ELECTRONIC **WORKSHOPS IN COMPUTING**

Series edited by Professor C.J. van Rijsbergen

C.R. Roast and J.I. Siddiqi, Sheffield Hallam University, UK (Eds)

BCS-FACS Workshop on Formal Aspects of the Human Computer Interface

Proceedings of the BCS-FACS Workshop on Formal Aspects of the Human Computer Interface, Sheffield Hallam University, 10-12 September 1996

Dialogue Graphs - A Formal and Visual Specification Technique for Dialogue Modelling

E. Schlungbaum and T. Elwert

Published in Collaboration with the
British Computer Society

BCS
FACS

Springer

ISBN 3-540-76105-5

Dialogue Graphs - a Formal and Visual Specification Technique for Dialogue Modelling

Egbert Schlungbaum

Computer Science Department, University of Rostock
Rostock, Germany

Thomas Elwert

Computer Science Department, University of Rostock
Rostock, Germany

Abstract

This paper presents the formal definition and visual presentation of Dialogue graphs -- a visual specification technique for dialogue modelling. The Dialogue graphs are used in TADEUS to support an explicit dialogue modelling stage in a task-oriented and model-based approach to the development of interactive systems and automatic generation of the user interface code. The requirements of such a specification technique are discussed from different points of view. The formal definition of the Dialogue graphs is based on Coloured Petri net theory. The interactive tool for handling the Dialogue graphs is described shortly. The use of Dialogue graphs is demonstrated describing the OpenLook e-mail tool.

1 Introduction

When developing an interactive system the developer spends a lot of work and time on designing the graphical and window-oriented user interface. This task is difficult and error-prone and largely unsupported by methodologies and recent commercial software development tools. Furthermore, there are many high level tools, models, and techniques to support the early stages of user interface development, but these lack a close relation to the implementation stage of the user interface software and to the development of the application software.

Researchers are working to overcome the gap between Software Engineering and Human-Computer Interaction. Over the last years extensive research has focused on merging the user interface engineering with general software engineering approaches, what leads into the model-based approach to user interface development as one example. In this approach, developers work with high-level specifications (declarative models) of the desired user interface to define its layout characteristics and dynamic behaviour. One of the key ideas of model-based systems is that declarative models are used to automate user interface design tasks. There are several of different declarative models, e.g., data models of the application are used to generate the static layout (UIDE [6], GENIUS [15]), pre- and post-conditions added to the application model are used to generate the dynamic behaviour (UIDE [8, 25]), object models or domain models are used to generate both parts of the user interface (JANUS [2], MECANO [21]). But the application of these tools gives no guarantee for a task-oriented, user-centered design of the user interface. One of the aims of task-based techniques in user interface development is to meet this shortcoming (e.g., ADEPT [18], TRIDENT [5]).

The TADEUS approach (*TA*sk-based *DE*velopment of *US*er interface software) is a task-oriented and model-based approach to the development of interactive systems [7]. The main goal and rationale of the TADEUS approach include a continuous development process from the requirements analysis to the automatic generation of the user interface prototype, the development of four declarative models (task, user, problem-domain, and dialogue model) to specify the interactive system, the connection with software engineering methods and tools (the problem domain model corresponds to the OMT object model; existing UIMS are the target of the automatic generation). The user interface designer's work, which follows the TADEUS approach, is supported by the TADEUS system.

This article concentrates on Dialogue graphs which are a more usable formal and visual specification technique for dialogue modelling. They are used in TADEUS to support the explicit dialogue modelling stage and the automatic generation of the user interface code. The paper is organised as follows. In section 2 we argue the requirements of a dialogue model specification technique from different points of view. Section 3 gives a short overview on related work. In section 4, the main section of this paper, we introduce the Dialogue graph specification technique. Section 5 illustrates the use of Dialogue graphs describing the OpenLook e-mail tool. The paper concludes with further details of the new specification technique and a summary.

2 Requirements for a dialogue modelling technique

Focus on dialogue modelling

The specification of the user interface should be an integrated part of software development (e.g., [12, 24]). But recent software engineering methods, irrespective of whether they are structured or object-oriented, support the development of the application and give a very simple level of support for the design and specification of graphical and window-oriented user interfaces which satisfy software-ergonomics criteria.

Today, user interface development is mostly done through prototyping supported by high level tools like user interface development systems or user interface builders. The use of such tools makes the implementation of the user interface easier but does not support the design of the user interface, especially the course of dialogue (e.g., the description of navigation between windows). Furthermore, it is very difficult to implement correct code with an event-oriented user interface management system. The dialogue designer is insufficient supported to identify the application control structure, because of the control structure is implicit in event protocols and difficult to visualise.

Consequently, a support for dialogue modelling is necessary and should satisfy the following requirements (e.g., [13]):

- The specification of the graphical and window-oriented user interface should be easy to do and to understand. In TADEUS it should be possible to specify the navigation dialogue between windows and the processing dialogue on the user interface objects inside the windows.
- The specification should be precise. The behaviour of the system should be clear for each possible input.
- It should be easy to check for consistency, absence of deadlocks, and reachability in the dialogue specification.
- The specification technique should be powerful enough to express nontrivial system behaviour with a minimum of complexity, e.g. concurrency between windows, multiple instances of a window type, modal dialogue states, dynamic creation of user interface objects.
- It should separate what the system does (function) from how it does it (implementation). The technique should make it possible to describe the behaviour of a user interface, without constraining the way in which it will be implemented.
- It should be possible to construct a prototype of the system directly from the specification of the user interface. In TADEUS the prototype of the user interface is generated automatically.

These requirements of a dialogue modelling technique can be fulfilled by a formal specification.

Focus on formal specification

Formal specification techniques have been applied to many aspects of software development. They permit a designer to describe the external behaviour of a system precisely without specifying its internal implementation. A specification allows the formal description of requirements and design ideas at different levels of abstraction expressed by a particular notation.

Several reasons justify the importance of formal specification for interactive system design (e.g., [11, 13]). The main reason arises from the cost-intensive nature of user interface implementation in the interactive system development process.

Formal specification can help to reduce the communication effort between designer and end-user by providing a common ground for discussion of design decisions. The use of formal specification allows to get closer to the desired user interface from an abstract point of view step by step through refinement. A formal specification permits formal analysis for the user interface evaluation in early stages of the design process. The designer can check absence of deadlocks and the validity of design decisions by reasoning about the specification instead to wait for a finished interactive software system.

