Published in Proc.: ECAI'96, Budapest, Hungary, August, 1996.

Knowledge-Based Dialog Structuring for Graphics Interaction

Ralf Möller[1]

Abstract. For interface development, it is important that decisions about the structure of the interface and the appearance of graphical objects be based on conceptual information about domain objects and user actions because, at development time, concrete objects are not available. This paper presents a new approach to model dialog structuring knowledge for interactive inferfaces to realize dialog structuring on the basis of a Description Logic knowledge base.

1 INTRODUCTION

For many knowledge-based systems the interface for the *end-user* is an integral part of the whole system architecture. For these applications, the knowledge-based development approach should also be applied to the interface, i.e. interfaces should not be treated merely as backends for the "real" applications. Research on Human-Computer Interaction has shown the problems of prototyping approaches for system development. While some HCI approaches use formal models to prepare manual system implementation, more and more semi-automatic User Interface Development Environments have been introduced (see e.g. the survey in [5]) that allow the automatic construction of interface components from models about domain objecs, user actions etc. There are several mutual influences between domain objects and user actions. On the one hand, domain knowledge required for the implementation of application functions is influenced by interface design constraints (e.g. for a movement action, it must be possible to compute the possible positions in beforehand). On the other hand, domain knowledge also influences the selection of appropriate interaction techniques for user actions.

This paper describes a new approach to formally model these influences using the derivation processes of Description Logics. In contrast to other approaches which also model user actions with task models for interface design (see e.g. [6]), the approach presented in this paper is used (i) to automatically derive concepts for user actions (or user tasks) from partial information about domain objects at system construction time (ii) to structure the interface and (iii) to automatically derive the contents of graphical interfaces based on models for knowledge about user tasks and *geometric* information about domain objects rather than standard interaction gadgets (see e.g. [12]) or objects known from drawing programs (see e.g. [14]).

In order to support a system development methodology where graphical interface components are first-class citizens which directly shape the specification of required program components, the system HAMVIS has been developed (HAMburg VIsualization System, see [9]).

1.1 The context: HAMVIS

HAMVIS supports a declarative way of specifying interface services and allows visualizations of domain objects to be systematically composed. The idea is to specify the interface of an application from an action-oriented point of view. HAMVIS provides generic concepts for user actions and domain objects. The application designer uses these concepts to define subconcepts to model domain objects and to specify an application-specific Action Decomposition Model. The Action Decomposition Model is used to derive (i) what has to be displayed, (ii) when this should take place, (iii) how the objects are displayed and (iv) how the displayed objects can be acted upon using a generic library of interface gadgets defined for a class of applications. Within an Action Decomposition Model, the designer describes actions of end-users by selecting generic action concepts provided by the HAMVIS. Instead of focussing on surface acts (e.g., gestures like "click" or "drag-and-drop") the generic action knowledge base of HAMVIS models concepts for user actions on a deep level, i.e. at the level of manipulations of domain objects rather than at the level of manipulations of graphical objects.

Interface development with the methodology behind HAMVIS roughly consists of the following substeps which are dependent on one another:

1. Development of models for domain objects using a Description Logic as an extension to a basic model provided by HAMVIS (HAMVIS uses CLASSIC [12] as a concrete DL implementation).
2. Interactive definition of an Action Decomposition Model using predefined generic action concepts for user actions.
3. Derivation of a *dialog structure* to prepare the mapping to UIMS services.
4. Determination of presentation attributes like color, line thickness or additional metagraphical objects (e.g. arrows for highlighting) in relation to the information encoded in the dialog structure.
5. Automatic code generation for a UIMS.

For the development of the HAMVIS prototype, the application class of interactive construction systems has been considered. In this paper, an example application for laying out the interior of an aircraft (called XKL) is discussed. Concrete physical objects must be interactively located while layout constraints are automatically maintained and possible placement areas are automatically computed (for a handmade Object-Oriented Design Environment of this class see e.g. Fischer et al. [4]).

Inspired by Intelligent Multimedia Presentation Systems (IMPS, see e.g. [15]), HAMVIS supports a communication-oriented perspective for UI development. In contrast to automatic IMP systems, HAMVIS examines how interaction and presentation knowledge can be used in an *interactive* system development approach. The goal is to support an interface designer with high-level specification subsystems. Thus, HAMVIS strictly distinguishes between *development-time activities* and *runtime activities* which are supported by standard UIMS architectures.

