Refining Formal Specifications of Human Computer Interaction by Graph Rewrite Rules

Bettina Eva Sucrow

Data Management Systems and Knowledge Representation
Dept. of Mathematics and Computer Science
University of Essen, 45 117 Essen, Germany
sucrow@informatik.uni-essen.de

Abstract. Human computer interaction can be specified successfully using the concept of information resources and the formal notation of graph grammars. In order to achieve a precise and continuous specification process between the requirements and design stages, however, a suitable strategy for refining abstract specifications into more concrete ones correctly and consistently is highly necessary.
In this paper it will be proposed to apply graph rewrite rules at a meta level to abstract graph grammar specifications of human computer interaction in order to achieve correct and consistent refinements of the specifications wrt important requirements. A safety-critical system concerning the interaction between the pilot and the flight management system on the flight deck of an aircraft will be used as an example. A graph grammar specification of this interaction at an abstract level will be refined wrt mode visualization by a graph rewrite rule at a meta level.

1 Introduction

A successful design of an interactive system requires a suitable specification of human computer interaction. Specifications using the concept of information resources (cf. [23] and [3]) and the formal notation of graph grammars (cf. [17]) provide a precise understanding of interaction as well as a correct and consistent formal description at least at an abstract level (cf. [21]).

However, in order to achieve a continuous specification process between the requirements and design stages a suitable strategy for refining abstract specifications into more concrete ones in a correct and consistent way is highly desirable.

A variety of techniques for specifying states and behaviour of interactive systems can be found in the literature. As shown in [11] state transition diagrams form the basis of a variety of description techniques for user interfaces. However, as pure sequential techniques they are only suitable for mask and menue dialogs. Describing user interfaces using statecharts permits modelling parallel dialogs but does not show the context between such parallel subdialogs. Dialog nets ([10]), a special form of Petri nets, overcome this problem with the features of modal subdialogs, a clear net structure and hierarchy. However, dialog net specifications of graphical user interfaces result quickly in very complex descrip-

tions of even simple user interfaces. A detailed presentation and comparision of techniques for specifying interaction processes can be found in [18].

Graph grammar descriptions have also been used as specification technique. In [1] a customized user interface design environment is generated. First, a conceptual framework for task-oriented user interface specification is specified as a visual language. The specification is then applied to a visual language generator yielding a visual syntax-directed editor for the specification language. In this approach the visual language is specified with graph transformation systems. Specification and representation of user interfaces based on end user tasks using attributed graphs and related graph rewriting systems can also been seen in [4].

In order to achieve specifications of interaction between human and machine which are expressive enough on the one side and which remain understandable on the other side the formalism of graph grammars is used in [7]. Dialog states describing user interface objects with their current appearance and their respective relationship to the underlying application are formally specified by directed attributed graphs. Transformations of one dialog state into another one by an event are formally specified by graph rewrite rules. Hence, graph grammars provide formal as well as clear interaction specifications allowing correct specification changes and reflecting the intuitive comprehension that designer and customer have about interaction, respectively, in every specification stage. In [15] this approach has been used in order to formalize the control window of a complex real numeric system by graph grammars (see also [19]), and the correct and comprehensive specification led to a far better interaction process. [20] and [22] show an approach for integrating software-ergonomic aspects in formal specifications of graphical user interfaces using graph grammars in order to improve human-machine interaction.

In [21] it has been presented how interaction may be modelled using the concept of information resources and formally specified by the notation of graph grammars. In order to develop a strategy for refining such specifications at an still abstract level the idea of constructing *meta rules* will be introduced in this paper. A graph rewrite rule of this kind specifies a specific task concerning the refinement of the abstract specification wrt a specific requirement. It specifies a refinement resulting from a regular communication between designer and customer during the specification process. A safety-critical system concerning the interaction between the pilot and the flight management system (FMS) on the flight deck of an aircraft will be used as an example for an abstract specification. A meta rule specifying the visualization of modes with the goal of refining the abstract interaction specification correctly and consistently will be constructed.

The formal notation of graph grammars which is based on the algebraic double pushout approach (DPO) will be introduced first. The construction of graph rewriting in this approach will help in understanding the refinement of graph grammar specifications by graph rewrite rules. A short presentation will then be given about how interaction between the pilot and the FMS can be modelled using the concept of information resources and specified formally by a graph grammar at a still abstract level. After the presentation of some proves already

at that abstract specification stage wrt some important requirements the specification has to be refined in order to move towards the design stage coming from the requirements stage. For this purpose a graph rewrite rule will be constructed with the goal of refining the abstract specification wrt the visualization of the modes involved. Considering this construction within the DPO will provide a suitable insight into the conditions under which an abstract specification could be refined correctly and consistently by a meta rule.

2 The Graph Grammar Formalism

The graph grammar formalism introduced below is based on the algebraic *double pushout (DPO) approach*. The definitions will sometimes be only semi-formal and incomplete due to lack of space. We therefore refer to [5], [17] and [13] for a detailed and complete description.

