A DESIGN TOOL FOR AUTONOMOUS GROUP AGENTS

Thomas Kreifelts, Frank Victor,
Gerd Woetzel, Michael Woitass

Gesellschaft für Mathematik und Datenverarbeitung (GMD)
P. O. B. 1240
5205 St. Augustin 1, West Germany

ABSTRACT

Autonomous group agents coordinate cooperative activities in spatially distributed groups of people who communicate via electronic mail. The highly asynchronous nature of remote communication poses a problem for the design of such programmed agents. This paper describes a tool supporting the consistent design of the communication behaviour of autonomous group agents. The tool will support the designer in detecting possible deadlocks, inconsistent terminations, or unforeseen situations. The approach chosen is based on a conversational model of cooperation. Essential parts of the tool have been successfully implemented in Prolog.

1. Introduction

Autonomous group agents serve the purpose of coordinating cooperative activities in a group of people who communicate via electronic mail. Examples of such cooperative activities are the distribution of information, the monitoring of office procedures, or the scheduling of meetings, to name a few. Autonomous group agents are fully programmed agents which are accessible within an electronic mail network. The implementation of such an agent which mediates and coordinates off-line cooperation requires a careful design in order to avoid inconsistencies or breakdowns of the cooperation. Ideally, the resulting specification should be easily understandable by the intended users of the agent.

In the following, we propose a model and an according tool to support the design of autonomous group agents. The model allows for a formal specification of the communication behaviour of the agent. The tool is able to check this specification for completeness, consistency, unforeseen situations and possible deadlocks. At the user level, the communication behaviour is described as reactions to certain events like the arrival of messages or the occurence of certain states via production rules.

2. Conversational systems: A model of cooperation

The modeling approach which underlies a lot of cooperation support systems conceives of cooperation as a formalized language game governed by certain rules [Flores & Ludlow 1980, Winograd & Flores 1986, De Cindio et al. 1986], or similarly, as a formalized exchange of messages governed by certain protocols [Holt & Cashman 1981, Sluizer & Cashman 1984]. The monitoring facilities of the office procedure system DOMINO [Kreifelts & Woetzel 1987] also fall within this cooperation paradigm. Building on the experiences with DOMINO, we have developed the conversational systems model [Kreifelts & Woetzel 1988] which is intended to serve as a basis for various group support applications. We have included autonomous group agents into this model, and have called them *mediators*. Before we can describe the role of mediators within the conversational systems model in more detail, we have to give a short account of the basic concepts of the model.

A common aspect of cooperative activities is the use of language to advance the activity towards its goal. The speaker of an utterance intends to achieve a certain effect with the listener. He wants the listener e.g. to do something now, to commit himself to do something in the future, to accept or refuse the result of an activity, to view the state of a cooperation in a certain way (e.g. as successfully completed, or failed) etc. The recurring patterns of action which form a cooperative activity can thus be viewed as a conversational game, where the moves are exchanges of messages, and the players are the participants in an activity, filling certain roles with respect to the cooperation. The role-players may send or receive certain types of messages, depending on the situation or state they are in. Sending and receiving messages changes the state of the role-players.

For different forms of cooperation there are different conversation types. A conversation type can be adequately described by its message types, roles, role states, and the rules which govern the message exchange between the roles. We can have conversation types of a more generic nature, e.g. the "conversation for action" [Winograd & Flores 1986, p. 65], or similarly, the action conversation of the DOMINO system [Kreifelts & Woetzel 1987] with rather general message types (request, promise, rejection, counter offer, cancellation etc.). But we can also have conversation types more tailored to a specific application, e.g. a meeting scheduling conversation with specialized message types like date proposal and date change.

A conversation type consists of message types, roles and their possible states, and the respective conversation rules. For a concrete conversation, role-players are assigned to the roles of a conversation. The conversation rules describe in which state the role-players may receive or send which types of messages and which consequences the reception or dispatch of these messages will have on their state. The content and the internal structure of a message do not occur in conversation rules, only the type of a message.

