Managing Design Knowledge with Mixed-Initiative Dialogue

Sambit Datta
Built Environment Research Group
School of Architecture and Building
Deakin University
Australia.

ABSTRACT

This paper is based on ongoing work in developing interactive interfaces to formal methods for encoding design knowledge. It reports on the development of a shared graphical notation to support user interaction with design knowledge based on mixed-initiative. Mixed-initiative provides a model of interaction where both the designer and the knowledge formalism may share responsibility over decisions. The paper discusses how a formal visual notation can support the mixed-initiative mode for developing and managing formal design knowledge. The notation addresses on the dialogue problem between the user and a knowledge based formalism and illustrates a model of interaction in which the user and the formalism can share and input data through a common shared resource, on a common shared task. The paper demonstrates the use of this notation in common decision tasks and the implications for seamless interaction with design support systems.

1 INTRODUCTION

In generative design systems, the interaction between the designer and the generative formalism is typically transactional. The system responds to a single query or utterance with one or more results, defining the transaction. A neat conceptual separation exists between user tasks and system tasks, rendering the modeling of joint responsibility for generation difficult. A mixed-initiative model of interaction is necessary to support forms of dialogue where both the designer and the formalism may share responsibility over the same task. Mixed-initiative provides a robust basis for combining design interaction with automated or intelligent systems. In such systems, the role of both user and formalism are integrated into an incremental based model, where the user and the formalism can share and input data through a common shared resource, on a common shared task.

Generative design systems provide formal representations that aid designers in exploring spaces of designs. Woodbury et al (Burrow, 1999; Woodbury and Burrow, 1999) develop a formal representation of design spaces and unification based algorithms for generation in the specification and implementation of design space explorers, computer programs for constructing, navigating and exploring spaces of design problems using feature structure theory (Carpenter, 1992). They adopt the unification view of incremental generation and provide a representation for the constraints on a design space, relate those constraints to the generation of partial designs and supports intermediate representations of design states. Extending this work with exploration systems, we report on the development of a graphical notation to support mixed-initiative dialogue in generative design systems. We demonstrate the use of this notation in design space exploration and discuss its implications for interaction with generative design systems.

1.1 **Design space exploration**

The exploration formalism (Woodbury et al., 1999) is made out of three sets of components: types ***T***, structures ***F*** and descriptions ***D***, put into a representation scheme within the building design domain. The types comprising ***T*** stand for expressed knowledge of the domain of interest, in this case building design. Structures from ***F*** represent models of particular designs, in this case, buildings and/or their components, either physical or conceptual. Structures from ***F*** are expressed in terms of the information expressed in ***T***. Descriptions from ***D*** are utterances in a formal textual language and correspond to sets of structures from ***F*** in the representation scheme.

The primary representational device for design space exploration is provided by the typed feature structure. Symbols are augmented with types and recursively defined attributes called features. Symbol rewriting is based upon the incremental unification of typed feature structures. The problems associated with symbol matching in earlier systems such as combinatorial explosion are resolved by using feature unification in the generative process. Unification is an efficient algorithm to determine whether one view is more specific than another. The formal generation is incremental and supports stepwise refinement rather than global search for solutions. The use of constraints facilitates the view of domain objects at varying levels of resolution subject to satisfaction. The representation of intermediate states facilitates a collection of partial views of a single domain object at varying levels of specificity. The representation of two structures that possibly represent the same object is recognisable eliminating redundancy in the generative process.

The term *mixed-initiative* refers broadly to methods that explicitly support an efficient, natural interleaving of control by users and automated services aimed at converging on solutions to problems (Allen, 1999). Mixed-initiative dialogue provides a robust and well understood basis for addressing the problem of interaction with generative design systems. In this dialogue view of interaction, instructions are entered via spoken input, typed commands or the direct manipulation of graphical symbols. All modalities of input can be interpreted in a common symbolic representation.