1.2 The dialog structuring phase

The communication-oriented view of HAMVIS assumes that each object to be displayed has a certain status in the discourse of interaction and fulfills a specific discourse purpose that constrains the way the object is presented. In addition, visualizations should be constructed in such a way that more than one action is supported by a single graphic in order to save screen space. The resulting interface can be specified by a dialog structure (DS) which describes required windows, panes and pane classes, view types and interaction techniques. The development of a DS for the interface of an application must be based on *conceptual information* about domain objects (to be actually computed at runtime) because, at *system development-time*, concrete objects are not necessarily available.

This paper focusses on the third system development step (see above) and presents a new approach to *model dialog structuring knowledge for interactive inferfaces* that realizes dialog structuring on the basis of a DL knowledge base. Due to the nature of the reasoning process involving *conceptual* information of domain objects, a DL provides an adequate representation mechanism for dialog structuring knowledge because conceptual reasoning about domain objects is easily extended to conceptual reasoning about communication purposes.

1 University of Hamburg, Computer Science Department, D-22257 Hamburg, Vogt-Kölln-Straße 30, Germany

ECAI 96, *12th European Conference on Artificial Intelligence*,
Edited by W. Wahlster
Published in 1996 by John Wiley & Sons, Ltd.

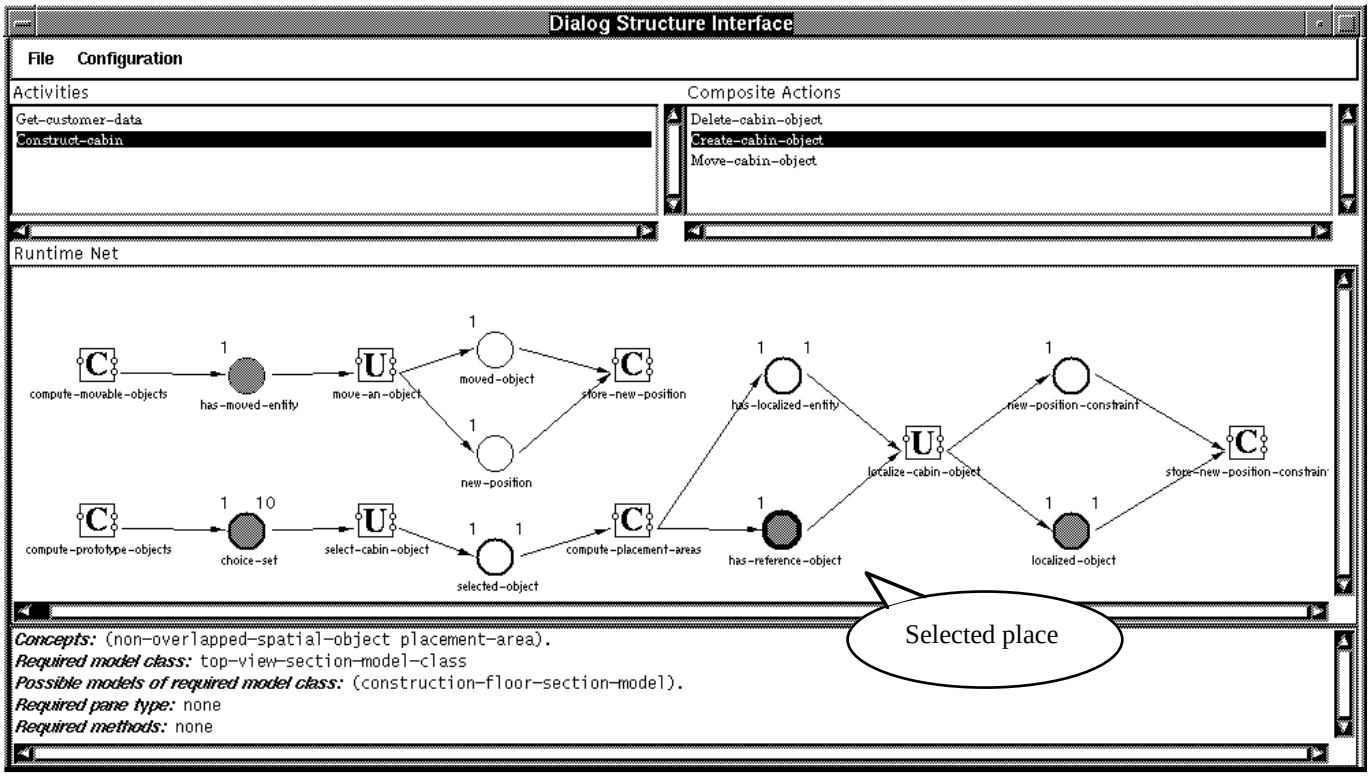

Figure 1. Action model of the XKL application visualized as a Petri net (user actions are indicated by a "U" icon, functions are marked by a "C").

The paper is organized as follows. The next section discusses aspects of the action-based specification approach for defining the *basic information content*. The example describes the input to the dialog structuring phase, gives a flavor of the generic HAMVIS knowledge bases and sketches the mapping to UIMS services. The third section formalizes the notion of a dialog structure, sketches how a DS can be derived using DL concepts and presents some extensions to DL used to model the DS derivation process. The paper concludes with a comparison to IMP Systems.

2 THE DIALOG STRUCTURING TASK

2.1 Prerequisites: Basic information content

An action decomposition model specifies the domain-specific action structure and declares the relationships between *user actions* and *application functions*. The term "application function" is used as an abstraction for program components which do *not require user interaction* but might be realized by knowledge-based subsystems. The results of application functions must be displayed in order for user actions to become applicable. An action is actually carried out by the end-user with an appropriate gesture (e.g., via drag and drop or using a specific gadget) and, as a result, new objects are passed to subsequent functions which are automatically applied when the input objects become available. This kind of dataflow can be formally modeled using Petri nets (so-called runtime nets). HAMVIS provides an interactive interface for the application developers to support a convenient way to specify such a model. An example for the action decomposition for XKL is shown in Figure 1. The XKL application basically consists of two separate activities (Get-customer-data and Construct-cabin) which are realized at the UIMS level by different windows which are presented one after the other. The selected activity Construct-Cabin consists of three alternative subactions (Delete-cabin-object, Create-cabin-object, Move-cabin-object). Composite actions are action *alternatives* which can be repeatedly executed until the associated activity is finished. The main window describes the decomposition of the composite actions. The first Petri net defines Move-cabin-objects. The Petri net presented below models the action Create-cabin-object which consists of two elementary user actions: a selection action (select-cabin-object) and a localization action (localize-cabin-object). Information about selected transitions and places is presented in the pane below (in this case for the place has-reference-object). The objects to be put into the place at runtime must inherit from the concepts non-overlapped-spatial-object and placement-area. The action model for the localization action requires a certain view (called model class).

When the window of an activity is presented at runtime, the interaction cycle starts. The initial transitions of the runtime net fire, i.e. the initial application functions (without input parameters) are evaluated. The results are put into the corresponding output places in the runtime net. Some places contain objects that must be displayed in a pane of the activity window. In the screen shot of the HAMVIS interface in Figure 1, these places are indicated with dark gray. The goal is to present objects in such a way that they are mouse-sensitive and can be used as gadgets. For instance, the HAMVIS library contains gadgets for moving objects within certain boundaries. A movement gadget generates a new value for the position which is passed to the subsequent application function (in addition to the moved object itself). When the last transition of a subnet has fired, the interaction cycle starts again, i.e. the initial application functions are evaluated etc. The UIMS is responsible for managing the resulting drawing requests and screen updates.

Conceptual knowledge about user actions is modeled with DL concepts and roles. A sketch of the definition of the concept localization-in-xy-bounding-rectangle for the action localize-cabin-object is given below. Roles and concepts are presented with some extensions to the KRSS syntax [11]. The keyword :asserted-concepts is an abbreviation for a rule that triggers on the concept being defined and asserts the conjunction of the specified concepts.

```
(define-primitive-role has-localized-entity
                       has-manipulated-object)

(define-concept spatial-localization
  (and localization
       (all has-localized-entity spatial-object)
       (all has-reference-object spatial-object)))

(define-concept localization-in-xy-bounding-rectangle
   (and spatial-localization ...
        (all has-loc-entity-ref-object-relation in))
  (:asserted-concepts
    (all has-reference-object
         non-overlapped-spatial-object) ...))
```

The generic action knowledge base of HAMVIS models concepts for user actions (with case roles) with respect to a *HAMVIS Upper Model*. Domain-specific concepts for domain objects must inherit from concepts defined in the HAMVIS Upper Model. For instance, the XKL concept placement-area (see the place has-reference-objects in Figure 1) must inherit from a concept spatial-object defined in the HAMVIS Upper Model.

