A model of the communicative behavior of designers involved in cooperative software design

Antonio Carlos Pereira Maia [*]
Hugo Fuks
Carlos José Pereira de Lucena

Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro
R.Marquês de S.Vicente, 225,
Rio de Janeiro, RJ, 22453-900, BRAZIL

e-mail: {maia, hugo, lucena@inf.puc-rio.br}

[*](on leave from PETROBRÁS - Petróleo Brasileiro S.A.)

PUC-RioInf.MCC41/ 95 December,1995

ABSTRACT

In this paper we present the use of Statecharts for modeling the communicative behavior of a team involved in cooperative software design. Design episodes were previously represented using diagrams called Dynamics of Interaction Diagrams from which we have now drawn up a cooperative design activity model. The modeling activity itself led to the recognition of a number of underlying properties, most of which comprised temporal relations between speech objects (representations of verbal communication actions). We also present an analysis showing the advantages and disadvantages of Statecharts for modeling software designers' communicative behavior. Finally, we propose the use of another concurrent automaton, Colored Petri Nets, for the same purpose.

Keywords = cooperative software design, communication, Statecharts.

RESUMO

Neste trabalho, apresentamos o uso de Statecharts para modelagem do comportamento comunicativo de uma equipe de desenho de software. Episódeos de desenho foram previamente representados usando diagramas, denominados Diagramas da Dinâmica de Interação, a partir dos quais extraímos um modelo de atividade de design cooperativo. A modelagem proporcionou também o reconhecimento de várias propriedades, muitas das quais compreendem relações temporais entre objetos de fala, ou seja, representações de ações de comunicação verbais. Analisamos também as vantagens e desvantagens de Statecharts como uma notacão para modelar o comportamento comunicativo de desenhistas de software. Por fim, propomos o uso de um outro autômato concorrente, Redes de Petri Coloridas, para o mesmo propósito.

Palavras-chave= desenho cooperativo de software, comunicação, Statecharts.

1. THE MOTIVATING PROBLEM

This paper addresses the modeling of cooperative software design, a form of design where all information exchange and decision-making are collectively undertaken during design meetings. The design activity of a multi-disciplinary group from between five to eight people was first videotaped and then analyzed. The team designed an educational software system.

Working alone or in groups, designers create, develop, refine, analyze and document their conceptualization of a product. In addition to the number of participants, its empirical nature [Ara94] and the scope of problems that could arise contribute to the complexity of the design process. There are many methods for observing the design activity which vary from simple documentation analysis to video tape analysis.

Although video tape technology was previously used to record the design activity, until recently systematic methods had not been developed to analyze video data. For more information on this recent development see our paper [Mai95] detailing a representational scheme for such analysis of team design activity recorded on videotape. The representational scheme focused on the dynamics of interaction between people involved in design. It was based on the belief that the interaction established between designers would provide important insights for a better understanding of design. In this paper, we present the way in which we could draw up a cooperative software design model from that representational scheme. We also introduce a number of underlying properties, which state temporal relations between speech objects (representations of verbal communication action) and the conditions for and constraints to enabling the utterance of specific speech objects.

Finite-state-machines and state-transition diagrams are classical formalisms for the specification and analysis of computational systems. These formalisms have been extended and modified to match the requirements of complex systems. In particular, since state-machines are inherently sequential in nature, a number of extensions have been proposed to deal with the concurrent nature of some applications. This has led to the proposition of a visual formalism to represent concurrent systems named Statecharts [Har88,Har90]. In this paper, we claim about the reactive nature of cooperative software design and propose the use of Statecharts to model it.

Statecharts are a powerful visual formalism for the specification and analysis of reactive systems. Statecharts are an extension of finite-state machines and have been formulated to

overcome some of the drawbacks of finite-state machines notation, such as state explosion and lack of structure.

This paper presents the use of the Statecharts notation for generating a behavioral specification of the problem-solving activity in cooperative software design. In the following section, we provide a brief review of the representational scheme mentioned above. Next, we discuss the reactive behavior of cooperative software design. Then, we introduce the proposed model and related properties. Finally, we present what we have learnt about the usage of the notation to model this particular type of social activity. We also comment on other related research and discuss how current results will affect our future work.

2. A METHOD FOR ANALYZING TEAM DESIGN ACTIVITY

In this section we briefly introduce a method, called the Dynamics of Interaction Method, for analyzing audiovisual records of teamwork design. The method leads to a representational scheme which focus on the dynamics of interaction between team members. A thorough description of the method can be found in a previous paper [Mai95].

The handling of the information produced during design meetings is the kernel of the method. In cooperative design all information exchange is performed during meetings. The whole design is based on collective decisions. Individual activities outside the meetings are restricted to personal documentation, recording the discussion from a meeting and preparation of materials of various kinds to be submitted for consideration by the design team. Although this particular kind of design is not so common in current design practice, it is very interesting and appropriate for our research goals because it is strongly based on cooperation between people of various abilities and from different areas of expertise which leads to the great diversity of cooperation episodes to be analyzed. We believe that what is learnt from these episodes may be used to specify intelligent computer environments for facilitating cooperation between users during problem-solving processes.

Videotape based analysis methods used so far, adapted from [Pur94], have followed the procedure of : recording the design activity on videotape, transcribing the videotape data, segmenting the transcription (by inferring categories by repeatedly reading the transcripts or viewing the tapes, or by imposing an external conceptual coding scheme)

and counting the frequency of occurrence of the categories. In this approach there is a huge abstraction gap between the crude original data (recorded on videotape) and the data used for analysis (the frequency of occurrence of the categories). To reduce this abstraction gap we [Mai95] propose a method to produce dynamic models out of co-presence based teamwork activities.

We could summarize the method through the following six steps:

1. Record the design activity on the videotape;

2. Select relevant sequences (by whatever criteria are relevant to the specific analysis);

3. Define a representation language;

4. Translate the selected relevant sequences into the representation language (specific to the domain problem and research goals) thus creating a less complex description of the activity while preserving relevant information;

5. Search for interaction patterns;

6. Apply effort measures to this representation.

The first step towards the creation of the representational scheme is to define a language that will show the dynamics of interaction between team members. This language is used to express visually relevant sequences of the design practice, abstracting from the specifics of the application domain by providing a representation that reveals communication actions (both verbal and non-verbal ones- such as gestures and looks). In this paper we do not present all possible sentences, but only those that are really necessary to describe design episodes. We also decided to simplify this language in order to facilitate the understanding of the resulting representational scheme. Therefore, in this paper we present communication actions that are expressed by the second version of the language. A list of these communication actions are presented in Appendix A.

In order to provide a visual organization upon the new description of the design activity we propose to display communication actions in diagrams - we named them Dynamics of Interaction Diagrams. These diagrams preserve the temporal relations and parallelism among communication actions. We would also like to register turn-taking allocation. The design episode description should identify turn transitions and indicate which turn

allocation mechanism was applied, not only verbal ones (by naming the person to take the turn) but also non-verbal ones (staring at the person to whom you want to address a query). In Appendix B we present and comment on a diagram representing a genuine design episode.