Let $Attr_V$ and $Attr_E$ be two type variables denoting types for labelling nodes and edges of graphs respectively. The sets used to actualize $Attr_V$ and $Attr_E$ are not necessarily simple flat alphabets, but could also be sets of functions, relations, etc. as attributes. We define:

Definition 1. An $(Attr_V, Attr_E)$-*graph* M is a system
$M = (V_M, E_M, s_M, t_M, l_M, m_M)$, where
V_M set of nodes,
E_M set of edges,
$s_M : E_M \to V_M$ source function of edges,
$t_M : E_M \to V_M$ target function of edges,
$l_M : V_M \to Attr_V$ node labelling,
$m_M : E_M \to Attr_E$ edge labelling.
M represents a *directed node and edge labelled* graph.

For describing modifications of graphs by graph rewrite rules we need the definition of a match between two graphs:

Definition 2. A *graph match* g of a $(Attr_V, Attr_E)$-graph M in a $(Attr_V, Attr_E)$-graph N is given by a 4-tupel $g = (g_V, g_E, p_V, p_E)$, where
$g_V : V_M \to V_N$ and $g_E : E_M \to E_N$ mappings,
$p_V : (Attr_V \times Attr_V) \to \{true, false\}$ node attribute match predicate,
$p_E : (Attr_E \times Attr_E) \to \{true, false\}$ edge attribute match predicate,
such that $\forall e \in E_M, v \in V_M$:
$g_V(s_M(e)) = s_N(g_E(e))$,
$g_V(t_M(e)) = t_N(g_E(e))$,
$p_V(l_M(v), l_N(g_V(v))) \wedge p_E(m_M(e), m_N(g_E(e)))$.

For the induced match $g(M)$ it has to be assured on the one hand that g_V and g_E map consistently the source and target of each edge in graph M onto nodes in the target graph N such that the graph structure of M - i.e. without considering labels - is mapped onto a proper subgraph of N. On the other hand the label of a graph element (node or edge) in M and the label of the corresponding picture

of that graph element in graph N (under the mapping g_V or g_E respectively) have to be *compatible* with respect to the predicates p_V or p_E respectively.

Modifications of graphs are described by graph rewrite rules:

A *modification of a graph* M into a new graph N by applying a graph rewrite rule $r = (L, R, K)$ is realized by the following two principal steps:
1) a graph match is chosen between the graph L and the graph M to be modified,
2) the induced match of graph L is removed in graph M and graph R is added.
The connection of graph R to the remaining part of graph M is given by the glueing graph K (see below).

As can be seen, a *graph rewrite rule* $r = (L, R, K)$ is a triple of graphs, where graph L is the left hand side of the rule, graph R is the right hand side of the rule and graph K is the so called glueing graph. K takes care that no dangling edges appear in the new graph after applying rule r to the old graph. Hence, K identifies some anchor elements which have to remain unchanged by the modification and is a subgraph of L and R as well.

At this point we have all necessary preliminaries to define:

Definition 3. A *graph grammar* is a system $G = (Attr_V, Attr_E, P, P_r, Z)$, where
$Attr_V$, $Attr_E$ are node and edge attributes respectively,
P is a set of graph rewrite rules,
P_r is a set of attribute match predicates required for a graph match,
Z is the start graph (Z is a $(Attr_V, Attr_E)$-graph).

For the sake of simplicity a graph grammar G will be described in the following by $G = (Z, P)$, where Z denotes the start graph and P the set of graph rewrite rules.

In order to prepare the use of the graph grammar formalism for specifications of interaction as well as for refining such specifications a view onto direct derivation diagrams in the DPO helps much in understanding the rewrite mechanism introduced above (see also [2]). Direct derivations, i.e. modifications of graphs by a rewrite rule, are modelled by glueing constructions of graphs, that are formally characterized as pushouts in suitable categories having graphs as objects and graph homomorphisms as arrows. A graph rewrite rule p is given by a pair $L \xleftarrow{l} K \xrightarrow{r} R$ of graph homomorphisms from a common *interface* or *glueing graph* K, and a direct derivation consists of two glueing diagrams as (1) and (2) in the diagram in figure 1. The *context graph* C is obtained from the given graph G by deleting all elements of G which have a pre-image in L, but none in K. This deletion is modelled as an inverse glueing operation by diagram (1), while the actual insertion into H of all elements of R which do not have a pre-image in K is modelled by the glueing diagram (2). The match m must satisfy the so-called *glueing condition* which takes care, that the context graph C will have no dangling edges and that every element of G that should be deleted by the application of p has only one pre-image in L.

Diagrams as the one in figure 1will help much in investigating the refinement of graph grammars by graph rewrite rules at a meta level as will be seen in

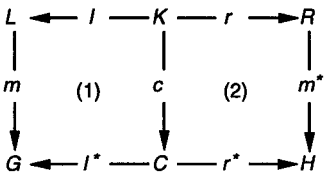


Fig. 1. Direct derivation in the DPO

chapter 4. After this introduction to the graph grammar formalism a specification of the interaction between the pilot and the FMS on the flight deck of an aircraft will be presented using the concept of information resources and the formal notation of graph grammars (for a detailed explanation cf. [21]).