Formal specification should be used for the purpose of correct implementation, if possible in an automatic generation process. In this way, it can contribute to more consistent and standardised user interfaces. This can support team work in the design process and the formalisation of design principles of the designer for further use.

The success of a specification technique depends on its simplicity and powerfulness, on its precise defined syntax and semantics, and on its tool support.

As shown, a formal specification can support the design process of an interactive system but simultaneously many software designer have problems to use kinds of formal specification. The reasons are various and show that formal specification have to become easier in use and application.

Focus on visual representation

In general, formal specifications are expressed in two different forms, as a textual or a graphical notation. In the context of specifying systems requirements, many authors argue that graphical notations are better at the higher

levels of abstraction, and that textual notations more suited to providing lower level, detailed description. For example, Tse et al [26] considers that graphical representation of complex material is much more comprehensible than its textual counterpart, because

- it can show naturally both hierarchy and concurrency;
- it can be read selectively rather than linearly;
- it reduces the number of concepts to be held in short term memory;
- the reader can move naturally from higher to lower levels of detail.

Harel [10] suggests that specifications, including behavioural aspects, should be based on visual formalisms depending on a small number of carefully chosen diagrammatic paradigms. This message is very important because in any visual formalisms is a trade-off between the visual complexity of the graphical notation and the effort required to use it. As the visual complexity increases, the visual effort required to use the notation effectively also increases.

This suggests an approach to interface design in which a high level graphical specification is augmented by non-graphical detail, and subsequently mapped into a more formal dialogue specification language for implementation; that is an approach based upon multiple specifications. At the highest level of design what is needed is a visual vocabulary suitable for sketching the interface, a representation able to capture both user and system interaction with the visual display.

The main thesis of this section (see also [10]) is that the intricate nature of an interactive system can, and in our opinion should, be represented by a visual formalism: *visual*, because they are to be generated, comprehended, and communicated by humans; and *formal*, because they are to be manipulated, maintained, and analysed by computers.

3 Related work

Specification techniques for user interfaces are reported in a lot of papers over the last decade (e.g., [1, 4, 9, 11, 20]). Our observation will be limited on visual formalisms in conformity with the requirements discussed above.

The transition networks and their extensions are well known, frequently used visual dialogue specifications. They tend to emphasis the states of the user interface and the sequence of transitions from one state to another. Certainly transition networks are unsuited to the description of window-oriented and graphical user interfaces. They are useful and powerful for the specification of non-graphical user interfaces, but they have to be modified in order to handle direct manipulation within a user interface. Jacob [14] extends state transition diagrams by multiple parallel diagrams. However, the global relations of concurrent dialogues cannot be visualised by this approach.

A further approach to visual specification of user interfaces is the use of Petri nets or Petri net based techniques, like the event graphs [22], the Petri net objects [3, 19], or the dialogue nets [15, 16]. Compared to the user interface specification with transition networks, the advantage of Petri nets is that they already have a concept for the specification of concurrency in the basic form. That is needed for the description of window-oriented graphical user interfaces. In addition, another important point of all these approaches is that the user interface software code can be directly created by using the dialogue specification.

Event graphs can be used to describe the dynamic visibility of objects within a graphical user interface. Compared with Dialogue graphs introduced later on, the specification with event graphs is more expensive and is not so easily surveyed, because the places of the event graph are not in correspondence with the user interface objects. There are not means for dialogue structuring and for visualisation of global relations between separate event graphs describing one user interface.

Petri net objects are more powerful for the description of the general object flow and the object manipulation. But they are more complex than event graphs and less suited for the earlier design phases. Petri net objects are based on high-level Petri nets and so the dynamic creation of user interface objects is natural to model.

Dialogue nets are similar to event graphs. Compared with event graphs, they offer some extensions like hierarchical dialogue structuring, declaration of modal dialogue windows, declaration of macros of dialogue structures, and dynamic creation of user interface objects by dynamic repeated execution of sub nets. The dialogue nets are based on the condition/event nets, which are working with a single unnamed token. From this, it is clear that the modelling of multiple instances of a window is difficult and violates our requirement of powerfulness for expressing non-trivial system behaviour with a minimum of complexity. Our Dialogue graphs are an extension of dialogue nets and based on Coloured Petri Nets (CPN) [17].

From our point of view, Coloured Petri Nets are a suitable visual formalism to fit our requirements on formal dialogue specification. But many user interface designers are not used to work with CPN, a lot of authors argue that it is better to use a more simple notation, e.g. the condition/event-nets [15]. Difficulties come from the complexity of the CPN elements such as node function, guard functions, arc expressions, transition functions, or colour functions. We decide to develop an abstraction level higher than a CPN, called Dialogue graphs, which is understandable for the dialogue designer and expressive enough for dialogue modelling.

4 Description of Dialogue graphs

4.1 Excursus: Dialogue modelling within the TADEUS-approach

The user interface and the application core of an interactive system can be developed independently by means of TADEUS. The user interface development process is divided into three stages [7]. In the first stage, the requirements analysis, the designer specifies three domain models (task, problem domain, and user model) which contain the requirements for the desired user interface. In the second, the dialogue design stage, the designer develops the dialogue model, the initial form of which is generated from the already created domain models. The dialogue model describes the static characteristics and the dynamic behaviour of the user interface. The third stage is the automatic generation of the prototype of the final user interface by using the specified models, a software ergonomics knowledge base, and auxiliary dialogue with the dialogue designer in order to request non-specified information.

The development of the dialogue model starts with the definition of dialogue views. A *dialogue view* is a collection of tasks defined in the task model and/or objects defined in the problem domain model. These tasks and objects are put together because of the end user needs their user interface representation to perform a certain task. A dialogue view can be seen as a class definition. That means a dialogue view can have one or more instances (see example below). One instance of a dialogue view becomes one window of the final user interface while the automatic generation stage. A dialogue view instance has essential properties which are important for the modelling of the dynamic behaviour of the user interface. A dialogue view instance can be *visible*, that means the corresponding window is visible on the screen. A dialogue view instance can be *active*, that means the corresponding window is visible and possesses the input focus (all end user interactions are directed to objects inside this window). And it can be manipulable, that means the corresponding window is visible and can be activated.

The TADEUS dialogue model distinguishes between two different types of dialogue, the navigation and the processing dialogue. The *navigation dialogue* describes the dialogue between different dialogue views. It can be specified by means of Dialogue graphs. The *processing dialogue* deals with the description of the dialogue within a view and is expressed by interaction tables.