The generic HAMVIS Action Model determines *which* case roles describe *domain objects* that must be displayed and *which concepts* the objects will have. The application-specific Action Decomposition Model defines *when* this should take place. The interfaces for the declaration of this model and the reasoning services for action modeling provided by HAMVIS are explained in detail in [10].

2.2 Bottom-up dialog structuring

In contrast to IMP systems which compute the dialog structure top-down, HAMVIS uses a bottom-up approach. The basic information content for HAMVIS is not given with (agent-oriented) "presentation goals" to be expanded top-down (for instance with plan operators that operate on the mental state of the perceiver). The Action Decomposition Model declares information about the objects to be communicated in order to support the end-user's actions and represents restrictions on the view (e.g., perspective) that will be used to display the objects (see Figure 1). However, like IMP systems, HAMVIS must extend the basic information content given by the Action Decomposition Model to support information processing activities of the end-user. For example, the actions in the XKL example presented above require a reference frame (e.g., the cabin body) to be presented. In addition to content completion, panes of certain types for displaying objects for interactive manipulation must be allocated (e.g., for the selection action in Figure 1 a palette pane is required). The composite actions Create-cabin-object and Move-cabin-object can be realized by a single presentation because the reference frame can be shared (the associated action concepts allow the same view). As an example, the final interface for the XKL activity Construct-cabin which has been generated with the services provided by HAMVIS is presented in Figure 2. The former action can be realized by a drag-and-drop gesture, the latter by clicking on a cabin object and dragging it inside its associated placement area.

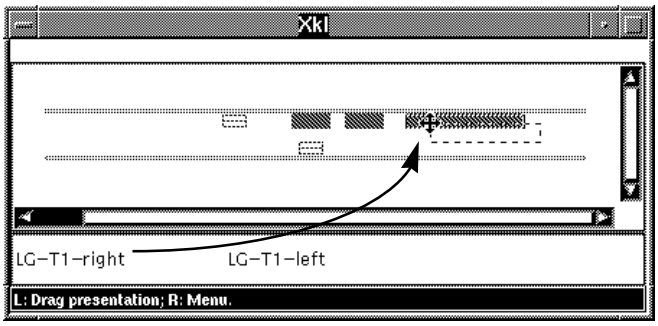

Figure 2. The final interface for the Construct-cabin activity generated by HAMVIS with mouse documentation. Missing cabin objects (e.g., a "Lateral Galley of Type 1") are shown in a palette window and can be dragged into automatically computed placement areas (dark rectangles) being presented in the graphic pane when dragging starts (cf. Figure 1). Other cabin objects previously located are also shown and can be moved with the mouse.

In addition to a Petri net interpreter, the runtime system to be generated from a dialog structure includes a *display manager* which is responsible for managing presented objects. For instance, the placement areas required for the action Create-cabin-objects are displayed only "temporarily" while the display status of a localized object (see the place localized-object in Figure 1) should be "permanent". Furthermore, a reference system (the cabin body) must be "statically" presented. Thus, in order to generate adequate presentations that reflect the end-user's expectations, objects found in places of the runtime net are transferred to the display manager which needs additional information concerning the *discourse status* and the *discourse purpose* of presented objects. The discourse purpose represents the reason why an object is displayed and influences the selection of drawing attributes. The discourse information for the display manager depends on the user actions and can be derived at system development time. The definition of the information content, visualization composition, the definition of required display panes and the derivation of discourse purposes and status of displayed objects should be realized using declarative dialog structuring models rather than by programming and rapid prototyping.

