State-Based Modeling Method for Multiagent Conversation Protocols and Decision Activities

Ralf König

Department of Process Automation Techniques
Technical University of Hamburg-Harburg
21073 Hamburg, Germany
(+49) 40 42878 – 3547 (– 2493 for fax)
r.koenig@tu-harburg.de

Abstract In this paper an approach to model conversations and agents decision activities by using state-based models is presented. These models are parts of a layered architecture which separates the aspects regarding the communication from the decision activities of the agents. By this, application independent conversation protocols are defined. The application dependent parts of the model are formed by a sequence of activities an agent has to perform to solve a problem. The different layers of the architecture are separated from each other by well-defined interfaces.

Keywords: decision activity, conversation, conversation specification, multiagent communication

1. Introduction

Research in multiagent communication is still an ongoing process. Topics of this research are communication languages, either for semantics of messages as well as for representation of contents, representation of communication patterns or coordination descriptions, to name only a few. In the area of agent communication languages (ACL), speech act based languages (like KQML) have shown, that the basic idea to adopt the speech act theory [20] to model multiagent communication is seminal. Using these speech act typed messages as "building blocks" for multiagent communication, the structure of message exchange between two or more agents (termed as *conversation*) can be specified. A good definition of *conversation policy* resp. *conversation* can be found in [9, p. 123]

> Any public declaratively-specified principle that constraints the nature and exchange of semantically coherent ACL messages between agents can be considered a conversation policy.

For these specifications of conversations different approaches have been made. Depending on the focus of the approaches, different representations are used, including formal grammars, petri nets, temporal logic, state machines, state transition diagrams, to name only a few. Up to now there is no agreement among researchers

which is the best way to define specifications for conversations. In this paper an approach is presented to model conversations as well as the agent internal decision activities by using state-based models. The approach bases on a layered architecture build upon a speech act based ACL for defining conversation specifications between agents (on the lower level), for defining communication services using these conversations (at the medium level) and for defining the sequence of the activities of the agent solving a problem (on the upper level). Doing this, the agent uses the communication services provided by the medium level. Thus communication is a means which gives the agent the opportunity to reach a goal by making use of help of other agents.

This approach aims to form the base for a framework, which helps the designer of agents to develop cooperating multiagent systems by using the communication services provided by the framework. As a means for modeling the well known notation of state charts defined in the UML is used.

2. Fundamentals

2.1. Speech acts and conversations

The speech act theory forms one of the fundaments of research in multiagent communication today. A speech act is an utterance which is pronounced by a (human) speaker to cause a reaction at the (human) hearer. The utterances are classified by their types, e. g. question, assertion, request [20].

Concerning computer sciences the speech act theory gave the motivation to define multiagent languages like the KQML (currently the most used ACL [7, p. 3]) and FIPA-ACL. To be not determined to one ACL, only the type of the speech act will be declared in the following.

This paper follows the definition made in [25, p. 80] saying that a conversation is a structured exchange of messages. For the following explanation it is assumed that only two agents are involved in the conversation.

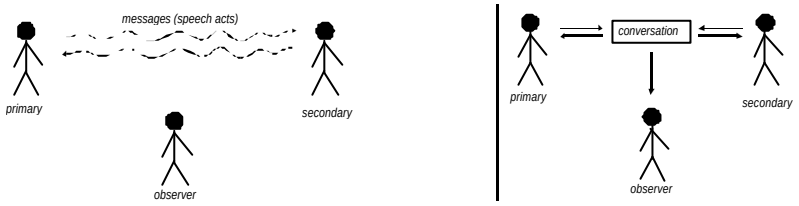


Fig. 1: Perspectives for modeling a conversation

The conversation can be modeled from three different viewpoints. The first one is the viewpoint of the agent who starts the conversation (it will be called the “primary”). An other viewpoint is the one of the agent who is addressed by the primary (the

"secondary"). As a third party the observer is introduced (Fig. 1). It is not included in the conversation.

Thinking of a spoken conversation between humans, the primary and the secondary play the roles of speaker and hearer whereas the observer is still a hearer. This means that primary and secondary are making inputs and outputs to the conversation whereas the observer only gets the outputs.