Various formalisms for the representation of conversations have been suggested, e.g. the state transition diagrams of [Winograd & Flores 1986] or the state graphs for the protocols of [Holt & Cashman 1981]. These representations serve as specifications of what exactly can happen in a given conversation and, additionally, as a means for the analysis of conversations with regard to possible breakdowns. Note that there is a bigger potential for breakdowns in conversations mediated by electronic mail than in face-to-face conversations because there is no direct feedback for the immediate correction of misinterpretations, and not even the temporal order of utterances is necessarily preserved by the medium.

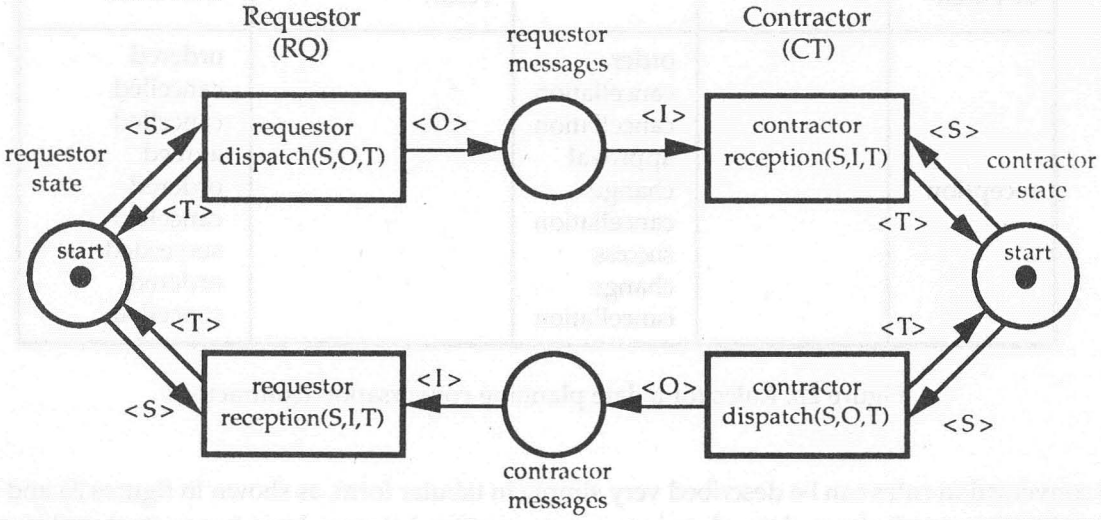


Figure 1. Conversation net.

We have suggested predicate/transition nets as a formalism for specifying conversations [Woetzel & Kreifelts 1989]. Predicate/transition nets (Pr/T nets) [Genrich & Lautenbach 1981, Reisig 1983] are a kind of high-level Petri nets with individualized tokens and according firing rules for the transitions. Figure 1 shows the conversation net. It has two places representing the states of the two partners, one opening up the conversation (generic role name: requestor), the other being the addressee of the opening message (generic role name: contractor). The other two places represent channels for message transmission. Reception and dispatch of messages is modeled by the respective transitions for each partner (actually, an additional synchronization transition is necessary for the immediate reaction to incoming messages because of the asynchronous nature of conversations in store-and-forward message handling systems). The conversation rules of a certain conversation type are represented by the inscriptions (firing rules) of the transitions which determine a possible dispatch or reception of a message and the state changes connected with such an event.

transition	state (S)	input (I) message	output (O) message	successor (T) state
dispatch	start		order	ordered
	ordered		cancellation	cancelled
	submitted		approval	agreed
	submitted		change	ordered
	submitted		cancellation	cancelled
	agreed		success	succeeded
	agreed		change	ordered
	agreed		cancellation	cancelled
reception	ordered	result		submitted

Figure 2a. Rules for a date planning conversation (requestor).

transition		input (I) message	output (O) message	successor (T) state
dispatch			result	submitted
reception		order		ordered
		cancellation		cancelled
		cancellation		cancelled
		approval		agreed
		change		ordered
		cancellation		cancelled
		success		succeeded
		change		ordered
		cancellation		cancelled

Figure 2b. Rules for a date planning conversation (contractor).