3 Specification of the Interaction between Pilot and FMS

Interaction can be modelled based on the concept of *information resources* (more deeply handled in [23] and [3]). This idea is influenced by the approach of distributed cognition [9] which sees resources as distributed across components of the whole system. The user tries to achieve *goals* by interacting with a system and he has *plans* in his head during certain time periods to achieve these goals. *Action-affordances* refer to the set of possible next actions that can be taken, given the current state of the system. An *action-effect mapping* is a statement of the effect that an action will have if it is carried out. Such information resources are highly suitable criteria for getting an understanding of interaction.

Graph grammars represent a highly suitable formalism for specifying statics and dynamics of interactive systems. Graphs describe states, graph rewrite rules describe changes of states in a powerful but understandable manner (cf. [7]).

Based on these preparations the specification of a safety-critical system, of the interaction between the pilot and the flight management system (FMS) on the flight deck of an aircraft, can be constructed. It comprises the stages beginning with the take-off and ending up with the landing (for a detailed description we refer to [21]). The first state of the interaction may be specified by the start graph G_{Start} in figure 2 where the small pictures denoted by `Pilot` and `FMS` are only depicted for orientation purposes.

The root node on the pilot side specifies the top level goal `TLG` with the meaning of a particular flight. This goal is decomposed into the three subgoals `Start`, `Fly` and `Land`, landing again is decomposed into the subgoals `CNM` (Changing Navigation Mode), `EDI` (Entering Descent Input) and `TD` (Touch Down). All goals not further decomposed are associated to subparts of the system's user interface on the FMS side. Action-effect edges indicate that actions of type `<a>` can be performed within parts of the system's user interface in order to complete the respective goals. All possible actions constitute the actual action-affordances.

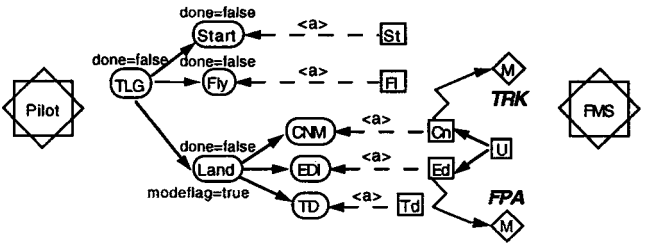
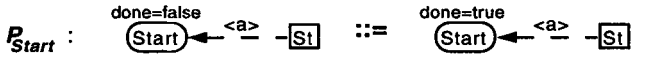


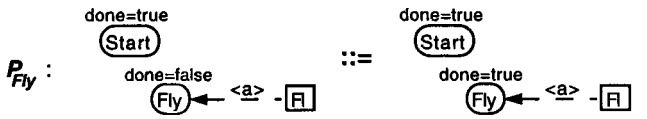
Fig. 2. Start Graph G_{Start}

In the autopilot system the *modes* of the navigation and descent are automatically coupled (cf. [12]). For that reason the corresponding parts of the system's user interface can be grouped together by specifying them, e. g., as subwidgets of a superwidget. `Cn` and `Ed` are examples for such subwidgets with a common superwidget `U`. Moreover, because of the importance of the actual modes of `Cn` and `Ed` in every situation during the flight a node `M` of a new type denoted by a different shape is introduced. It has an attribute indicating the current mode value of its associated system part, e.g. mode value *TRK* (*Track*) for `Cn`, and a specific edge connects this mode node with its respective user interface part.

Graph rewrite rules specify the interactions. The first rule specifies the start of the flight.

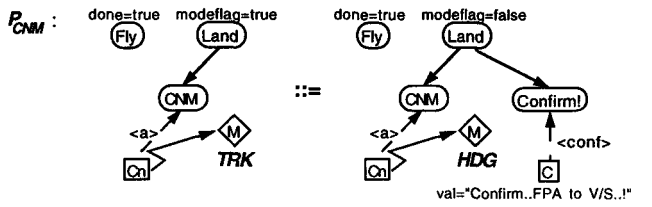


This rule matches in a state graph under the condition that the goal `Start` has not yet been completed as indicated by the value *false* of the attribute `done` on the left hand side of the rule. In the case of a match the left hand side of the rule will be substituted by its right hand side. This specifies that the action `<a>` performed within the user interface part `St` leads to the completion of the goal `Start` as indicated by the value *true* of the attribute `done` on the right hand side of the rule. (Glueing graphs necessary due to technical reasons will not be considered further in this context.) The next rule specifies the flying stage.



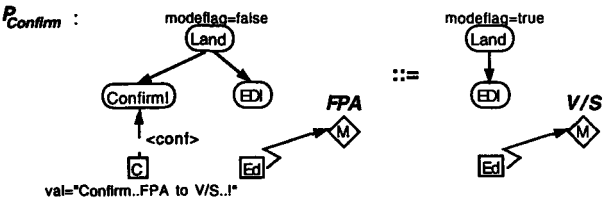
This rule matches under the condition that the goal `Start` has already been completed but the goal `Fly` has not yet been completed, and it works analogously

to the previous rule. The next rule specifying the changing of the navigation mode applies under the condition that the goal `Fly` has already been completed.