Now, let's introduce our example. Figure 1 gives a snapshot of a situation using the OpenLook e-mail tool. The end user (in this example Thomas) has opened the main window, two mail viewer windows, and a mail composer window. He works on an answer to Egbert's e-mails while preparing this article. Thomas tried to send the message, but he forgot the recipients address (usually he uses the reply function). That is why he got the message window (a modal dialogue, which must be finished first). While composing the new message Thomas was able to navigate freely between all opened windows.

4.2 Views and transitions of a Dialogue graph

A Dialogue graph consists of nodes and directed arcs between these nodes. Each node represents a dialogue view. The directed arcs represent transitions, which cover the possible interactions between the connected views.

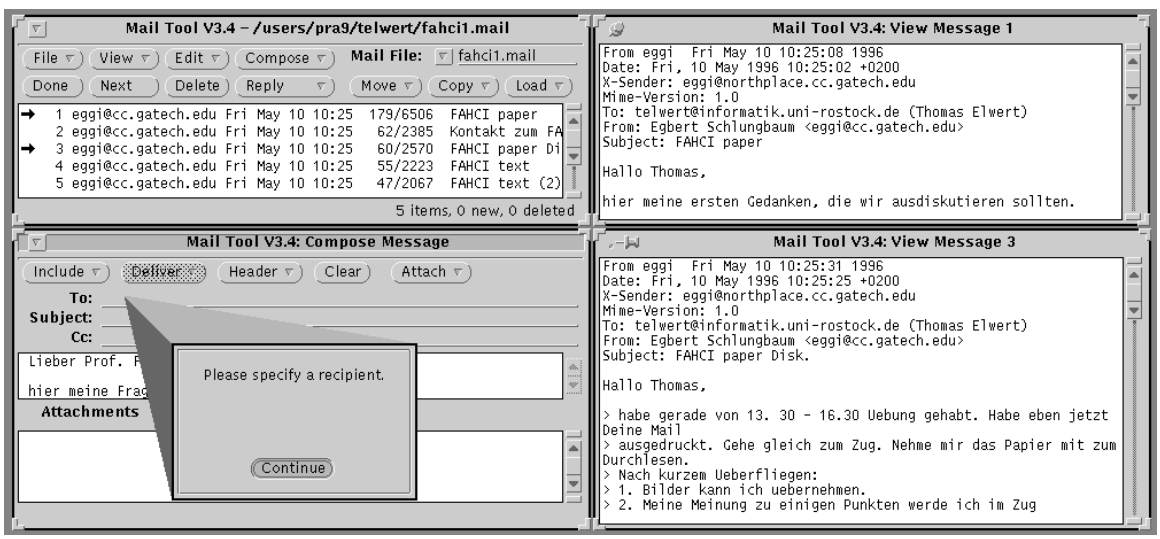

Figure 1: User Interface of the OpenLook e-mail Tool

The views are drawn as circles or ellipses labelled with a name. There are single (non-modal), multi (non-modal), modal and complex views. The non-modal views contain neither refinement nor interaction limitations unlike the modal view. The modal view is a starting point for a modal dialogue. If the end user achieves this view by an interaction he must first complete this modal dialogue in order to continue the other dialogue. A complex view allows the dialogue designer to refine a dialogue view. The difference between a single and a multi view is that a single view has only one instance but the multi view can have one or more instances. The instances of a multi view have all the same layout characteristics but they represent different objects of the same type. Furthermore, there is an end node which does not represent any dialogue view. Therefore, it is drawn as a black point. This node is necessary to complete the graph structure.

The end user can navigate from a source view V_1 to the destination view V_2 by interactions represented by directed transitions. The direction of the transition is additionally explained with the black point near by the destination view. Between two nodes (views) there can exist a concurrent or a sequential transition. The transitions of the Dialogue graph describe explicit the change of the view visibility. The behaviour of the transitions is described with control rules below. The representation of views and transitions are shown in figure 2.

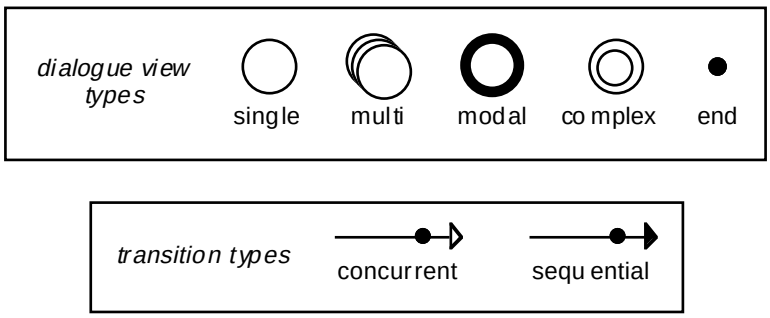

Figure 2: Views and Transitions of a Dialogue Graph

Definition 1:
A Dialogue graph is defined as a directed graph DG=(DV,DVT,T,TT,n,Nm) with:

DV is a finite set of nodes, called dialogue views and DVT={single, multi, modal, complex, end} is a set of dialogue view types: $\forall dv \in DV$: TYPE(dv)$\in$DVT.

T is a finite set of transitions and TT={concurrent, sequential} is a set of transition types: $T \subseteq V \times V$; $\forall t \in T$: TYPE(t)$\in$TT.

Nm is a finite set of names and n a label-function $n: DV \cup T \rightarrow Nm$.

4.3 Modelling the behaviour of a user interface

In order to specify the state of a user interface Dialogue graph views are marked. The marking of a view describes its state (in detail: the state of the user interface window associated with a concrete instance of a dialogue view). The Dialogue graph with all it's views and their markings describes the total state of the modelled user interface. The tokens represent the properties of a dialogue view:

- activity (a) describes whether a view instance has the input focus or not,
- visibility (v) describes whether a view instance is visible or not,
- manipulability (m) describes whether a view instance is manipulable or not by the end-user.

This properties are modelled by tokens and their value. Furthermore each instance of a dialogue view possesses an identifier, the fourth token. This token allows to specify multiple instances for a view. The view instances are known at the design time or are created at run time dynamically. All four tokens form a tuple that resides in a dialogue view. The tuple exists for each instance of a view and describes the state of the dialogue view instance. The existence of tuples on a view with particular token values is controlled by the following *existence rules*:

1. Only one dialogue view instance can be active in the Dialogue graph at the same time. That means only one tuple with an a-token can exist in the Dialogue graph.
2. If a dialogue view instance is active then it is also visible and manipulable.