The conversation rules can be described very simply in tabular form, as shown in figures 2a and 2b for a simplified example from date planning: a conversation between boss (requestor) and secretary (contractor) concerning the scheduling of a meeting. The tables are nearly symmetric, i.e. can almost be generated from one another by exchanging dispatch for reception and input for output. The only exception is an extra entry for the contractor covering the case where a subsequent cancellation

overtakes the opening order and arrives at the contractor's site in start state. The relation between requestor and contractor table are not necessarily that simple. The conversations which the secretary has to conduct with the prospective participants in order to find an agreeable date for the meeting are not covered by the above tables. These conversations belong to a different type with different rules.

The representation of conversations as Pr/T nets avoids assuming a global state for a conversation, and it also catches the dynamic aspect of (asynchronous) conversations: the exchange of messages in a conversation corresponds to the token game in the conversation net of figure 1. We have used a Pr/T net simulator to prove certain important properties of conversation types, like deadlock freeness and consistent termination.

3. Mediators

So far we have assumed conversations to be bilateral, i.e. to involve only two people in the role of requestor and contractor. Most cooperative activities, however, include more than two people, so the bilateral conversation model can only cover parts of such activities. As a possible way out of this situation, one could try to extend the conversational model in order to include multilateral conversations as well. However, the resulting nets and tables tend to be rather complex, and the advantage of the simple conversational model is lost. Therefore we have decided to model multilateral cooperative activities as a set of bilateral conversations coordinated by an autonomous group agent, which we call a *mediator*. Mediators are completely programmed agents which participate in conversations and also can initiate new conversations. They carry out certain tasks in a conversational context on behalf of a whole group or organization, where the responsibility is with the group or organization. The task and the behaviour of a mediator is known to, and agreed upon by, the other participants in the cooperative activity.

A simple example of a mediator is a group representative which distributes messages addressed to a group according to interest and competence of the group members. A more complex example which we will keep to for the rest of this paper is a date planning agent [Prinz & Woitass 1989], which to a certain extent would play the role of the secretary in the above example. This mediator tries to schedule meetings on behalf of an initiator in conversations with the prospective participants. Date planning is considered as a process of negotiation between the persons involved without disclosure of private calendars. In order to keep the example short, we have chosen a restricted functionality of the date planning mediator (we have excluded e.g. timeouts during the planning process). But the selection will still be sufficient for illustration purposes. The date planning agent operates as follows (message types italicized):

> The initiator ("requestor") gives an *order* to the date planning agent to start a *query* among the desired participants ("contractors").
>
> If the *answers* of all participants have arrived, they are evaluated and the *result* is presented to the initiator.
>
> The initiator now can approve or disapprove of the result, or change the original order.
>
> In case of *approval* the participants are informed of the date found and in response send a *receipt* message back to the mediator.
>
> In case of *cancellation* — the initiator can also cancel the whole process at other stages — the participants are informed and the date negotiation has definitely failed.
>
> In case of *change* which also can be issued after an agreement has been reached, another round of queries is started by the mediator.
>
> The process terminates when the initiator confirms the *success* of the meeting which is then propagated to all participants.

The mediator fulfils its task as a link between one person initiating a cooperative activity and a set of persons completing the activity. Apart from internal calculations and evaluations, the mediator

coordinates the communication with the initiator on the one side and the communication with the contractors on the other side, complying to the rules of the conversation types used. Thus, in a mediator system the multilateral relationship within the cooperating group is resolved into a series of bilateral conversations and the internal mediator rules. The mediator rules are not part of the conversation rules, they state how the mediator is to behave in the diverse conversational situations. In this way, the specification of the communication structures within a multilateral group activity is factorized into bilateral conversation rules and mediator rules. This leaves us with the problem of the mediator specification.