The same applies to different modes of output, whether generated speech, natural language explanations or graphical visualization. Mixed-initiative dialogue involves the exchange of initiative in a flexible and opportunistic manner, shifts in focus of attention to meet user needs and the maintenance of shared contexts. This approach has been developed, applied and tested in the areas of AI planning (Ferguson and Allen, 1994; Burstein and McDermott, 1996), simulation (Amant and Cohen, 1997; Cohen et al., 1997), knowledge engineering (Tecuci G. and Lee, 1999), scheduling (G. Ferguson and Miller, 1996) and computational dialogue systems (Guinn, 1993).

In order to support mixed-initiative dialogue in design space exploration, we develop a unified notation that enables the designer and the formalism to exchange initiative, maintain shared context and supports control, coordination and communication during exploration. The visual notation is based on the following assumptions,

-Typed Feature structures as a representation.
The typed feature structure is a common basis for representing the components of design space exploration as feature nodes. Feature structures provide a rigorous and well-understood basis for representing the objects and entities in a design space as feature nodes.

-Integration of input and output modalities.
The typed feature structure can support the multiple input and output modalities arising out of interaction between the user and the generative system. Feature nodes provide and maintain a shared context for dialogue interactions between the user and the formalism. The unification of typed feature structures integrates both generative and interactive actions of exploration. Given this conception of a feature node and its attributes and mixed-initiative dialogue, the components of the visual notation are examined in the next section.

2 THE VISUAL NOTATION

This section presents the visual notation for representing dialogue using feature structures. This visual form of feature structures is for encoding dialogue in mixed-initiative systems. First, the straightforward adaptation of feature structure notation to represent feature nodes is described. The representation is then extended with direct manipulation to support interaction dialogue between the designer and the generative formalism. All communication between the designer and the generative formalism is based on graphical interaction with the notation.

2.1 **Feature structures as a representation**

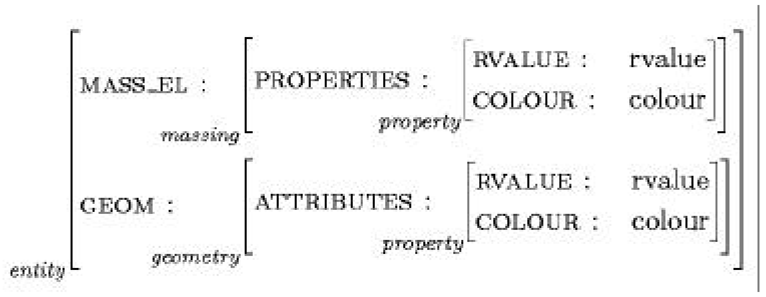


Figure 1: **Feature structure in AVM notation. The feature structure entity has the features, MASS_EL and GEOM. These features have two substructures of type massing and type geometry.**

The analogy between feature structures and frame-based representations provides a more standard graphical notation for large collections of feature structures. By interpreting each node as a frame, the features on arcs can represent slot labels, and the arcs themselves point to the slot fillers. The only difference is the natural enforcement of unique-value restrictions on slot values in feature structures, since the arcs are modelled by using a partial function. This frame-like notation, called attribute-value matrix or AVM notation is the standard form for feature structures used in the description of linguistic fragments modelled by feature structures. In AVM notation, each node is represented with the frame delimiters "[" and "]". The frame is annotated with the type of the node, as shown in Figure 1. Tags numbered with labels such as 4 indicate re-entrancy, or structure sharing. The slots are the features and the values are written next to them, as shown in Figure 2. We develop this AVM notation to represent the feature nodes and their attributes. First, the elements of a typed feature structure, types, features and their values are mapped to a visual representation of feature structures using elements of the AVM notation.

The visual representation comprises the representation of types, features, their values and co-references. The smallest element of the visual feature structure notation is the feature-value pair, comprising the relation between a feature and its value, which may be atomic, complex or another feature structure. For example, the feature-value pair, [MASS_EL : **massing**] represents the functional relation between the feature, MASS_EL and its value, which is minimally the type, **massing.**

In standard AVM notation, the value of a feature may also be another feature structure. For example, the value of MASS_EL may be itself be an arbitrary feature structure. We extend the usual AVM notation to include feature values which are complex, such as a query description, resolution step or function application or external complex data type. The representation of the value of a feature *f*, is given by an element called a **feature-value map.** This term denotes the relation between a feature and the feature structure that is its value.