2.2. Related Work

In the last years a lot of research in modeling conversation policies has been done. These works can be separated into two main directions [26, p. 384]. One direction understands conversations between agents as a process emergent from within the society, which can not be defined by hard-wired social laws (like fixed protocols). The other direction (termed as the *off-line approach*) understands a conversation as a predefined process, which can be ruled by a fixed protocol used by the participating agents. This paper follows the off-line approach in which three different representations have been used mainly ([21, p. 155]). The most significant are state transition diagrams (finite state machines) and petri nets. Beside this the Dooley Graph (as well as the Enhanced Dooley Graph) can be found as a state-based representation for conversations [21, p. 155].

In a petri net, the places can be marked by a number of identical tokens. Colored petri nets (CPN) and predicate transition nets (Pr/T-Nets) are extended petri nets using individualized tokens. Approaches using petri nets are made from von Martial (Pr/T-Nets) [15], Fallah-Seghrouchni et al. (CPN) [8], Cost et al. (CPN) [5] and Lin et al. (CPN) [14].

In this paper, state transition diagrams (STD) will be separated from finite state machines (FSM) in the manner that a STD models the states of a system and the transitions between them. These transitions can occur of a multitude of reasons which can have their causes inside or outside the modeled system. Following the idea of Winograd and Flores to model a conversation as a state-based process, the following approaches base on state transition diagrams (STD): Winograd and Flores [27], Belakhdar and Ayel [3], Bradshaw et al. [4], Greaves et al. [9] and Pitt and Mandami [19]. An extension of STD, the state charts [10], were used by Moore [17].

In contrast, for a FSM it is claimed that transitions only occur in case of an input (represented by an input symbol) from outside the modeled system. Models using FSMs or extensions of them where presented by Barbuceanu and Fox (FSM) [1], Barbuceanu and Lo (FSM) [2], von Martial (FSM) [15], Martin et al. (PDT) [16], Nodine and Unruh (FSM) [18] and Wagner et al. (extended FSM) [24].

Comparing these approaches, it can be seen that all approaches modeling conversations from the viewpoint of an observer are using either STD or petri nets, whereas the approaches using FSM (or state charts) are representing the conversation from the viewpoint of a participating agent. For modeling a conversation from the point of view of a participating agent who receives and sends messages, a model supporting input and output operations is more suitable, because the interaction between the agents can be expressed by these inputs and outputs. When a conversation should be modeled from an overall view (the observer's view), it is

sufficient to use a model which is able to express, that a message has been transmitted from one agent to another, like a transition in a STD or in a petri net.

In contrast to STD, petri nets are able to express concurrency, which is important if the overall system behavior (including the agents internal decisions) is modeled. Models of this kind can be found in [14] for instance.

Most of the approaches focus on modeling the communication between two agents, without providing the opportunity to model communication among a greater number of participants. Furthermore, only a few approaches include the agents internal activities (the decision activities) in the model.

The approach presented in this paper models the decision activities of the agents as well as the communication between one agent and a group of agents. It bases on state-based modeled components which interact by inputting / outputting symbols (see Chap. 3.1 and 3.5 for details). The components are modeled by STD. The use of interacting, state-based modeled components ensures that these model components can easily be implemented as objects, which is helpful for the development of a software framework using this model.

2.3. Protocol design

The presented approach uses the conversation as one layer of an architecture for agent communication. As agents are (at least from the authors point of view) software programs running on computers, it is helpful to take a look at the protocol design of communication protocols for computer networks.

Communication between computers is regulated by protocols which are organized in different layers (protocol stack). A protocol is a collection of rules and conventions defining how a dialogue has to take place [23, p. 33]. The ISO-Model defines a protocol layer as two connected protocol machines.

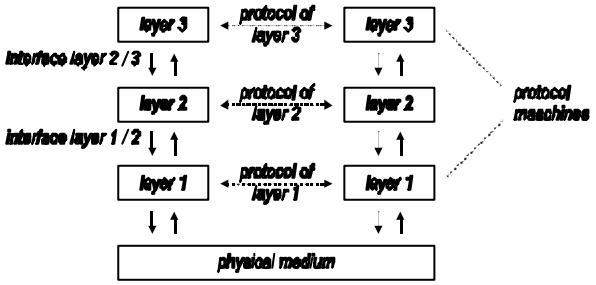


Fig. 2: Protocol stack

Each protocol machine consists of two interfaces, one to provide services to the user of the protocol (*user interface*), e. g. the next higher architecture layer, the other for sending and receiving data to / from the lower architecture layer. To simplify the understanding it is often assumed, that the protocol machine communicates directly with its peer (using its *peer interface*) by omitting the protocols on the lower levels (indicated by the dashed lines in Fig 2).