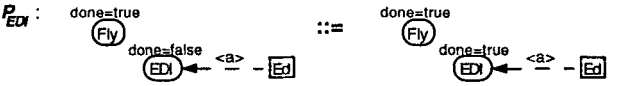
This rule changes the value of the navigation mode from *TRK* to *HDG* (*Track* to *Heading*) as can be observed by considering the mode node connected to the related subpart `Cn` of the system's user interface. Such a change is sometimes necessary in order to comply with radar guidance ([12]). But additionally, the rule does something else. A new goal `Confirm!` appears with related user interface part `C` within which an action `<conf>` can be performed to complete this goal. One can imagine `C` as a modal subdialog or a dialogbox. The goal `Confirm!` is added here in order to force the pilot to confirm the change of the descent mode from *FPA* to *V/S* (*Flight Path Angle* to *Vertical Speed*) which is automatically coupled with the change of the navigation mode from *TRK* to *HDG*. This is indicated to the pilot by the value of the attribute `val` attached to the node `C`. In this specification the pilot ist forced to confirm the automatically coupled change of the descent mode in order to prevent mode errors which in the past have led already to accidents claiming casualties (cf. [12] and [14]).

Now the specification has to assure that the pilot indeed can not do anything else except this confirmation. Therefore, the boolean attribute `modeflag` is attached to the goal node `Land`. Its value is always *true* except in the situation that the pilot changes the navigation mode where its value becomes *false*. This can be observed by comparing left and right hand side of the rule P_{CNM}. If the `modeflag` is *false* in the current state graph then only one specific rule matches, namely $P_{Confirm}$ below specifying the expected confirmation by the pilot as well as the automatically coupled change of the descent mode. The right hand side of the rule shows that after its application the value of `Land.modeflag` is *true* again and that the subgraph containing the goal `Confirm!` has disappeared.

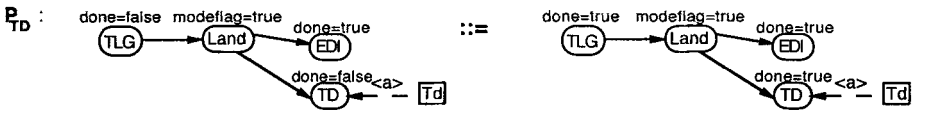


The penultimative rule specifies the entering of the descent input which is a

subgoal of the landing stage and is therefore performed under the condition that the goal Fly has already been completed.



Finally, the last subgoal of the landing stage, the touch down goal, is specified by the following rule P_{TD} applicable under the condition that the entering of the descent input has already been completed.



This rule, finally, completes the entire top level goal, the particular flight which is indicated by the value *true* of attribute done attached to node TLG.

Despite existing techniques in the theory of graph grammars for aggregating set of rules differing only in the labels of their graphs (cf. [6]) two further rules $P^{\diamond}_{CNM}$ and $P^{\diamond}_{Confirm}$ specifying the change of the navigation and descent modes also in the other direction, from *HDG* to *TRK* (*Heading* to *Track*) and from *V/S* to *FPA* (*Vertical Speed* to *Flight Path Angle*) respectively, are added in this example. Thus, the interaction between the pilot and the FMS can be specified at a still very abstract level by the

Graph Grammar $GraGra = \{G_{Start}, P\}$ with

- G_{Start} Start Graph,
- $P = \{P_{Start}, P_{Fly}, P_{CNM}, P_{Confirm}, P^{\diamond}_{CNM}, P^{\diamond}_{Confirm}, P_{EDI}, P_{TD}\}$.

Specifications of this kind are very useful in order to prove already at abstract stages of the development process whether important properties and requirements are fulfilled by the specification or not. In order to give an impression some properties and requirements already considered in [21] will be presented in the following. In the safety-critical system introduced above it is highly important to be able to prove already at early specification stages that certain actions take place before or after other ones. A first claim could therefore concern the required order in which the three goals Start, Fly and Land are intended to be completed.

Claim 4. *The goals* Start, Fly *and* Land *will by all means be completed in the required order!*

Proof. The set of nodes and edges of a graph G will be denoted by V_G and E_G.

a) $P_{Start} = (L_{Start}, R_{Start}, K_{Start})$ is the only rule the left hand side L_{Start} of which matches a subgraph of the start graph G_{Start}!

   $\forall v \in V_{Start} : v.\texttt{done} = false \wedge v.\texttt{modeflag} = true$
   $\forall L_x \text{ of } P_x \in P \backslash \{P_{Start}\} : (\exists v \in V_{L_x} : v.\texttt{done} = true \vee v.\texttt{modeflag} = false)$

b) P_{Fly} is applicable only under the condition that goal `Start` has been completed!

   L_{Fly} only matches graphs of the kind $L^\diamond$ where
   $\exists v \in V_{L^\diamond}$ with $v = \texttt{Start} \wedge v.\texttt{done} = true$

c) All rules contributing to the completion of goal `Land` are only applicable if goal `Fly` has been completed!