3. THE REACTIVE NATURE OF TEAM DESIGN PROBLEM-SOLVING

The use of concurrent automata to represent team design is possible due to its reactive nature. A reactive system was defined by [Har88] as being characterized by its event driven nature, continuously reacting to stimuli. The behavior of a reactive system is really the set of allowed sequences of input and output events, conditions, actions and information flow. Team design behavior is also a set of permitted sequences of input and output events, particularly, communication actions. We may characterize an interaction state by the relationship between its input or incoming communication actions and its permitted output communication actions.

Dynamics of Interaction Diagrams represent parallel sequences of input and output communication actions that actually took place during design meetings. We studied these diagrams, looking for sets of input/output for each particular input communication action. As a result we derived for each input event (communication action) a fixed set of permitted output communication actions. From this we drew up a model of the reactive behavior of the team design activity. While involved in the modeling process, we also identified several temporal relations between communication actions.

4. TEAM DESIGN ACTIVITY MODEL

In this section we present the resulting model of the communicative behavior of the team design activity and its underlying properties. We used Statecharts to represent the model. In the next sub-section we introduce some basic concepts and terminology of Statecharts. Then, we present the model itself.

4.1. STATECHARTS BASIC CONCEPTS

State-based notation specifies states and transitions between states. In some applications, states represent sets of operational and internal conditions of physical machines, while in others they may represent sets of buttons enabled or disabled, and information displayed or not on the screen of a user interface, and so on. In our

modeling approach states represent sets of predicates that hold or not as a result of the communicative behavior among designers. Transitions between states are triggered by communication actions. In a particular state a number of communication actions are made available or not for designers use.

Statecharts are an extension of finite-state-machines which support state decomposition and broadcast communication mechanisms. The former mechanism adds hierarchy and concurrence to the notation while the latter one allows for the propagation of events through concurrent components.

Hierarchy allows common transitions to be clustered and adds structure to the chart [Car94]. It is achieved by the exclusive or (XOR) decomposition. In the next section, we use XOR decomposition to describe starting conditions of a design episode.

4.2. DESIGN EPISODE INITIATION

In fig. 1 we represent a design meeting by a state, named Meeting-in-progress. During a design meeting designers are engaged in a cooperative design task (we consider they are taking part in a design-episode) or are performing another meeting activity, for example, planning the meeting itself, or they are simply engaged in side digressions (not taking part in a design-episode). We represent these mutually exclusive situations by decomposing Meeting-in-progress into two sub-states (components): In-design-episode and Not-in-design-episode, respectively. To be in the state Meeting-in-progress means to be in the component state In-design-episode or to be in the component state Not-in-design-episode.

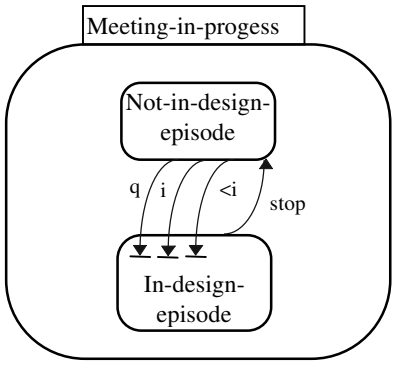


Figure 1. A high level description of a design meeting.

To enter the In-design-episode state there are three events available: **i** (to raise a new issue), **<i** (to raise a former issue) and **q** (to make a query). These events fire transitions from the state Not-in-design-episode into the state In-design-episode. On the other

hand, the event **stop** fires a transition that leaves the In-design-episode state and enters the Not-in-design-episode state. For modeling purposes we use two different types of events. The first one comprises all events that correspond to communication actions, such as **i**, **<i** and **q**. The second type comprises events that correspond to procedures that take place during a design meeting but do not have a direct correspondence to a particular communication action. There are few events of this type, such as **stop** and **resume**. The event **stop** takes place whenever the team finishes or simply quits a design episode. On these occasions, designers simply change to another issue or another activity. Usually there is no particular identifiable verbal or non-verbal action taking place.

With the high level description of the team design activity introduced above we are now able to state two properties of the model. These first properties deal with the initiation of a design episode, i.e., how designers become engaged in a design task:

I. At the beginning of every design episode communications actions **i** , **<i** and **q** must be available.

II. Only the communication actions **i** , **<i** and **q** are available at the beginning of every design episode.

These two properties state that the beginning of a design episode is determined by performing one of the listed communication actions: **i** (to raise a new issue), **<i** (to raise a former issue) or **q** (to make a query).

4.3. CONCURRENT PROCESSES

Teams work in a parallel manner. Hence, a design episode description should be able to represent a number of concurrent processes. In a design episode designers may be engaged in five different processes which we represent by independent and concurrent states.

The ability to describe independent and concurrent states is provided in Statecharts by the AND decomposition of states. AND components are said to be orthogonal. In figure 2 we show how the In-design-episode state is decomposed into orthogonal components representing concurrent processes. We identify the following processes taking place in a

parallel manner during a design episode: Design-problem-solving, Feedback, Addressing, Documentation and Retrieval.

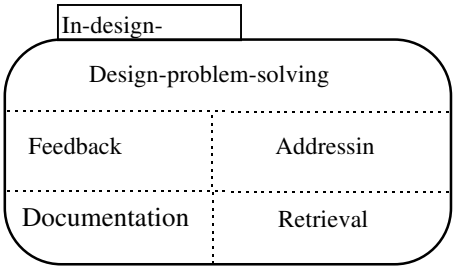


Figure 2. Description of concurrent processes in a design episode

4.4. DESIGN JUSTIFICATION

In cooperative design designers exchange pieces of rationale in order to support or clarify their position, to justify their preference, to resolve a conflict or simply to provide their personal experience. Hence, in cooperative design justification is part of the overall process of producing the design. However, it is a different problem-solving activity from design itself, in particular, with different goals and eventually different knowledge (e.g., strategic knowledge about negotiation) [Bro94]. Consequently, it makes sense to represent Design Justification and Design as internal states of Design-problem-solving - fig. 3.

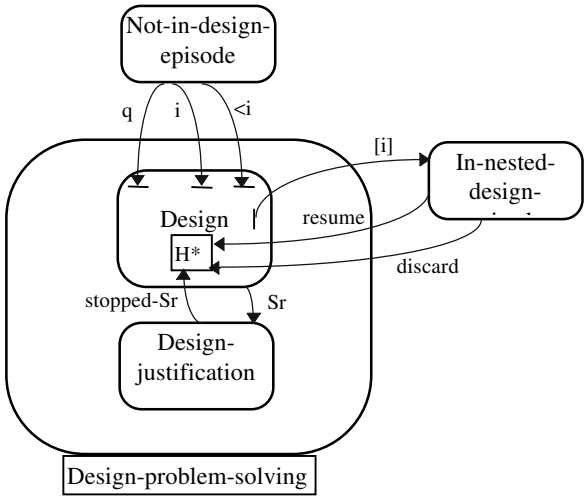


Figure 3. Description of Design and Design Justification as different activities

A powerful notation to represent a transition into a composed state is known as H-entrance (H stands for history). Using this notation one is able to specify the resumption of a context - that is, to specify that when returning to a previously visited composed state the system shall enter the component state it last visited. The H-entrance notation

is applied to the level that appears. Nevertheless, an extension of this notation, called H*-entrance, allows one to specify that it should be applied to all levels.