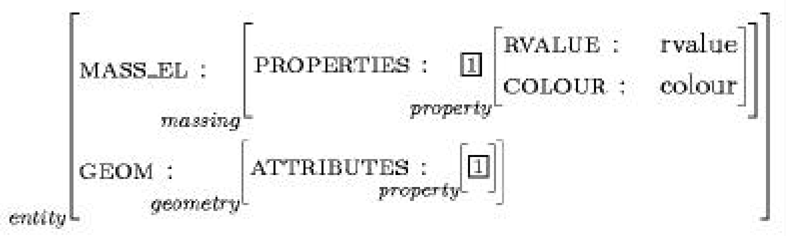


Figure 2: **A folded feature structure with co-reference notation. The shared feature structure of type property is indicated by the co-reference tag**

The feature-value map shown above represents the functional relationship between the type **geom** and its value. The feature-value map is enclosed by the delimiters "[" and "]" and annotated by its type, **geometry**. Since feature structures are recursive, the values of feature structures may be another feature structure; the attribute-value notation is easily adopted for a visual representation of the feature-value map. The feature-value map can be conceptualized as a recursive container of entities of type feature-value pair.

Finally, in a visual feature structure, two or more features can share the same information. Each shared structure is represented by co-references, also called tags. The co-reference ***n*** denotes an index value and the identity of the node that is shared between one or more feature structures. Co-references and their denotation by indices are a straightforward adaptation from the AVM notation.

2.1.1. Interaction with feature nodes

The formal properties of visual feature nodes provide a means for combining human interaction with the generative process. In this section, we extend the representational properties of feature nodes to incorporate graphical interaction and direct manipulation of feature nodes. The functionality of the visual components of feature nodes can be extended by interaction logic to incorporate complex behavior attributes. The feature node is extended with an interaction logic providing the ability to unfold feature nodes through zooming and imploding substructures, interaction with conjuncts, disjuncts, functions and commands.

2.1.2. Zooming and imploding substructures

Feature structures can be nested recursively up to arbitrary levels, and can contain many recursively nested substructures. It is necessary to provide a mechanism for folding the nested substructures of a feature-value map to hide their underlying notation and for unfolding the imploded structure, when it is necessary to see the details of a feature-value map. In the visual representation, this is realised by incorporating interaction logic on a feature-value map. This is described in Figure 3.

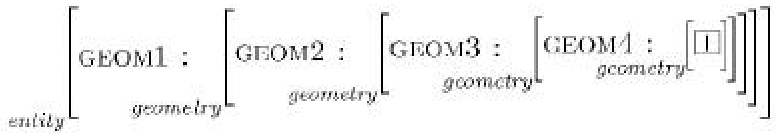


Figure 3: **An example of an automatically imploded feature structure during deep containment. The symbol indicates that the feature-value map of type entity is hidden automatically after three levels of nesting. User interaction on this node is necessary to unfold these structures, but on regeneration will resume their default behavior.**

The feature-value map can be in one of two modes, either open or closed. This is represented visually by annotating the feature-value map with the symbol, "+".

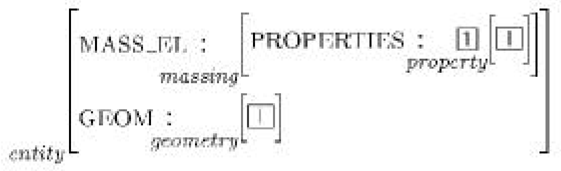


Figure 4: **An example of an imploded feature structure. The symbol indicates that the feature-value map of type property and geometry contains nested substructures which can be unfolded by user interaction.**