   This has to be shown for all rules in the set $P \backslash \{P_{Start}, P_{Fly}\}$:
   The proofs for P_{CNM} ($P^\diamond_{CNM}$) and P_{EDI} work analogously to proof 4b). $P_{Confirm}$ ($P^\diamond_{Confirm}$) is only applicable directly after P_{CNM} ($P^\diamond_{CNM}$) (because of the attribute `Land.modeflag`). P_{TD} is only applicable after P_{EDI} (because of the attribute `EDI.done`).

One very important requirement is to assure that the pilot is always aware of the actual mode values.

Claim 5. *Any indirect (caused by another action) change of the descent mode will be performed through* **confirmation** *by the pilot!*

Proof. Changing the descent mode is performed by rule $P_{Confirm}$ ($P^\diamond_{Confirm}$) forced in turn by rule P_{CNM} ($P^\diamond_{CNM}$). $P_{Confirm}$ ($P^\diamond_{Confirm}$) specifies an action `<conf>` within a subpart `C` of the system's user interface to be performed by the pilot.

Finally, it is important to be sure that certain actions are always performable by the pilot at certain stages during the flight. The next claim considers that more concretely.

Claim 6. *The navigation mode can always be changed during the stage* `Land`.

Proof. A switch between the two navigation modes *TRK* and *HDG* is specified by the two rules P_{CNM} and $P^\diamond_{CNM}$ respectively. Both work under the condition `Fly.done` $= true$ according to proof 4b).

A specification describing the interaction between the pilot and the FMS at an abstract level has been constructed so far. Some important requirements could also proved to be true already at this abstract specification stage. In the next chapter it will be investigated how such an abstract specification could be refined continuously by considering a graph rewrite rule at a meta level.

4 Refining the Interaction Specification by a Meta Rule

An important requirement wrt the specification of the interaction between the pilot and the FMS described in chapter 3 is the visualization of the modes

involved. The consideration of the start graph G_{Start} in figure 2 shows that modes are specified already but they are not visualized to the pilot who has to be aware of their values in every situation to increase safety for the flight. An idea in order to refine this graph wrt to mode visualization is to apply a graph rewrite rule at a certain meta level. The notion *meta level* indicates that such a rule does not **belong** to the interaction specification, but it has the task to **refine** this specification. A meta rule extending the state graph wrt to mode visualization could look like the graph rewrite rule P_{Meta} in figure 3.

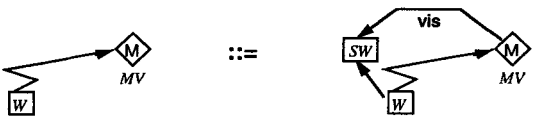


Fig. 3. Graph Rewrite Rule P_{Meta}

The left hand side matches whenever a system's user interface part is connected to a mode node by a respective edge. The free variable W indicates that application of the rule is possible wrt any system part under the conditions specified by the left hand side of the rule. The same holds for the value of the current mode indicated by the free variable MV. Substitution by the right hand side of the rule implies an extension wrt the required mode visualization. An additional system part denoted by the free variable SW will be created by which the current mode value MV becomes visualized. One can think about this new part as a label with a bright background colour or the like. The fact of visualization is explicitly specified by a directed edge labelled by the constant attribute `vis` and leading from the mode node M to the node SW describing the new system part.

At this point the following should be noted: application of the graph rewrite rule $P_{Meta} = (L_M, R_M, K_M)$ to a graph G leads to a substitution of the induced match $g(L_M)$ (cf. definition 2) in G by R_M for every occurence $g(L_M)$ (the glueing graph K_M is equal to graph L_M in this case). According to a regular communication between designer and customer, however, meta rule P_{Meta} should only be used for refinement in cases in which mode visualization has not yet been integrated into the specification. Thus, to prevent the application of P_{Meta} in cases where modes just have been visualized *negative application conditions* (cf. [8]) can be integrated elegantly into the rule specification. This, however, is omitted here for the sake of simplicity.

Refining the specification of an interaction by a meta rule means applying the meta rule to the respective graph grammar specification, i.e. applying it to

- the *start graph* specifying the first state of the interaction,
- the *set of rewrite rules* specifying the dynamics of the interaction.

Based on these considerations the refinement of graph grammar *GraGra* in

chapter 3 by meta rule P_{Meta} in figure 3 will be investigated in the following two sections.

4.1 Refining the Start Graph G_{Start} by the MetaRula P_{Meta}

According to the previous explanations the twofold application of the meta rule P_{Meta} in figure 3 to the state graph G_{Start} in figure 2 would lead to the state graph H_{Start} in figure 4 where the visualization of the navigation as well as of the descent mode is specified now. The visualization of the two modes is realized by the new user interface parts SC and SE respectively.