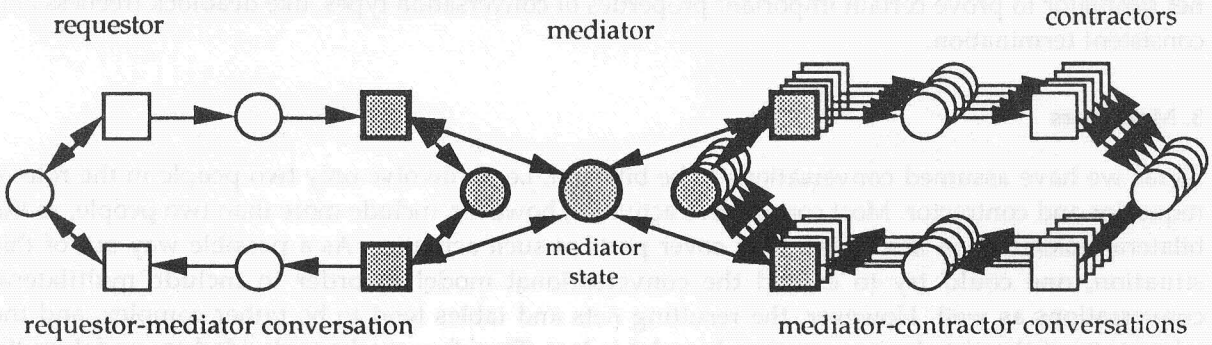


Figure 3. Mediator system in a net representation.

As a first formal model of mediator systems we have developed a Pr/T net mediator model which is shown in figure 3. The conversation nets of the mediator-contractor conversations are mapped ("folded") onto each other and coupled to the requestor-mediator conversation by introducing an additional mediator state. In this way, the complete mediator system can be formally specified. The mediator behaviour can be isolated from the complete system (shaded subnet of figure 3), and it is possible to investigate mediator and conversations separately, e.g. by formal methods of net theory or simulation. The drawback of this approach is that mediator rules in the form of net inscriptions are not as simple and easily understandable as the corresponding conversation rules. Therefore we have been looking for a more user oriented description of mediators. We have chosen a kind of production rules, much in the flavour of the activity specification of the AMIGO activity model [Pankoke-Babatz 1989]. A production rule approach for describing communication structures can also be found in [Bowers & Churcher 1988]. The content of our mediator rules will of course be tailored to the notions introduced in the conversational systems model, i.e. roles, message types and states.

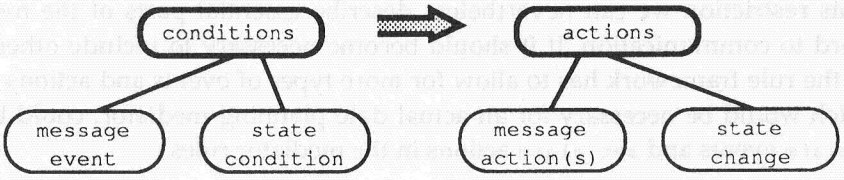
4. Mediator specification by production rules

When we analyze the role of a mediator in a conversational setting, we can distinguish two different task domains of a mediator: the problem specific and the communication specific task domain. In the date planning example, a problem specific task would be the calculation of the optimal date from answers given by the participants. The respective methods and algorithms, like the evaluation algorithm of a date planning mediator, are tailored to the special problems to solve. They differ from mediator to mediator, and we are not concerned with their specification in the context of this paper. The communication specific task of a mediator consists of the synchronization and coordination of the concurrent conversational activities it is engaged in. This task is common to all mediators, and it is the specification of this mediator task we are interested in.

In the date planning example given above, all contractors, i.e. the participants of a meeting, play the same role, so the mediator has to coordinate only two types of conversations, one with the initiator and the other with the participants, which is the *same* for all participants. We call this type of mediator a *1-to-n multiplexer*, and will consider only this type of mediator for the rest of

the discussion. The mediator's coordination task thereby is the synchronization of the bilateral conversations with one requestor on the one side and a number of (equal) contractors on the other.

The design of a mediator starts out from the two conversation types for requestor and contractors, i.e. the message types, possible states and state changes in these conversations. The mediator has to react adequately to arriving messages, so we think it natural to specify the communication behaviour of a mediator in form of production type rules with the notions of occuring events and subsequent actions. A mediator rule then has the general form:



In order to illustrate the use of rules for mediator specification, we take the example of the (simplified) date planning agent given above. The behaviour described in the last section may be specified by rules as follows.

```
R1: on_recv(order,requestor) =>
        send(query,allContractors).

R2: on_recv(cancellation,requestor) =>
        send(cancellation,allContractors).

R3: on_recv(approval,requestor) =>
        send(result,allContractors).

R4: on_recv(success,requestor) =>
        send(success,allContractors).

R5: on_recv(change,requestor) when(med_state = result_to_allContractors) =>
        set(med_state = changed).

R6: on_recv(change,requestor) when(med_state = agreed) =>
        send(query,allContractors).

R7: on_recv(answer,allContractors) =>
        send(result,requestor).

R8: on_recv(receipt,allContractors) when(med_state = result_to_allContractors) =>
        set(med_state = agreed).

R9: on_recv(receipt,allContractors) when(med_state = changed) =>
        send(query,allContractors).
```