2.3 Interactive approach

HAMVIS supports an interactive approach to build a DS. The interface designer can build a DS by clicking on the gray places of the runtime net. For instance, if the place has-reference-object was the first place to be selected, HAMVIS would compute the initial dialog structure presented in Figure 3 (called ds4345). A DS can be devided into three levels: the pane level, an intermediate level (called aggregate level) and the level of DS segments which describe information about displaying objects found in places of the runtime net. For displaying placement areas for a localization action (place has-reference-object) at least one pane is required (called pane-2). Furthermore, Figure 3 indicates that in the pane a reference object and the objects in the environment of the reference objects must be displayed.

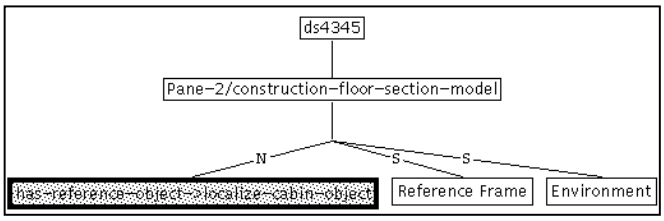

Figure 3. Snapshot of the initial dialog structure derived for considering the place has-reference-object.

HAMVIS computes several alternatives for considering a place in the DS. Information about the discourse status can be examined by clicking on DS nodes etc. The interface designer can keep several alternatives for exploring them in parallel (using the interactive interfaces of HAMVIS) or can reject structuring alternatives. If a dialog structure cannot be derived, HAMVIS informs the interface designer about the reason. It might be the case that either the domain models or the Action Decomposition Model must be modified or extended. The rest of the paper focusses on the mechanisms behind the surface, presents a formalization of the associated knowledge sources and describes the incremental reasoning processes for building dialog structure alternatives.

3 THE INTERNAL LOGICAL VIEW

In IMP systems, presentation knowledge is usually expressed from the "external" viewpoint of a presentation agent which reasons about the mental state(s) of the perceiver. For instance, the fact that a reference frame is required might be represented in such a way that the perceiver "wants to know" something about the reference system (see [15]). The advantage of this approach over simpler schema-based approaches is that presentation constituents can be related to each other (via their effects on the mental states of the perceiver).

However, even for evaluation purposes of interactive graphical interfaces, defining mental models is very difficult. The HCI literature is full of different proposals that cover only a small range of effects. HAMVIS uses a different approach which avoids the explicit definition of mental states of the user. Instead of using an external point of view to relate presentation constituents, HAMVIS models the influences between presentation constituents form an internal points of view. We say that a presentation constituent requires a *modification* (see [7]) which is induced by another constituent (the modifier). In the following it is shown how the construction of a dialog structure can be interpreted as the fulfilment of modification requirements which, in turn, can be realized by conceptual reasoning with DL.

3.1 "Driving" Bottom-Up DS construction with DL

From an internal, logical point of view, selecting a place for DS integration corresponds to generating a set of assertions for a certain DS segment and assuming that a DS can be built, i.e. the assertions are logically consistent. Using CLASSIC as a DL, HAMVIS represents each of the DS layers with appropriate concepts (DS-unit, DS-aggregate and DS-segment). The graph structure is defined by the relation has-ds-component, its inverse ds-component-of and subrelations for distinguishing main objects (has-nucleus) and additional objects which represent information about required modifiers for the nucleus (has-satellite).

```
(define-primitive-role has-nucleus has-ds-component)
(define-primitive-role has-satellite has-ds-component)
(define-primitive-concept ds-segment
  (and (at-least 1 ds-component-of)
       (all ds-component-of ds-aggregate)
       (at-most 1 generated-by-action)
       (at-most 1 has-discourse-purpose)))

(define-primitive-concept ds-aggregate
  (and (at-least 1 has-nucleus) (at-most 1 has-nucleus)))
```

For each place to be integrated into a DS, internally an individual representing known information about a DS segment is created:

```
(state (instance dss-1 ds-segment))
(state (instance dss-1
        (all describes-displayed-object placement-area)))
(state (related dss-1 localize-cabin-object
        generated-by-action)))
```

Though a DS segment is related to at least one DS node via the role ds-component-of (see the definition of ds-segment), this node is not necessarily known and the DL system does not automatically generate an individual representing this node. Creating the corresponding individual (node at the aggregate level) requires some set of additional assertions (either that a newly created node is not equal to an existing one or that an existing node might by "reused"). In other words: the initial set of assertions is in some sense *incomplete* and a set of additional assertions must be generated.