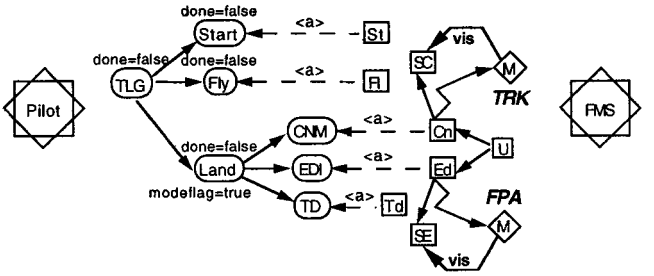


Fig. 4. Start graph H_{Start} (refinement of G_{Start}) with integrated mode visualization

The application of rule P_{Meta} to graph G_{Start} can easily be understood by considering the corresponding derivation diagram in the DPO in figure 5.

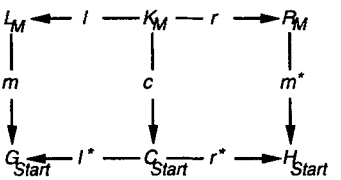


Fig. 5. Application of rule P_{Meta} to graph G_{Start} yielding graph H_{Start} in the DPO

A comparision of the diagrams in figures 1 and 5 provides the proof idea for the fact that applications of graph rewrite rules like P_{Meta} to state graphs like G_{Start} result in correct and consistent refinements of the resulting graphs.

The more complicated question is, how meta rules could refine graph grammar rules specifying interactions. This will be investigated in the next section.

4.2 Refining Set P of Graph Rewrite Rules by Meta Rula P_{Meta}

Application of rule P_{Meta} (cf. figure 3) to the set P of graph rewrite rules (cf. capter 3) at first means application to every graph rewrite rule contained in set P. Obviously, the left hand side L_M of P_{Meta} would only match in graphs involved in the four rules P_{CNM}, $P_{Confirm}$, $P^{\diamond}_{CNM}$ and $P^{\diamond}_{Confirm}$. Additionally, it is intuitively clear that a refinement of either of these rules wrt to mode visualization would make sense: a change of the navigation as well as of the descent mode by the pilot during a flight would have to be reflected in the respective user interface part. Thus, a consistent refinement of a graph rewrite rule $p = (L, R, K) \in \{P_{CNM}, P_{Confirm}, P^{\diamond}_{CNM}, P^{\diamond}_{Confirm}\}$ would require the application of the meta rule P_{Meta} to L as well as to R. Application of rule P_{Meta} to graph rewrite rule $P_{Confirm}$ in chapter 3 would lead to the new rule $P^{\star}_{Confirm}$ in figure 6.

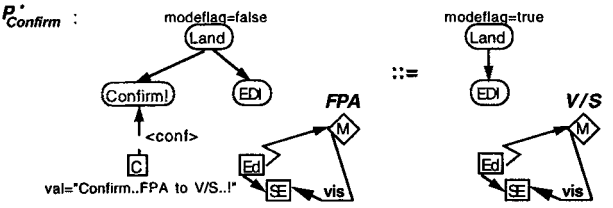


Fig. 6. Graph rewrite rule $P^{\star}_{Confirm}$ with integrated mode visualization

The DPO diagram in figure 7 describes this idea: the application of the meta rule $P_{Meta} = (L_M, R_M, K_M)$ to a graph rewrite rule $p = (L, R, K)$ yields the refined graph rewrite rule $h = (H_L, H_R, H_K)$ with $H_K = K$.

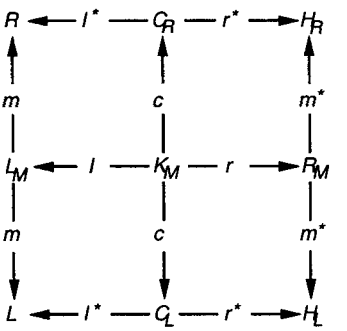


Fig. 7. Refinement of $p = (L, R, K)$ by P_{Meta} to $h = (H_L, H_R, H_K)$ in the DPO

In order to achieve a correct and consistent refinement by the application of the meta rule P_{Meta} to a graph rewrite rule of the kind $p = (L, R, K) \in \{P_{CNM}, P_{Confirm}, P^{\diamond}_{CNM}, P^{\diamond}_{Confirm}\}$ the following two requirements have to be ensured for which proof ideas will follow:

- P_{Meta} rewrites L of rule p exactly in the same way as R (visualization of a mode has to be updated **together** with the mode value (consistency)).
- $H_K = K$ is a correct glueing graph for the new rule h (the new rule is defined correctly).

The first requirement can be ensured by considering the graph homomorphisms m, $m^\star$ and c in figure 7: as in the DPO diagram in figure 1 they take care for constructing a correct graph H_L in case of a match of L_M in L. Thus, because of using exactly the same homomorphisms as well in order to refine the right hand side R of rule p the requirement is ensured.

Generally, also the case in which L_M matches only in one of the graphs L and R has to be considered. Due to the goal of refining interaction specifications according to the result of a regular communication between designer and customer only meta rules within a sensible context are considered. Thus, if L_M matches only in L this would mean that rule p specifies the *deletion* of a specific feature, and this has also to be refined. E.g., switching off a mode by the pilot should also imply the deletion of the respective user interface part for visualizing this mode in the specification **together** with the mode specification part itself. Analogously, if L_M matches only in R this would mean that rule p specifies the *addition* of a specific feature, and this has to be refined as well. E.g., switching on a mode by the pilot should also imply the addition of a respective user interface part for visualizing this mode in the specification **together** with the mode specification part itself.