In the Statecharts for the Design-problem-solving state the event **Sr** (Supply rationale) fires one transition from the Design sub-state to the Design-justification sub-state. On the other hand, the event **stopped-Sr** is used to represent that the designer completed hers justification (and, consequently, stopped supplying design rationale). This event fires a transition that takes it back (see the H*-entrance symbol) to the last visited internal state of Design (whose decomposition will be introduced soon).

4.5. DEPENDENT ISSUES

There are situations during design episodes in which a designer may propose the discussion of a new issue strongly related to the current issue (sometimes a generalization or a specialization of the current one). This situation usually takes place when an issue is raised but the team perceives that they first need to solve another issue upon which the current one is dependent, i.e., there is an order in which issues have to be solved. This is represented in figure 3 where event **[i]** (to raise a nested issue) fires one transition into a new state called In-nested-design-episode (and satisfies the predicate *nested-issue*). After discussing the nested issue, the team may return to the previous context, which is represented by the transition fired by the event **resume**. Nevertheless, when discussing the nested issue a designer may propose a new nested issue. This situation leads to a chain of nested issues or a dependence net (alternatives of values for dependent issues are assigned after the issues on which they depend are solved). In this case, the event **resume** will always take it to the previous (dependent) context. For the sake of brevity, an arbitrary number of nested issues is not represented in the figure.

When discussing a nested-issue the team may want to give up the whole reasoning line (formed by a number of nested issues) and return to the context of the first issue. This is made possible in the model by the event **discard**. The event **discard** is available at the In-nested-design-episode state. This event leaves the current nested design episode and takes it back to the initial context in the structure of nested-issues (see fig. 5).

A number of new properties deal with the availability of the events previously described:

III. There is some design-episode in which during some interaction cycle *nested-issue* is satisfied and **[i]** (to raise a nested issue) is available.

> This property states that a nested issue may also be further generalized or specialized. In practice we have observed chains of up to three nested issues.

IV. In all design episodes if the event **resume** is available then the *nested-issue* is satisfied.

> This property states that one can always resume the context of the primary issue (the one that was generalized or specialized) and that *nested-issue* is a pre-condition for enabling the event **resume** to take place.

V. In all design episodes if the event **resume** is available then **discard** is also available .

> This property states that one is not obliged to resume the context of a primary issue, there is also the possibility of quitting the whole line of reasoning. Both events (**resume** and **discard**) are enabled by the predicate *nested-issue*.

VIa. There is no design episode in which during some interaction cycle both events **stop** and **resume** are available .

VIb. There is no design episode in which during some interaction cycle both events **stop** and **discard** are available.

> These properties state that one should resume previous issues or quit the whole line of reasoning before leaving a design episode (via the event **stop** which fires a transition that leaves the In-design-episode state and enters the Not-in-design-episode state).

VIIa. There are some design episodes in which **resume** is never available.

VIIb. There are some design episodes in which **discard** is never available.

> These situations take place whenever the pre-condition - *nested-issue* - of both events (**resume** and **discard**) is not satisfied. That is, **resume** and **discard** are available if and only if a nested issue has been raised.

VIII. **Stop** or (**resume** and **discard)** are available at any moment.

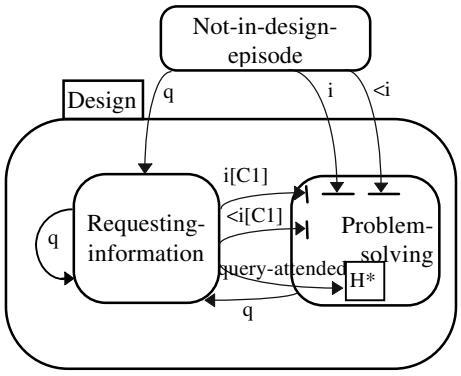
IX. It is possible to begin a new design episode at any moment.

X. From the beginning of a design episode, it is true that one can never reach an interaction cycle in which no communication action can be taken.

These properties state that one can always reach the initial state Not-in-design-episode in which it is possible to begin a new design episode.

4.6. STARTING DESIGN EPISODES

Designers start a design episode by requesting for problem-solving (raising an issue) or requesting for information (making a query). This is represented by decomposing the Design state into two sub-sates, namely: Problem-solving and Requesting-information. In fig. 4 we can observe that whenever the current state is Not-in-design-episode and a designer raises a new issue (event **i**) or raises a former issue (event **<i**), transitions will be fired from it to the Problem-solving state (actually to its internal states - not yet introduced). On the other hand, if a query is made (event **q**) a transition to the Requesting-information state will be fired. It is also specified that the event **q** is available from Problem-solving's internal states as well, which represents that, whenever the team is solving an issue, a designer can always ask for clarification. After clarification the event **query-attended** takes it back to the last visited sub-state of Problem-Solving.



C1 = condition
not(*issue-raised*)
and
not(*issue-revisited*)

Figure 4. Requesting problem-solving and information

There are situations during design episodes in which a request for information may introduce the discussion of an issue, i.e., call the team's attention to the fact that a particular issue should be considered/solved. It may be a new issue or an issue already discussed beforehand. The issue is brought under consideration as a consequence of the query. This situation allows for the observation of another important feature of Statecharts which is the possibility of representing conditions. In figure 4, conditioned events **i[C1]** and **<i[C1]** will fire transitions from the Requesting-information state to

the Problem-solving state if and when events **i** (to raise a new issue) and **<i** (to raise a former issue) respectively occur since the condition **C1** is true at that time. So, events **i** and **<i** are available from the Requesting-information state, although conditioned to the fact that neither a new issue nor a former one have already been raised before.

The state decomposition presented above allows for the introduction of a couple of new properties related to queries:

XI. There are some design episodes in which during some interaction cycle *query-posed* is satisfied and **i** and **<i** are available.

This property states that a new or a former issue is brought under consideration by the design team as a consequence of a previous query.

Let us illustrate this situation with an extract of a genuine example:

Context: the software designer was trying to build a relation between teaching goals (items) and concepts. In particular, some teaching goals of a preliminary unit and the concept of variables.

Role players: Software designer (SD), Domain expert (DE)

Simplified (translated - original in Portuguese) dialogue:

/*begin of extract

(SD) The query is: when I was drawing the diagram I considered that all these items were bound to the concept of variables. Is that correct?

(DE) Yeah. it's always bound to variables.

....

(DE) But, what I thought about these items is that they should be re-organized according to the procedure we defined in the last meeting.