The unfolding symbol, "+" shows up in two different situations. A restriction may be placed on the depth of display of a feature-value map. Any substructure in a feature-value map that exceeds that depth is represented by the symbol, "+". This is automatically managed by the dialogue layer and the nesting levels set through preferences. The user can also manipulate the feature-value map interactively. In this situation, the feature-value map will be shown as folded, until it is explicitly unfolded. Thus, the "+" symbols on the feature-value map coming from depth restriction are generated and removed dynamically while the user navigates a feature structure. In contrast, the maps that are unfolded manually need explicit interaction to change their mode. This enables the user to control the level of detail, shown in Figure 4, while zooming and imploding substructures.

$$
{}_{entity}\begin{bmatrix}
\text{MASS\_EL}: & {}_{massing}\begin{bmatrix}\text{PROPERTIES}: \boxed{1}\;{}_{property}\begin{bmatrix}\text{RVALUE}: & \text{rvalue}\\ \text{COLOUR}: & \text{colour}\end{bmatrix}\end{bmatrix} \\
\text{GEOM}: & {}_{geometry}\begin{bmatrix}\text{ATTRIBUTES}: {}_{property}\begin{bmatrix}\boxed{1}\end{bmatrix}\end{bmatrix} \\
\text{FUNCTION\_UNIT}: \text{fu} & \\
\text{DESIGN\_UNIT}: \text{du} &
\end{bmatrix}
$$

Figure 5: **An example of substitution of a feature-value map with a coreference tag, 1. The coreference tag is an index to the nested typed feature structure of type property that is shared by the features, PROPERTIES and ATTRIBUTES**

Co-reference tags in the visual representation indicate structure sharing. They are used in two ways. Firstly, a co-reference is used to annotate a feature-value pair that structure-shares a feature-value map. Secondly, it is used to simplify the visual representation of feature structures, by simple substitution of the shared feature-value map by the co-reference tag, denoted by n. This is shown in Figure 5. The co-reference tag can also be substituted by the feature structure it denotes through user interaction. If the feature structure is represented, the co-reference appears outside the feature-value map, as shown in the value of properties. If the co-reference is used to nest the feature structure, it appears inside the feature-value map as shown in the value of attributes.

2.1.3. Interaction with functions and commands

The operations of the description language, conjunctions, disjunctions, implications, lists and function applications, need to be incorporated into the graphical representation of feature structures. Such an interface, must allow for the following, namely,

- Extensions to the graphical interaction to enable the display of conjunctions and disjunctions.
- Provide the interaction logic for integrating implications, resolution steps and function applications.

2.1.4. Interaction with conjunctions and disjunctions

Conjuncts and disjuncts can be represented using visual feature structures using the same notation as feature value pairs. In place of the feature labels, the labels conjunct and disjunct are used, with the values as feature structures. This common representation can be scaled to represent the conjunction of disjuncts and the disjunction of conjuncts. In linugistic attribute-value formalisms, conjuncts and disjuncts are denoted by special delimiters, such as " [" and "]" and their edge names are either omitted, suppressed or obscured (Kiefer and Fettig, 1995). This is not necessary in the interactive representation outlined above. An example of a conjunct of disjunctive feature nodes is shown in Figure 6. User interaction is necessary to resolve the structures associated with the features mass_el and geom of the feature structure of type, entity.

The nodes conjunct 1 and conjunct 2 are represented as a feature value map. Each feature-value map has a type conjunct n, where n is an index over conjunct nodes. Each conjunct feature value map has four possible disjuncts, each of which are represented as a feature-value pair, whose features are defined by disjunct n where n is an index over disjuncts.

