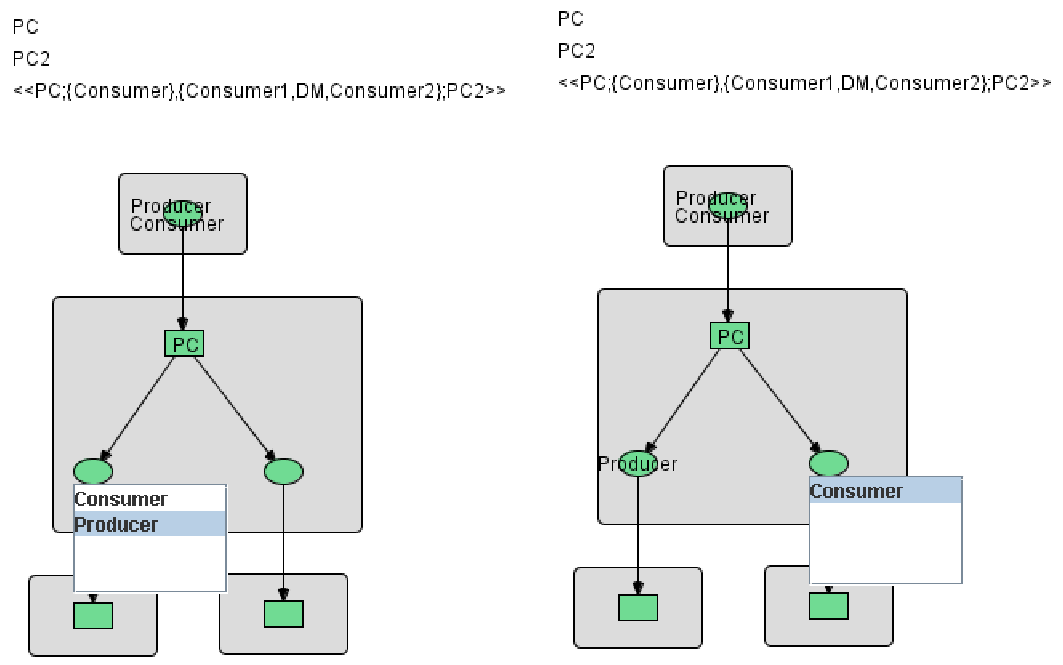
  - An empty suggestion list for an unassigned transition t means error. The postset of t has to be diminished by removing places from it.

  - An empty suggestion list for a place p where there are some unassigned $p_i \in ({}^\bullet p)^\bullet$ means that there is an error. The distribution of roles to the siblings of p is faulty or if $|{}^\bullet p| > 1$ there may be no roles that satisfy all conditions for every $t \in {}^\bullet p$. To rule out a faulty distribution, shared places between transitions should be assigned first.

These rules can be easily visualized which helps in their quick learning. Most of all, they do not require from users to make any calculation with set operations whatsoever. Also, the error rules allow users to also incrementally debug the well-formed R/D nets without a validation of the whole net. If errors occur, they can be checked and removed immediately.

6.2.5. Validation and Evaluation of Nets

The validation of the nets created with OREDI includes the validation if nets comply to the rules of organization nets and the validation if nets comply to the rules of well-formed R/D nets. Especially, the validation for well-formed R/D nets is particularly helpful as there is no guarantee that users create well-formed R/D nets even if supported by the context based suggestions. Also checking if the local context of a transition is well-formed can help to remove errors during the construction process of the organization net and not after it. The validation of an organization net is realized by going through all transitions and checking if their postset is in the same position and if their preset is in another position. Similarly, all places are checked if their postset is in another position


PC
PC2
<<PC;{Consumer},{Consumer1,DM,Consumer2};PC2>>
Producer
Consumer
PC
Consumer
Producer
PC
PC2
<<PC;{Consumer},{Consumer1,DM,Consumer2};PC2>>
Producer
Consumer
PC
Producer
Consumer

(a) Suggestion list for a place.

(b) All the suggestion list for the last unassigned place $p \in t^\bullet$ has to be selected.

Figure 6.14.: Suggestion list for places.

and if their preset is in the same position. The validation of organization nets can be started by the organization validation tool in the Renew GUI.

The validation of well-formed R/D nets is also realized by checking all transitions if they comply to the static and dynamic rules of well-formed R/D nets. For the validation of R/D nets to be successful the service and refinement definitions have to be loaded. If the net does not comply to the rules of well-formed R/D nets the first transition encountered which does not fulfill the conditions is highlighted and a message about the cause is displayed in the Renew GUI. Thus, this feature can also be used for debugging. The action of validating well-formed R/D nets can be started by the validation tool of well-formed R/D nets in the Renew GUI. The validation for a single transition if its local context is well-formed is straightforward and can be started by a right mouse click on a transition with a service net assignment. If the local context of the transition is well-formed a message is displayed otherwise the transition is highlighted. The validation of single transitions is useful if it is used immediately after assigning the local context with roles.

The evaluation of R/D nets for processability is realized by locating the productive variables. To locate productive variables the algorithm described in the Formula 5.1 is used with small changes. A set of places and transitions is used as $PV(G)$ representing the productive variables. Also, for the set X a set of transitions is used. If a place has a leaf transition (transition with no postset) as a postset it is added to $PV(G)$. If a place p contains a transition t in its postset and the postset of the transition t is made only of places contained in $PV(G)$, then p is added to $PV(G)$ too. In this way can be continued until all the places representing the productive variables are added to $PV(G)$. Finally, all the places of the set representing the productive variables are highlighted. The highlighted places $P_{productive}$ describe that all markings m with $m(p) > 0 \wedge p \in P_{productive}$ are processable.

The evaluation of R/D nets for strong processability is done in a similar fashion. First, a set of places representing the productive variables is constructed. Then, for each place p the set of reachable places $reach(p) \subseteq P$ is determined. If the set representing the productive variables contains all the elements of the set $reach(p)$ then p is added to a set P_{strong}. At the end, the places in the set P_{strong} are highlighted. The set P_{strong} describes that all markings m with $m(p) > 0 \wedge p \in P_{strong}$ are strongly processable. In Figure 6.15, all places are highlighted, which means that the R/D net displayed is a strongly processable net.

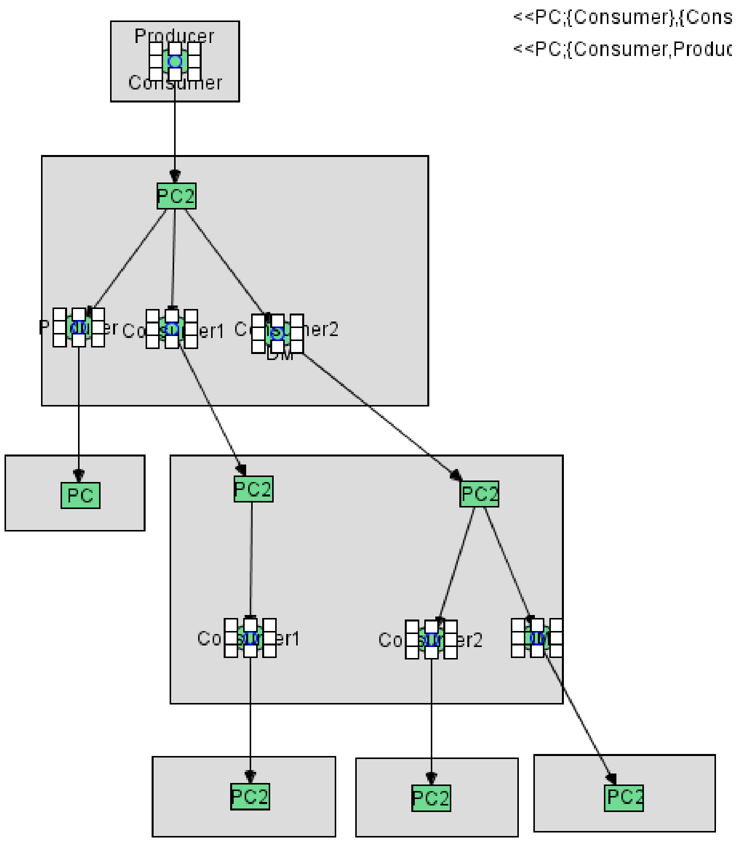


Figure 6.15.: A strongly processable net.

6.2.6. Parsing of Nets and XML Generation

The XML file where the organization or R/D net is exported, is created by parsing the net, extracting all needed information and writing it simultaneously in the XML file in the format described above. The net is parsed into a DOM tree model which is then written to a new file. Before the parsing process begins, all positions of the net are assigned to an id value that is unique within the set of all positions. Similarly, all places and all transitions of the net are assigned to id values which are unique within the set of all places and the set of all transitions respectively. First, positions are parsed for neighbors and transitions within it. The neighbor positions found are not parsed and written again, but a reference element with the id assigned at the beginning is written in the XML file instead. For transitions their preset and postset are found and written as reference elements with unique id values in the XML file and so on. A position is also parsed for the extra information that is written additionally to the XML file which includes the list of places which are the preset of all the transitions and the list of places which are the postset of its transitions. These lists are written in the XML file as the lists of role profiles that the position implements and the role profile that the position delegates. To start the action of exporting an organization or R/D net to an XML file, the XML generation tool in the Renew GUI should be selected. Then, a Java file chooser is prompted and after providing a name for the new file, the generation and writing of XML is completed and the file is saved.

6.3. Summary

In this chapter, OREDI, a modeling tool for the creation and manipulation of SONAR formal organizations was presented. The main requirements for OREDI included, among offering a graphical interface for the creation and manipulation of the models, supporting the users in the process of building models complying to the formal rules of SONAR organization nets and R/D nets. OREDI should also supply a validation feature that checks the formal correctness of the created organization or R/D nets along with an evaluation feature for the processability or strong processability of markings. Another requirement for OREDI was a feature that exports the created nets in a standard format.

Ensuring that users are supported into building formally correct organization or R/D nets was achieved by two main methods. For the correctness of organization nets restricting user interactions while building these nets to always comply with the formal conditions of organization nets was enough. Supporting the user in the inscription assignment process while building well-formed R/D nets involved the generation of context based suggestions. The generation of suggestions required the specification of special nets, namely, the service definition nets and the refinement definition nets. These nets were defined as petri nets with some additional conditions. They can be loaded in the organization nets and the generation of suggestions can then start. The generation of correct suggestions which lead users to well-formed R/D nets was analyzed and important cases were specified. The generation of correct suggestions can be made basing on some assumptions which can be learned from the users as simple thumb rules. However, the context based suggestion do not guarantee the creation of well-formed R/D nets. Thus, validation of R/D nets becomes important for debugging. The validation of organization or R/D nets is achieved by going through all transitions and places and by checking if they comply to the formal conditions for these type of nets. The determination of all processable and strongly processable markings for R/D nets was achieved by using the corresponding context free grammars and by determining the set of the productive variables and reachable variables. The implementation of the required features of OREDI was chosen to be made in Renew as two different plug-ins. One plug-in enables the creation of service and refinement definition nets. The other plug-in enables the creation and manipulation of organization and R/D nets and an extensive support of users during the process of building these nets through the organization and R/D net editor. In this plug-in, the validation evaluation and the export of the created nets as an XML file is also supported.

Chapter 7.

Deploying SONAR Formal Organizations into Mulan Agent Organizations

This chapter, describes the deployment of SONAR formal organizations into *Mulan* agent organizations and builds on the results of [KWE07]. The main focus will be on the specification and implementation of the process of generating *Mulan* agent organizations from SONAR formal organizations. With OREDI, a tool for the creation of SONAR formal organizations is now available. Thus, the generation of *Mulan* multi-agent organizations from the organization and R/D nets created with OREDI can proceed. In Section 7.1, the general approach for the deployment process is described. The main results from this section will be used in Section 7.2 and Section 7.3 which focus on the specification of the generation of position agents and the assignment of member agents to position agents.

7.1. From Formal Organization Models to Mulan Agent Organizations

In this section the process of generating Mulan multi-agent organizations from SONAR formal organization will be discussed. First, the multi-agent system design is described in Subsection 7.1.1. In Subsection 7.1.2, the general approach for generating position agents out of organization nets and assigning member agents to the position agents is discussed.

7.1.1. Multi-Agent System Design

This subsection describes the general design of multi-agent organizations in *Mulan* corresponding to SONAR formal organizations and is mostly drawn from [KWE07]. In [KWE07], a decoupling of the parts of the multi-agent system that specify the organizational structure from those that act as members of the organization is proposed. Each organizational position in a SONAR organization net is mapped to a *position agent*. Position agents are derived from the structure of the organization net and are owned by the organization. *Member agents* are external agents that actually do the work. They carry out actions and make decisions. However, they must use the position agents as a gateway to the organization. Member agents take part in the organizational processes only through their respective position agents. In Figure 7.1 an example is displayed.

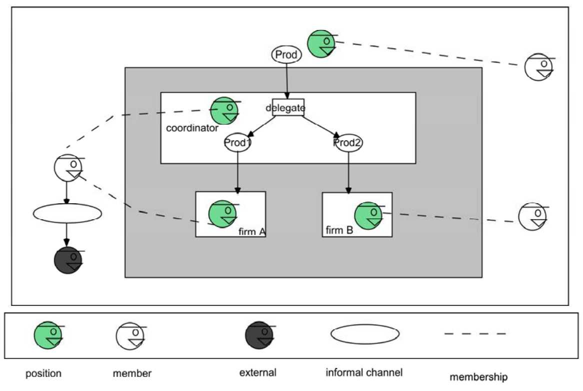


Figure 7.1.: A multi-agent design derived from an organization net ([KWE07]).