/*end of extract

From this point on, designers start discussing how to structure teaching goals in an hierarchy according to a procedure already established.

XII. At any interaction cycle of every design episode **q** is available.

Finally, the last property of this group states that queries may be made at any time during design episodes.

4.7. PROBLEM-SOLVING

Problem solving is a major activity in design. It involves the generation, refinement and evaluation of the designers' conceptualization of a solution. Given that it is a complex activity we introduce its components step-by-step as we go deeper into an hierarchy of states. Let us first take a look at what happens when a design member requests problem-solving.

4.7.1. REQUESTING

When requesting problem-solving there are two possibilities: a new issue (not solved) is raised or a former issue (already solved in a previous occasion) is revisited. In the model, a number of transitions, previously introduced, fired by events **i** (to raise a new issue) and **<i** (to raise a former issue) enter the Requesting-problem-solving state. These events fire transitions into sub-states Issue-raised and Issue-revisited satisfying predicates *issue-raised* and *issue-revisited* respectively.

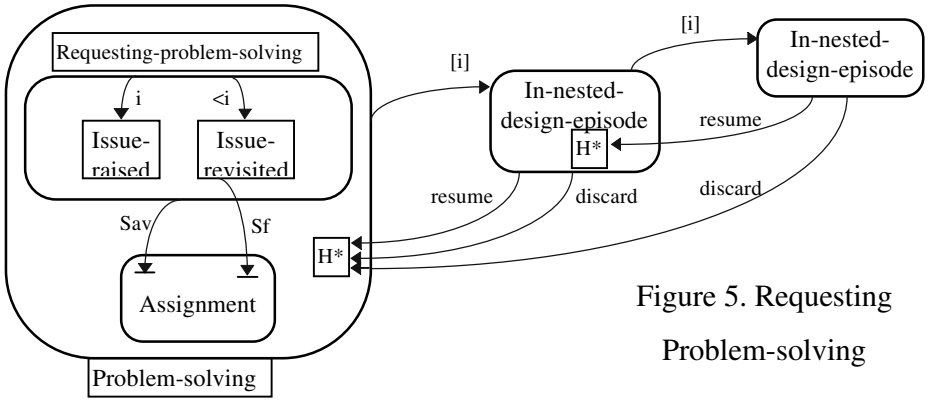


Figure 5. Requesting Problem-solving

Once an issue is raised or revisited designers supply alternatives of value (generate possible solutions) to this issue. This is represented by the transition, labeled by the event **Sav** (Supply alternative of value)**,** from Requesting-problem-solving into (an internal sub-state of) Assignment. Figure 5 shows that the Problem-solving state comprises two sub-states: Requesting-problem-solving and Assignment. The arrow labeled by the event **Sav** leaving the Requesting-problem-solving state means that the transition is fired from any of its internal sub-states (Issue-raised and Issue-revisited).

An issue being revisited brings with it the alternative of value assigned to it when he was first solved - that is, the decision made. This alternative of value previously assigned to this issue is liable to elaboration, that is, designers may suggest that features

should be added or subtracted from the original decision (alternative of value). This is represented by the event **Sf** (Supply feature). This event fires a transition from a particular sub-sate - Issue-revisited into the Assignment state.

4.7.1.1. NESTED-ISSUE AVAILABILITY

In figure 3, decomposition of the Design-problem-solving state, the event **[i]** (to raise a nested issue) was first introduced. Now we resume its description to show that it is only available when there is an issue under consideration. This is stated by the following property:

XIII. In all design episodes if **[i]** is available then *issue-raised* is satisfied or *issue-revisited* is satisfied.

This property states that an issue under consideration is subject to an abstraction (generalization or specialization) or must be solved after another issue. Thus, leading to new issue (a nested issue). Raising a nested issue does not take it out of the current design episode. It just starts a new process of decision (a nested design episode) after which one may resume the issue (the context of the primary design episode) or discard the current reasoning line.

4.7.1.2. MUTUAL EXCLUSION PROPERTIES

Cooperative design is based on collective decision-making. Designers in the meeting are supposed to take part in all decisions. To be able to participate in all decisions, designers discuss one issue at a time. This is represented in figure 5 in the decomposition of the Problem-solving state. It allows for the statement of a number of new properties:

XIV. There is no design episode in which during some interaction cycle *issue-raised* is satisfied and **i** is available.

XV. There is no design episode in which during some interaction cycle *issue-revisited* is satisfied and **<i** is available.

XVI. There is no design episode in which during some interaction cycle *issue-raised* is satisfied and **<i** is available.

XVII. There is no design episode in which during some interaction cycle *issue-revisited* is satisfied and **i** is available.

> The last four properties state that pairs of communication actions (**i** and **i**), (**<i** and **<i**) and (**i** and **<i**) are mutually exclusive. This is a modeling decision that we believe adheres to the observed behavior. Let us take the pair **i** (to raise a new issue) and **<i** (to raise a former issue). If designers are considering an issue and then abandon it to focus on a former issue, this situation is modeled as the end of the current design episode and the beginning of a new design episode in which a former issue is revisited. The same reasoning is applied in relation to the remaining pairs of communication actions.

4.7.1.3. PROPOSITION AND ELABORATION OF ALTERNATIVES OF VALUE

In this sub-section we state some properties related to the proposition and the elaboration of alternatives of values:

XVIII. In all design episodes if the communication action Sav (Supply an alternative of value) is available, then *issue-raised* is satisfied or *issue-revisited* is satisfied.

> This property states that one may only supply an alternative of value if there is an issue under consideration.

Sometimes designers first need to solve an issue upon which the current one is dependent. Consequently, a nested issue is raised. Raising a nested issue brings up a new process of decision (a nested design episode) similar to the current one, which allows designers to supply alternatives of value to the nested issue, as stated by the following property:

XIX. There are some design episodes in which during some interaction cycle *nested-issue* is satisfied and the communication action **Sav** (Supply an alternative of value) is available.

> In this case, from XIII, *issue-raised* is satisfied or *issue-revisited* is satisfied.

XX. In all design episodes if *issue-revisited* is satisfied then *alt-of-value-supplied* is satisfied.

We have already mentioned that a former issue may be revisited. Nevertheless, an issue can only be considered a former issue, and consequently revisited (as stated by the previous property), if it was previously raised and an alternative of value assigned to it. Otherwise, it is considered a new issue, and not a former issue.

XXI. In all design episodes if the communication action **Sf** (Supply belief) is available then *alt-of-value-supplied* is satisfied.

The last property of this group states that first a solution (alternative of value) is proposed and then it admits modification of its characteristics by adding or suppressing features.

In the next sub-section we look at the Assignment state and its internal components.