To avoid confusion, this paper distinguishes between the *protocol* and the *protocol machine*. Whereas the protocol defines the way how the communication takes place, the protocol machine is the realization of the protocol which can be used to transfer data between two computers or agents.

3. State based modeling method

This chapter presents a state-based model for multiagent communication. The first section introduces the underlying architecture and it illustrates the purpose of its layers in short. The following sections describe the layers in depth. Chap. 3.6 describes roughly the more complex contract net protocol and Chap. 3.7 dwells on the framework implemented by using this model.

3.1. Structure of the layered architecture

In the layered architecture, communication services (which are defined in the lower part of the hierarchy) are used by the decision activities of the agent, which are forming the upper part of the hierarchy.

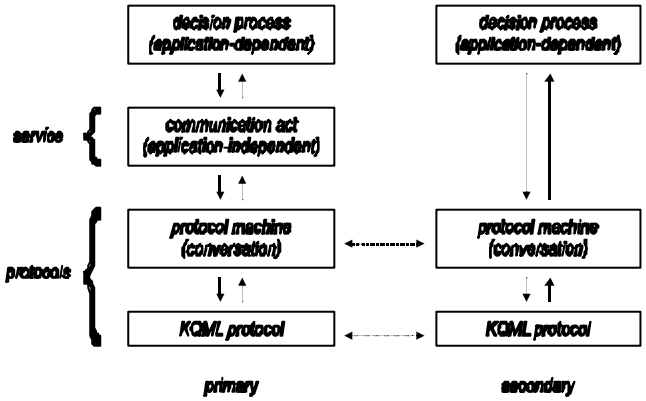


Fig. 3: Architecture

These decision activities will be termed as decision processes. A *decision process* consists of a number of activities (time consuming acts) performed by the agent to solve a problem. Hence the state of a decision process is defined by the execution of an activity. The communication service is defined in a *communication act*. Such an act is a means to serve a purpose, making use of the help of other agents. The message exchange among the agents is organized by making use of conversation policies, defined in a conversation protocol.

Considering the protocol stack the decision process uses the communication act which makes use of the protocol machine (which implements the conversation protocol). Underneath the conversation, the KQML protocol and transport protocols of the network are placed. Whereas the conversation policy and KQML are protocols

(because either on the primary as well as on the secondary side a layer exists) the communication act is no protocol, because there exists no corresponding layer on the secondary's side. The separation between the decision process and the communication act is made, to have the opportunity to define communication acts independently from any application.

To summarize, every protocol stack layer is seen as a service provider to the upper layer. Because the services on an upper layer are build upon the services of the lower layer, these lower layers form "building blocks" for the upper ones. As an example, a conversation consists of a set of KQML-performatives. A communication act consists of conversations and so on. On the upper level of this hierarchy, the decision process makes use of communication acts to fulfill its task.

This approach separates the aspects concerning the communication (which are independent from a specific application) from the aspects regarding the decision activities of the agent (which are dependent from the application). By this, application-independent conversation protocols and communication acts can be defined and modeled in a framework. Using this framework, the agent developer only has to model the (application-dependent) decision activities of the agent as decision processes (which extends predefined patterns (skeletons)) and can make use of the framework to give the agents the ability to communicate.

The remainder of this section will show how to define conversation protocols, protocol machines, communication acts and decision processes.

3.2. Conversation protocols and protocol machines

In the following, the conversation will be modeled as a FSM from the point of view of the observer. This FSM will serve as a base to represent the viewpoint of two participating agents. The resulting two FSM define the protocol machines (primary / secondary). A conversation is defined as a structured exchange of speech act typed messages between two agents. Furthermore the conversation is restricted to be synchronous, that means that the roles of speaker and hearer are defined for every message. It is not allowed, that in one state both agents are speakers, for instance.

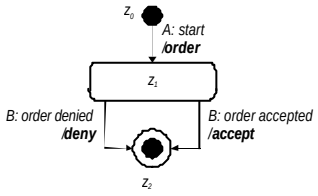


Fig. 4: order-protocol

Looking from the viewpoint of the observer, the heard speech acts are outputs of the conversation. Thus they form the output alphabet of the FSM. As an example, a simple kind of conversation is used. In this conversation the primary gives the order to perform a task to the secondary which answers with a denial or an acceptance (Fig. 4). Depending on the result of an agents decision, the conversation chooses one of the transitions. As a result, the agents decisions serve as inputs to the conversation. In

state z_1 the secondary (agent B) decides to fulfill the order (indicated by the event *order accepted*) or to deny it (indicated by *order denied*). Depending on this decision, the appropriate speech act type is selected.