The second requirement can be ensured by considering the fact that the glueing graph K will not be refined by the meta rule P_{Meta}. The important point here is that P_{Meta} does only add graph elements (nodes/edges), but does not delete such elements. Thus, after application of P_{Meta} to a rule $p = (L, R, K)$ yielding $h = (H_L, H_R, H_K)$, K is still a subgraph of both, H_L and H_R, so that $H_K = K$ is still a correct glueing graph for the new rule h.

The last requirement can not be ensured for meta rules which delete graph elements from graph rewrite rules. In such a case it may happen that K is not a subgraph of H_L and/or H_R anymore. Even, if application of the meta rule to K would be allowed, a match of L_M in K could possibly not exist.

According to the approach presented in this paper[1] the graph grammar *GraGra* in chapter 3 can be refined by the meta rule P_{Meta} in figure 3 yielding the

Graph Grammar $GraGra^{\star}_{Ref} = \{H_{Start}, P^{\star}\}$ where

- H_{Start} Start Graph,

[1] A similar approach to graph grammar transformation can be seen in [16]

– $P^{\star} = \{P_{Start}, P_{Fly}, P^{\star}_{CNM}, P^{\star}_{Confirm}, P^{\diamond\star}_{CNM}, P^{\diamond\star}_{Confirm}, P_{EDI}, P_{TD}\}$,

H_{Start} is the refinement of G_{Start}, and $P^{\star}_{CNM}$, $P^{\star}_{Confirm}$, $P^{\diamond\star}_{CNM}$ and $P^{\diamond\star}_{Confirm}$ are the respective refinements of P_{CNM}, $P_{Confirm}$, $P^{\diamond}_{CNM}$ and $P^{\diamond}_{Confirm}$. In this way the abstract graph grammar specification $GraGra$ of the interaction between the pilot and the FMS on the flight deck of an aircraft could be refined correctly and consistently wrt the so important mode visualization by the meta rule P_{Meta} yielding the more concrete graph grammar specification $GraGra^{\star}_{Ref}$.

As the main result of this paper refinements using meta rules that do not delete any graph elements can be carried out correctly and consistently. Constructing such meta rules according to the DPO diagrams in figures 5 and 7 and based on the ideas introduced above in order to refine abstract graph grammar specifications of human computer interaction successively into more concrete ones highly encourages the development of a suitable strategy in order to achieve a continuous specification process between the requirements and design stages.

5 Conclusions and Future Work

Specifying human computer interaction in a suitable manner requires a deep understanding of interaction as well as a powerful and understandable formal specification formalism. Interaction can be specified successfully using the concept of information resources and the formal notation of graph grammars.

In order to achieve a continuous specification process between the requirements and design stages a suitable strategy for refining abstract specifications into more concrete ones correctly and consistently is highly desirable. This paper proposes an approach by applying graph rewrite rules at a meta level to abstract graph grammar specifications. A meta rule of this kind refines the start graph as well as relevant graph rewrite rules of an interaction specification by adding new graph elements specifying new requirements resulting from a regular communication between the designer and customer.

As main research goals for future work important issues concerning the correct and complete definition of a strategy or of a calculus in order to achieve a continuous specification process for human computer interaction will be considered. For a requirement arising as a result of discussions between customer and designer it has to be investigated how it could be formulated suitably by a correct meta rule. Beyond the ideas of proofs given in the last chapter the correctness and consistency of this rule has then to be proved. Another important question will concern the problem of how the deletion of certain features of specifications could be specified by suitable meta rules in order to deal with inconsistencies arising during the design process or with contradictory properties possibly being existent from earlier specification stages, becoming evident, however, only later on. Further, the construction of meta rules contributing to collaborative specification processes is one of the most important future goals.

Considering these and future questions highly encourages to develop a suitable specification calculus in order to achieve a continuous specification process of human computer interaction between the requirements and the design stages.

Acknowledgments

I would like to thank Michael Goedicke who always finds the time for discussing ideas, questions and problems, as well as Torsten Meyer for fruitful discussions. Thanks also to Reiko Heckel for valuable conversations and reference hints.