Member agents can be any kind of agents, even non Mulan agents. The decisive point is that an interface to the position agents is offered. Member agents can also occupy multiple positions. In the example in Figure 7.1 the same agent occupies the "coordinator" and "firm A". The division between position agents and member agents is conceived to support the software development process in application areas where organizations have to be developed separately from their members. Position agents are connected through *formal channels* which correspond to the delegation relationships of the organization nets. Also, informal channels may emerge between member agents due to interactions that were not foreseen by the formal specification. Position and member agents share the planning of the organizational processes. Practically, organizational processes are teamwork processes to carry out organizational services. Teamwork encompasses the stages of *team formation*, *team plan formation* and *team plan execution* [WJ99].

However, before a multi-agent organization can start teamwork processes, the position agents and the organizational structure should exist. Member agents should also be assigned to position agents. In the next subsection, the approaches taken for both the creation of the position agents and the assignment of member agents to position agents will be discussed.

7.1.2. Approach

Deploying a SONAR formal organization to a *Mulan* agent organization involves the generation of position agents and the assignment of member agents to position agents. After these two phases, team formation, team plan formation and team plan execution phases can follow as specified in [KWE07, Köh07]. In this work, only the generation of position agents and the assignment of members to positions are handled. The generation of positions and the assignment of members pave the way for the initiation of the team processes which will be handled in future works.

At first, a *Mulan* platform should be generated where one or more agent organizations can be embedded. The position agents generated for each position in the SONAR organization net have to be placed inside the created platform. Additionally, suitable *Mulan* protocols have to be developed to handle agent conversations. *Mulan* agents can use protocols proactively or reactively as a response to specific messages. The decision which protocol to use for a specific received message is made in knowledge base where a mapping between message templates and protocols is consulted.

The position agents should have the same information of their corresponding positions in the SONAR organization net specified in the XML file. This information includes the position's relative place in the organization (knowledge about neighbor positions), the roles they are implementing/delegating and their tasks. Generating position agents out of an organization net specification can be accomplished in agent-oriented fashion by an initial agent. The initial agent is responsible for the generation of the position agents and their initialization with informations extracted from the organization net specification. The initial agent will be called the *organization agent* as it has a global view on all positions.

The information needed from a position agent includes which other position agents are its neighbors. This requires the identity of the neighbors. At least in Mulan, the identity of agents and their location can only be known after the creation of these agents. This means that information about the neighbor positions has to be provided for a position agent only after, not during its creation. Thus, the information about the place of a position agent in an organization has to be conveyed through a conversation with the organization agent. During the conversation, in order to make sure that the messages come from the right parties they have to be signed with a public key mechanism which requires that parties know their respective public keys. In Figure 7.2, an AUML sequence diagram displays the conversation between position agents and the organization agent during which the organization agent communicates to the positions all relevant informations extracted from the organization net specification. In FIPA terminology, conversations between agents are called *protocols*. Mulan *protocols* describe the behavior of agents during conversations. The AUML diagram in Figure 7.2 also serves as an overall sketch of *Mulan* protocols.

The conversation displayed in Figure 7.2 is based on the assumption that the position agents already know the identity of their organization agent but the organization agent does not know the identities of its position agents. Position agents send a message with their identifiers to their organization agent requesting their local structure which should include all the relevant information extracted from the respective positions in the SONAR organization net such as the neighbors, the implementing and delegating roles, the tasks, etc. After receiving the requests for the local structure and the identifiers from all the position agents, the organization agent can proceed and send the respective local structure to each position agent. If the conversation partner know their respective public keys, all the messages of the conversation can be signed with the private keys of the sending par-

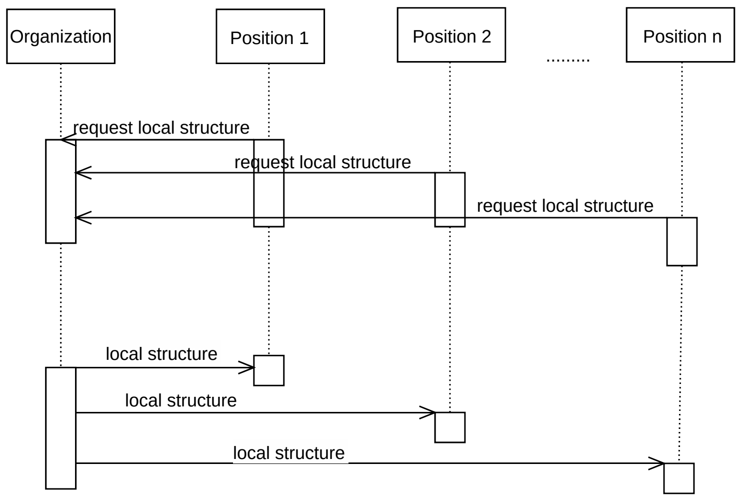


Figure 7.2.: The organization agent communicating to the generated positions the informations extracted from the respective positions in the SONAR organization net.

ties. However, the aspect of authentication has been left out from Figure 7.2 for simplicity reasons.

After the generation of the position agents and the communication of the local structures to them, the assignment of member agents to position agents can be started. The approach for the assignment process is leaned on [Köh07]. As the organization agent represent some kind of a service provider and logical platform to the positions and the potential members, it can assume at this point the management of the assignment of members to positions. If a position agent has an open position either because its member agent resigned or it has been fired, the position agent sends a request to the organization agent to start the procedure for the occupation of the open position. The organization agent publishes a job description for the open position to a central registry component named DF [FIP04] (**D**irectory **F**acilitator). A DF is a mandatory component of an agent platform in FIPA that provides a yellow pages directory service to agents. Agents can advertise their services through the DF. In *Mulan* the DF is also an agent with which the organization agent can communicate. The external agents that are interested in occupying open positions in the organization can search through the DF and apply to the organization agent for a specific open position. The initial assignment of position agents with member agents is a special case where all position agents have open positions. In Figure 7.3a, an AUML sequence diagram displays the procedure of occupying a vacant position during the initialization process, while in Figure 7.3b, the diagram displays the case when the resignation of a member triggers the start of the procedure to assign a new member for the position. The organization agent sends a description for a new job to the DF. The job description contains the identifier of the vacant position agent, the requirements that applicants have to fulfill, and a time period during which applications for the job are accepted. Agents that find that job description interesting after a search in the DF, apply to the organization agent. Their application includes the description of the job for which they are applying, their public key, their personal abilities as a response to the requirements specified in the job description, and the costs for their service. The applications are received from the organization agent. After the application period for a vacant position expires, the organization agent sends all received applications to the respective position agent, The position agent selects the new member and sends to it the information that it has been hired as well as the public key for the authentication during their future communication. In the case of resignation from a member, the member is dismissed only after finishing its ongoing activities on behalf of the organization. Even if a new member may have been hired since the resignation request of the old member, the old member is dismissed only after finishing all its ongoing activities. This means that a position can have more than one member agents for limited time periods. An alternative way for an agent to get the list of open positions within an organization is to send a message to the organization agent itself with a request for the open positions that the organization has. In Figure 7.4 this case is displayed as an AUML diagram.

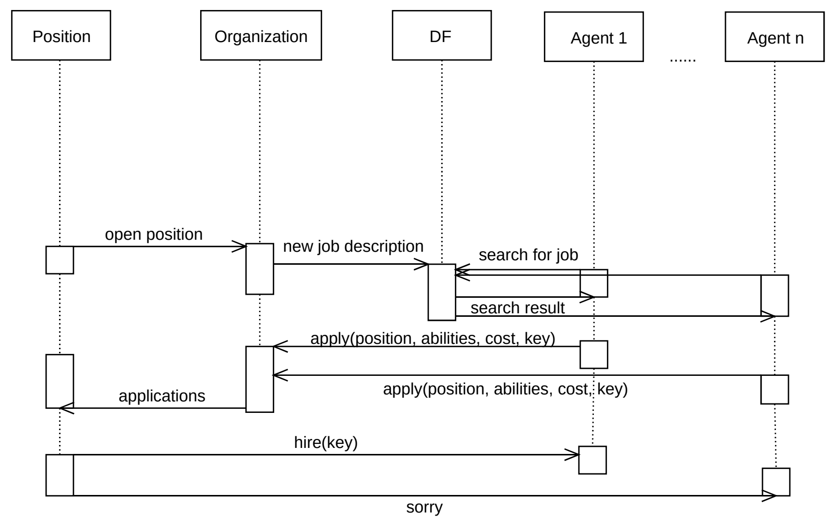
In the following subsection the ontology used for the communication of the local structures to the position agents and the assignment of member agents to position agents will be described.

7.1.3. Ontology

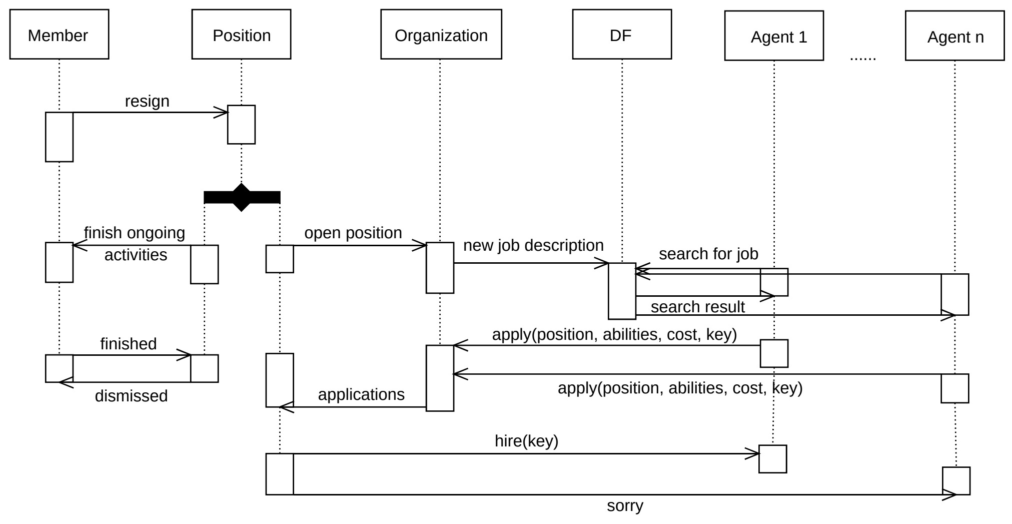
Both the communication of their local structure to the position agents and the assignment of member agents have to be accomplished by conversations with the organization agent. A shared vocabulary between the agents participating in the conversations is obviously indispensable for the agents to correctly understand each other. Thus, a common ontology has to be developed that provides the common basis for the communication between the organization agent and the position agents. the ontology developed in this work builds on the CAPA [Duv02] ontology. Here, a few notions build the basis of the ontology.

- The *Thing* notion describes everything possible.
- The *Predicate* notion describes when agent A asks agent B if a given proposition is true.
- The *Concept* notion is everything that is not a *Predicate*.
- The *AgentAction* notion is a *Concept* which describes the request of an agent A to an agent B to perform a specific task.

Obviously, *Predicates* and *Concepts* are *Things*. Every ontology element defined in this work is a specialization of these notions. The main ontology elements used for the com-



(a) During initialization.



(b) After the resignation of the member.

Figure 7.3.: The assignment of an agent as a member to a position agent.

munication of the local structure to the positions and the assignment of members to positions are defined below.

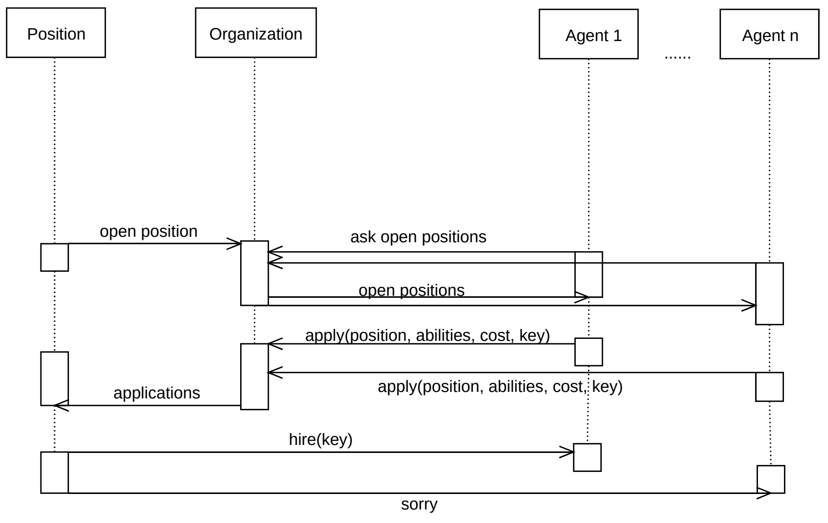


Figure 7.4.: Agents searching the organization directly for open positions.

Role is a *Concept* which describes an agent role. It has only a *name* attribute as a string

Roles is a *Concept* which describes an aggregation of *Role* concepts.

ServiceNet is a *Concept* which describes a service net. It has a *name* attribute as a string representing the name of the service net, and *roles* an aggregation of *Role* concepts representing the roles involved in the service net.

PublicKey is a *Concept* which describes an RSA [Lab07] public key. It contains the attributes *modulus* and *exponent* as hexadecimal numbers.

JobApplicationPeriod is a *Concept* which describes the time period in which applications for a vacant position are accepted. It contains the strings *beginTimestamp* and *endTimestamp* as attributes.

PositionJob is a *Concept* which describes a job description for a vacant position. It contains the following attributes: *applicationTime* as an instance of *JobApplicationPeriod*, *position* as an agent identifier of the vacant position, and *roleAbilities* as an instance of *Roles*. The *roleAbilities* define the role capabilities that applicant agents should have in order to be chosen as member of the vacant position.

PositionJobs is a *Concept* which describes an aggregation of *PositionJob* concepts.

MemberApplication is a *Concept* which describes the application submitted by an agent for a vacant position. It contains the following attributes: *positionJob* as an instance of

PositionJob describing the job for which this application is submitted, *roleAbilities* as an instance of *Roles* describing the capabilities of the applicant in terms of the roles that it can play, *cost* is a number describing the price that the applicant requires for its service as a member, and *publicKey* as an instance of *PublicKey* describing the public key of the applicant.