After defining the conversation policy from the viewpoint of the observer, the protocol machines (modeled by FSMs) for the primary and the secondary are created. The conversation protocol is a means to transform the agents decisions into speech act typed messages. When an agent inputs its decisions into the conversation, the protocol machines convert them into speech act typed messages (on the senders side) and reconvert them from speech act typed message into agents decisions (on the receivers side). This is shown in Fig. 5.

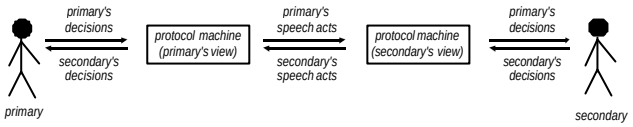


Fig. 5: Interaction of protocol machines and agents decisions

Both protocol machines consist of all states defined in the conversation protocol from the observers perspective. Every time the agent got control over the conversation, its decision result forms the input for its protocol machine. If the peer agent got the control, the speech act type of its message is used as input. In this conversation, the control over the conversation changes with every utterance from one agent to the other.

To clarify this, the order-protocol (defined in Fig. 4) is regarded. The protocol machines for the participating agents are created by changing transition labels. Fig. 6 shows the protocol machine for the primary, Fig. 7 the one for the secondary.

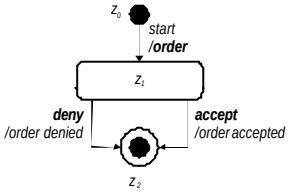


Fig. 6: Primary's protocol machine

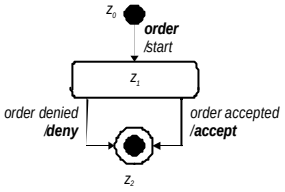


Fig. 7: Secondary's protocol machine

The primary starts the conversation by the decision result *start*. Its protocol machine changes to z_1 by outputting an *order*-typed message. This message type serves as the input of the secondary protocol machine which changes from z_0 to z_1 by outputting the decision result of the primary agent (*start*) to the secondary agent. Depending on its decision regarding the order (remember that the message got a content, in which the secondary agent was told to do something) either an *order accepted* or an *order denied* is input into the secondary protocol machine which changes to z_2 (final state) by outputting either an *accept*-typed or a *deny*-typed message. The primary protocol machine receives the message via its peer interface and moves to z_2 either by outputting *order accepted* or *order denied* depending on the speech act type of the received message.

3.3. Communication acts

As the conversation describes the message exchange between two agents, an agent needs to be able to communicate with more than one agent. For that a "component" is needed which manages several conversations.

This component is termed *communication act*. In general a communication act fulfills a special purpose by making use of a conversation policy. Examples for a communication act are the answer to a question by a number of agents, the assignment of a task to a number of agents or to exactly one agent out of a group of agents. Please note that a communication act is not a protocol because there exists no peer layer at the peer agent (see Fig. 3).

This part of this section defines the communication act for the order-protocol defined in Fig. 4. Such a communication act can be seen as a general schema of using a protocol when communicating with several partners. Considering the order protocol, the following situation can be assumed.

Agent A needs to assign a task to other agents by using the order-protocol. As the agent A can not be sure that the first agent it gives the order to execute the task is able to fulfill it, it has to continue giving orders until one agent states that it is able to fulfill the task. On the other hand, if agent A would give the order to all agents at the same time, possibly more than one agent would fulfill it and execute the task. But only one agent should fulfill it. Hence agent A needs to give orders to the agents one by one, until an agent reports that the order can be fulfilled by executing the task in question.

The communication act modeled as a STD is shown in Fig. 8. It interacts with more than one object (a decision process and a set of conversations). Therefore the events and actions are extended by a prefix defining the affected object. The prefix *dec* stands for the decision process, *conv* for a conversation.