4.7.2 SOLVING THE PROBLEM

In this sub-section we introduce processes observed once the team was confronted with a issue (problem) to be solved. We identified three distinct processes represented, Figure 6, by three sub-states: Generation, Evaluation and Selection. The Generation state represents a problem-solving activity component in which designers generate possible solutions (alternatives of value) to the problem (issue) under consideration. The event **Sav** (Supply alternative of vale) represents the proposition of an alternative of value. It fires a transition into the sub-state Alt-of-value-supplied satisfying predicate *alt-of-value-supplied*.

Proposed alternatives of value are refined (elaborated) - that is, designers supply additional features or suppress features from an alternative of value. This is represented by the transition fired by the event **Sf** (Supply feature) from the sub-state Alt-of-value-supplied to the sub-state Alt-of-value-elaborated. This transition satisfies the predicate *alt-of-value-elaborated*.

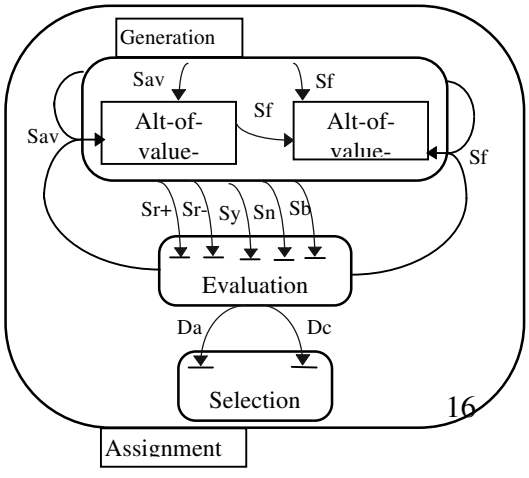


Figure 6. Alternatives of Value Generation, Evaluation and Selection

After or even concomitant with the generation of solutions, designers evaluate proposed solutions expressing their agreement or disagreement to one or another solution, supplying rationale to support their position or just expressing their preference for a particular solution. These communication actions are represented by a number of transitions from the Generation state into the Evaluation state (satisfying the predicate *alt-of-value-evaluated*) as shown in figure 6. These transitions are labeled with the corresponding events: **Sr+** (Supply supporting rationale), **Sr-** (Supply against rationale), **Sy** (Supply positive reply, i.e., agreement), **Sn** (Supply negative reply, i.e., disagreement) and **Sb** (Supply preference). Figure 7 presents a classification of these communication actions into two categories: positive and negative evaluation.

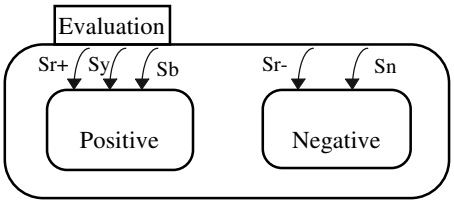


Figure 7. Positive and negative evaluations

Besides generating and evaluating possible solutions, designers also have to select one of them to be assigned to the design parameter (issue) under consideration. In figure 8 we introduce the decomposition of the Selection state which represents this activity.

After discussing one or more alternatives of value the team should converge to one of them. We have observed two common strategies normally applied when designers feel that they are reaching a consensus in relation to one of the proposed solutions (alternatives of value): verification and confirmation. The former is applied when a designer wants to check if another designer (who has not yet committed to the candidate alternative of value) agrees to this alternative of value. This situation is represented by a transition labeled with event **Da** (Demand agreement). It leads to the Selection-candidate state, which means that there is a candidate alternative of value to be assigned to the issue (*selection-candidate* is satisfied). A designer expresses agreement or disagreement to the candidate alternative of value by supplying a positive reply (**Sy**) or a negative reply (**Sn**) respectively. In the first case, the transition, labeled with the event **Sy,** is fired leading into the Alt-of-value-selected state and satisfying the predicate *alt-*

of-value-selected. To sum up, a decision is made. In the second case, the transition, labeled with the event **Sn,** is fired leading into the Negative (Evaluation) state. Once one alternative of value is selected, designers are likely to ask for its confirmation. This is represented by the event **Dc** (Demand confirmation). This transition leads to the Demanded-confirmation state (satisfying *demanded-confirmation* predicate).

Confirmation is the second strategy that is applied when, as a result of the discussion there is no doubt concerning the alternative of value to be selected (usually all designers are already committed to one solution), designers skip the agreement verification procedure described above and go straight to confirm a particular alternative of value. This is represented by the transition, labeled with the event **Dc**, from the Evaluation state into the Demanded-confirmation state. The selected alternative of value is confirmed by restating it, which is represented by the event **Ssav**.

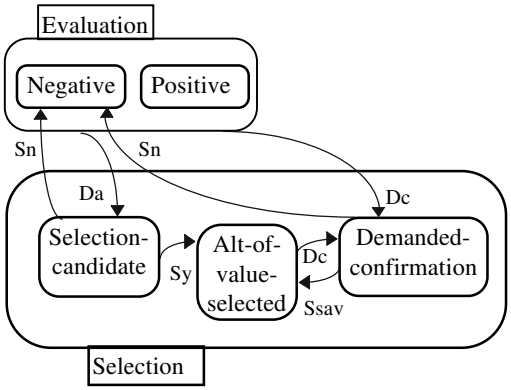


Figure 8. Decision-making

We are now ready to introduce some properties dealing with agreement to and confirmation of alternatives of values:

XXII. In all design episodes, if the communication actions **Da** (Demand agreement) and **Dc** (Demand confirmation) are available, then *alt-of-value-evaluated* is satisfied.

The first property states that an alternative of value must first be evaluated in order to be liable for selection.

XXIII. In all design episodes, if *alt-of-value-selected* is satisfied, then *selection-candidate* is satisfied or *demanded-confirmation* is satisfied.

This second property is about the two strategies mentioned above. If the team reached a decision one of the two strategies, verification or confirmation, was applied.

4.8. DESIGN DOCUMENTATION

In this sub-section we introduce two In-design-episode state's ortoghonal components. The first one deals with the documentation of information exchanged during the design problem solving activity. Designers record not only decisions but also related information such as the rationale behind a decision or discarded alternatives. The second one deals with the retrieval of information to support the design problem solving activity. Designers access former design documentation, for example design specifications as well as manuals or personal notes. These states are named Documentation and Retrieval respectively.

Often during a design episode designers take notes or write and draw on a shared workspace. They usually annotate decisions made on their personal notebooks and use public workspaces to explain something. This type of action takes place parallel to utterances. Figure 14 shows the Documentation state decomposition into two orthogonal components: Public-record and Private-record. Each one is composed of two sub-states representing mutually exclusive situations: writing (or drawing) and not writing (or drawing).