References

1. F. Arefi, M. Milani, and C. Stary. Towards Customized User Interface Design Environments. *Journal of Visual Languages and Computing*, pages 146–151, 1991.
2. A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Löwe. Algebraic Approaches to Graph Transformation – Part I: Basic Concepts and Double Pushout Approach. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 1, pages 163–245. World Scientific, 1997.
3. B. Fields, P. Wright, and M. Harrison. Designing Human-System Interaction Using The Resource Model. In *Proc. APCHI'96: Asia-Pacific Conference on Human Computer Interaction*, Singapore, June 1996.
4. R. Freund, B. Haberstroh, and C. Stary. Applying Graph Grammars for Task-Oriented User Interface Development. In W. Koczkodaj, editor, *Proceedings IEEE Conference on Computing and Information ICCI'92*, pages 389–392, 1992.
5. M. Goedicke. On the Structure of Software Description Languages: A Component Oriented View. Research Report 473/1993, University of Dortmund, Dept. of Computer Science, Dortmund, May 1993. Habilitation.
6. M. Goedicke, T. Meyer, and B. Sucrow. Modularization of Graph Grammars for Describing Grahical User Interfaces. Technical Report 01/96, University of Essen, Essen, 1996. Informatik-Berichte (in German).
7. M. Goedicke and B. Sucrow. Towards a Formal Specification Method for Graphical User Interfaces Using Modularized Graph Grammars. In *Proceedings of the Eighth International Workshop on Software Specification and Design*, pages 56–65, Schloss Velen, Germany, March 22-23, 1996. IEEE Computer Society, IEEE Computer Society Press.
8. A. Habel, R. Heckel, and G. Taentzer. Graph Grammars with Negative Application Conditions. In *Special issue of Fundamenta Informaticae*, volume 26, 1996. no. 3,4.
9. E. Hutchins. *Cognition in the Wild.* MIT Press, 1994.
10. C. Janssen. Dialog Nets for describing Dialog Behaviour of Graphical Interactive Systems. In K.-H. Rödiger, editor, *Software Ergonomics '93*, volume 39 of *German Chapter of the ACM*, pages 67–76, Stuttgart, 1993. Teubner. (in German).
11. C. Janssen, A. Weisbecker, and J. Ziegler. Generation of Graphical User Interfaces using Data Models and Dialog Net Specifications. In H. Züllighoven, W. Altmann, and E.-E. Doberkat, editors, *Requirements Engineering '93: Prototyping*, volume 41 of *German Chapter of the ACM*, pages 335–347, Stuttgart, 1993. Teubner. (in German).
12. V. D. Keyser and D. Javaux. Human Factors in Aeronautics. In F. Bodart and J. Vanderdonckt, editors, *Design, Specification and Verification of Interactive Systems '96*, Editors W. Hansmann and W.T Hewitt and W. Purgathofer, pages 28–45, Wien, 1996. Proceedings of the Eurographics Workshop in Namur, Belgium, June 5-7, 1996, Springer.

13. A. Martini, H. Ehrig, and D. Nunes. Graph Grammars - An Introduction to the Double-Pushout Approach. Technical Report 96-6, Dept. of Computer Science, Technical University of Berlin, Berlin, Germany, 1996.
14. A. Monnier. Rapport préliminaire de la Commission d'enquête administrative sur l'accident du Mont Saint-Odile du 20 janvier 1992. Technical report, Ministère de l'Equipement, du Logement, des Transports et de l'Espace, Paris, France, 1992.
15. U. Nowak, U. Pöhle, R. Roitzsch, and B. Sucrow. Formal Specification of the ZIB-GUI Using Graph Grammars. In W. Mackens and S. Rump, editors, *Software Engineering im Scientific Computing*, pages 290–296. Vieweg, July 1996. (in German).
16. F. Parisi-Presicce. Transformations of Graph Grammars. In J. Cuny, H. Ehrig, G. Engels, and G. Rozenberg, editors, *Graph Grammars and Their Application to Computer Science*, volume 1073 of *Lecture Notes in Computer Science*, pages 428–442. Society for Design and Process Science, Springer, 1996. 5th International Workshop, Williamsburg, VA, USA, November 1994, Selected Papers.
17. G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 1. World Scientific, 1997.
18. C. Stary. *Interactive Systems.* Vieweg, 1994. Software Development and Software Ergonomics (in German).
19. B. Sucrow. Formal Specification of Graphical User Interfaces Using Graph Grammars. In W. Mackens and S. Rump, editors, *Software Engineering im Scientific Computing*, pages 279–289. Vieweg, July 1996. (in German).
20. B. Sucrow. Towards an Integration of Software-Ergonomic Aspects in Formal Specifications of Graphical User Interfaces. In *Proceedings of The Second World Conference on Integrated Design & Process Technology*, volume 1, pages 194–201, Austin, Texas, December 1-4 1996. Society for Design and Process Science, IEEE Computer Society Press.
21. B. Sucrow. Formal Specification of Human-Computer Interaction by Graph Grammars under Consideration of Information Resources. In *Proceedings of the 1997 Automated Software Engineering Conference (ASEC'97)*, pages 28–35, Incline Village at Lake Tahoe, Nevada, November 3-5, 1997. IEEE Computer Society, IEEE Computer Society Press.
22. B. Sucrow. On Integrating Software-Ergonomic Aspects in the Specification Process of Grafical User Interfaces. In *Transactions of the SDPS Journal of Integrated Design and Process Science.* Society for Design and Process Science, IEEE Computer Society Press, 1998. (Accepted).
23. P. Wright, B. Fields, and M. Harrison. Distributed Information Resources: A New Approach to Interaction Modelling. In T. Green, J. Canas, and C. Warren, editors, *Proceedings of ECCE8: European Conference on Cognitive Ergonomics*, pages 5–10. EACE, 1996.