These rules describe how the date planning mediator is to react to messages which may be received according to the conversation types used for the communication with initiator and participants (cf. figures 2a, 2b for the conversation with the initiator). In the above example the rules are rather straightforward. The only (mildly) complicated situation arises when the initiator sends a change message before all receipts of a former result have arrived from the participants. Accordingly, the mediator action in case of a change message depends on the current state as can be seen from the respective rules covering change and receipt messages (cf. rules R5, R6 and R8, R9).

The general idea with the rule based specification of mediators is that a set of mediator rules should cover the communication behaviour of the mediator in the sense that for the occurence of every possible message event (`on_recv`-construct), there is a rule which specifies the appropriate action. This action can optionally also depend on the current mediator state, as e.g. in rule R6. The

actions consist of sending messages to the requestor and/or the contractors, which changes the state of the mediator. If no immediate message action is possible, only the mediator state is changed in the action part of the rule (cf. rule R5). If the new state is not explicitly set in a rule (cf. rule R3), it can nevertheless be referenced in other rules following a naming convention: if the last action in the action part of a rule consists of sending a message of type `msgtype` to a `recipient`, the new mediator state is called `msgtype_to_recipient`. This convention is used in rules R5 and R8.

The only event type we have admitted for a mediator rule in the above example is the arrival of a conversational message, i.e. mediator action is triggered exclusively by the arrival of such messages. With this restriction we can nevertheless describe essential parts of the mediator's behaviour with regard to communication. If it should become necessary to include other trigger mechanisms as well, the rule framework has to allow for more types of events and actions. A clock mechanism, e.g., which would be necessary for an actual date planning mediator, could be integrated by admitting `on_alarm` events and `set_alarm` actions in the mediator rules.

In the simplified date planning example, the actions of the mediator are completely determined by the type of the messages received. In a more elaborated model of such a mediator, however, the behaviour would also depend on the content of the messages received. If we want to do without message content, we have to accept a possibly nondetermistic rule set, i.e. several rules specify different reactions to the "same" event. This is perfectly acceptable for the level of design and analysis of mediators we propose in the following section, and we will leave out message content in mediator specifications for our present discussion.

In order to keep the mediator rule sets rather compact we have allowed for sender/recipient specifications like `allContractors`, describing sets of contractors which a message is sent to, or expected from. With the dispatch of messages, the meaning of such a specification is obvious. On the recipient side, an event specified by `on_recv(msgtype,allContractors)` occurs when the message of type `msgtype` has arrived from the *last* contractor. With more detailed descriptions of mediators we need further constructs of this type, e.g. `someContractors`, describing a certain subset of contractors. What excatly is meant by `someContractors` depends on problem specific criteria of the mediator, e.g. in the date planning context `someContractors` could mean a subset of participants which needs a separate round of queries and answers in order to reach a date agreement. A precise specification would again require the consideration of message content in addition to message type and is not necessary for our level of modeling.

5. A mediator design tool

So far, we have presented a way of describing the coordinating behaviour of a mediator which is based on the conversational model of cooperation. Because a mediator is a system which coordinates asynchronous communication between people, the demands on reliability and correctness of a mediator system are high. In the following we describe a tool which supports the design of mediators in that it allows to check out a rule based mediator specification.

As we have seen in the last section, it is fairly easy to specify an (admittedly simple) mediator by way of mediator rules. Now, what types of errors or possible inconsistencies can such a specification contain? Apart from pure *syntactical errors* we have a first class of errors which we call *static errors*. Examples are:

- A message type referenced in a rule does not exist, or may not be received by, or sent to, the mediator according to the conversation rules.
- Not all message types that can arrive from either requestor or contractors are covered by corresponding rules.