3. If a dialogue view instance is visible and manipulable then this view instance can become active through an end user interaction[1].
4. If a dialogue view instance is visible but not manipulable then it cannot be activated. That means a modal dialogue view is active.
5. A non-marked dialogue view in the Dialogue graph means that there is no instance of the view.

Now we must extend the definition of the Dialogue graph DG in order to describe the behaviour of a user interface. For it we define a finite set of tuples. The elements of this set will be used to mark the dialogue views:

Definition 2:

Marking of nodes is defined as a finite set TU with: $TU \subset \Sigma$, Σ is the finite set $\Sigma = A \times V \times M \times Id \mid A=\{a,\_\}, V=\{v,\_\}, M=\{m,\_\}, Id=\{0, ..., n\} \mid n \in N$;

In consideration of the existence rules there are some valid examples of marking tuples $tu \in TU$:

- active instance of a view: $(a,v,m,X) \mid X \in Id$
- visible and manipulable instance of a view: $(\_,v,m,X) \mid X \in Id$
- all other visible instances of views, if a modal dialogue is activated: $(\_,v,\_,X) \mid X \in Id$.

The definition of the general Dialogue graph is derived as an extension of Definition 1 considering Definition 2:

Definition 3[2]:

A Dialogue graph is a tuple $DG=(DV, DVT, T, TT, n, Nm, S, TU, C, M_0, INIT, \Omega, GF)$ with:

C $DV \rightarrow TU$ is a colour function which maps each view into a set TU;

M_0 is a finite set, which contains for each $dv \in DV$ the initial marking $m_0 \in M_0$, $m_0 \in TU$;

INIT $DV \rightarrow M_0$ is an initialisation function, which maps each view into a set $M_0 \subset TU$;

Ω is a finite set of interaction events[3];

GF is a guard function and maps each transition $t \in T$ into an expression of type Boolean. The expression evaluates to TRUE, if the pre-conditions described in the control rules are satisfied. Pre-conditions include the marking on source and destination views and the occurrence of the assigned interaction event $\varpi \in \Omega$.

4.4 Control rules - semantics of transitions

Each transition type has its own behaviour which is described by control rules. They are explained in verbal form. For the explanation we assume that V_1 is the source view and V_2 is the destination view. The textual description of each transition can be specified by a CPN. There the interaction events are described by transitions labelled IT. Additionally, each control rule is explained with a state table in which time t_1 describes the state before and t_2 after occurrence of the related transition.

Concurrent transition: The view V_1 is active and the view V_2 is non-marked or visible and manipulable. After the execution of a related event, the concurrent transition is carried out and the view V_1 will become visible and manipulable and the view V_2 will become active (figure 3).

Sequential transition: The view V_1 is active and the view V_2 is non-marked. After the execution of a related event, the transition is carried out and the view V_1 will become invisible. The view V_2 will become active (figure 4).

[1] This change of activity is not expressed by a special transition in the Dialogue graph; but there is not any lose of information because of this property is very clear recognisable by means of the simulation tool (see section 6 below).

[2] We reuse the specifications from previous definitions and describe only new information.

[3] An interaction event can be external (end user interaction) or internal (application event).

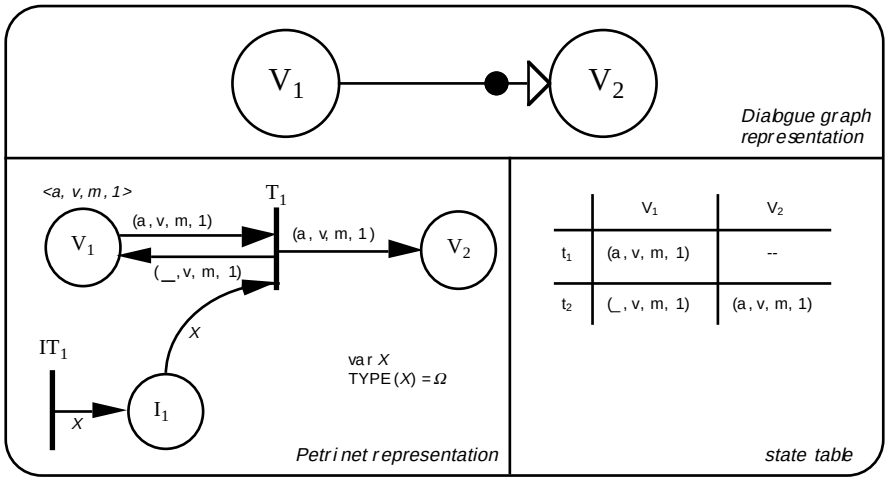

Figure 3: Concurrent Transition

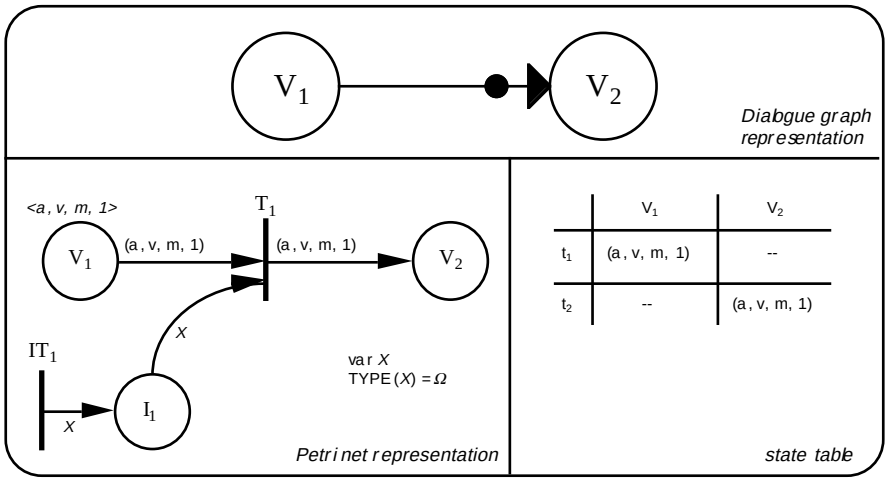

Figure 4: Sequential Transition

4.5 Special views and their transitions

In the paragraph above, the semantics of transitions were described for transitions between single nodes. In this paragraph, we will concentrate on some special transitions. Firstly, we describe the transitions around a modal dialogue view.