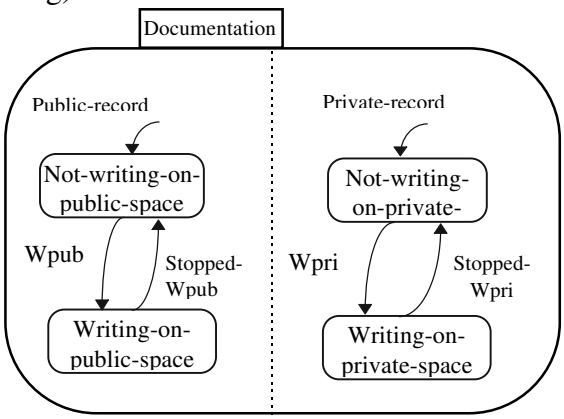


Figure 9. Design Documentation

In figure 10 we represent the use of documentation. Designers browse documents, represented by event **B**, when, for example, they are looking for a piece of information. Designers also read the contents of documents to other designers. This transition is labeled by event **R**.

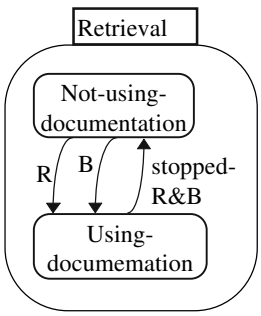


Figure 10. Documentation Retrieval

4.9. EXPRESSING AGREEMENT AND UNDERSTANDING

A common parallel communication action takes place when a designer wants to show agreement, disagreement or understanding of something that is being uttered by another designer. This situation is represented in the Feedback state in figure 11.

Designers verbalize their agreement or disagreement to something or just make gestures (such as nodding one's head or shrugging one's shoulder) with the same intention. As different designers may eventually give this sort of feedback using different channels (verbal and non-verbal) concurrently, these two possibilities are represented as orthogonal states: Gesturing and Verbalizing.

Broadcast communication is the communication mechanism used in Statecharts. It makes it possible for an event that was fired in a given state be propagated through all its orthogonal components. All the transitions caused by this event will be fired simultaneously. Hence, in figure 11, the occurrence of event **Sy** will cause the transition from Silent into Expressing-in-words in the Feedback state as well as the transition into the Evaluation state (figure 6) if it happens to be in the in the Generation state or into the Alt-of-value-selected (figure 8) if it happens to be in the Selection-candidate state in its orthogonal component Design-problem-solving.

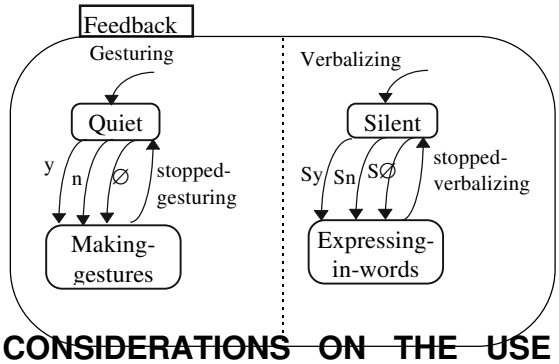


Figure 11 Feedback

5. FINAL CONSIDERATIONS ON THE USE OF STATECHARTS TO REPRESENT TEAM DESIGN ACTIVITY

The use of Statecharts allowed us to generate a concise representation of the communicative behavior of a team involved in a design activity avoiding some disadvantages of finite-state machine representation, such as state explosion and lack of structure [Har92].

The use of XOR decomposition besides adding structure to the chart itself, also allowed us to zoom in and out while working at different levels of abstraction, so that, at one moment we could concentrate on details and at another moment we could abstract from

them and take a global view. This step-wise refinement modeling technique was also facilitated by a number of mechanisms to represent transitions concisely. There are also mechanisms to represent transitions into structured states that add to concision. In particular, the H*-entrance, a notation to represent a transition into the state most recently visited in a structured state. This notation allowed a simple and elegant representation of the procedure of resuming a previous context that frequently takes place in team design meetings. Concurrence, a major characteristic of our application problem, was also easily represented by the use of AND decomposition.

We also observed some disadvantages which are concerned with generation and consumption of resources. Let us first consider the problem of generation of resources illustrating it with an example. Whenever an alternative of value is provided by a designer, that is an event **Sav** is fired, it enters the state Supplied-alt-of-value. If this event is fired twice, the state is entered twice and there is nothing to represent the fact that although it is again in the same state something has changed - now one has got two alternatives of value under consideration instead of one. Actually, the state is not exactly the same. One could always create a new state to represent it but this would lead to an explosion in the number of states.

The problem of consumption of resources is almost a consequence of the former one. Let us consider the scenario described above. There are two alternatives of value under consideration and then one designer supplies a supporting rationale to one of them while another designer expresses preference for the other. In order to properly model this situation one needs to distinguish each alternative of value. This is necessary because the different instances of alternative of value are taking part in different transitions, that is, are being consumed by different transitions. One of them is likely to be selected while the other discarded. Similar situations take place in relation to agents. Similar communication actions taken by different agents configure different situations. For example, if a designer who was previously supplying rationale against an alternative of value then expresses agreement or supplies a supporting rationale to it, it is a different situation than if it were another agent that had expressed agreement or supplied a supporting rationale to the alternative of value under consideration. In summary, we felt the need to identify and distinguish some particular communication actions and the agents of these actions in order to provide a more accurate model of the activity. Consequently, we propose the use of another concurrent

automaton, Colored Petri Nets, for overcoming these drawbacks and providing a more accurate model of cooperative software design.

6. RELATED RESEARCH AND RESEARCH DIRECTIONS

In the field of Computed Supported Cooperative Work there are a number of tools to support communication among people. Some of them take conversation-oriented approaches [Flo88, Con88, Cin86, Kap90). Searle's Speech Act theory [Sea69] is the underlying theory of most of them.

According to this theory when a person utters a sentence, he or she also realizes an act which may commit him or her to do something or to adhere to the truth of something or to repute the listener to do something. An utterance actually involves three kinds of acts: *locutionary act* - the speaker utters something, *illocutionary act* - the speaker aims at the realization of a social or illocutionary act, and *perlocutionary act* - the speaker tries to convince the hearer. An utterance also conveys its *propositional content*, i.e. the object of the utterance. Searle also provides a classification of speech acts covering all possible types of utterances from the speaker's point of view [VEG83], disregarding their *propositional content*. Another classification of speech acts was also proposed by Ballmen and Brennenstuhl [BaB81].

Winograd and Flores [Win86] propose a conversation model based on speech acts that recognize four types of conversations: Conversation for Action (CfA), Conversation for Possibilities (CfP), Conversation for Clarification(CfC) and Conversation for Orientation(CfO). This conversation model led to the development of a work group productivity system, called **The Coordinator**. **The Coordinator** assists conversations in general and does not deal with the contents of exchanged messages. **CHAOS** (Commitment handling Active Office System) [Cin86] also structures the contents of messages using the speech act theory but relies on knowledge about the *organization structure* and the users *linguistic context*. **The Conversation Builder** is another system that adds improvements to the initial functionality of the Coordinator. It is a framework for collaborative open systems which works with the concept of protocols. Conversations are instances of protocols. It allows the composition of complex protocols by smaller ones and it has the ability to invoke a protocol from within a

protocol. Based on the Conversation for Action concept [Lau95] proposes the use of Conversation Clichés for Cooperation.