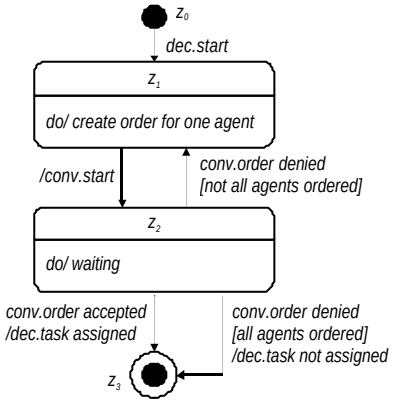


Fig. 8: Communication act (order-protocol)

After the occurrence of the event *start* (*dec.start* means that the event was caused by the action *start* from the decision process *dec*) the act changes to z_1, creates the order for the first agent and moves to z_2 by performing the action *start* which affects the conversation (*conv.start*). The act waits in z_2 until an external event, caused by the

conversation, occurs. In case of the event *order accepted* the communication act moves to z_3 (final state) by performing the action (outputting) *task assigned* to the decision process. In the other case (event *order denied*) the act moves to z_1 or to z_3. When all available agents have been addressed (ordered), the act moves to z_3 by outputting *task not assigned* to the decision process, otherwise to z_1 without producing an output.

The communication act doesn't matter what kind of task is assigned, it is independent from the message content and the application. It describes only how conversations all using the order-protocol have to be managed, to make sure that one task is assigned to one agent out of a group of several agents.

3.4. Decision Processes

The highest layer of the architecture is formed by the application dependent decision process. The decision process is modeled by a STD. In this chapter only the events, actions and essential states of the process are mentioned. These "skeletons" will be expanded in Chap. 4.2. to application specific decision processes for planning agents.

Like shown in Fig. 3 different decision processes for primary and secondary agent are needed. The decision process on the primary's side interacts with a communication act, the one on the secondary's side with a protocol machine (conversation).

Starting with the decision process on the primary's side, Fig. 9 is shown. This process interacts with the communication act, indicated by the prefix *comm* at the events and actions.

After creating the task to assign, the process moves to z_2 (*waiting*) by starting the communication act with the action *start*. Depending on the behavior of the addressed agents (whether they fulfill the order or not) the communication act causes either the event *task assigned* or *task not assigned*, which lets the decision process continue to z_3 (in case of a successful assignment) or z_4 (in case of failure) to execute activities mentioning the success of the assignment. The process finishes in z_5.

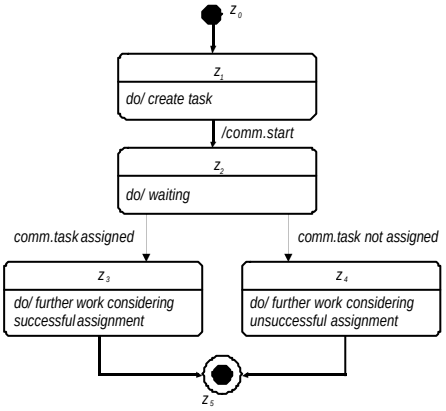


Fig. 9: Decision process on the primary side

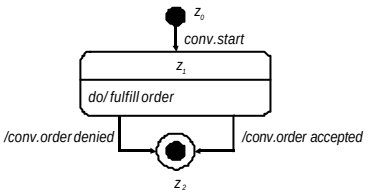


Fig. 10: Decision process on the secondary side

On the secondary side, the decision process interacts with the protocol machine of the conversation (indicated by the prefix *conv*). This process is shown in Fig. 10. The process is started by the event *start* caused by the conversation. In z_1 the process tries to fulfill the order and continues to z_2 by causing the events *order accepted* or *order denied*, depending whether the order could be fulfilled or not.

3.5. Interaction between layers

To explain the interaction between the layers of this architecture, an example is assumed. Considering the case that a secondary agent receives a message, a decision process is started to work on this message. Assuming that the agent needs the help of others, the decision process selects one or more appropriate communication acts which starts one or more conversations. Fig. 11 shows the resulting structure.

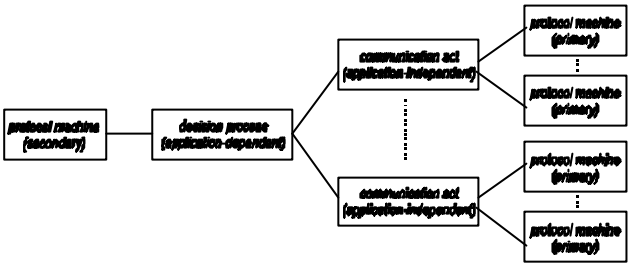


Fig. 11: Structure of protocol machines, decision processes and communication acts

3.6. Contract net protocol

With regard to the spare space a simpler protocol was used to be able to present the approach in detail. The intention of Chap. 3.1 – 3.5 was to explain the structure of the presented model for multiagent communication as well as the function of the different layers of the architecture.