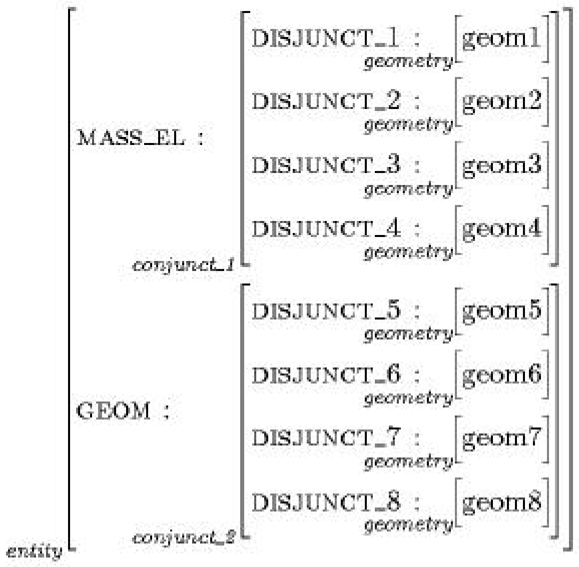

MASS_EL :
DISJUNCT_1 : geometry [geom1]
DISJUNCT_2 : geometry [geom2]
DISJUNCT_3 : geometry [geom3]
DISJUNCT_4 : geometry [geom4]
conjunct_1
GEOM :
DISJUNCT_5 : geometry [geom5]
DISJUNCT_6 : geometry [geom6]
DISJUNCT_7 : geometry [geom7]
DISJUNCT_8 : geometry [geom8]
conjunct_2
entity

Figure 6: **An example of a conjunct of disjunctive feature nodes. The conjunct nodes conjunct 1 and conjunct 2 are represented as a feature-value map and each disjunct is represented as a feature-value pair, whose features are defined by disjunct *n* where *n* is an index over disjuncts.**

2.1.5. Incorporating resolution steps

There are two ways to visualize functions for mixed-initiative interaction. Functions can be encoded within the representation such that the functor annotates the feature-value map and the arguments are features. An example of the duality of a function and its arguments with a feature node representation is shown in Figure 7. This representation allows the feature structure to encode the traditional commands found in geometry-based design systems. The specification of a command or function then returns a value, which can be atomic, complex or a feature structure. The use of feature structures to encode functions can also be used to pass commands. The expressiveness of feature structure command representations need to address the possibility of cyclic feature structures and structure sharing. Cyclic feature structures present challenges for the use of feature structures as a representation of visual commands. A cycle arises when following a non-empty sequence of features out of a node leads back to that node, which is a useful property in state-space models and the finite modelling of knowledge (Carpenter, 1992). A recognition mechanism is necessary to interrupt infinite loops in the commands.

$$append(X, Y) \Longleftrightarrow {}_{append}\left[\begin{array}{l} \text{ARG1} : {}_{arg}\left[\text{X}\right] \\ \text{ARG2} : {}_{arg}\left[\text{Y}\right] \end{array}\right]$$

Figure 7: **Encoding a function as a feature-value map. The function *append(X, Y)* which concatenates values can be represented as the feature-value map of type, append and the two features ARG1 and ARG2. The features, ARG1 and ARG2 encode the values *X* and *Y* as two feature value pairs**

When functions occur as feature values, structure sharing is not considered to be a valid part of the command syntax. If co-references occur, the structures they represent are copied uniquely within each command. The second way of visualising functions within feature structures is to encode the functional definition as the value of a feature-value pair.

$${}_{design\_unit}\left[\begin{array}{l} \text{GEOM: geom1} \\ \text{TRANSFORM: } \langle translate(a, b, c) \rangle \end{array}\right]$$

Figure 8: **The function *translate(a,b,c)* can be represented a feature-value pair and contained within a visual feature structure, with feature TRANSFORM and value, *translate(geom)***

In this scheme, for a function append(X,Y) with two arguments, there exists a type function which introduces a feature function, such that its value is a function definition with the syntax, append(X,Y).

An example of the feature-value pair representation of a procedural function is shown in Figure 8. User interaction on this node involves three possible behaviours, the application of the function to an appropriate node results in a new feature structure, consistent with the application.

The unification of a functional node with an appropriate feature structure, results in a new feature structure, following the unification of typed feature structures. Finally, the function can be unfolded into its constituent subparts following the standard interaction and its values subject to exploration. An example of the latter is shown in Figure 9. The unfolding of a functional representation shows the feature structure dual of the function, translate(a,b,c) is of type **translate** and the arguments are the three feature value pairs, TX, TY and TZ.