Hired is a *Concept* which specifies that the agent to which the message is sent has been chosen as member of a position for which the agent has applied before. It contains the attribute *memberApplication* describing the initial application of the agent for the position, and the attribute *publicKey* describing the public key of the position.

Sorryis a *Concept* which specifies that the agent to which the message is sent has not been chosen as member of a position for which the agent has applied. It contains the attribute *memberApplication* describing the initial application of the agent for the position.

Dismissed is a *Concept* which is sent to a member agent from its position as a notification that it has been removed.

DismissalAfterLastTask is a *Concept* which is sent to a member agent from its position as a notification that it has will be removed after it finishes all its ongoing tasks.

LastTaskFinished is a *Concept* which is sent to a position agent from its member as a notification that it has finished all its tasks.

SignedContent is a *Concept* which describes a signed content sent from an agent to another. It contains a *signedContent* attribute as an instance of *Concept* or *Predicate*, and a *signature* attribute as a string. The *signedContent* attribute represents the actual content sent. The *signature* attribute represents the signature generated from the *signedContent* sent. The signature is an SHA1 [31701] hash of the content which is then signed with the RSA private key of the sender.

RolesFromPosition is a *Concept* which describes the a set of roles that a position has to implement from another position. It contains the attributes *roles* as an aggregation of instances of *Role* representing the set of roles, and *aid* as the identifier of the position which delegates the set of roles.

RolesToPosition is a *Concept* which describes the a set of roles that a position delegates to another position. It contains the attributes *roles* as an aggregation of instances of *Roles* representing the set of roles, and *aid* as the identifier of the position which implements the set of roles.

Task is a *Concept* which describes a task and corresponds to a task in a SONAR organization net. It contains the following attributes: *implementingRole* as an instance of *RolesFromPosition*, *delegatingRoles* as an aggregation of instances of *RolesToPosition*, and *serviceNet* as an instance of *ServiceNet*.

NeighborInPosition is a *Concept* which describes a neighbor position from which a specific position implements. Its attributes are: *aid* representing the identifier of the neighbor position, *delRoles* as an instance of *Roles* representing all the roles that the neighbor delegates to the specific position for implementation.

NeighborOutPosition is a *Concept* which describes a neighbor position to which a specific position delegates. Its attributes are: *aid* representing the identifier of the neighbor position, *implRoles* as an instance of *Roles* representing all the roles that the specific position delegates to the neighbor for implementation.

SelfPosition is a *Concept* which describes the representation of a formal position. It contains the attributes: *aid* the agent identifier of the position, *delRoles* as an instance of *Roles* representing all roles that this position delegates, *implRoles* as an instance of *Roles* representing all roles that this position implements, *tasks* as an aggregation of instances of *Task* representing all the tasks of the position, *neighborIns* as an aggregation of instances of *NeighborInPosition* representing the set of positions from which this position implements, and *neighborOuts* as an aggregation of instances of *NeighborOutPosition* representing the set of positions to which this position delegates.

LocalStructure is a *Concept* and represents the local structure of a position. It contains an attribute named *self* as an instance of *SelfPosition*.

SendLocalStructure is an *AgentAction* which is sent from a position agent to the organization agent as a request to send the *LocalStructure* of the position. It contains the attribute *formalPositionId* as a number which represents the id of the corresponding position in the SONAR organization net.

OpenPositionProcedure is an *AgentAction* which is sent from a position agent to the organization agent requesting it to start the procedure for a vacant position. It contains the attributes *applicationTime* as an instance of *JobApplicationPeriod*, and *requirements* as an instance of *Roles* representing the requirements that applicants should fulfill for the vacant position.

SendOpenPositions is an *AgentAction* which is sent from an agent to the organization agent requesting it to send a list of all its open positions. It contains no attributes.

Resign is an *AgentAction* which is sent from a member to its position as a request to start the procedure for a vacant position.

All these ontology elements have to be embedded inside the content of the ACL messages that are exchanged between the Mulan agents. They are implemented as Java classes whose instances can be serialized as simple strings. The ontology objects and their string representations are actually organized as KVT (key value tuples) or VT (value tuples), complying to the FIPA specifications. They are used inside the SL0 content of ACL messages exchanged between *Mulan* agents.

7.2. Generating Position Agents from Organization Nets

For the generation of position agents from the organization net specification, the XML specification[1] is parsed. The file of the XML specification is made available through a simple file chooser at the beginning of the deployment process. As soon as the XML specification of the organization net is loaded, the parsing process can begin. At first, the organization agent is generated. The organization agent assumes the responsibility for the generation of the position agents. After its creation and initialization, the organization agent starts the protocol net "Organization_create_positions" which is displayed in Figure 7.5.

The "Organization_create_positions" protocol creates a position agent for every position contained in the organization net. In fact, the organization agent sends requests to the AMS (Agent Management System) for the creation of the position agents. The AMS is a mandatory component for FIPA platforms which is responsible for the lifecycles of the agents in the platform. In *Mulan* platforms, AMS is an agent itself which can communicate with other agents through ACL messages. Before the position agents are finally created, in the protocol "Organization_create_positions" their knowledge base is initialized with some important information such as the public key and the agent identifier of the organization agent as well as their own public and private key. Additionally, the id of their corresponding position in the SONAR organization net is saved in their knowledge base. All this initial information for the position agents is also available to the organization agent which is responsible for their creation. However, immediately after their creation the position agents still lack the information about their organizational context which includes neighbor positions, implementing roles, tasks etc. After their creation, all position agents start the protocol "Position_init" (Figure 7.6) which basically sends a request for their respective local structure to the organization agent. The request for the local structure contains the *SendLocalStructure* ontology object which includes the id of the corresponding position of the formal organization. The organization agent waits until it has received the *SendLocalStructure* requests from all its position agents (Figure 7.7). Through the *SendLocalStructure* requests, the organization agent gets the agent identifiers of the position agents and can construct a mapping of the agent identifiers to the id-s of the corresponding positions in the SONAR organization net. Then, the organization agent starts sending the local structure to every position agent. The local structure of position agents is individual and is determined from the organization agent basing on the id of the positions in the organization net to which the position agents correspond (Figure 7.8). Finally, the position agents receive the local structure sent to them from the organization agent and save it in their knowledge base (Figure 7.9). The local structure in the knowledge base of the position agents represent their organizational relationships and constraints.

[1]The XML file into which the organization net is exported.

7.3. Assigning Member Agents to Position Agents

The assignment process of member agents to position agents is based on several protocols which are displayed in Figures 7.10-7.18. The assignment of an agent to a position agent as its member can start either from the position agent itself or from the resignation of a member agent. If a member agents resigns it sends to its respective position agent a signed request with the ontology object *Resign* as its content. The reception of such a message from the member agent causes the position agent to initiate an instance of the protocol "Position_member_resignation" in Figure 7.10. During the execution of this protocol a signed request with the ontology object *OpenPostionProcedure* as content is sent to the organization agent and then it is checked if the member agent has any unfinished task that it still has to process. If this is the case, a message is sent to the member agent informing it that it will be dismissed after it has finished its last task. If the member agent had no unfinished tasks, the position agent sends to itself a request to remove the member so that the "Position_remove_member" protocol (Figure 7.11), which basically removes the member agent, is started. The initialization phase, where all positions do not have members, is a special case. In this phase, all positions send a request with an *OpenPostionProcedure* as content to the organization agent.

If the organization agent receives from one of its positions a signed request with the ontology object *OpenPostionProcedure* as a content, it starts an instance of the "Organization_openposition_process" protocol (Figure 7.12). In this protocol, a message is sent to start an instance of the protocol "Organization_register_job_DF" (Figure 7.13). This protocol serves to modify the organization agent description in the DF. In the modified organization agent description, the open positions can be found as service descriptions. In addition, the "Organization_openposition_process" protocol, sends to the organization agent a request for the start of a timer, which specifies the begin and the end time for the acceptance of applications for the open position. The request for the timer contains the description of the job (ontology object *PositionJob*) for which the timer should be started. The timer is represented by the protocol "Organization_timer_application" (Figure 7.15). This protocol makes sure that applications for the open position described in the request are received only during the time specified. The reception of applications is handled by the protocol "Organization_receive_application" (Figure 7.16) which saves the applications for the open positions to the knowledge base of the organization agent. After the end of the application period, the protocol Organization_timer_application sends all received applications to the respective position agent.

The position agent starts the protocol "Position_select_best_application" (Figure 7.17) after receiving all the applications made. In this protocol the best application is chosen. The best application is the application of the agent that fulfills the requirements of the job description and has the smallest costs for its service as a potential member. Then the agent that has the best application is hired and the public key of the position is sent to this agent in order to enable signed communication in the future. The position agent sends a message to the other applicants informing them that they were not selected. Finally, hiring the agent with the best application involves the protocol "Position_add_new_member" (Figure 7.18). This protocol adds the new member if the old member has finished the last task and is dismissed. However, if the old member of the position has not finished its last task yet, the agent with the best application is added as a passive member which becomes active as soon as the old member finishes its last task. This means that for limited periods of time, positions may have more than one member agents.

7.3.1. Member Assignment Interface

Member agents can not be generated from the SONAR organization net specification. Rather, they have to be developed and interact with the generated position agents. Interaction with the position agents requires the specification of interfaces. However, unlike object-oriented application programming interfaces (API-s), interfaces made for agent interaction are represented by agent communication messages and not by methods. These messages are already defined in the protocols displayed in this section. Nevertheless, in this subsection the interface messages will be specified in a bundled form.

Search in the DF for Open Positions inside Organizations

The first task to accomplish for an agent aspiring to become a member of a position agent is to search in the DF (Directory Facilitator) for agents which deliver the service "organization" and one or more services with the name "position-job-offer". Specifically, the agents looking for jobs should search for agent descriptions by sending a search request to the DF (s. [FIP04]) with the following agent description template:

```
(df-agent-description
    :services (set
    (service-description
    :name organization)
    (service-description
    :name position-job-offer
    :properties (set
    (property
    :name position-job-description))))
)
```

As a response to the search request, the DF delivers all agent descriptions that match the above template. In the agent descriptions delivered by the DF, inside the service description "position-job-offer" the property with the name "position-job-description" contains the ontology object *PositionJob* as value. The string representation of the object *PositionJob* is:

```
(positionjob
    :applicationTime (jobapplicationperiod ...)
    :position (agent-identifier ...)
    :roleAbilities (roles (sequence
    (role ...) (role ...)...))
)
```

In this way an agent that is looking for a job can get all job descriptions of all organizations.

Ask the Organization Agent for Open Positions

As an alternative to searching in the DF, the organization agents can be asked by agents for their open positions. The agents have to send a request to an organization agent with the ontology object *SendOpenPositions* as content.

```
(request
    :sender ...
    :receiver ...
    :content "((action (agent-identifier ...)
    (sendopenpositions)))"
    :language FIPA-SL0
)
```

In return the organization agents sends an inform with all its open positions as the content. The description of the open positions are enclosed inside the ontology object *PositionJobs*.

```
(positionjobs (sequence
    (positionjob ...) (positionjob ...) ...
    ))
```

Applying for Membership

To apply for the membership of a position an agent has to send an ACL request to the organization agent with the ontology object *MemberApplication* as content. The message should have the performative, content and language properties defined as shown below:

```
(request
    :sender ...
    :receiver ...
    :content "((action (agent-identifier ...)
    (memberapplication
        :aid (agent-identifier ...)
        :cost ...
        :positionJob (positionjob ...)
        :publicKey (publickey ...)
        :roleAbilities (roles (sequence
        (role ...) (role ...)...))
    )))"
    :language FIPA-SL0
)
```

After the end of the application time for an open position the agent receives as a response for its application either a message with the *Hired* ontology object or with the *Sorry* ontology object contained in its content.

```
(inform
    :sender ...
    :receiver ...
    :content "((hired
        :publicKey (publickey ...)
        :memberApplication (memberapplication ...)
    ))"
    :language FIPA-SL0
)
```

```
(inform
    :sender ...
    :receiver ...
    :content "((sorry
        :memberApplication (memberapplication ...)
    ))"
    :language FIPA-SL0
)
```

Member Resignation

For the member resignation, the member agent has to send to its position agent a request with the *Resign* ontology object.

```
(request
    :sender ...
    :receiver ...
    :content "((action (agent-identifier ...) (resign)))"
    :language FIPA-SL0
)
```

If the member has finished all its tasks, the position agent sends the final dismissal of the member agent.

```
(inform
    :sender ...
    :receiver ...
    :content "((dismissed))"
    :language FIPA-SL0
)
```

If the member agent has not yet finished its tasks, it receives a message with the ontology object *DismissalAfterLastTask*.

```
(inform
    :sender ...
    :receiver ...
    :content "((dismissalafterlasttask))"
    :language FIPA-SL0
)
```

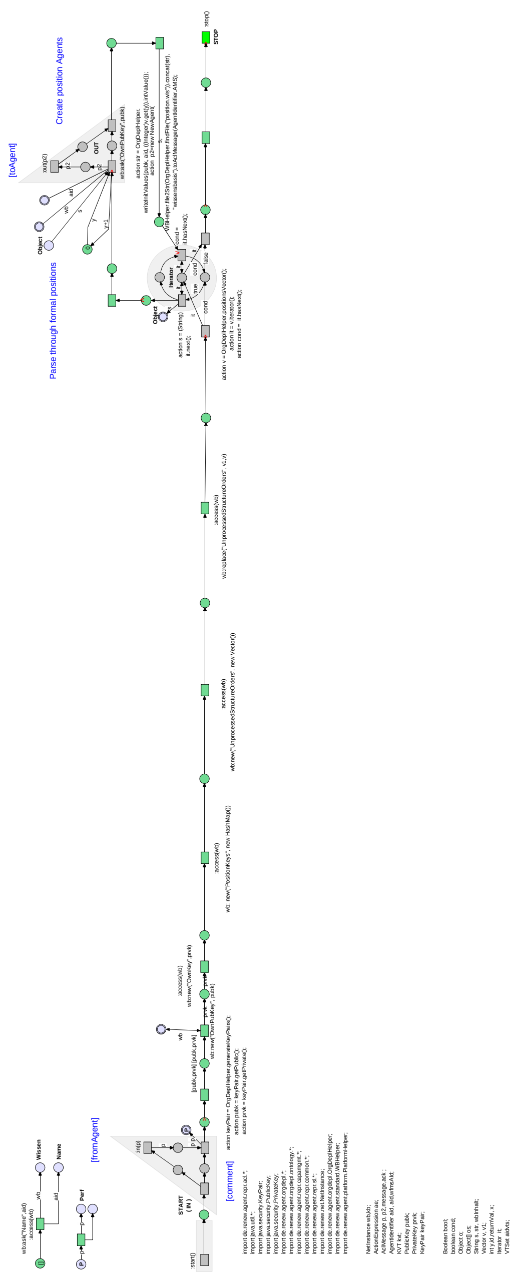
After receiving a *DismissalAfterLastTask* and after finishing all its tasks, the member agent should send a *LastTaskFinished* to the position agent.

```
(inform
    :sender ...
    :receiver ...
    :content "((lasttaskfinished))"
    :language FIPA-SL0
)
```

Then, the position agent removes the member agent and sends him the *Dismissed* message which is also described above.