3.1.1 Notion of "incompleteness"

In order to define which individuals require a set of known fillers for a certain role, a new concept schema has been introduced to KRSS-CLASSIC. In lower-bound cardinality restrictions, a so-called K-operator may be used in the consequences part of a rule (K-cardinality restriction). An example is given by the following definition:

```
(define-concept spatial-action-ds-segment
  (and ds-segment
    (all describes-displayed-object spatial-object)
    (all generated-by-action
       (and user-action
          (all has-manipulated-object spatial-object))))
  (:asserted-concepts
    reference-frame-requiring-dss
    environment-requiring-dss
    (at-least 1 (k ds-component-of))
    (at-least 1 (k has-discourse-purpose))
    (all has-discourse-purpose
         descriptor-for-manipulated-object)))
```

When a ds-segment is classified as a spatial-action-ds-segment by ABox reasoning (which is actually the case for dss-1, s.b.), a function compute-role-fillers is automatically applied to the individual and the role (actually the role name). The function must return the required fillers for the relation ds-component-of. In the example from above, one filler has to be created. It will be set into relation with dss-1 and will be automatically classified as a ds-aggregate (see the definition of ds-segment).

If less than the number of required fillers are returned, the individual is called incomplete. When an instance is found to be incomplete, this means that there exists not enough information to prove the initial set of assertions, i.e. a dialog structure cannot be found.

3.1.2 Computation of assertions

When a concept like spatial-action-ds-segment is declared (during the development phase of HAMVIS itself), corresponding definitions to compute the respective role fillers are defined. This kind of incremental definition of functions is most easily supported by generic functions and methods. Therefore, CLASSIC has been extended with CLOS-like generic functions which dispatch on CLASSIC concepts and on CLOS classes [8]. For the dialog structuring concepts defined by HAMVIS, the corresponding methods have already been defined. As an example, the method for spatial-action-ds-segment and the role ds-component-of is presented.

```
(define-method compute-role-fillers
               ((ind spatial-action-ds-segment)
                (role (eql 'ds-component-of)))
  (cl-create-ind (gensym "DSA")
                 `(fills has-nucleus ,(cl-name ind))))
```

This function computes an instance (say dsa-1) which is used as a filler for the role ds-component-of of dss-1. In addition, an additional assertion is generated: dss-1 is the nucleus of the dsa (has-nucleus is a subrole of has-ds-component, see above).

3.2 Derivation of display information

The initial ds-segment dss-1 is classified as a spatial-action-ds-segment because (i) placement-areas are subsumed by spatial-object and (ii) localize-cabin-object is a user-action whose manipulated object is a spatial-object (see the definition of the action and its case roles mentioned in Section 2). Cabin objects like galleys and lavatories are also spatial objects. The defined concept spatial-action-ds-segment serves as a trigger for a rule that asserts modification requirements expressed with the concepts reference-frame-requiring-dss and environment-requiring-dss. Modifiers (satellites) are indirectly set into relation to nuclei via a ds-aggregate individual which is generated by the K-operator. The HAMVIS dialog structuring knowledge base contains several additional concepts and roles for ds-aggregates. For example, reasoning about reference frames at the aggregate level is modeled by the following definitions.

```
(define-primitive-role has-reference-frame-satellite
                       has-satellite)

(define-concept dsa-with-ref-frame-requiring-nucleus
  (and ds-aggregate
       (all has-nucleus reference-frame-requiring-dss))
  (:asserted-concepts
    (at-least 1 (k has-reference-frame-satellite))
    (at-least 1 (k ds-component-of
                 / (at-least 1 (k has-nucleus))
                   (at-least 1
                    (k has-reference-frame-satellite))))
    (all has-reference-frame-satellite
         reference-frame-ds-segment)))
```

The ds-aggregate dsa-1 is subsumed by dsa-with-ref-frame-requiring-nucleus. As a consequence, the ds-aggregate requires a role filler for the role has-reference-frame-satellite. This role filler is the modifier for the nucleus segment. Furthermore, the ds-aggregate must be inserted into the dialog structure. However, this should only happen if the nucleus and the associated reference frame modifier are actually known. In order to support a *conditional role filler computation* in a K-operator, a set of K-cardinality restrictions can be given after a slash (read as "under the condition that"). Filler computation for the role ds-component-of is delayed until the other fillers are actually known as concrete individuals.