A modal view describes a starting point for a modal dialogue in the navigation dialogue. The description of a modal dialogue is necessary if the end user has to carry out a special task or to confirm a message. In this case it must be impossible to change to another instance of a dialogue view. The end user must finish all interaction in the modal dialogue and only then he can work in other dialogue view instances.

For the Dialogue graph we can specify control rules describing the change of tuples (and with it the state of the view instances) if the end user reaches a starting point of a modal dialogue:

- If a tuple with an `a`-token reaches a modal view, all views of the entire Dialogue graph which have a `m`-token lose this token. This rule is controlled by the transition function $TF_{modal\text{-}in}$.
- After the modal dialogue, the view instances which lost a `m`-token from a tuple get the `m`-token back. This rule is controlled by the transition function $TF_{modal\text{-}out}$.
- There is no way to move an `a`-token from a tuple of a view instance to a view instance which is not in the modal dialogue (see existence rules 3 and 4).

With it, it is necessary to extend definition 3:

Definition 3a[4]:
A Dialogue graph is a tuple DG=(DV,DVT,T,TT,n,Nm,S,TU,C,M_0,INIT,Ω,GF,TF) with:

TF $TU \rightarrow TU$ is a transition function, which can be assigned to t, $t \in T$ and is executed if t occurs;
TF_t //TF assigned to the transition t, $t=(dv_i,dv_j)$// can have an effect on all $dv \in DV \setminus \{dv_i, dv_j\}$

Now we can define the transition functions $TF_{modal\text{-}in}$ and $TF_{modal\text{-}out}$ as specialisations of the general transition function TF.

Definition 4:

$TF_{modal\text{-}in}$ $\{(\_, v, m, X)\} \rightarrow \{(\_, v, \_, X)\}$, $X \in Id$ is assigned to all transitions $t, t \in T$ with $t=(dv_i, dv_j)$, $dv_i, dv_j \in DV$, TYPE(dv_j)=modal, $TF_{modal\text{-}in}$ can have an effect on all $dv \in DV \setminus \{dv_i, dv_j\}$

$TF_{modal\text{-}out}$ $\{(\_, v, \_, X)\} \rightarrow \{(\_, v, m, X)\}$, $X \in Id$ is assigned to all transitions $t, t \in T$ with $t=(dv_i, dv_j)$, $dv_i, dv_j \in DV$, TYPE(dv_i)=modal, $TF_{modal\text{-}out}$ can have an effect on all $dv \in DV \setminus \{dv_i, dv_j\}$

A modal dialogue view can be integrated by a concurrent and a sequential transition as shown in figure 5. Transitions functions are assigned to both transitions. The modal-in function is attached to the concurrent transition and the modal-out function to the sequential transition. The assignment of the transition function to the appropriate transition in case of a modal dialogue is automatically done by the TADEUS system. In the example in figure 5 we present one possibility of a modal dialogue situation: the end user works inside the source view V_1, the modal dialogue occurs on the screen (e.g., an error message[5]), the end user confirms the message and continues the interaction inside the source view V_1.

Modal view and its transitions: The view V_1 is active and the modal view V_2 is non-marked. The modal dialogue becomes active and the source view is still visible but not manipulable. In order to achieve the modal dialogue for the whole Dialogue graph the transition function $TF_{modal\text{-}in}$ is assigned to the transition T_1 which is executed when the transition T_1 occurs. The interaction for finishing the modal dialogue is modelled by the sequential transition from V_2 to V_1. The transition function $TF_{modal\text{-}out}$ is assigned to the transition T_2.

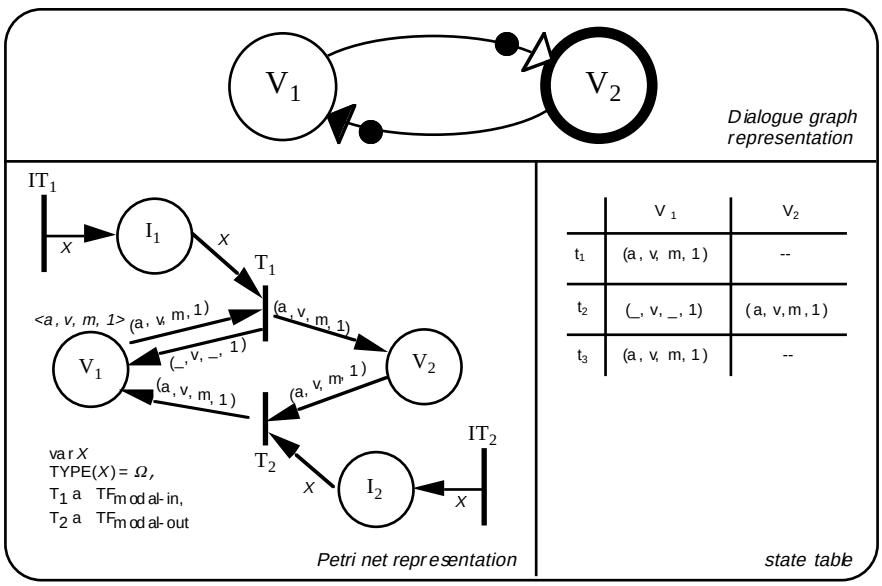

Figure 5: Transitions around a Modal View

The existence of the multi view demands a specialisation of the concurrent transition into an Id-preserving concurrent transition from a single to a multi view and between two multi views, and an Id-independent concurrent transition

[4]We reuse the specifications from previous definitions and describe only new information.

[5]Referring to our example the error message occurs as a result of an internal event. This event was generated after the end user's interaction Send mail without recipients address.

from a multi to a single view. In figure 6 we explain the multi view in combination with a concurrent transition due to it is used in our example. Because of the lack of space we do not describe the other in detail.

Multi view with concurrent transition: The behaviour of this transition is similar to the concurrent transition. After the related end user interaction, the views V_1 and V_2 are visible. An important property of this transition is the possibility to create more than one instances of the view V_2 dynamically at runtime. Each instance of view V_2 represents one object (instance-window) of a related class. The end user interaction for the firing the transition is connected with the selection of an object in view V_1 at runtime.