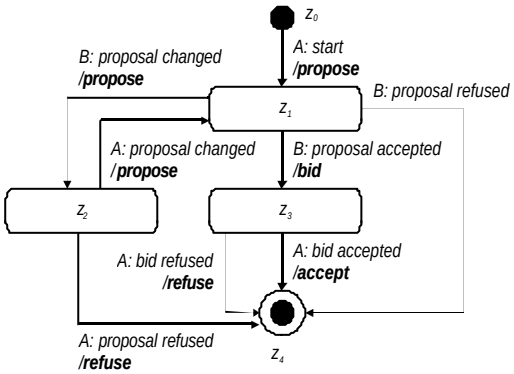


Fig. 12: Contract Net Protocol

By using this model, several protocols were modeled, including protocols for task assignment using the contract net approach [6], [22]. Fig. 12 shows the conversation protocol from the perspective of an observer.

Here the agent B is told to react on a proposal send by agent A. Receiving the proposal, agent B has in z_1 the opportunity to create a bid (*proposal accepted*), to vary the proposal by sending a counterproposal (*proposal changed*) or simply to leave the conversation by doing nothing (*proposal refused*). In case of a bid send by agent B (conversation is in z_3), agent A can accept or refuse the bid. In case of a proposal send by agent B, the conversation moves to z_2, where agent A can refuse the proposal or vary it by sending a counterproposal.

3.7. About the framework

The model presented in this paper has been implemented in a framework using the object oriented language Smalltalk (VisualWorks 2.5.2). The framework provides communication services (communication act, protocol machines for primary and secondary) for task assignment and for information seeking. Due to its structure, the framework can easily be extended by further communication services. By this a tool for designing application independent communication services is provided.

4. Application example

The developed framework was used to build up an agent-based distributed planning system for production planning. The structure of the planning system as well as the types of agents used in it are explained in Chap. 4.1. Chap. 4.2 clarifies by an example how to extract the skeletons of the primary and secondary decision process (defined in Fig. 9 and Fig. 10) to application dependent decision processes.

4.1. Short description of the planning system

The planning system consists of different types of agents, two of them will be described in detail. The first one is responsible for planning the manufacturing process of products (*product planner*), the second one for creating schedules of workstations (*workstation planner*). Fig. 13 shows the structure of an example planning system containing two product planners and three workstation planners. The arcs connecting the agents show the communication structure. Two connected agents communicate with each other while solving a problem.

The planners can be seen as service providers able to perform services by executing processes on workstations or by producing products. In its planning process a planner solves its planning task by working on it up to the point where it needs the help (information or resources) of other planners. By using communication, the planner requests for service, which is given by the fulfillment of tasks. This help enables the planner to fulfill its own task successfully.

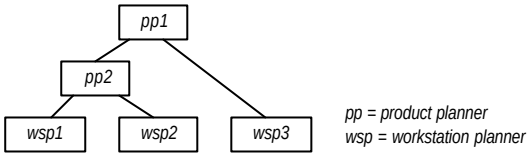


Fig. 13: Structure of an example multiagent system

The product planner conducts conversations with other product planners or with workstation planners (like *pp1* in Fig. 13). Every product planner needs the help of workstation planners to fulfill its task. In contrast to, workstation planners are able to fulfill their tasks without the help of other agents. Therefore, a workstation planner still acts as a secondary in a conversation because every time it is involved in a communication, a product planner asks it for its help. A product planner can be either a primary as well as a secondary in a conversation, because it can be asked by an other product planner for its help (secondary) or it can ask other planners for their help (primary).

In the following example in Chap. 4.2, the skeletons of the decision processes for the order-protocol (defined in Fig. 9 and Fig. 10) will be expanded to an application specific decision process.

4.2. Decision process for planning agent

The decision process belongs to the product planner. This agent receives messages telling it to manufacture products in a specified quantity and time window. Its decision process is shown in Fig. 14.

The agent acts as a secondary. Solving its problem, it has to conduct conversations, which makes it also to a primary. The decision process expands the skeleton of the secondary decision process (Fig. 10). In z_1 the agent acts as a primary, so within z_1 the skeleton of a primary decision process (Fig. 9) is expanded.