7.4. Summary

For the deployment of SONAR organization nets into *Mulan* agent organizations, a decoupling of the parts of the multi-agent system that specify the organizational structure from those that act as members of the organization is proposed [KWE07]. Each organizational position in a SONAR organization net is mapped to a *position agent*. Position agents are derived from the structure of the organization net and are owned by the organization. *Member agents* are external agents that actually do the work. They carry out actions and make decisions. However, they must use the position agents as a gateway to the organization. Member agents take part in the organizational processes only through their respective position agents. Further, the deployment process involves the generation of position agents for each of the position elements in the organization net specification and the assignment of existing agents to position agents as their members. Both stages are implemented in an agent-oriented way. A *Mulan* agent is created that parses the XML specification of the organization nets and generates *Mulan* position agents from it. The position agent are placed in a *Mulan* platform. For both the deployment stages, *Mulan* protocols are provided. For the assignment of members to positions the developed *Mulan* protocols represent some kind of interface or API. An explicit specification of the member assignment interface, which the agents that are interested to occupy open positions can use, is also described in detail.



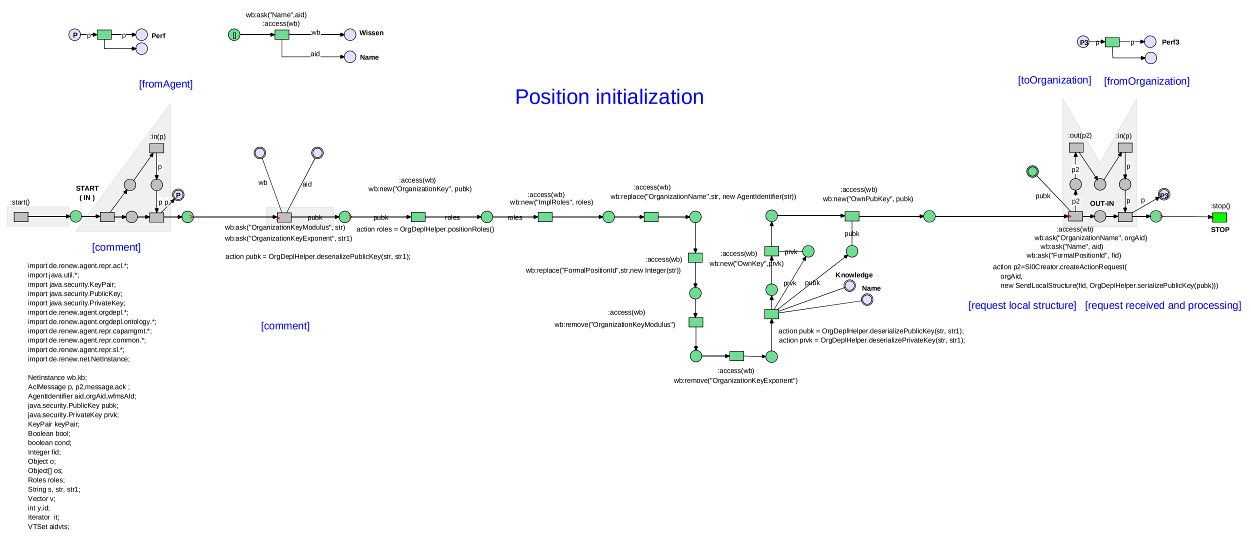
```
import de.renew.agent.repr.acl.*;
import java.util.*;
import java.security.KeyPair;
import java.security.PublicKey;
import java.security.PrivateKey;
import de.renew.agent.orgdepl.*;
import de.renew.agent.orgdepl.ontology.*;
import de.renew.agent.repr.capamgmt.*;
import de.renew.agent.repr.common.*;
import de.renew.agent.repr.sl.*;
import de.renew.net.NetInstance;
import de.renew.agent.orgdepl.OrgDeplHelper;
import de.renew.agent.standard.WBHelper;
import de.renew.agent.platform.PlatformHelper;

NetInstance wb,kb;
ActionExpression ae;
AclMessage p, p2,message,ack ;
AgentIdentifier aid, ald,wfmsAld;
KVT kvt;
PublicKey pubk;
PrivateKey prvk;
KeyPair keyPair;

Boolean bool;
boolean cond;
Object o;
Object[] os;
String s, str, wbInhalt;
Vector v, v1;
int y,id,returnVal, x;
Iterator  it;
VTSet aidvts;
```

```
action keyPair = OrgDeplHelper.generateKeyPairs();
    action pubk = keyPair.getPublic();
    action prvk = keyPair.getPrivate();
```

Figure 7.5.: The *Mulan* protocol net for generating positions (simplified).


Position initialization
P
p
p
Perf
wb:ask("Name",aid)
:access(wb)
[]
wb
aid
Wissen
Name
[fromAgent]
:start()
START
(IN)
:in(p)
p
p p
P
[comment]
wb
aid
wb:ask("OrganizationKeyModulus", str)
wb:ask("OrganizationKeyExponent", str1)
action pubk = OrgDeplHelper.deserializePublicKey(str, str1);
pubk
:access(wb)
wb:new("OrganizationKey", pubk)
pubk
roles
action roles = OrgDeplHelper.positionRoles()
[comment]
:access(wb)
wb:new("ImplRoles", roles)
roles
:access(wb)
wb:replace("OrganizationName",str, new AgentIdentifier(str))
:access(wb)
wb:replace("FormalPositionId",str,new Integer(str))
:access(wb)
wb:remove("OrganizationKeyModulus")
:access(wb)
wb:remove("OrganizationKeyExponent")
:access(wb)
wb:new("OwnKey",prvk)
prvk
prvk
pubk
Knowledge
Name
action pubk = OrgDeplHelper.deserializePublicKey(str, str1);
action prvk = OrgDeplHelper.deserializePrivateKey(str, str1);
:access(wb)
wb:new("OwnPubKey", pubk)
pubk
P3
p
p
Perf3
[toOrganization]
[fromOrganization]
:out(p2)
:in(p)
p2
p2
pubk
OUT-IN
p
p
p
P3
:stop()
STOP
:access(wb)
wb:ask("OrganizationName", orgAid)
wb:ask("Name", aid)
wb:ask("FormalPositionId", fid)
action p2=Sl0Creator.createActionRequest(
orgAid,
new SendLocalStructure(fid, OrgDeplHelper.serializePublicKey(pubk)))
[request local structure]
[request received and processing]

```
import de.renew.agent.repr.acl.*;
import java.util.*;
import java.security.KeyPair;
import java.security.PublicKey;
import java.security.PrivateKey;
import de.renew.agent.orgdepl.*;
import de.renew.agent.orgdepl.ontology.*;
import de.renew.agent.repr.capamgmt.*;
import de.renew.agent.repr.common.*;
import de.renew.agent.repr.sl.*;
import de.renew.net.NetInstance;

NetInstance wb,kb;
AclMessage p, p2,message,ack ;
AgentIdentifier aid,orgAid,wfmsAId;
java.security.PublicKey pubk;
java.security.PrivateKey prvk;
KeyPair keyPair;
Boolean bool;
boolean cond;
Integer fid;
Object o;
Object[] os;
Roles roles;
String s, str, str1;
Vector v;
int y,id;
Iterator  it;
VTSet aidvts;
```

Figure 7.6.: The *Mulan* protocol net that position agents use to request the local structure from their organization agent (simplified).

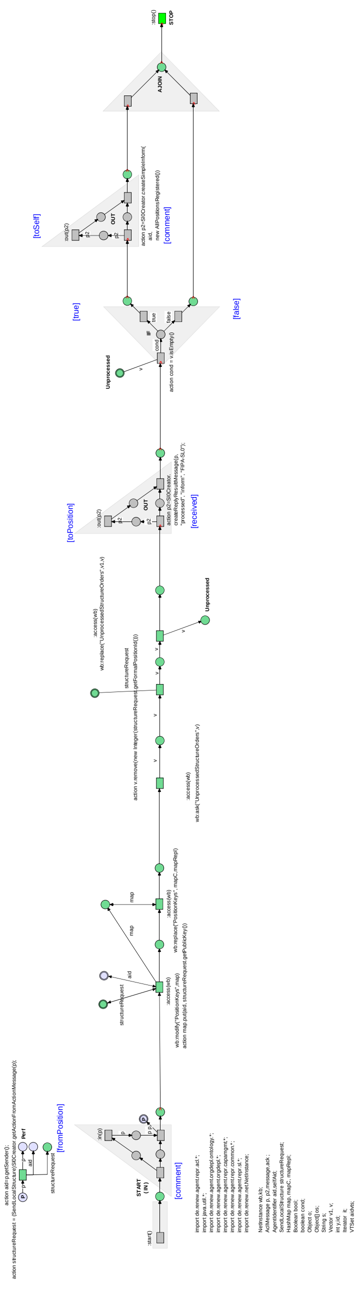
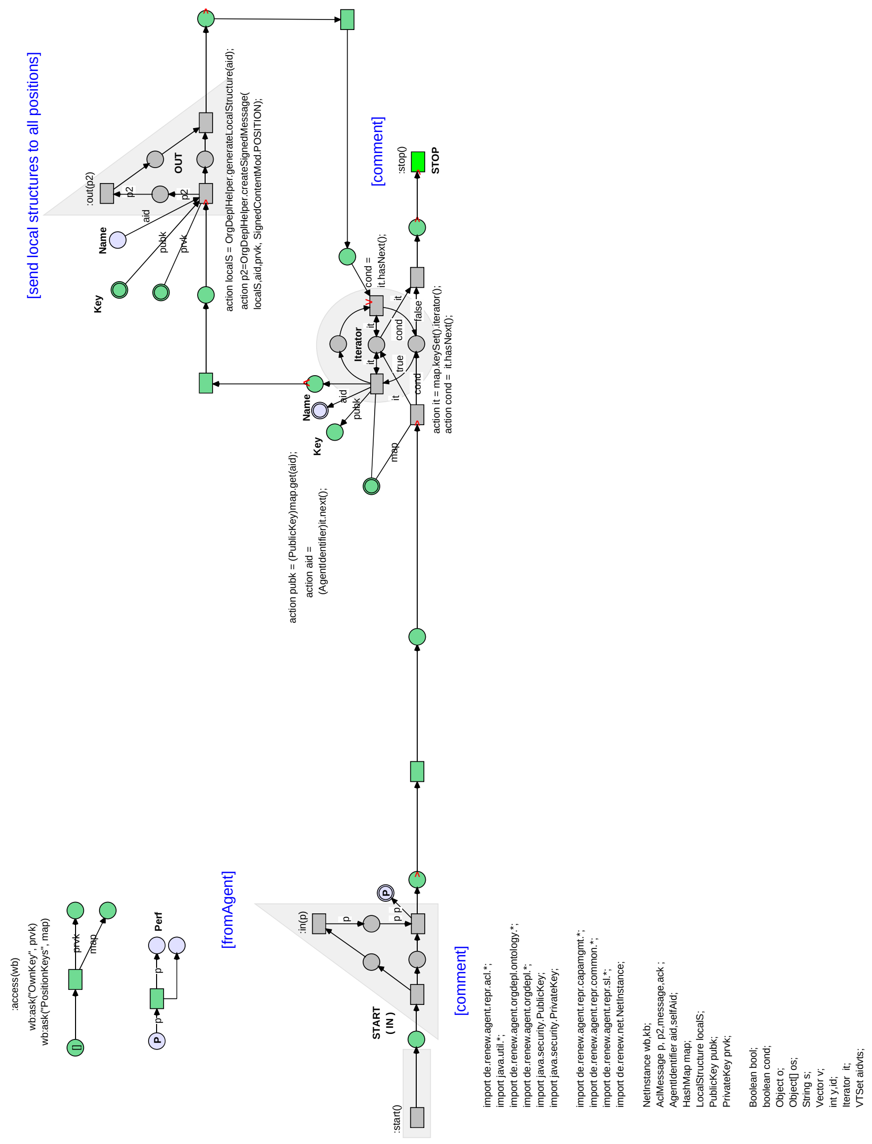


Figure 7.7.: The *Mulan* protocol net used by the organization agent for receiving all position requests for the local structure(simplified).



```
action localS = OrgDeplHelper.generateLocalStructure(aid);
action p2=OrgDeplHelper.createSignedMessage(
localS,aid,prvk, SignedContentMod.POSITION);
```

```
action pubk = (PublicKey)map.get(aid);
action aid =
(AgentIdentifier)it.next();
```

```
cond =
it.hasNext();
```

```
action it = map.keySet().iterator();
action cond =  it.hasNext();
```