3.2.1 Discourse purposes represented as individuals

The display manager needs information about the discourse purpose associated with a place in the runtime net in order to derive constraints for presentation attributes. Each DS segment is associated with an individual describing the discourse purpose (see the use of the K-operator in the definition of spatial-action-ds-segment). The HAMVIS knowledge base contains several different concepts for describing discourse purposes. With each discourse purpose concept, a set of contraints for selecting presentation attributes and a discourse status is associated. The discourse status is required for display management. Currently, the HAMVIS display manager distinguishes the following status descriptors: dynamic, dynamic-persistent and static. A dynamic object will be presented right after the place has been filled. It will be erased at the end of the action. A

segment with discourse status dynamic-persistent describes places that will contain objects that should not be erased after the action is finished.

```
(define-concept descriptor-for-manipulated-object
  (and discourse-purpose
    (fillers has-discourse-status dynamic-persistent)))
```

A static object will be displayed before any action is carried out. An example is a reference system. For a reference frame modifier, the DS segment knowledge base contains the following definitions.

```
(define-concept reference-frame-ds-segment
  (and (all has-discourse-purpose
            reference-frame-modifier) ...)
  (:asserted-concepts
    (at-least 1 (k has-discourse-purpose)))

(define-concept reference-frame-modifier
  (and discourse-purpose
       (fillers has-discourse-status static))
  (:asserted-concepts
     (at-least 1 (k has-static-reference-object))))
```

The current version of HAMVIS requires that reference objects (a filler of the role has-static-reference-objecct) be known at system development time. Thus, the corresponding method for compute-role-fillers must be able to derive a reference object right from the conceptual information known about the object to be found in the place at runtime. The reference object is found by considering restrictions for part-whole relations defined in the HAMVIS Upper Model. This is possible when concepts for domain objects (e.g. cabin objects) are defined with respect to static individuals:

```
(define-primitive-concept cabin-object
  (and spatial-object ...
       (at-most 1 spatially-enclosed-by)
       (fillers spatially-enclosed-by uac-cabin-body)))
```

Part-whole relations like spatially-enclosed-by are also considered for automatically deriving modification functions for computing objects in the environment of a set of other objects. This is required for environment modifiers (see Figure 3) but due to space limitations, this cannot be discussed in detail here.

3.2.2 DS construction variants

The dialog structure constructed so far can be extended by selecting another gray place. If the next place was has-moved-entity (see Figure 1), HAMVIS would allocate another pane with an associated reference frame etc. For XKL, it would be advantageous to use a single pane for object movement and for object creation, especially because the reference system (the cabin body) will be the same for both actions. In the context of IMP systems, André calls this effect “structure sharing” [1]. In order to support this reasoning step, HAMVIS computes a DS variant which restructures the initial DS, “reuses” the first pane and “shares” the reference system etc. The knowledge sources and restructuring algorithms cannot be described in this paper.

3.2.3 Interpreting a DS: Runtime net extension

When all places of a runtime net for a certain activity of an application are considered in the DS, each place is associated with its pane, the discourse purpose and status of the objects etc. Some modification requirements lead to an expansion of the runtime net. Modification functions (e.g., a function for computing the objects in the environment of a set of objects or a function for computing a reference frame) are represented by new transitions with new output places. The extended runtime net and the dialog structure provide the input to the code generation component of HAMVIS.

4 CONCLUSION

With HAMVIS it is shown that design models can be formalized such that knowledge-based UI design support frameworks can be built. Interface “services” provided by HAMVIS are defined as action concepts in a DL knowledge base (compare this to a specification of network “services” in a distributed object-oriented system using type descriptions).