In case of an event *start* from the conversation, the decision process moves to z_1 and onwards to $z_{1,1,1}$ to define the subtasks which are needed to manufacture the product in question. A subtask is either a product order for a subproduct, needed to manufacture the product or a process order for a process, needed to be processed on a workstation. For these subtasks planners have to be found, which are able to fulfill these tasks. If the product planner wasn't able to specify all needed subtasks or if it couldn't find at least one agent for each subtask, the decision process leaves to $z_{1,1,3}$. Otherwise the decision process moves to $z_{1,1,2}$ and creates one communication act (described in Fig. 8) for each subtask. After leaving $z_{1,1}$ the process starts the created communication acts by the action *start* and enters $z_{1,2}$ to wait in $z_{1,2,1}$ until all communication acts have been finished.

Depending whether all subtasks were assigned, the process moves to $z_{1,3}$ or $z_{1,4}$. In $z_{1,3}$ the process stores the assigned subtasks and the agents responsible for them. If not all subtasks could be assigned, the process withdraws made assignments in $z_{1,4}$. By leaving z_1, the process moves to z_2 by outputting *order accepted*, if the product could be manufactured indicated by assigning all needed subtasks to other agents. Otherwise *order denied* is output.

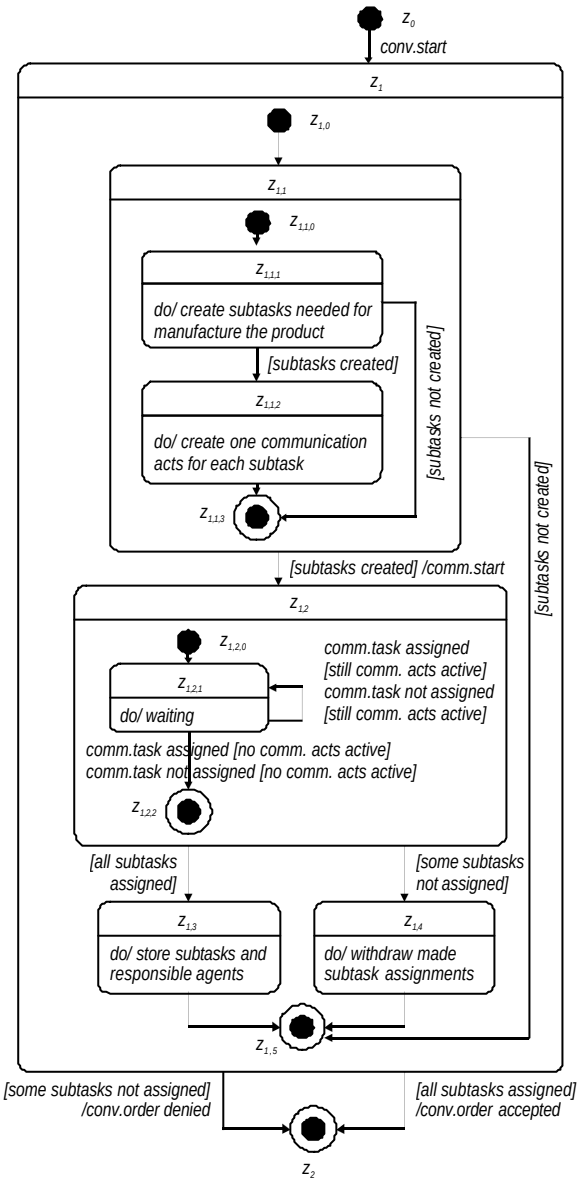


Fig. 14: Decision process of product planner

5. Conclusion

This paper presented an approach making use of a layered architecture, which ensures the reusability of the defined protocols and the communication acts by defining a clear separation between the application-independent and the application-dependent parts. The approach models conversation policies by making use of finite state machines. These policies were transformed into protocol machines, which provides communication services to agents. The modeled conversation protocols are application-independent as well as the communication acts, which are a mean for agents to fulfill their goals by making use of communication. The sequence of activities an agent has to perform to reach its goal (which is a state-based process), was called a decision process. It was modeled by a state transition diagram. The conversation, the communication act and the decision process are parts of a layered architecture. The layers are separated by well-defined interfaces. The interaction between the layers is modeled by the exchange of input/output symbols between the participating layers.

6. Acknowledgements

The author thanks the anonymous reviewers for their thoughtful comments and suggestions to improve this paper.