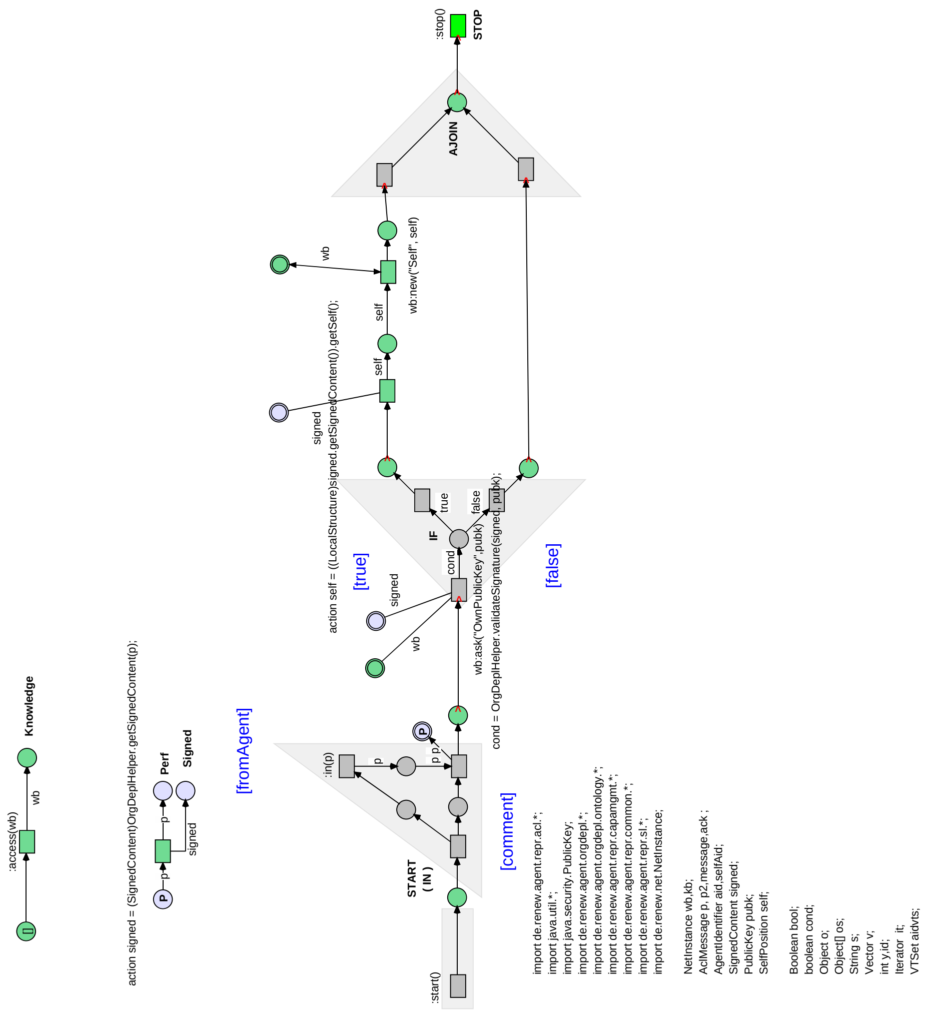
```
import de.renew.agent.repr.acl.*;
import java.util.*;
import de.renew.agent.orgdepl.ontology.*;
import de.renew.agent.orgdepl.*;
import java.security.PublicKey;
import java.security.PrivateKey;

import de.renew.agent.repr.capamgmt.*;
import de.renew.agent.repr.common.*;
import de.renew.agent.repr.sl.*;
import de.renew.net.NetInstance;

NetInstance wb,kb;
AclMessage p, p2,message,ack ;
AgentIdentifier aid,selfAid;
HashMap map;
LocalStructure localS;
PublicKey pubk;
PrivateKey prvk;

Boolean bool;
boolean cond;
Object o;
Object[] os;
String s;
Vector v;
int y,id;
Iterator  it;
VTSet aidvts;
```

Figure 7.8.: The *Mulan* protocol net for sending the local structure to all positions (simplified).


:access(wb)
wb
Knowledge
action signed = (SignedContent)OrgDeplHelper.getSignedContent(p);
P
p
p
Perf
signed
Signed
[fromAgent]
:start()
START
(IN)
:in(p)
p
p p
P
[comment]
wb
signed
[true]
IF
cond
true
false
wb:ask("OwnPublicKey",pubk)
cond = OrgDeplHelper.validateSignature(signed, pubk);
[false]
signed
action self = ((LocalStructure)signed.getSignedContent()).getSelf();
self
self
wb
wb:new("Self", self)
AJOIN
:stop()
STOP

```
import de.renew.agent.repr.acl.*;
import java.util.*;
import java.security.PublicKey;
import de.renew.agent.orgdepl.*;
import de.renew.agent.orgdepl.ontology.*;
import de.renew.agent.repr.capamgmt.*;
import de.renew.agent.repr.common.*;
import de.renew.agent.repr.sl.*;
import de.renew.net.NetInstance;

NetInstance wb,kb;
AclMessage p, p2,message,ack ;
AgentIdentifier aid,selfAid;
SignedContent signed;
PublicKey pubk;
SelfPosition self;

Boolean bool;
boolean cond;
Object o;
Object[] os;
String s;
Vector v;
int y,id;
Iterator  it;
VTSet aidvts;
```

Figure 7.9.: The *Mulan* protocol net that position agents use to save the representation of their organizational relationships to their knowledge base.

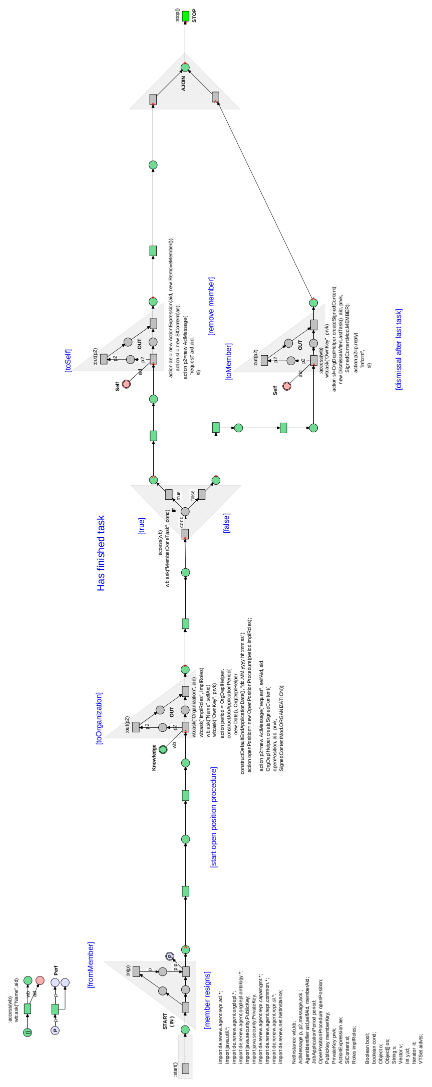


Figure 7.10.: The *Mulan* protocol net for handling the resignation of a member agent.

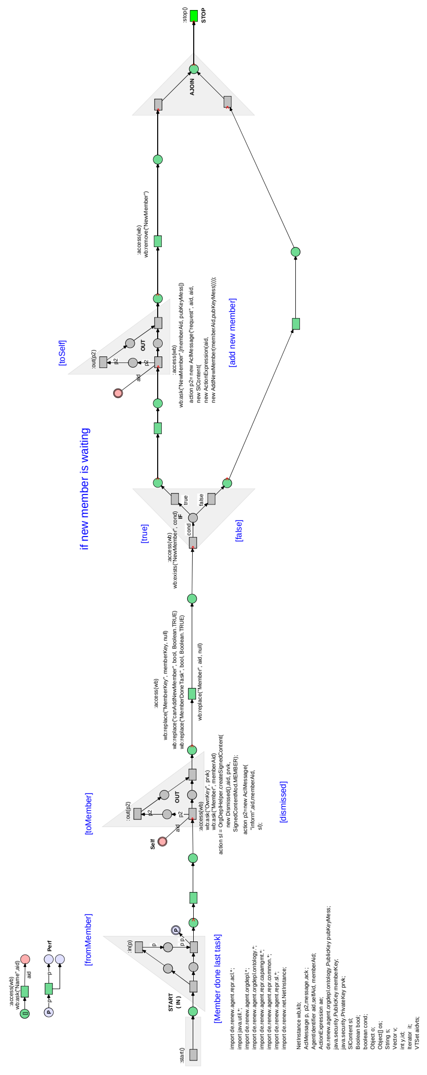
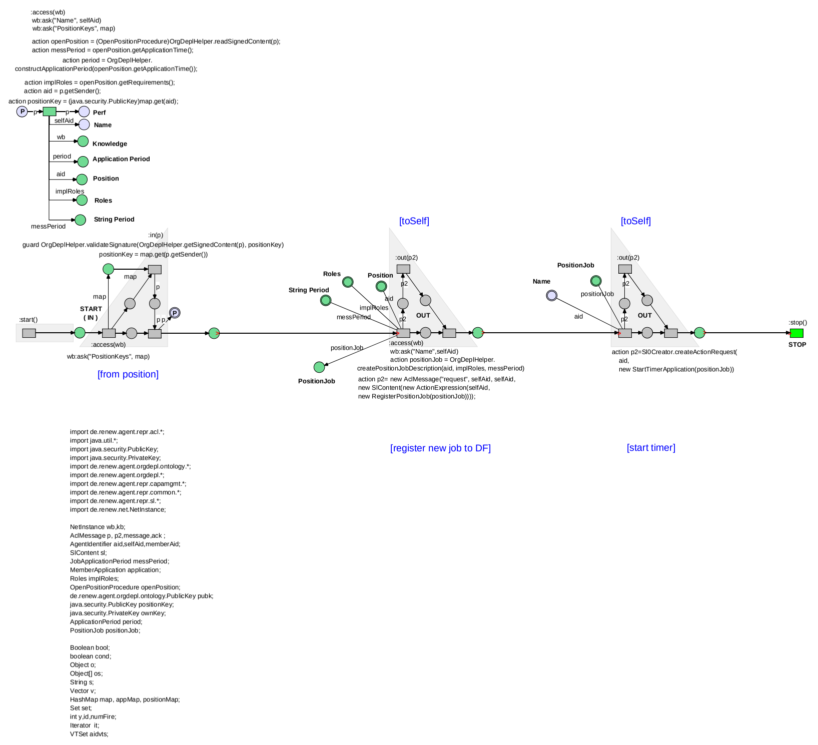


Figure 7.11.: The *Mulan* protocol net for removing a member agent from a position agent.



```
:access(wb)
wb:ask("Name", selfAid)
wb:ask("PositionKeys", map)

action openPosition = (OpenPositionProcedure)OrgDeplHelper.readSignedContent(p);
action messPeriod = openPosition.getApplicationTime();
                    action period = OrgDeplHelper.
constructApplicationPeriod(openPosition.getApplicationTime());

action implRoles = openPosition.getRequirements();
action aid = p.getSender();
action positionKey = (java.security.PublicKey)map.get(aid);
```

```
guard OrgDeplHelper.validateSignature(OrgDeplHelper.getSignedContent(p), positionKey)
positionKey = map.get(p.getSender())
```

```
:access(wb)
wb:ask("PositionKeys", map)
```

```
:access(wb)
wb:ask("Name",selfAid)
action positionJob = OrgDeplHelper.
createPositionJobDescription(aid, implRoles, messPeriod)
action p2= new AclMessage("request", selfAid, selfAid,
new SlContent(new ActionExpression(selfAid,
new RegisterPositionJob(positionJob))));
```

```
action p2=Sl0Creator.createActionRequest(
  aid,
  new StartTimerApplication(positionJob))
```

```
import de.renew.agent.repr.acl.*;
import java.util.*;
import java.security.PublicKey;
import java.security.PrivateKey;
import de.renew.agent.orgdepl.ontology.*;
import de.renew.agent.orgdepl.*;
import de.renew.agent.repr.capamgmt.*;
import de.renew.agent.repr.common.*;
import de.renew.agent.repr.sl.*;
import de.renew.net.NetInstance;

NetInstance wb,kb;
AclMessage p, p2,message,ack ;
AgentIdentifier aid,selfAid,memberAid;
SlContent sl;
JobApplicationPeriod messPeriod;
MemberApplication application;
Roles implRoles;
OpenPositionProcedure openPosition;
de.renew.agent.orgdepl.ontology.PublicKey pubk;
java.security.PublicKey positionKey;
java.security.PrivateKey ownKey;
ApplicationPeriod period;
PositionJob positionJob;

Boolean bool;
boolean cond;
Object o;
Object[] os;
String s;
Vector v;
HashMap map, appMap, positionMap;
Set set;
int y,id,numFire;
Iterator  it;
VTSet aidvts;
```

Figure 7.12.: The *Mulan* protocol net for starting the open position procedures.