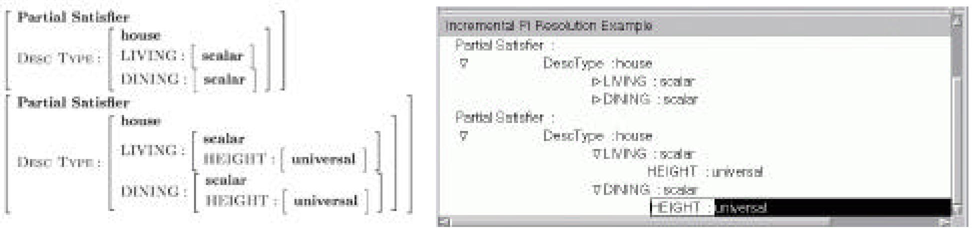
$$\left[\begin{array}{ll} \text{GEOM: geom1} & \\ \text{TRANSFORM:} & \left[\begin{array}{l} \text{TX : } {}_{arg}\left[\text{a}\right] \\ \text{TY : } {}_{arg}\left[\text{b}\right] \\ \text{TZ : } {}_{arg}\left[\text{c}\right] \end{array}\right]_{translate} \end{array}\right]_{design\_unit}$$

Figure 9: **An unfolding of a functional representation shows the feature structure notation of the function, translate(a,b,c). The type of the function is translate. The arguments are unfolded into the three feature value pairs, TX, TY, TZ**

This representation of functions, commands and their arguments extends the visual feature structure notation for user interaction. The behaviour of functions and commands during interaction, namely function unfolding, function application and function unification can be added to the interaction logic necessary for exploration.

3 MIXED-INITIATIVE DIALOGUE

The exposition thus far concentrates on the generic attributes of a visual notation for enabling mixed-initiative dialogue in generative design systems. The combination of typed feature structure unification, the recursive containment of feature nodes and the interaction behavior of the notation for feature nodes provides a sound representation for combining human interaction with the generative process. In this section we provide an example of the behaviour of the notation in a mixed-initiative interface implemented in an experimental mixed-initiative system, FOLDS. The visual representation of a description query and translation of the visual representation into an interactive feature node in FOLDS is shown in Figure 10. User interaction in FOLDS involves three possible behaviours, the application of an unfolding operation to an appropriate node results in a new feature structure, consistent with the application. The unification of a type with an appropriate feature structure, results in a new feature structure, following the laws of unification for typed feature structures.


```
Partial Satisfier
            house
Desc Type : LIVING : [ scalar ]
            DINING : [ scalar ]
Partial Satisfier
            house
Desc Type : LIVING : [ scalar
                       HEIGHT : [ universal ] ]
            DINING : [ scalar
                       HEIGHT : [ universal ] ]
```

```
Incremental FI Resolution Example
Partial Satisfier :
▽           DescType  :house
                ▷LIVING  :scalar
                ▷DINING  :scalar
Partial Satisfier :
▽           DescType  :house
                ▽LIVING  :scalar
                        HEIGHT  :universal
                ▽DINING  :scalar
                        HEIGHT  :universal
```

Figure 10: **The visual representation of a description query and translation of the visual representation into an interactive feature node in FOLDS**

Finally, the feature node can be unfolded into its constituent subparts following the standard interaction and its values subject to exploration.

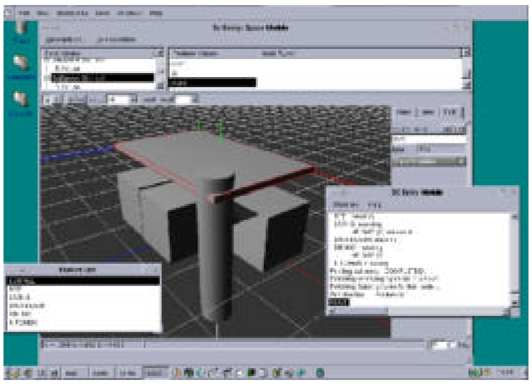


Figure 11: **Implementation of interactive unfolding of a feature-value map using the visual representation of a feature node in FOLDS**