7. References

1. Barbuceanu, M.; Fox, M. S.; COOL: A Language for Describing Coordination in Multi Agent Systems; in: First International Conference on Multi-Agent Systems – ICMAS ’95; pp. 17 – 24; The MIT Press, California, 1995
2. Barbuceanu, M.; Lo, W.-K.; Conversation Oriented Programming for Agent Interaction; in: [7], pp. 220 – 234
3. Belakhdar, O.; Ayel, J.; Modelling Approach and Tool for Designing Protocols for Automated Cooperation in Multi-agent Systems; in: Agents Breaking Away (MAAMAW ’96), LNAI 1038; Van de Velde, W.; Perram, J. W.; pp. 100 – 115; Springer, Berlin, 1999
4. Bradshaw, J. M.; Dutfield, S.; Benoit, P.; Woolley, J. D.; KAoS: Toward An Industrial-Strength Open Agent Architecture; in: Software Agents; pp. 375 Bradshaw, J. M.; The MIT Press, California, 1997
5. Cost, R. S.; Chen, Y.; Finin, T.; Labrou, Y.; Peng, Y.; Using Colored Petri Nets for Conversation Modeling; in: [7], pp. 178 – 192
6. Davis, R.; Smith, R. G.; Negotiation as a Metaphor for Distributed Problem Solving; in: Artificial Intelligence, Vol. 20, pp. 63 – 109; Elsevier Science Publishers, North, 1983
7. Dignum, F.; Greaves, M.; Issues in Agent Communication, LNAI 1916; Springer, Berlin, 2000

8. Fallah-Seghrouchni, A. E.; Haddad, S.; Mazouzi, H.; Protocol Engineering for Multi-agent Interaction; in: Multi-Agent System Engineering (MAAMAW '99), LNAI 1647; Garijo, F. J.; Boman, M.; pp. 89 – 101; Springer, Berlin, 1999
9. Greaves, M.; Holmback, H.; Bradshaw, J.; What is a Conversation Policy?; in: [7], pp. 118 – 131
10. Harel, D.; Statecharts: A visual formalism for complex systems; in: Science of Computer Programming, Vol. 8, pp. 231 – 274; Elsevier Science Publishers, North, 1987
11. Holzmann, G. J.; Design and validation of computer protocols; Prentice Hall, New Jersey, 1991
12. Jennings, R. J.; Woolridge, M. J.; Agent Technology, Foundations, Applications and Markets; Springer, Berlin, 1998
13. König, R.; Modeling Conversations and Decision Processes by using state-based models; Technical Report ET8-2002-01; Department of Process Automation Techniques; Technical University of Hamburg-Harburg, Hamburg, 2002
14. Lin, F.; Norrie, D. H.; Shen, W.; Kremer, R.; A Schema-Based Approach to Specifying Conversation Policies; in: [7], pp. 193 – 204
15. von Martial, F.; Coordinating Plans of Autonomous Agents, LNAI 610; Springer, Berlin, 1992
16. Martin, F. J.; Plaza, E.; Rodríguez-Aguilar, J. A.; Conversation Protocols: Modeling and Implementing Conversations in Agent-Based Systems; in: [7], pp. 249 – 263
17. Moore, S. A.; On Conversation Policies and the Need for Exceptions; in: [7], pp. 114 – 159
18. Nodine, M. H.; Unruh, A.; Constructing Robust Conversation Policies in Dynamic Agent Communities; in: [7], pp. 205 – 219
19. Pitt, J.; Mamdani, A.; Communication Protocols in Multi-agent Systems: A Development Method and Reference Architecture; in: [7], pp. 160 – 177
20. Searle, J. R.; Speech Acts; Cambridge University Press; Cambridge, 1969
21. Shen, W.; Norrie, D. H.; Barthès, J.-P., A.; Multi-agent systems for concurrent intelligent design and manufacturing; Taylor & Francis, London, 2001
22. Smith, R. G.; The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver; in: IEEE Transactions on computers, Vol. C-29, No. 12, pp. 1104 – 1113, 1980
23. Tanenbaum, A. S.; Computernetzwerke; Pearson Studium, München, 2000
24. Wagner, Th.; Benyo, B.; Lesser, V.; Xuan, P.; Investigating Interactions between Agent Conversations and Agent Control Components; in: [7], pp. 314 – 330
25. Weiss, G.; Multiagent Systems, A Modern Approach to Distributed Artificial Intelligence; The MIT Press, Cambridge, 1999
26. Walker, A.; Wooldridge, M.; Understanding the Emergence of Conventions in Multi-Agent System; in: First International Conference on Multi-Agent Systems – ICMAS '95; pp. 384 – 389; The MIT Press, California, 1995
27. Winograd, T.; Flores, F.; Understanding Computers and Cognition: A New Foundation for Design; Ablex Publishing Corp., New Jersey, 1986