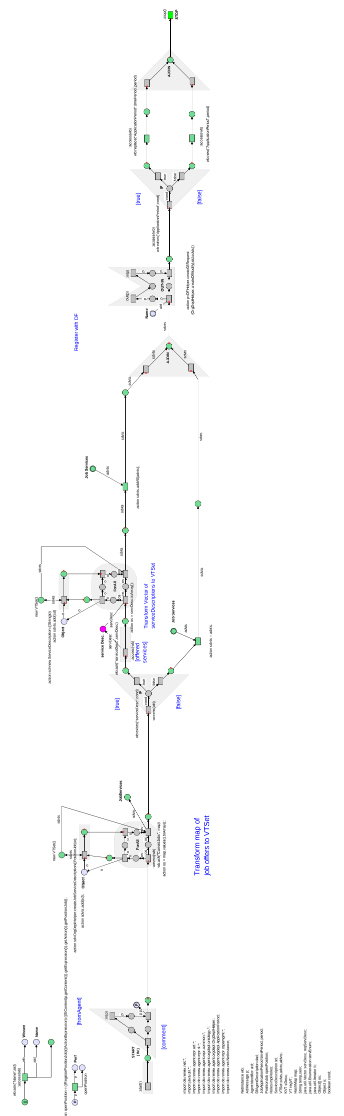
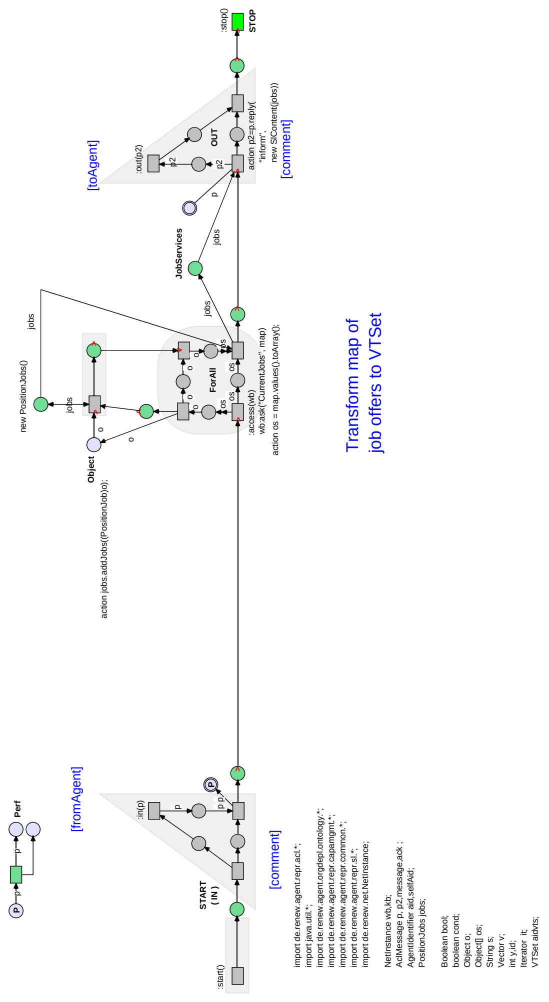


Figure 7.13.: The *Mulan* protocol net which registers the open positions of an organization to the DF.



```
import de.renew.agent.repr.acl.*;
import java.util.*;
import de.renew.agent.orgdepl.ontology.*;
import de.renew.agent.repr.capamgmt.*;
import de.renew.agent.repr.common.*;
import de.renew.agent.repr.sl.*;
import de.renew.net.NetInstance;

NetInstance wb,kb;
AclMessage p, p2,message,ack ;
AgentIdentifier aid,selfAid;
PositionJobs jobs;

Boolean bool;
boolean cond;
Object o;
Object[] os;
String s;
Vector v;
int y,i,d;
Iterator it;
VTSet aidvts;
```

Figure 7.14.: The *Mulan* protocol net which sends the list of open positions within an organization if requested from agents.

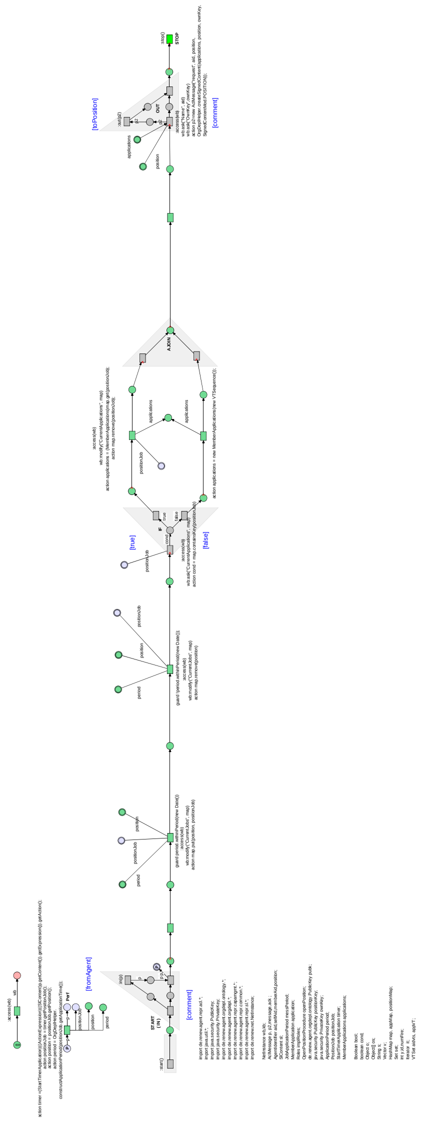


Figure 7.15.: The *Mulan* protocol net which controls the time where applications for a specific open position are accepted.

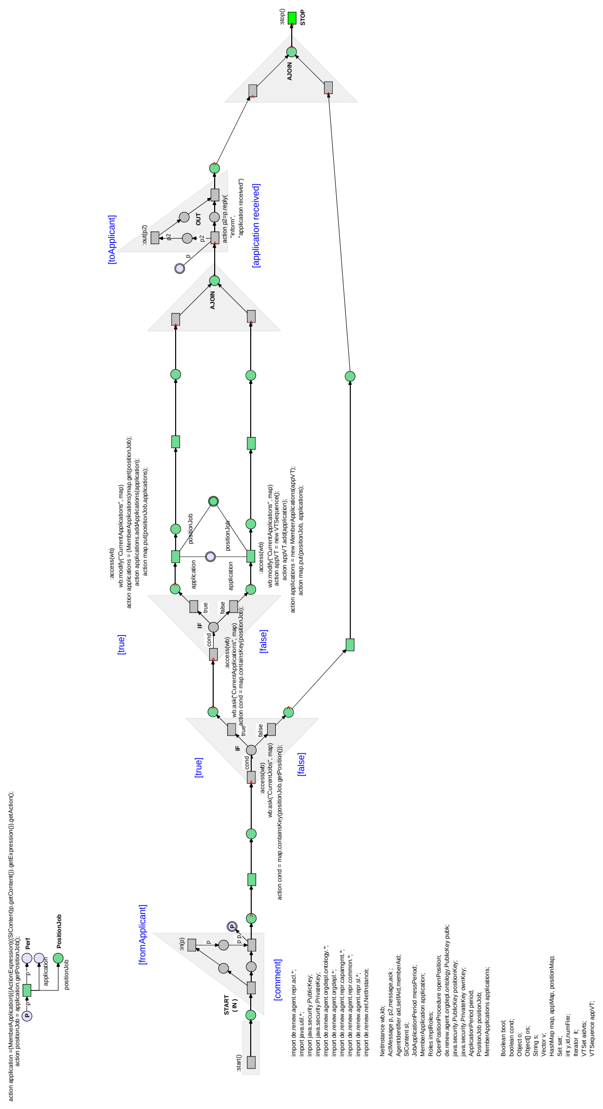


Figure 7.16.: The *Mulan* protocol net which handles the reception of an application for an open position.

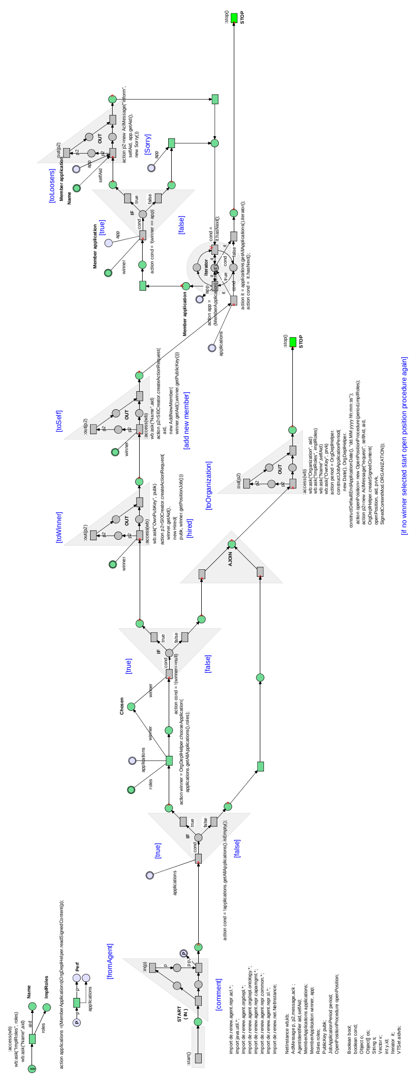
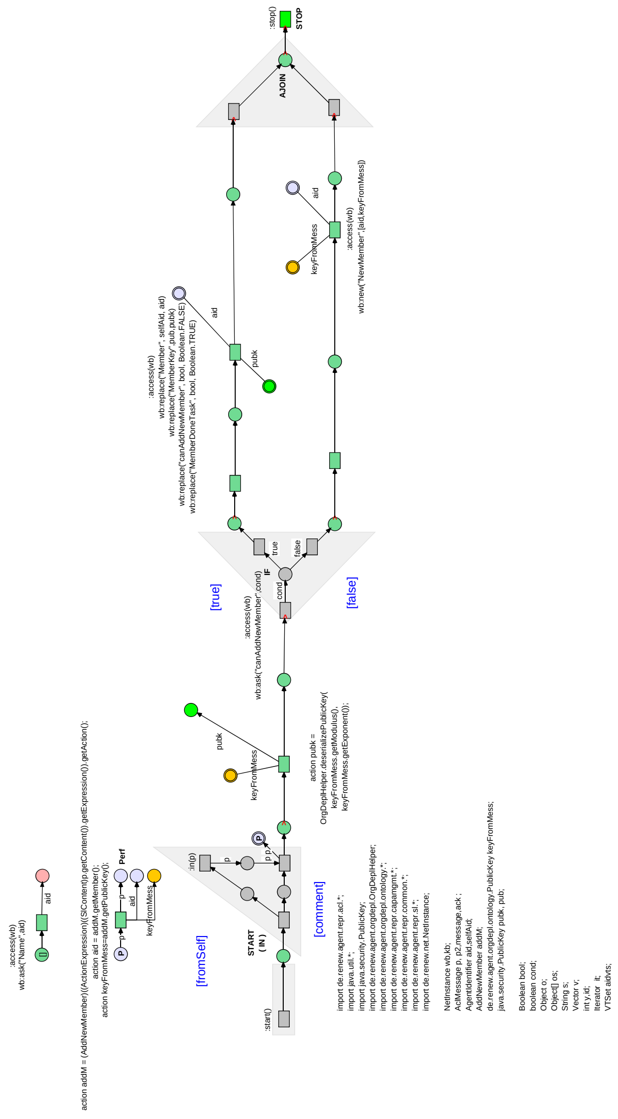


Figure 7.17.: The *Mulan* protocol net which selects the best application for a position.



```
:access(wb)
wb:ask("Name",aid)
```

```
action addM = (AddNewMember)((ActionExpression)((SlContent)p.getContent()).getExpression()).getAction();
action aid = addM.getMember();
action keyFromMess=addM.getPublicKey();
```

```
action pubk =
OrgDeplHelper.deserializePublicKey(
        keyFromMess.getModulus(),
        keyFromMess.getExponent());
```

```
:access(wb)
wb:ask("canAddNewMember",cond)
```

```
:access(wb)
wb:replace("Member", selfAid, aid)
wb:replace("MemberKey",pub,pubk)
wb:replace("canAddNewMember", bool, Boolean.FALSE)
wb:replace("MemberDoneTask", bool, Boolean.TRUE)
```

```
:access(wb)
wb:new("NewMember",[aid,keyFromMess])
```

```
import de.renew.agent.repr.acl.*;
import java.util.*;
import java.security.PublicKey;
import de.renew.agent.orgdepl.OrgDeplHelper;
import de.renew.agent.orgdepl.ontology.*;
import de.renew.agent.repr.capamgmt.*;
import de.renew.agent.repr.common.*;
import de.renew.agent.repr.sl.*;
import de.renew.net.NetInstance;

NetInstance wb,kb;
AclMessage p, p2,message,ack ;
AgentIdentifier aid,selfAid;
AddNewMember addM;
de.renew.agent.orgdepl.ontology.PublicKey keyFromMess;
java.security.PublicKey pubk, pub;

Boolean bool;
boolean cond;
Object o;
Object[] os;
String s;
Vector v;
int y,id;
Iterator  it;
VTSet aidvts;
```

Figure 7.18.: The *Mulan* protocol net which adds a new agent as a member agent.

Chapter 8.

Conclusion

Providing a graphical tool that enables the creation and editing of SONAR formal organizations supports the modeling stage in agent oriented software development. The tool developed as part of this thesis successfully supports users to build correct and well formed organization nets without previous knowledge of the rules underlying the formal model. In this chapter, an overview and summary of the present work (Section 8.1) will be given as well as an outlook at further development.

8.1. Summary and Results

After a short introduction (Chapter 1) where the motivation and the main goals of the thesis are presented, the work continues with Chapter 2 in which agents and multi-agent systems as well as the main concepts and architectures related with them are presented. Agents are introduced as autonomous entities that can sense part of their environment and have the ability to make decisions. They can be reactive or proactive meaning that they can only react to changes of their environment or act on their own initiative as well. Multi-agent systems are systems where many agents interact with each other. The social abilities of agents are crucial for their interactions in a multi-agent system and agents are required to have a common language and a shared vocabulary in order to understand each other and interact successfully.

In Chapter 3, a perspective on multi-agent systems is presented which is based on the concept of organizations. This organizational perspective on multi-agent systems has been inspired and has drawn much from modern organization theory. Since the rise of the organizational perspective the focus in multi-agent systems has shifted from the internal agent states toward organizational concepts such as groups, communities, organizations, roles, etc. Organizations on top of agents provide a concept that enables a better structuring and level of control on multi-agent systems. Moreover, organizations represent a further step toward more modular multi-agent systems. This ensures that the levels of predictability and manageability of multi-agent systems increase. Chapter 3 also provides an overview of the main organization types mostly used in multi-agent systems.