Conversation for Possibilities (CfP) is intended to define the range of possibilities for analyzing a problem and for preparing a decision. This particular type of conversation also inspired a model of deliberation, called Issue Based Information System (**IBIS**). The **IBIS** model focuses on the articulation of the key *issues* in order to solve a problem. Each issue may have a number of *positions* (alternatives of solutions) and each *position* may have a number of *arguments* that support it or object to it. It is the underlying model of a hypertext system called **gIBIS** [Con88], which facilitates and encourages the capture of deliberations. **gIBIS** is a general tool independent of any application domain. Some other models, such as **QOC** [MaL91] and **DRL** [Lee90], were also suggested in the field of Design Rationale for the capture and use of the rationale provided during deliberation in design activities.

Both Conversation for Possibilities and the **IBIS** model are based on the same underlying theoretical foundation known as Toulmin's argumentation theory [Tou58]. In Toulmin's model, *evidence* is presented for a *conclusion* based on *warrant*. To this warrant are given evidences (called *backings*) and conditions of exception (*rebuttals*). Qualifiers are used to indicate the universality of arguments.

In the above paragraphs we presented argumentation and deliberation models (Toulmin's model, **IBIS**) as well speech acts based frameworks for the structuring of conversation. These conversation-oriented approaches were used as a basis for designing a number of systems that support communication among people involved in technical-social activities. As we are modeling the communicative behavior of a team involved in a technical-social activity - cooperative software design - we considered it important to compare our approach with that of other studies previously conducted in this field.

We have proposed a grammatical approach to address the problem of modeling team design activity. To define the language for describing team design activity we partially drew on models of argumentation and deliberation (**IBIS**). Toulmin's argumentation model was too general for our needs since it was conceived to deal with any sort of argumentation and we were just concerned with argumentation for deliberation. **IBIS**,

on the other hand, was to specific for our needs. Besides the capture of deliberations we were also interested in the capture of information exchange during design problem-solving.

Another important aspect to be pointed out is that our goal was to describe communicative behavior. Conversation-oriented approaches focus only on the verbal channel. Hence, they are not appropriate to describe non-verbal communication action (such as, gestures and visual contact).

Conversation-oriented approaches usually consider two parties involved in conversations. As we are modeling collaborative design, we have proposed a visual representation that is able to represent communication actions performed by any number of participants.

7. REFERENCES

[Ara94] G. Arango. Domain analysis methods. In.W. Schäfer, R. Prieto-díaz and M. Matsumoto (Eds.) *Software Reusability* (Ellis Horwood, New York, 1994) 17-49.

[BaB81] T. Ballmer and W. Brennenstuhl. *Speech Act Classification: A Study in the Lexical Analysis of English Activity Verbs.* (Springer-Verlag,1981).

[Car94]. L.M.F. Carneiro-Coffin. *Jasminum: joining ADVs and State Machines in a Notation for user-Interface Modeling*. Ph.D. Thesis, University of Waterloo, 1994.

[Cin86] F. de Cindio, G. de Michelis, C. Simone, R. Vassalo and A. Zanaboni. CHAOS as a Coordination Technology. In *Proceedings of the First Conference on Computer-Supported Cooperative Work*, Austin, (1986) 325-342.

[Con88] J. Conklin and M.L. Begeman. gIBIS: A Hypertext Tool for Exploratory Policy Discussion. *ACM Trans. on Office Info. Systems*, 4 (1988) 303-331.

[Bro94].D.C. Brown. Rationale in Design. In *Notes of the Representing and Using Design Rationale Workshop in the Third Intl. Conference on Artificial Intelligence in Design,* (Lausanne, Switzerland, 15-18 August 1994) 1-3.

[Flo88] F. Flores., M. Graves, B. Hartfiled and T. Winograd. Computer Systems and the Design of Organizational Interaction. *ACM Trans. on Office Info. Systems*, 2 (1988) 153-172.

[Har88] D. Harel. On Visual Formalisms. *Communications of the ACM*, **31**(5) (1988) 514-530.

[Har90] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring, and M. Trakhtenbrot. STATEMATE: A Working Environment for the Development of Complex Reactive Systems. *IEEE Transactions on Software Eng.*, **16**(4) (1990) 403-414.

[Kap90] S.M. Kaplan. Conversation Builder: an Open Architecture for Collaborative Work. *Interact'90*. (Elsevier, 1990) 917-922.

[Lau95] C. C. Laufer and H. Fuks. ACCORD: Conversation Clichés for Cooperation. *In Proceedings of The International Workshop On The Design Of Cooperative Systems*, Juan-les-Pins, France, (1995)

[Lee90] J. Lee. SIBYL: A Qualitative Decision Management System. In P. Winston & S. Shellard (Eds.), *Artificial intelligence at MIT: Expanding Frontiers* vol. 1 (MIT Press, Cambridge, MA, 1990) 104-133.

[MaL91] A. MacLean, R. Young, V. Belloti and T. Moran. Questions, Options and Criteria: Elements of a Design Rationale for User Interfaces. *Human Computer Interface,* **6**(3/4) (1991).

[Mai95] A.C.P. Maia, A.C.B. Garcia and J.C.P. de Lucena. A Method for Analysing Team Design Activity. In *Proc of the Symposium on Designing Interactive Systems*, Ann Arbor, MI, August 23-25, (ACM, New York, 1995).

[Pur94] A.T. Purcell, J.S. Gero and H.M. Edwards. The data in Design Protocols: The Issue of Data Coding, Data Analysis in The Development of Models of the Design Process. In *Preprints of papers to be presented at the workshop Analysing Design Activity*, Delf, The Netherlands, September 20-22 (1994)169-187.

[Sea69] J.R.Searle.*Speech Acts*. (Cambridge University Press, Cambridge, 1969).

[Sea75] J.R.Searle. A taxonomy of illocutionary acts. In K. Gunderson (Ed.) *Language, Mind and Knowledge*. Minnesota Studies in the Philosophy of Science vol. 7, (Univ. of Minnesota Press, Minneapolis, 1975) 344-369

[Tou58] S.E. Toulmin. *The Uses of Arguments*. (Cambridge Univ Press, Cambridge,1958).

[Win86] T. Winograd and F. Flores*: Understanding Computers and Cognition: A New Foundation for Design*.(Ablex, Norwood, New Jersey, 1986).

[VEG83] F.H. van Eemeren and R. Grootendorst. *Speech acts in Argumentative Discussions.* (Floris Publication,Dordrecht,1983).

APPENDIX A : Dynamics of Interaction Language

Table 1 presents the group of target communication actions. The occurrence of any of these actions determines the beginning of a design episode.

Communication Action	Meaning	Comment
i	raised a new issue	asked for decision (problem-solving)
<i	raised a former issue	decision already made is reviewed (the issue is revisited)
[i]	raised a nested issue	raised an abstraction (usually a generalization or specialization) of the current issue
q	made a query	asked for information

Table 1. Target communication actions

Table 2 we presents the group of communication actions in which speech objects are supplied.