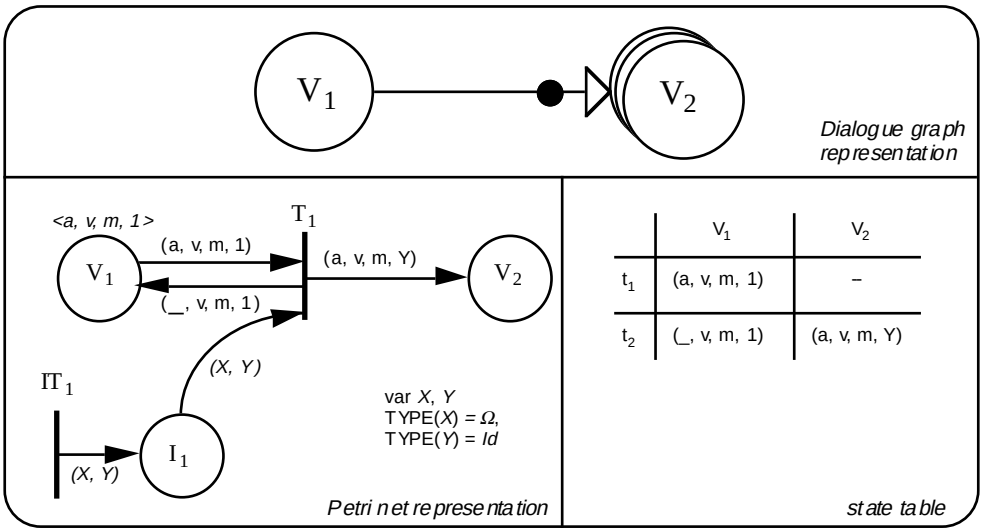

Figure 6: Multi View with Concurrent Transition

The dialogue designer can describe the closing of a user interface by the usage of the end node. This end node does not represent any actual view of the desired user interface. The end node is always connected via a sequential transition with the view where the closing is initiated. The behaviour of a sequential transition with a end node can be described as follows. Its source view is active. If this transition fires, the current dialogue view instance will be deleted together with all other existing dialogue view instances. Therefore the transition function $TF_{closing}$ is defined which removes the tokens from all other visible view instances which are not connected by the transition to the end node.

Now we can extend the definition 4:

Definition 4a:

$TF_{closing}$ $\{(\_, v, m, X)\} \rightarrow \emptyset$, $X \in Id$ is assigned to all transitions t, $t \in T$ with $t=(dv_i, dv_j)$, $dv_i, dv_j \in DV$, TYPE(dv_j)=end, $TF_{closing}$ can have an effect on all $dv \in DV \setminus \{dv_i, dv_j\}$

This paragraph introduced transition functions which extend the behaviour of the sequential and concurrent transition to fit some special requirements. These requirements arise from modal and multi dialogue views and the end node.

4.6 Specification of navigation dialogue

In order to specify the navigation dialogue the designer uses an Abstract Dialogue Graph (ADG). The abstract Dialogue graph is a Dialogue graph without loops. On definition 3 and their extension 3a the abstract Dialogue graph is defined as:

Definition 5:

An ADG is a tuple $ADG=(DV, DVT, T, TT, n, Nm, S, TU, C, M_0, INIT, \Omega, GF, TF)$ with:

$t=(dv_i, dv_j)$, $t \in T \subset DV \times DV$, $i \neq j$.

5 Example

The use of the Dialogue graph specification will be explained on the OpenLook e-mail tool. It was introduced in paragraph 4.1 (see figure 1). The example shows how the Dialogue graph can describe the dynamic behaviour of this e-mail tool comprehensibly.

In figure 1 the reader can see three typical windows of this e-mail tool. There is the mail administration window, the mail viewer windows and the mail composer window. Furthermore, there is one message window. Each of these window types can be represented by a dialogue view: view `mail administrator`, view `mail viewer`, view `mail composer`, and view `message`. Figure 7 shows these views, the specification of transitions between them, and the initial marking of the Dialogue graph. The `mail viewer` and the `mail composer` are multi views. That means the end user can use one or more instances of these views. In figure 1 it was shown for the `mail viewer`, the end user uses two incoming mails (instances of the `mail viewer` view) to write one answer (instance of the `mail composer` view).

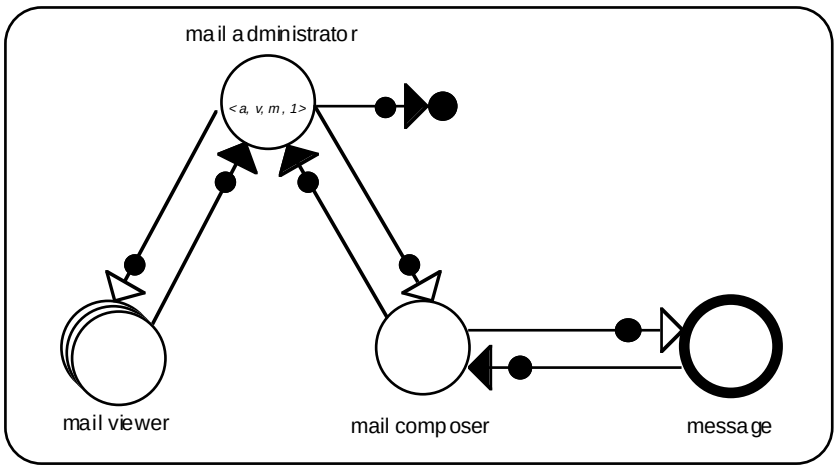

Figure 7: Dialogue Graph Specification of the OpenLook e-mail Tool

The state of each instance of a view can be expressed by the related tuple. The initial marking describes the initial state of the Dialogue graph. In this state one instance of the `mail administrator` view is active, visible and manipulable (state at time t1 in table 1). The end user can select one mail in order to read this mail in detail. As the result of this interaction event (selection of the mail) the `mail viewer` view will be marked by an instance-related tuple (a,v,m,<chosen mail>), that means an instance of the view will be created (state at t2). This instance is related to the chosen mail. If there are more incoming mails the end user can repeat this interaction and will get further instances of the `mail viewer` view. But for it it is necessary that at first the end-user moves to `mail administrator` view back (states at t3 and t4). The end user can finish the dialogue with an instance by an interaction. That will remove the instance-related tuple. This instance does not exist anymore. The same behaviour is specified for the `mail composer` view. According to figure 1 the end user works with one instance of `mail composer` view (states at t5 and t6).