- A mediator state referenced in the condition part of a rule is not set in any other rule (either explicitly, or implicitly by naming convention).
- A mediator state set explicitly in a rule is not referenced in the condition part of any other rule.
- An alarm event appears in the condition part of a rule that has not been set in the action part of a rule elsewhere.

Static errors can be detected by lexical analysis of the rules, and the mediator design tool should support the user with the elimination of these errors. But a rule set which is consistent and complete in the sense that it contains no static errors can still contain errors of another class which we call *dynamic errors*. These errors concern dynamic properties of the system which the rule set specifies. Examples of such errors are:

- A mediator rule will never fire, i.e. will never become applicable.
- The conversations with requestor and contractors stop before reaching a final state (positive or negative). This is referred to as a *deadlock*, and means that requestor and/or contractors wait for messages that will never arrive.
- The conversation with requestor and contractors stop in inconsistent final states, e.g. the contractors see the mediator process as succesfully terminated whereas the requestor thinks it has failed (usually, conversations have at least two possible outcomes: success or failure).
- Not all possible consistent final states of the mediator process can be reached, e.g. the conversations never reach a successful ending.
- In a specific mediator state, an arriving message is not covered by any rule and thus cannot be processed by the mediator. This may be on purpose because the message is no longer relevant, but usually this situation is unforeseen by the designer and signals an error.

Unlike static errors, dynamic errors cannot be detected by lexical analysis of the rules. In addition, the dynamic properties of the mediator rules cannot be completely separated from the dynamic properties of the conversations the mediator has to deal with. Let us consider our simple date planning mediator as an example. In order to decide whether it runs into a deadlock after having sent a date query to all participants (rules R1, R6, R9), we have to know how a participant can respond to a query. If the mediator-contractor conversation requires a *receipt* as the reaction to a query, the date planning mediator will run into a deadlock, but if *answer* is the correct message type, it will not because rule R7 applies.

For the analysis of dynamic errors we propose the following method: The mediator rules are translated into net inscriptions (firing rules) of the Pr/T net representation of a mediator as shown in figure 3 (shaded subnet). The inscriptions of the mediator net also contain the relevant rules of the conversation types involved. The dynamic properties of this net are then investigated by means of exhaustive simulation, i.e. all situations are generated which the mediator can run into according to the mediator and conversation rules. We have applied this simulation technique successfully to the verification of conversation designs [Woetzel & Kreifelts 1989] using a Pr/T net simulator implemented in Prolog [Wißkirchen et al. 1984].

The translation of mediator rules into inscriptions of the mediator net can be completely automated. A first version of the translator has been implemented in Prolog. The translation process starts with the mediator and the conversations in the initial start state. All rules which are applicable in the start state are translated into net inscriptions. This leads to new mediator states, and further rules are translated until no more rules qualify for translation. Rules left over will never fire, and are

reported back to the designer as possible errors. During translation, a state transition diagram is built up which shows how the mediator rules change the mediator state. This diagram presents the designer with an overview of the mediator behaviour as specified by the rules. It can also help in visualizing problem situations like deadlocks, e.g. by highlighting the node that represents the deadlock state and the path that represents the rule firing sequence leading to the deadlock.

The simulation of the mediator net which has been generated by the translator program will detect any deadlocks, ignored messages and inconsistent terminations. All consistent final states of the mediator system which have been reached during simulation are also reported. In this way, possible errors in the mediator specification concerning coordination and synchronization can be found and eliminated. As a test case, we have applied this technique successfully to a complex date planning mediator [Prinz & Woitass 1989].