Communication Action	Meaning	Comment
Sav	Supplied an alternative of value	to an issue (design parameter)
Ssav	Supplied selected alternative of value	re-statement of the alternative of value selected to be assigned to the parameter
Sf	Supplied additional or suppressive feature	elaborating an alternative of value
Sr	Supplied rationale	explanation, reason for preference, criterion
Sb	Supplied belief	expressing belief, opinion or indication of preference
Sv	Supplied value	any data or information
Sy	Supplied positive reply	for example: 'yes', 'I agree'
Sn	Supplied negative reply	for example: 'no', 'I disagree'
S∅	Supplied lack of knowledge, doubt or uncertainty	for example: 'I don't know', 'I'm not sure'
Su	Supplied understanding	expressing understanding
Snu	Supplied incomprehension	expressing incomprehension

Table 2. Supplying speech objects

Table 3 presents the group of communication actions in which speech objects are demanded.

Communication Action	Meaning	Comment
Da	Demanded agreement	in relation to a selection candidate alternative of value
Dc	Demanded confirmation	of the selected alternative of value

Table 3. Demanding speech objects

Table 4 presents the group of actions which represent interactions with physical objects in the setting.

Communication Action	Meaning	Comment
Wpri	Write on private workspace	taking notes on a personal notebook
Wpub	Write on public or shared workspace	on the blackboard or on flipcharts
B	Browse a document	for example: design documentation
R	Read a document	for example: personal annotations

Table 4. Interacting with workspaces

Table 5 presents a group of non-verbal communication actions used to express agreement, disagreement or acknowledgment of understanding.

Communication Action	Meaning	Comment
y	showing agreement, approval, understanding	nodding one's head
n	showing disagreement, disapproval	shaking one's head
∅	showing indifference or doubt	with a shrug of the shoulders

Table 5. Some non-verbal communication actions

Table 6 presents graphic symbols (arrows) used to describe verbal or non-verbal addressing mechanisms. A verbal addressing mechanism takes place whenever a design member explicitly names another member while providing some particular speech object (for example, while making a query, raising an issue or demanding agreement). A non-verbal addressing mechanism takes place whenever a design member makes a gesture, for example, points to or looks at another member instead of naming her.

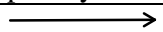
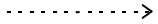
Graphic symbols	Meaning	Comment
	verbal addressing	by name
	non-verbal addressing	by gesture or look

Table 6 - Symbols to describe addressing mechanisms

Finally in table 7, we present three symbols (arrows) used to describe relations between speech objects. The first describes a speech object which refers to another speech object - for example, a positive reply being supplied in relation to a previously supplied alternative of value. The last two symbols are specialization of the first one. They are used to qualify rationales supplied in relation to alternatives of value, either for or against them.

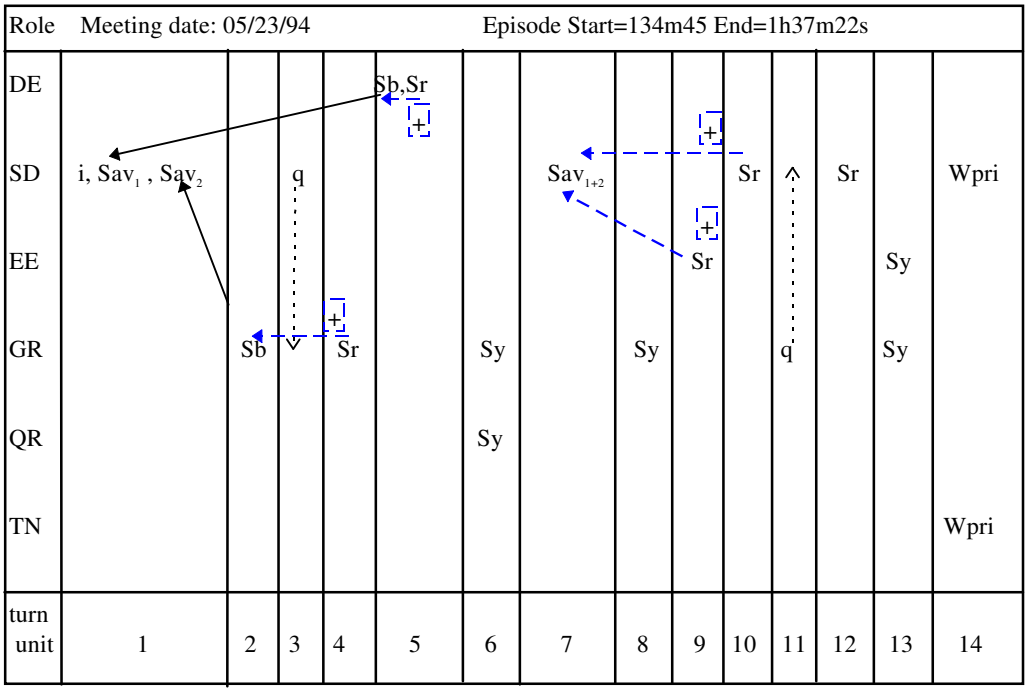
Graphic symbols	Meaning	Comment
	reference	between two speech objects
	supporting relation	rationale supporting alternative of value
	against relation	rationale against alternative of value

Table 7 - Symbols to describe relations between speech objects

APPENDIX B : Dynamics of Interaction Diagram

In this Dynamics of Interaction Diagram a sequence description (design episode) is presented in terms of the roles involved. The first column lists role players taking part in the design meeting. Subsequent columns present communication actions within turn units. If more than one communication action is taken within the same turn unit, all of them are represented. Each row presents communication actions performed by the corresponding role player (designer).

In the figure below we present a Dynamics of Interaction Diagram representing a genuine design episode. Role players present at the meeting were : Application Domain Expert (DE), Software Designer (SD), Education Expert (EE), Geoscience Representative (GR), Quality Control Representative (QC) and Technnical Annotator (TA). The Technical Annotator was a person not directly involved with the design itself, she was only responsible for making notes on whatever was considered important in relation to educational issues.



In this design episode, the Software Designer raises a new issue and also supplies two alternatives of value (Sav1 and Sav2) in the same turn unit (#1). Different designers express distinct preferences for different alternatives of value. The Geoscience Representative (GR) prefers Sav2 (as represented in turn unit #2) and, under request of

the Software designer (turn unit #3), supplies rationale supporting his preference (turn unit #4). The Application Domain Expert (DE), on the other hand, expresses preference for Sav1 and supplies rationale supporting her preference (turn unit #5). GR agrees (turn unit # 6) with the rationale provided by DE. Then, SD provides a new alternative of value Sav_{1+2} (actually a composition of the previous ones). Sav_{1+2} is justified in the following turn units by SD and by the Education Expert. In turn unit #11 GR demands more clarification. Designers express agreement to Sav_{1+2}. Finally, some designers made personal annotations. In actual fact they recorded the decision made, which is known because we had access to these notes *a posteriori*.