In table 1 the state t7 describes the state of the Dialogue graph after occurrence of the modal dialogue. Now all views excluding the `message` view are visible only. That means the end user is not able to continue the dialogue inside one of the instances of `mail administrator`, `mail viewer`, or `mail composer` views. A modal dialogue is active (see rule 4 of existence rules). The state t8 describes the state of the Dialogue graph after finishing the modal dialogue.

Time	Mail admin.	Mail viewer	Mail composer	Message
t1	(a,v,m,1)	-	-	-
t2	(_,v,m,1)	(a,v,m,1)	-	-
t3	(a,v,m,1)	(_,v,m,1)	-	-
t4	(_,v,m,1)	(_,v,m,1),(a,v,m,2)	-	-
t5	(a,v,m,1)	(_,v,m,1),(_,v,m,2)	-	-
t6	(_,v,m,1)	(_,v,m,1),(_,v,m,2)	(a,v,m,1)	-
t7	(_,v,_,1)	(_,v,_,1),(_,v,_,2)	(_,v,_,1)	(a,v,m,1)
t8	(_,v,m,1)	(_,v,m,1),(_,v,m,2)	(a,v,m,1)	-

t9	(a,v,m,1)	(_,v,m,1),(_,v,m,2)	(_,v,m,1)	-
t10	-	-	-	-

Table 1: Scenario using the OpenLook e-mail Tool: State Table of the Dialogue Graph

The mail tool can be exited by using the closing transition (quit interaction). For it the end user has to move to the mail administrator view in order to select the quit interaction (state t9). The state t10 describes the situation after occurrence of the closing transition; the work with the OpenLook e-mail tool was finished and all windows were removed from the screen.

As mentioned above, the exclusive change of activity between visible and manipulable views is not described by transitions. This behaviour is provide by the existence rule 3 (see paragraph 4.3).

6 Further details of Dialogue graphs

Complex dialogue refinement

In order to create and to develop complex and detailed dialogue description, it is necessary to refine a given Dialogue graph. This task is supported by a special view - the complex view. The dialogue designer can create a Sub-Dialogue graph (SDG) and assign this SDG to a complex view. Currently we are working on a method which support a well-defined refinement process.

Description of the processing dialogue

The processing dialogue covers the navigation within a dialogue view. Currently we use interaction tables to describe this dialogue. An extended description is given in [7]. In the future we want to substitute the interaction table by an adapted Dialogue graph.

User Interface code generation from Dialogue graphs

As mentioned in the requirements a formal specification for user interface development here for dialogue modelling should allow the generation of user interface code. In [23] a simple method was offered for the generation of dynamics. A generation pattern which describes the relation between two views was created for each transition type.

In the future we will support this by a knowledge-based generation tool which collaborates with the designer. The architecture with a formal specification editor on the one side and a generator on the other side allows to provide different generators for different platforms.

Tool support for manipulating Dialogue graphs

Currently we are working on a tool which supports the modelling, simulation and formal analysis of the Dialogue graph. This Dialogue graph editor is embedded into the TADEUS environment. The figure 8 gives a short impression from the editor.

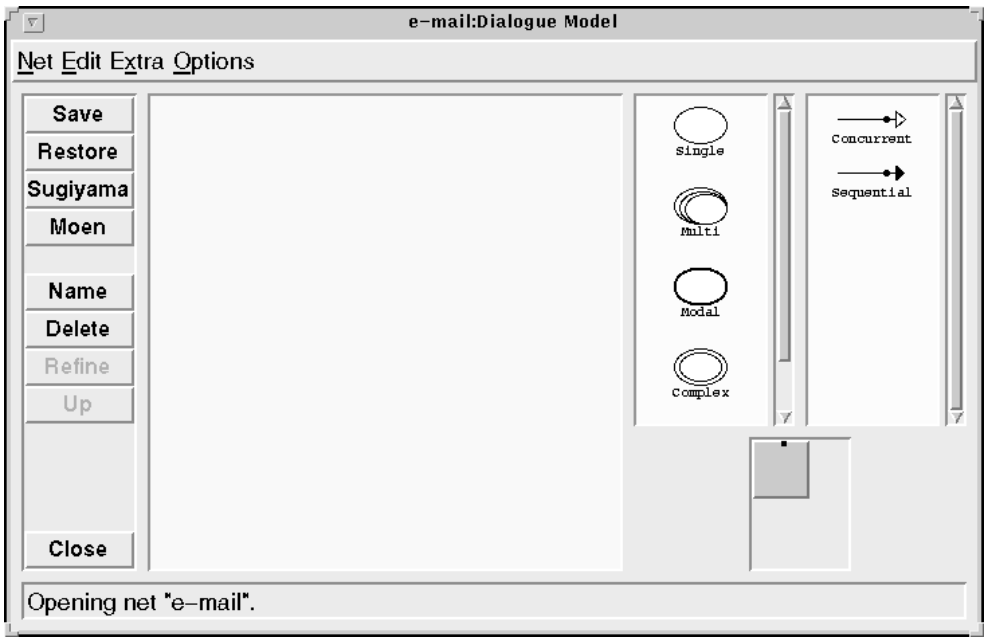

Figure 8: Dialogue Graph Editor in the TADEUS Environment

Verification of usability properties of the desired user interface

If the dialogue designer uses the Dialogue graphs he can check some usability properties of the desired user interface. This is because the Dialogue graphs are a formal specification technique. There are two possibilities to do this check. On the one hand, the dialogue designer can use the simulation tool or on the other hand, he can use formal analysis tool.

If a Dialogue graph consists of a small number of nodes (dialogue views; no more than 10) it is sufficient to use the simulation tool of the Dialogue graph editor in order to compare the designed user interface behaviour with the expected one. To allow the designer to percept the state of the Dialogue graph at a glance we code the states of views with only one instance by colour. The presence of activity can be expressed by a red coloured background of the view icon. The presence of visibility can be coded by blue colour and non-manipulability by grey. With the simulation tool, the designer can go step by step through the Dialogue graph and can compare the current with the desired behaviour. The development of the simulation tool is not finished yet. It is necessary to extend in order to handle multi and complex views.

If the number of nodes is larger than 10 the dialogue designer needs tool support to analyse the Dialogue graph. Using the state graph it is possible to check the reachability of all dialogue views (windows in the final user interface), the absence of deadlocks (the end user can always carry out an interaction, independently of the previous ones), the liveliness (from a dialogue view each interaction can be repeated after a finite number of interactions), and the number of opened windows the end user needs to perform a task. This formal analysis tool is not implemented yet.