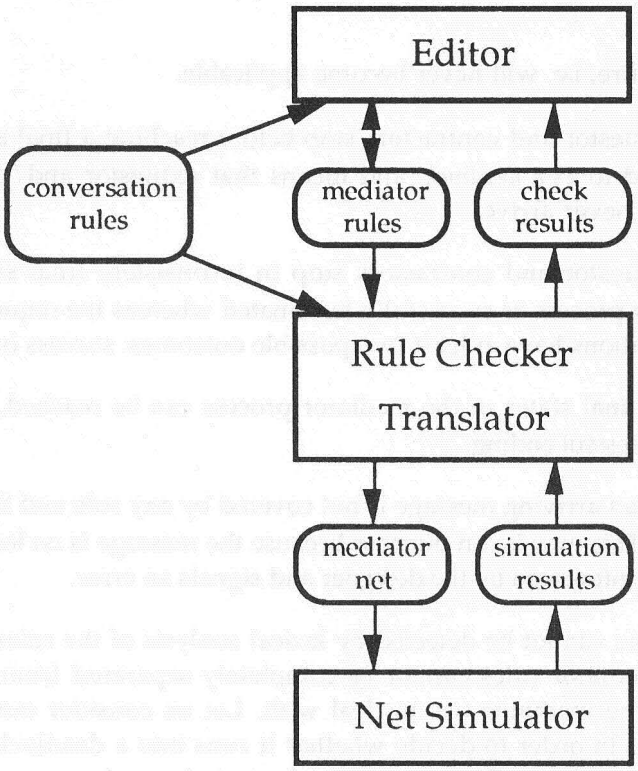


Figure 4. Components of the mediator design tool.

With these methods for eliminating static as well as dynamic errors from a mediator specification, we can support the design of mediators in an interactive and incremental fashion. The principal components of a mediator design tool are shown in figure 4: editor, rule checker, translator and net simulator. The editor is for input and modification of mediator rule sets. Rule editing is simplified by a simultaneous display of the conversation types which the mediator has to handle. The editor will also display the results of lexical analysis and simulation, possibly indicating which rules have to be modified, or in which situations further rules are needed. Figure 5 shows a mock-up of the user interface.

After input, a rule set is first analyzed by the rule checker for syntactical and static errors. If any errors are found, they are reported back to the user for elimination with the editor. If no errors are found, the rules are translated into a Pr/T net representation and turned over to the net simulator. The simulation results are translated back from Pr/T net level to rule level, and then presented to the user in textual as well as in graphical form, using the state transition diagram described above. Dynamic errors found during simulation can be corrected, and a new design cycle is started.

Currently, the mediator design tool is under development. The design method has been tested, and the basic algorithms concerning rule translation and simulation have been implemented. What still has to be done is the lexical rule checker, a suitable error reporting facility including graphical methods, and the user interface itself.

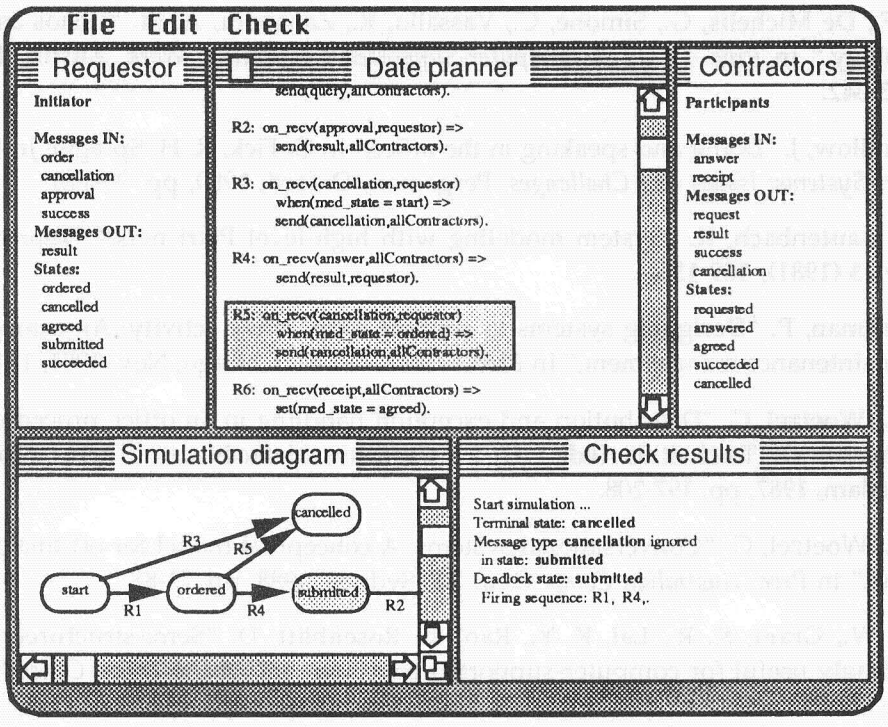


Figure 5. User interface of the mediator design tool.