In contrast to IMP systems which operate with concrete objects (at runtime), in the HAMVIS scenario it is important that decisions about the interface structure and about the appearance of graphical interface components be based (mostly) on conceptual information about domain objects and user actions. This paper has shown how this can be supported with DL knowledge bases. Compared to rule-based approaches to dialog (or discourse) structuring (cf. [2]), the approach presented in this paper does not require additional representation mechanims but allows the (bottom-up) construction of a concrete DS directly in terms of ABox statements. The logical mechanisms are hidden behind the HAMVIS user interfaces (see [9] for examples that demonstrate how a UI designer interacts with HAMVIS).

HAMVIS is a first step toward a knowledge-based interface development methodology that allows the UI designer to specify interaction components by explicity modeling relations between visualizations and visualization constituents (for more examples see [9]). The discourse status and discourse purposes of visualization constituents provide the basis for automatically selecting drawing attributes. It has been shown how communication-oriented modeling techniques known from IMP systems can be made available by DL reasoning at system development-time for the construction of standard interfaces which require *less computational resources*. HAMVIS currently supports code generation for CLIM [3] but other backends could also be supported.

REFERENCES

[1] E. André, *Ein planbasierter Ansatz zur Generierung multimedialer Präsentationen*, in German, Dissertation, Univserity of Saarbruecken, Computer Science Department, 1995.

[2] Y. Arens, E. Hovy, S. van Mulken, *Structure and Rules in Automated Multimedia Presentation Planning*, in: Proc. of the IJCAI-93, 1993.

[3] *Common Lisp Interface Manager, User Guide*, Franz Inc., 1994.

[4] G. Fischer, R. McCall, J. Ostwald, J. Reeves, F. Shipman, *Seeding, Evolutionary Growth and Reseedings: Supporting the Incremental Development of Design Environments*, in: Proc.CHI’94 Human Factors in Computing Systems, ACM Press, 1994, pp. 292-298.

[5] J.D. Foley, P.N. Sukaviriya, History, Results, and Bibliography of the User Interface Design Environment (UIDE), an Early Model-Based System for User Interface Design and Implementation, in *Interactive Systems: Design, Specification, and Verification* (ed. F. Paternó), Springer, 1995, pp. 3-14.

[6] H.R. Hartson, K.A. Mayo, A Framework for Precise, Reusable Task Abstraction, in *Interactive Systems: Design, Specification, and Verification*, (ed. F. Paternó), Springer, 1995, pp. 279-297.

[7] M. Leyton, *Symmetry, Causality and Mind*, MIT Press, 1994.

[8] R. Möller, *A Functional Layer for Description Logics: Knowledge Representations Meets Object-Oriented Programming*, to appear in: Proc. OOPSLA’96, San Jose, 1996.

[9] R. Möller, *HAMVIS: Generierung von Visualisierungen in einem Rahmensystem zur systematischen Entwicklung von Benutzungsschnittstellen*, in German, Dissertation, in preparation, University of Hamburg, Computer Science Department, 1996.

[10] R. Möller, *Reasoning About Domain Knowledge and User Actions for Interactive Systems Development*, in: Proc. IFIP Working Groups 8.1/13.2 Conference, Domain Knowledge for Interactive System Design, Geneva 8-10th May, Chapman & Hall Publishers, 1996.

[11] P.F. Patel-Schneider, B. Swartout, *Description Logic Specification from the KRSS Effort*, `ksl.stanford.edu:/pub/knowledge-sharing/papers/dl-spec.ps`.

[12] A.R. Puerta, H. Eriksson, J.H. Gennari, M. Musen, *Beyond Data Models for Automated User Interface Generation*, in People and Computer IX, Proceedings of HCI’94 (ed. Cockton, G., Draper, S.W., Weir, G.R.S.), Glasgow, August 1994, pp. 353-366.

[13] L.A. Resnick, A. Borgida, R.J. Brachman, D.L. McGuiness, P.F., Patel-Schneider, K.C. Zalondek, *CLASSIC Description and Reference Manual for the Common Lisp Implementation*, Version 2.2, 1993.

[14] P. Szekely, P. Luo, R. Neches, Beyond Interface Builders: Model-Based Interface Tools, in *Proc. of INTERCHI’93*, pp. 383-390.

[15] W. Wahlster, E. André, W. Finkler, H.-J. Profitlich, T. Rist, *Plan-Based Integration of Natural Language and Graphics Generation*, Artificial Intelligence, 63, 1993, pp. 387-427.