7 Conclusions

In this paper we have introduced the Dialogue graphs which are a formal and visual specification technique for dialogue modelling.

On the basis of the Dialogue graph the dialogue designer has the possibility to specify the navigation dialogue on a level which is detailed enough to describe the most common dialogue sequences between views and is abstract enough to get an overview about the desired user interface in an early stage of the user interface development process. Furthermore the Dialogue graph gives the possibility to evaluate the graph by means of simulation and formal analysis and to generate the user interface code automatically.

In the following we summarise the advantages of the Dialogue graph notation:

- representation of possible interactions by a manageable set of transitions;
- description of important properties of a user interface by tokens and view types;
- possibilities for simulation and formal analysis;

- starting point for the automatic generation of user interface code;
- graphical notation allow to get a quick overview about the dialogue structure.

Up to now our Dialogue graphs were tested by designing some user interfaces of window-oriented information systems on paper. These tests give a good evaluation of the communication between the dialogue designer and the end user of the desired user interface.

8 Acknowledgements

The authors would like to thank Peter Wright, Gregory Abowd, and Kurt Stirewalt who helped to improve an earlier version of this paper. Many thanks also to the anonymous reviewers for their detailed and helpful comments.

9 References

[1] Abowd G D, Dix A J. Integrating status and event phenomena in formal specifications of interactive systems. In: Proceedings of the ACM SIGSOFT'94, New Orleans, 1994

[2] Balzert H. From OOA to GUI - The JANUS-System. In: INTERACT'95 Conference Proceedings. Chapman & Hall, London, 1995, pp 319-324

[3] Bastide R, Palanque P. Petri nets with objects for the design, validation and prototyping of user-driven interfaces. In: INTERACT'90 Conference Proceedings. Elsevier Science Publishing, Amsterdam, 1990, 625-631

[4] Bay C. Analyse von Spezifikationsmitteln für graphische Dialoge zur Weiterentwicklung des THESEUS-Dialogmodells. Studienarbeit, FB Informatik, TH Darmstadt, Darmstadt, 1988

[5] Bodart F, Hennebert A-M, Leheureux J-M, Provot I, Vanderdonckt J. A Model-based Approach to Presentation: A Continuum from Task Analysis to Prototype. In Paterno F (ed) Designing, Specification, and Verification of Interactive Systems (Proceedings DSV-IS'94). Springer-Verlag, Berlin, 1995, pp 77-94

[6] deBaar D, Foley J D, Mullet K. Coupling Application Design and User Interface Design. In: Proceedings CHI'92. ACM Press, New York, 1992, pp 259-266

[7] Elwert T, Schlungbaum E. Modelling and Generation of Graphical User Interfaces in the TADEUS Approach. In: Palanque P, Bastide R (eds) Design, Specification and Verification of Interactive Systems '95 (Proceedings DSV-IS'95). Springer-Verlag, Wien, 1995, pp 193-208

[8] Gieskens D, Foley J D. Controlling User Interface Objects Through Pre- and Postconditions. In: Proceedings CHI'92. ACM Press, New York, 1992, pp 189-194

[9] Green M. A Survey of Three Dialogue Models. ACM Transaction on Graphics 1986; 5:245-275

[10] Harel D. On visual formalisms. Communications of the ACM 1988; 31:514-530

[11] Harrison M D, Duke D J. A Review of Formalismus for Describing Interactive Behaviour. In: Taylor R N, Coutaz J (eds) Software Engineering and Human-Computer Interaction. ICSE '94 Workshop. Springer-Verlag, Berlin, 1994, pp 49-75

[12] Hartson H R, Hix D. Toward empirically derived methodologies and tools for human-computer interface development. International Journal on Man-Machine Studies 1989; 31: 477-494

[13] Jacob R J K. Using Formal Specifications in the Design of a Human-Computer Interface. Communications of the ACM 1983; 26:259-264

[14] Jacob R J K. A Specification Language for Direct Manipulation User Interfaces. ACM Transaction on Graphics 1986; 5:283-317

[15] Janssen C. Dialognetze zur Beschreibung von Dialogabläufen in graphisch-interaktiven Systemen. In: Rödiger K-H (ed) Software-Ergonomie '93 - Von der Benutzungsoberfläche zur Arbeitsgestaltung. Teubner-Verlag, Stuttgart, 1993, pp 67-76

[16] Janssen C. Dialogentwicklung für objektorientierte, graphische Benutzungsschnittstellen. Springer-Verlag, Berlin, 1996 (PhD thesis, University of Stuttgart)

[17] Jensen K. Coloured Petri Nets: A High Level Language for System design and Analysis. In: Jensen K, Rosenberg G (eds) High-level Petri Nets. Springer-Verlag, Berlin, 1991, pp 44-119

[18] Johnson P, Johnson H, Wilson S. Rapid Prototyping of User Interfaces Driven by Task Models. In: Carroll J (ed) Scenario-Based Design. Wiley & Sons, London, 1995, pp 209-246

[19] Palanque P, Bastide R, Senges V. Automatic Code Generation From a High-Level Petri Net Based Specification of Dialogue. In: Proceedings of EWHCI'94. St. Petersburg, 1994

[20] Phillips C H E. Review of graphical notations for specifying direct manipulation interfaces. Interacting with Computers, 1994; 6:411-431

[21] Puerta A R, Eriksson H, Gennari J H, Musen M A. Beyond Data Models for Automated User Interface User Interface Generation. In: People and Computers IX HCI'94 Conference Proceedings, Cambridge University Press, Cambridge, 1994, pp 353-366

[22] Roudaud B, et.al. A New Generation UIMS. In: INTERACT'90 Conference Proceedings, Elsevier Publishing, Amsterdam, 1990, pp 607-612

[23] Schlungbaum E, Schmidt M. Automatische Erzeugung von Beschreibungen für Benutzungsoberflächen. Rostocker Informatik Berichte 1994; 15:97-106

[24] Sutcliffe A G, McDermott M. Integrating methods of human-computer interface design with structered systems development. International Journal on Man-Machine Studies 1991; 34:631-655

[25] Sukaviriya P, Foley J D, Griffith T. A Second Generation User Interface Design Environment: The Model and The Runtime Architecture. In: Proceedings INTERCHI'93. ACM Press, New York, 1993, pp 375-382

[26] Tse T, Pong L. An examination of requirements specification languages. Computer Journal 1991;34:143-152