An introduction to petri nets, reference nets, *Renew* and *Mulan* is made in Chapter 4. First, definitions for (simple) petri nets and some of their main properties are given. Reference nets are then introduced as high level petri nets complying to the "nets within nets paradigm" where tokens in a net can represent other nets. The presentation of *Renew* as a reference net editor follows. Lastly, *Mulan* is shortly described as a reference architecture for multi-agent systems basing on reference nets. Moreover, the four abstraction layers of Mulan including the system layer, the platform layer, the agent layer and the protocol layer are summarized as well.

In Chapter 5, some organizational models are presented to delineate the most important modeling dimensions of organizations. The models presented include AGR, MOISE$^+$, AUML and ISLANDER. The detailed description of SONAR, a petri net model of organizations which is the reference model for this work, follows. AGR is build around the concepts of agents, groups and roles and involves a structural and dynamic aspect. The structural aspect specifies how agents are assembled in groups, how groups are related to each other and the set of roles for each group. The dynamic aspect specifies the interaction patterns defined within roles. In MOISE$^+$, the structural (role, links, groups etc.), as well as the functional (goal, plans) and the deontic (norms, laws, etc.) aspects of the organization are used. AUML is an extension of UML for modeling agent based systems. The structural aspect of organizations in AUML is based on AALAADIN and also combines elements from the dependency theory and holonics. ISLANDER is a declarative language for specifying electronic institutions. Electronic institutions correspond to human institutions which structure human interactions and enforce norms for all individuals. SONAR is a petri net model built around the concepts of positions, task delegation and services. It provides a formal approach to modeling formal organizations. Organization nets in SONAR consist of organizational positions. Positions are responsible for several tasks and can also delegate tasks to other organizational positions. Organizational services are represented in SONAR by service nets. Service nets are petri net models that specify how agents interact with each other. R/D nets are also defined in SONAR as petri nets with inscriptions and are used for task delegation in conjunction with service nets. The combination of organization nets and R/D nets specifies SONAR formal organization.

The main results of this thesis are described in detail in Chapter 6 and Chapter 7. In Chapter 6 a modeling tool was developed with which users can build organization and R/D nets as defined in SONAR. Apart from offering a graphical interface that enables users to create and edit basic organization and R/D nets, the tool supports users into building SONAR correct and well formed organizations without requiring active knowledge of the formal rules underlying SONAR. The tool supports the construction of correct organization and R/D nets by a combination of interaction restrictions that ensure that the nets hold to the rules underlying basic organization and R/D nets. These interaction restrictions include restrictions to deleting, moving, connecting and creating the graphical components that represent the formal elements of SONAR (positions, transitions, places). The tool created in this thesis also supports the users on building well formed organizations by generating context based suggestions for the service net or role assignments to places or transitions. The context based suggestions generated by the tool comply to some formal conditions. The formal conditions to which the suggestions comply, ensure that the development process of well formed organizations is simplified if users hold to some easy and intuitive thumb rules. The tool additionally supports the validation of the created organization or well formed organization nets. The combination of the context based suggestions and the validation of the created nets provide the features needed for easy debugging. By using corresponding context free grammars for R/D nets, the evaluation of R/D nets for the properties of processability and strong processability is provided. The formal organizations modeled with the tool developed in this thesis can be saved in an XML format. This enables the use of formal organizations from other applications or platforms. Finally, an extra feature that the tool provides is the generation of service net definitions from AUML diagrams. This feature was implemented to support the agent-oriented projects at the University of Hamburg.

The deployment of SONAR formal organization models into *Mulan* agent organizations is described in Chapter 7. The deployment process is based on the decoupling of the elements of the multi-agent system that specify the organizational structure from those that act as members of the organization. Positions in the formal organization are deployed as *Mulan* agents (position agents) and are embedded in a Mulan *platform*. Any agent can be assigned to a position agent as its member and take over the execution of the necessary tasks. In the first step during the deployment process of a formal organization model, an organization agent is created. The formal organization model is parsed and the organization agent generates the position agents. The organization agent is involved in a conversation and delivers the respective organizational context to the generated position agents. In the second step of the deployment process, all position agents publish open position jobs and receive applications from other agents to become their members. Each position selects the best application and hires an agent as its member. At the end of the deployment process the *Mulan* position agents have knowledge on the organizational context in which they are operating and they have other agents assigned to them as their members.

8.2. Outlook

During this work it became clear that additional features can be developed that can further support the modeling stage for the development of multi-agent systems.

For large systems it makes sense modeling parts of it separately. The same principle holds for large SONAR formal organization models. Allowing the separate development of parts of the organizations is from a practical point of view an important step for OREDI toward modularization which is key to applications. The modularization of the formal organization models allows the distribution of the modeling process. Formal organizations can be partitioned into modules which can be developed and maintained in separate files. Modules can be linked to each other by delegation relationships. Linking modules can be achieved by adding extra graphical components to OREDI that represent the link to other modules of the organization. The validation and evaluation of a formal organization as a whole does not suffer from its partition into modules if the modules contain only transitions which comply to the rules for correct or well formed organizations. Thus, the organization as a whole is correct or well formed,if all its modules/parts are correct and well formed.

The improvement of the suggestion algorithm for an arbitrary place p within an organization net, is also important as the current algorithm does not guarantee that by making selections from the suggestion list users always reach well formed local contexts of p. At this point there is potential for improvements especially for the case $|^\bullet p| > 1$. For this case, a deeper mathematical analysis is required to determine if the suggestion list can be further reduced so that role sets that can not lead to well formed local contexts are not included.

Moreover, the organization and R/D net tool can be extended with features that allow the computation of workloads for positions. If the set of task processes and their frequencies are known, the workload for the positions can be computed. Features that provide the determination and evaluation of petri net processes of organization nets as defined in [Köh06] can also be implemented. Petri net processes of organizations are linked with the possible teams within an organization [Köh06]. This makes the determination and the evaluation of petri net processes of organization nets important.

Simulating firings of transitions and the resulting dynamic change of markings for organization and R/D nets can additionally be helpful in analyzing and understanding dynamic aspects of these nets. Renew already provides a simulation environment for petri nets and the organization and R/D net tool is built so that it can be easily extended with simulation capabilities. Thus, the extension of the tool with simulation capabilities can be achieved with moderate efforts.

The deployment of organization and R/D nets in Mulan agent organizations can also be extended to include the team formation, team plan formation and team plan execution phases (see [KWE07]). In [KWE07], the theoretical basis for the team formation phase has already been laid. Presumably, the theoretical basis for the two other stages will soon come. With the deployment of teams and the construction of the respective team plans, the agent organization can finally become operational which mostly means that the agent teams in the organization can cooperate to achieve a specified result.

The implementation and realization of the above features is key toward the adoption of SONAR as an organizational model in real world multi-agent or software projects. These additional features would increase the support for designers and programmers in the modeling phase during the development process of software applications. With the implementation of the tool for the creation and editing of organization and R/D nets, the work made in this thesis has laid the basis for the realization of the aforementioned features.

Appendix A.

XML Schema of Formal Organization Nets

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:m="http://orgedi.org"
        targetNamespace="http://orgedi.org">

    <xs:element name="formalOrganization"
    type="m:formalOrganizationType">

        <xs:keyref name="positionTypeIdKeyRef"
        refer="orgTypeIdKey">
            <xs:selector
            xpath="m:organizationPosition/m:neghborPositions/*"/>
            <xs:field xpath="m:positionTypeId"/>
        </xs:keyref>

        <xs:keyref
        name="positionRoleProfileTypeIdKeyRef"
        refer="roleProfileTypeIdKey">
            <xs:selector
            xpath="m:organizationPosition/m:positionRoleProfiles/*"/>
            <xs:field xpath="m:roleProfileTypeId"/>
        </xs:keyref>

        <xs:keyref
        name="positionRoleProfileTypeIdKeyRef"
        refer="roleProfileTypeIdKey">
            <xs:selector
            xpath="m:organizationPosition/m:positionRoleProfiles/*"/>
            <xs:field xpath="m:roleProfileTypeId"/>
        </xs:keyref>


        <xs:keyref
        name="roleTypeIdKeyRef"
        refer="roleTypeIdKey">
            <xs:selector
            xpath="m:refinements/m:refinement/m:roles |
            m:organizationPosition/m:delegatedRoleProfiles/m:roleProfile"/>
            <xs:field xpath="m:roleTypeId"/>
        </xs:keyref>

        <xs:keyref
        name="serviceNetTypeIdKeyRef"
        refer="serviceNetTypeIdKey">
            <xs:selector
            xpath="m:organizationPosition/m:implTasks/m:executiveTask |
            m:organizationPosition/m:implTasks/m:delegativeTask |
            m:refinements/m:refinement"/>
            <xs:field xpath="m:serviceNetTypeId"/>
        </xs:keyref>


        <xs:key name="orgTypeIdKey">
            <xs:selector xpath="m:organizationPosition"/>
            <xs:field xpath="m:typeId"/>
        </xs:key>
        <xs:key name="roleProfileTypeIdKey">
            <xs:selector
xpath="m:organizationPosition/m:delegatedRoleProfiles/m:roleProfile"/>
            <xs:field xpath="m:typeId"/>
        </xs:key>
        <xs:key name="roleTypeIdKey">
            <xs:selector xpath="m:allRoles/m:role"/>
            <xs:field xpath="m:typeId"/>
        </xs:key>
        <xs:key name="serviceNetTypeIdKey">
            <xs:selector
            xpath="m:allServiceNets/m:serviceNet"/>
            <xs:field xpath="m:typeId"/>
        </xs:key>
    </xs:element>

    <xs:complexType name="formalOrganizationType">
        <xs:sequence>
            <xs:sequence>
                <xs:element ref="m:organizationPosition"
                minOccurs="0" maxOccurs="unbounded" />
            </xs:sequence>
            <xs:element ref="m:allServiceNets" />
            <xs:element ref="m:allRoles" />
            <xs:element ref="m:refinements" />
        </xs:sequence>
    </xs:complexType>


    <xs:element name="organizationPosition"
    type="m:positionType" />

    <xs:complexType name="positionType">
        <xs:sequence">
            <xs:element ref="m:typeId" />
            <xs:element ref="m:name"
            minOccurs="0" maxOccurs="1" />
            <xs:choice"
            minOccurs="1" maxOccurs="1">
                <xs:sequence">
                    <xs:element
                    ref="m:positionRoleProfiles" />
                    <xs:element
                    ref="m:implTasks" />
                </xs:sequence">
                <xs:element
                ref="m:delegatedRoleProfiles" />
                <xs:sequence">
                    <xs:element
                    ref="m:positionRoleProfiles" />
                    <xs:element
                    ref="m:implTasks" />
                    <xs:element
                    ref="m:delegatedRoleProfiles" />
                </xs:sequence">
            </xs:choice">
            <xs:element ref="m:neighborPositions"
            minOccurs="0" maxOccurs="1"/>
        </xs:sequence>
    </xs:complexType>


    <xs:element name="neighborPositions">
        <xs:complexType>
            <xs:choice
            minOccurs="1"
            maxOccurs="unbounded">
                <xs:element ref="m:incoming" />
                <xs:element ref="m:outgoing" />
            </xs:choice>
        </xs:complexType>
```

```
</xs:element>

<xs:element name="incoming">
    <xs:complexType>
        <xs:sequence">
            <xs:element ref="m:positionTypeId" />
        </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:element name="outgoing">
    <xs:complexType>
        <xs:sequence">
            <xs:element ref="m:positionTypeId" />
        </xs:sequence>
    </xs:complexType>
</xs:element>


<xs:element name="implTasks">
    <xs:complexType>
        <xs:choice
        minOccurs="1" maxOccurs="unbounded">
            <xs:element ref="m:executiveTask" />
            <xs:element ref="m:delegativeTask" />
        </xs:choice>
    </xs:complexType>
</xs:element>


<xs:element name="executiveTask">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="m:typeId" />
            <xs:element
            ref="m:serviceNetTypeId"
            minOccurs="0" maxOccurs="unbounded"/>
            <xs:element
            ref="m:implementingFromRoleProfiles" />
        </xs:sequence>
    </xs:complexType>
</xs:element>


<xs:element name="delegativeTask">
    <xs:complexType>
        <xs:sequence>
```

```
                <xs:element ref="m:typeId" />
                <xs:element ref="m:serviceNetTypeId" />
                <xs:element ref="m:implementingFromRoleProfiles" />
                <xs:element ref="m:delegatingToRoleProfiles" />
            </xs:sequence>
        </xs:complexType>
    </xs:element>


    <xs:element name="serviceNet">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="m:typeId" />
                <xs:element ref="m:name" />
                <xs:sequence>
                    <xs:element
                    ref="m:roleTypeId"
                    minOccurs="0" maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:sequence>
        </xs:complexType>
    </xs:element>


    <xs:element name="positionRoleProfiles">
        <xs:complexType>
            <xs:sequence>
                <xs:element
                ref="m:roleProfileTypeId"
                minOccurs="1" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>



    <xs:element name="delegatedRoleProfiles">
        <xs:complexType>
            <xs:sequence>
                <xs:element
                ref="m:roleProfile"
                minOccurs="1"
                maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
```

```
<xs:element name="delegatingToRoleProfiles">
    <xs:complexType>
        <xs:sequence>
            <xs:element
            ref="m:roleProfileTypeId"
            minOccurs="1"
            maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>


<xs:element
name="implementingFromRoleProfiles">
    <xs:complexType>
        <xs:sequence>
            <xs:element
            ref="m:roleProfileTypeId" />
        </xs:sequence>
    </xs:complexType>
</xs:element>



<xs:element name="roleProfile">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="m:typeId" />
            <xs:sequence>
                <xs:element
                ref="m:roleTypeId"
                minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:sequence>
    </xs:complexType>
</xs:element>