6. Conclusion

Mediators as autonomous agents for the support of group activity are a useful concept for many group support applications based on electronic mail. An important task common to mediators is the coordination and synchronization of the asynchronous message exchange with a requestor on one side and a set of contractors on the other. We have presented a production rule oriented way of describing the communication behaviour of a mediator which allows for very compact and easily readable specifications. Because a mediator is an automated agent which coordinates remote communication between people, the demands on consistency of the mediator specification is high. We have shown what type of errors a mediator specification can contain and how they can be detected and eliminated. We have described a mediator design tool based on these ideas which supports the consistent design of mediators in an interactive and incremental fashion.

For the future, we plan to extend the mediator model to allow for the treatment of time (e.g. clock events, timeouts) and a more detailed specification through the inclusion of message content into the rules. As a major next step we plan to support not only the design of a mediator, but also the implementation of a mediator. In particular the communication monitoring modules of the mediator should be automatically generated from consistent specifications designed with the mediator design tool.

References

Bowers, J., Churcher, J. "Local and global structuring of computer mediated communication: Developing linguistic perspectives on CSCW in Cosmos," in *Proc. Conf. on Computer-Supported Cooperative Work*, Portland, OR, Sep. 1988, pp. 125-139.

De Cindio, F., De Michelis, G., Simone, C., Vassallo, R., Zanaboni, A. M. "Chaos as coordination technology," in *Proc. Conf. on Computer-Supported Cooperative Work*, Austin, TX, Dec. 1986, pp. 325-342.

Flores, F. , Ludlow, J. "Doing and speaking in the office," in G. Fick, R. H. Sprague Jr. (eds.) *Decision Support Systems: Issues and Challenges*, Pergamon, Oxford, 1980, pp. 95-118.

Genrich, H., Lautenbach, K. "System modeling with high-level Petri nets," *Theoretical Computer Science* 13 (1981), 109-136.

Holt, A., Cashman, P. "Designing systems to support cooperative activity: An example from software maintenance management," in *Proc. COMPSAC 81*, Chicago, Nov. 1981, 184-191.

Kreifelts, Th., Woetzel, G. "Distribution and exception handling in an office procedure system," in G. Bracchi, D. Tsichritzis (eds.) *Office Systems: Methods and Tools*, North-Holland, Amsterdam, 1987, pp. 197-208.

Kreifelts, Th., Woetzel, G. "Conversational systems: A conceptual model for off-line group support systems," in *Proc. Australian Comp. Conf. '88*, Sydney, 1988, pp. 71-88.

Malone, Th. W., Grant, K. R., Lai, K.-Y., Rao, R., Rosenblitt, D. "Semi-structured messages are surprisingly useful for computer-supported coordination," *ACM Trans. Office Inf. Systems* 5,2 (1987), 187-211.

Pankoke-Babatz, U. (ed.) *Computer Based Group Communication*, Ellis Horwood, Chichester, 1989.

Prinz, W., Woitass, M. "The date planning system – Example of a cooperative group activity," in E. Stefferud, O.-J. Jacobsen, P. Schicker (eds.) *Message Handling Systems and Distributed Applications*, North-Holland, Amsterdam, 1989, pp. 359-379.

Reisig, W. *Petri Nets, An Introduction*, Springer, Berlin, 1983.

Sluizer, S., Cashman, P. "XCP — an experimental tool for supporting office procedures," in R. W. Taylor (ed.) *Proc. IEEE First Int. Conf. on Office Automation*, New Orleans, LA, 1984, pp. 73-80.

Winograd, T., Flores, F. *Understanding Computers and Cognition: A New Foundation for Design*, Ablex, Norwood, NJ, 1986.

Wißkirchen, P., Niehuis, S., Victor, F. "Ein rechnergestützter Bürosimulator auf der Basis von Pr/T-Netzen und Prolog," *Angewandte Informatik*, 5 (1984), 181-188.

Woetzel, G., Kreifelts, Th. "Deadlock freeness and consistency in a conversational system," in B. Pernici, A. A. Verrijn-Stuart (eds.) *Office Information Systems: The Design Process*, North-Holland, Amsterdam, 1989, pp. 239-253.