<xs:element name="role">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="m:typeId" />
            <xs:element ref="m:name"
            minOccurs="1" maxOccurs="1"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:element name="allServiceNets">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="m:serviceNet"
            minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:element name="allRoles">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="m:role"
            minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:element name="roles">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="m:roleTypeId"
            minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:element name="refinements">
    <xs:complexType>
        <xs:sequence>
            <xs:element
            ref="m:refinement"
            minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:element name="refinement">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="m:typeId" />
            <xs:element ref="m:serviceNetTypeId" />
            <xs:element ref="m:roles" />
            <xs:element ref="m:serviceNetTypeId" />
            <xs:element ref="m:roles" />
        </xs:sequence>
    </xs:complexType>
</xs:element>


<xs:element name="name" type="string" />

<xs:element name="typeId" type="integer" />

<xs:element name="serviceNetTypeId" type="integer" />

<xs:element name="roleTypeId" type="integer" />

<xs:element name="roleProfileTypeId" type="integer" />

<xs:element name="positionTypeId" type="integer" />


</schema>
```

Bibliography

[31701] 3174, RFC: *US Secure Hash Algorithm 1 (SHA1)*. 2001. – http://tools.ietf.org/html/rfc3174

[Aus62] AUSTIN, John L. ; PRESS, Oxford U. (Hrsg.): *How to Do Things with Words*. Oxford, 1962

[BDC92] BERNARDINELLO, Luca ; DE CINDIO, Fiorella: A survey of basic net models and modular net classes. In: *Advances in Petri Nets 1992, The DEMON Project*. London, UK : Springer-Verlag, 1992, S. 304–351

[BG88] BOND, Alan H. ; GASSER, Les: An Analysis of Problems and Research in DAI. In: BOND, Alan H. (Hrsg.) ; GASSER, Les (Hrsg.): *Readings in Distributed Artificial Intelligence*. San Mateo, CA : Morgan Kaufmann Publishers, 1988, S. 3–35

[Cab03] CABAC, Lawrence: *Modeling Agent Interaction Protocols with AUML Diagrams and Petri Nets*, University of Hamburg, Department of Computer Science, Diplomarbeit, 2003

[Cas98] CASTELFRANCHI, Cristiano: Modelling social action for AI agents. In: *Artif. Intell.* 103 (1998), Nr. 1-2, S. 157–182. – ISSN 0004–3702

[CG99] CARLEY, Kathleen M. ; GASSER, Les: Computational organization theory. (1999), S. 299–330

[CH92] CHRISTENSEN, S. ; HANSEN, N. D.: *Coloured Petri nets extended with channels for synchronous communication*. 1992. – Technical Report DAIMI PB390, Computer Science Department, Aarhus University, DK-8000 Aarhus C, Denmark.

[Che96] CHEONG, F.C.: *Internet Agents: Spiders, Wanderers, Brokers and Bots*. New Riders, 1996

[DAR] DARPA: *DAML*. – DARPA Agent Markup Language. http://www.daml.org/.

[DL87] DURFEE, Edmund ; LESSER, Victor: Using Partial Global Plans to Coordinate Distributed Problem Solvers. In: *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, 1987, 875-883

[Dur88] DURFEE, Edmund H.: *Coordination of Distributed Problem Solvers*. Norwell, MA, USA : Kluwer Academic Publishers, 1988

[Duv02] DUVIGNEAU, Michael: *Bereitstellung einer Agentenplattform für petrinetzbasierte Agenten*, University of Hamburg, Department of Computer Science, Diplomarbeit, Dezember 2002

[EPS01] ESTEVA, Marc ; PADGET, Julian ; SIERRA, Carles: Formalizing a language for institutions and norms. In: MEYER, John-Jules (Hrsg.) ; TAMBE, Milind (Hrsg.): *Pre-proceedings of the Eighth International Workshop on Agent Theories, Architectures, and Languages (ATAL-2001)*, 2001, 106–119

[FG98] FERBER, J. ; GUTKNECHT, O.: Aalaadin: a meta-model for the analysis and design of organizations in multi-agent systems. In: *ICMAS'98*, 1998

[FGM03] FERBER, Jacques ; GUTKNECHT, Olivier ; MICHEL, Fabien: From Agents to Organizations: An Organizational View of Multi-agent Systems. In: GIORGINI, Paolo (Hrsg.) ; MÜLLER, Jörg P. (Hrsg.) ; ODELL, James (Hrsg.): *AOSE* Bd. 2935, Springer, 2003 (Lecture Notes in Computer Science), S. 214–230

[FIP00] FIPA: *FIPA ACL Specification*. 2000. – http://www.fipa.org/specs/fipa00061/SC00061G.pdf

[FIP02] FIPA: *FIPA SL Specification*. 2002. – http://www.fipa.org/specs/fipa00008/SC00008I.pdf

[FIP03] FIPA: *Foundation for Intelligent Physical Agents*. 2003. – http://www.fipa.org

[FIP04] FIPA: *FIPA Agent Management Specification*. 2004. – http://www.fipa.org/specs/fipa00023/SC00023K.pdf

[Gas91] GASSER, Les: Social conceptions of knowledge and action: DAI foundations and open systems semantics. In: *Artif. Intell.* 47 (1991), Nr. 1-3, S. 107–138

[Gas01] GASSER, Les: Perspectives on organizations in multi-agent systems. (2001), S. 1–16

[Gil04] GILBERT, Michael A.: Multi-modal argumentation. In: *Philosophy of the Social Sciences* 24 (2004), Nr. 2, S. 159–177

[Gin91] GINSBERG, Matthew L.: Knowledge Interchange Format: the KIF of Death. In: *AI Magazine* 12 (1991), Nr. 3, S. 57–63

[Gro] GROUP, K.: *An Overview of KQML: A Knowledge Query and Manipulation Language*. – KQML Advisory Group. An Overview of KQML: A Knowledge Query and Manipulation Language. http://retriever.cs.umbc.edu:80/kqml/.

[HL04] HORLING, Bryan ; LESSER, Victor: A survey of multi-agent organizational paradigms. In: *Knowl. Eng. Rev.* 19 (2004), Nr. 4, S. 281–316. – ISSN 0269–8889

[HS99] HUHNS, Michael N. ; STEPHENS, Larry M.: Multiagent systems and societies of agents. (1999), S. 79–120

[HSB02] HÜBNER, Jomi F. ; SICHMAN, Jaime S. ; BOISSIER, Olivier: MOISE+: Towards a Structural, Functional, and Deontic Model for MAS Organization. In: *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'2002)*. Bologna, Italy : ACM Press, 2002, 501–502

[Jen93] JENNINGS, N. R.: Commitments and Conventions: The Foundation of Coordination in Multi-Agent Systems. In: *The Knowledge Engineering Review* 8 (1993), Nr. 3, S. 223–250

[Jen00] JENNINGS, Nicholas R.: On agent-based software engineering. In: *Artificial Intelligence* 177 (2000), Nr. 2, S. 277–296

[JFL$^+$01] JENNINGS, Nicholas R. ; FARATIN, Peyman ; LOMUSCIO, Alessio R. ; PARSONS, Simon ; SIERRA, Carles ; WOOLDRIDGE, Michael: Automated Negotiation: Prospects, Methods and Challenges. In: *Journal of Group Decision and Negotiation* 10 (2001), Nr. 2, S. 199–215

[JV87] JESSEN, Eike ; VALK, Rüdiger: *Rechensysteme: Grundlagen der Modellbildung*. 1987 (Studienreihe Informatik). – 562 S.

[KB91] KLEIN, M. ; BASKIN, A. B.: A Computational Model for Conflict Resolution in Cooperative Design Systems. In: DEEN, S. M. (Hrsg.): *CKBS'90: Proc. of the International Working Conference on Cooperating Knowledge Based Systems*. Berlin, Heidelberg : Springer, 1991, S. 201–219

[KMR01] KÖHLER, Michael ; MOLDT, Daniel ; RÖLKE, Heiko: Modelling the Structure and Behaviour of Petri Net Agents. In: COLOM, J.M. (Hrsg.) ; KOUTNY, M. (Hrsg.): *Proceedings of the 22nd Conference on Application and Theory of Petri Nets 2001* Bd. 2075, 2001, S. 224–241

[Köh06] KÖHLER, M.: Formalising Multi-Agent Organisations. In: *Proceedings of Concurrency, Specification, and Programming CS&P'2006*, 2006

[Köh07] KÖHLER, Michael: *Koordinierte Selbstorganisation und selbstorganisierte Koordination*. 2007

[Kum02] KUMMER, Olaf: *Referenznetze*. Berlin : Logos Verlag, 2002

[KWD01] KUMMER, Olaf ; WIENBERG, Frank ; DUVIGNEAU, Michael: Renew - The Reference Net Workshop. In: *Tool Demonstrations - 22nd International Conference on Application and Theory of Petri Nets*, 2001

[KWE07] KÖHLER, Michael ; WESTER-EBBINGHAUS, Matthias: Petri Net-Based Specification and Deployment of Organizational Models. In: MOLDT, Daniel (Hrsg.) ; KORDON, Fabrice (Hrsg.) ; HEE, Kees van (Hrsg.) ; COLOM, José-Manuel (Hrsg.) ; BASTIDE, Rémi (Hrsg.): *Proceedings of the International Workshop on Petri Nets and Software Engineering (PNSE'07).* Siedlce, Poland : Akademia Podlaska, Juni 2007, S. 67–81

[Lab07] LABORATORIES, RSA: *RSA Cryptography Standard.* 2007. – http://www.rsa.com/rsalabs/node.asp?id=2125

[LCN90] LEVESQUE, H. J. ; COHEN, P. R. ; NUNES, J. H. T.: On Acting Together. In: *Proc. of AAAI-90.* Boston, MA, 1990, S. 94–99

[LJO05] L., Coutinho R. ; J., Sichman S. ; O., Boissier: *Modeling Organization in MAS: A Comparison of Models.* 2005. – In First Workshop on Software Engineering for Agent-oriented Systems

[MDOR03] MOLDT, D. ; DUVIGNEAU, M. ; ORTMAN, J. ; RÖLKE, H.: *Project: Agentenorientierte Softwareentwicklung.* 2003

[MKR01] MOLDT, D. ; KÖHLER, M. ; RÖLKE, H.: *Project: Agenten-orientierte Softwareentwicklung.* 2001

[New82] NEWELL, Allen: The Knowledge Level. In: *Artif. Intell.* 18 (1982), Nr. 1, 87-127. `http://dblp.uni-trier.de/db/journals/ai/ai18.html#Newell82`

[Ode02] ODELL, J.: *Objects and agents compared.* 2002. – http://www.jot.fm/issues/issue_2002_05/column4

[OMG99] OMG: *OMG Unified Modeling Language Specification, m Version 1.3*, June 1999

[OPB00] ODELL, J. ; PARUNAK, H. ; BAUER, B.: *Extending UML for Agents.* 2000

[PO01] PARUNAK, H. Van D. ; ODELL, James: Representing social structures in UML. In: MÜLLER, Jörg P. (Hrsg.) ; ANDRE, Elisabeth (Hrsg.) ; SEN, Sandip (Hrsg.) ; FRASSON, Claude (Hrsg.): *Proceedings of the Fifth International Conference on Autonomous Agents.* Montreal, Canada : ACM Press, 2001, 100–101

[RN95] RUSSEL, S. ; NORVIG, P.: *Artificial Intelligence.* Prentice Hall, 1995

[RZ94] ROSENSCHEIN, Jeffrey S. ; ZLOTKIN, Gilad: *Rules of Encounter: Designing Conventions for Automated Negotiation Among Computers*. Cambridge, Massachusetts : MIT Press, 1994

[RZ98] ROSENSCHEIN, Jeffrey S. ; ZLOTKIN, Gilad: Designing conventions for automated negotiation. (1998), S. 353–370

[San99] SANDHOLM, Tuomas W.: Distributed Rational Decision Making. In: WEISS, Gerhard (Hrsg.): *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA, USA : The MIT Press, 1999, S. 201–258

[Sch03] SCHUMACHER, J.: *Eine Plug-in-Architektur für Renew: Konzepte, Methoden, Umsetzung*, University of Hamburg, Department of Computer Science, Diplomarbeit, 2003

[Sco02] SCOTT, W. R.: *Rational, Natural, and Open Systems (5th Edition)*. Prentice Hall, 2002

[Smi88] SMITH, R. G.: The contract net protocol: high-level communication and control in a distributed problem solver. (1988), S. 357–366

[Val95] VALK, R.: *Petri nets as dynamical objects*. 1995. – In G. Agha and F. D. Cindio (Eds.), Workshop Proc. 16th International Conf. on Application and Theory of Petri Nets, Torino, Italy.

[WEM06] WESTER-EBBINGHAUS, Matthias ; MOLDT, Daniel: *Auf dem Weg zu organisationsorientierter Softwareentwicklung*. Available at: http://www.informatik.uni-hamburg.de/TGI/publikationen/public/data/2006/Wester+06/Wester+06.pdf. `http://www.informatik.uni-hamburg.de/TGI/publikationen/public/data/2006/Wester+06/Wester+06.pdf`. Version: September 2006

[WJ95] WOOLDRIDGE, M. ; JENNINGS, N. R.: Intelligent Agents: Theory and Practice. In: *Knowledge Engineering Review* 10 (1995), Nr. 200, S. 115–152

[WJ99] WOOLDRIDGE, Michael ; JENNINGS, Nicholas R.: The Cooperative Problem-solving Process. In: *Journal of Logic and Computation* 9 (1999), Nr. 4, S. 563–592

[Woo94] WOOLDRIDGE, M.: Coherent social action. In: COHN, Anthony G. (Hrsg.): *Proceedings of the Eleventh European Conference on Artificial Intelligence (ECAI-94)*. Amsterdam, The Netherlands : John Wiley & Sons, 1994, S. 279–283

[Woo95] WOOLDRIDGE, M.: *Agents*. 1995. – http://www.cs.mu.oz.au/682/wooldridge.pdf

[Woo01] WOOLDRIDGE, Michael J.: *Introduction to Multiagent Systems*. New York, NY, USA : John Wiley & Sons, Inc., 2001

[XML07] XML: *Extensible Markup Language*. 2007. – http://www.w3.org/XML/