In this example, a description query returns a partial satisfier of type **sfc** comprising three feature-value pairs, LIVING, PORCH, DINING and their most general substructure nodes. The elements can be expanded and imploded using the triangular arrows, while the selected feature-value pair, size of type **vector** can be subject to exploration through mixed-initiative. Note that the interactive visual feature node interface as well as the description interpreter in the background enables communication with the generative formalism. FOLDS comprises a graphical view of the satisfier space and the feature value nodes that it contains, shown in Figure 11. The visual representation of the current state of the dialogue facilitates the growth and traversal of the design space along the attributes of a feature node.

4 RELATED WORK

Linguistic formalisms as well as constraint programming languages use feature structures as a basic data structure. FEGRAMED (Kiefer and Fettig, 1995) develops a visual tool that is built to cope with the complexity of feature structures in constraint-based systems. St. Amant et al (Amant and Cohen, 1997) combine the dialogue view of mixed-initiative with direct manipulation techniques in the domain of abstract force simulation and exploratory data analysis. They focus on the ability of an interactive environment to constrain and guide both automated agent behavior as well as human effort. Rich and Sidner (Rich and Sidner, 1997; Rich and Sidner, 1998) discuss the design of a collaborative interface agent, which works on a plan with its user and communicates via mixed-initiative dialogue. Hartrum et al (Hartrum and DeLoach, 1999) proposes a mixed-initiative system in which humans interact directly with software agents in a collaborative framework for problem solving. A language processing architecture, based on typed feature structure unification, which supports both speech and gesture is proposed in (Johnston et al., 1997). This architecture allows simultaneous input from speech and gesture recognition interfaces. It is implemented in the QuickSet system (Cohen et al., 1997), a multimodal pen/voice system that enables users to set up and control distributive interactive simulations. The ICE, Incremental Configuration Engine project (Zeller and Snelting, 1995; Zeller, 1997) applies feature logic to the problem of incremental configuration management. Feature logic is used as a unifying formalism for the description of variants and revisions, where sets of versions rather than single versions are the basic units of reasoning. This approach allows for a configuration thread to be constructed by adding or modifying configuration constraints until either a complete configuration or an inconsistency can be deduced. Chien (Chien and Flemming, 1996; Chien, 1998) addresses the problem of design exploration in interactive contexts and employs the navigation and wayfinding metaphors arising in physical and information spaces to address the problem of interaction with generative design systems. Schulte (Schulte, 1997) presents a novel visual constraint programming tool, Oz Explorer, intended to support the exploration, visualization and development of constraint programs. It uses a user-driven and interactive search tree of a constraint problem as its central metaphor.

5 DISCUSSION

In conclusion, the notation we have described in this paper has potential for incorporating mixed-initiative dialogue in generative design systems. The notation extends the modelling of incomplete or partial situations in design space exploration. It enables the user and the formalism to communicate and exchange initiative through interaction.The notation addresses the requirements for exchange of initiative during exploration. Incrementality provides the basis for developing a model of turn-taking between the formal resolution process and the user during exploration. As the steps of exploration change the design space dynamically, the user and the resolution process can shift their focus of attention dynamically, synchronise their paths to meet the shifts in exploration and provide context for the next step of exploration. The notation imposes a consistent and shared model of dialogue against which dynamic change may occur.

The notation schema is computable from linguistic forms of generation based on constraint unification and it is independent from the surface syntactic structure of generative formalisms. While it is based on a logical form, it is not limited to supporting a specific generative formalism. In the example application, logic plays an important role as an inference mechanism, but the notation itself avoids this regimentation. While supporting dialogue in a principled way, it remains open to multiple modalities of input and output, including voice and gesture. While the work reported in this paper emerged out of research into supporting more intuitive forms of input based on a logical description language, extending this work through research on visual formalisms provides a possible direction of future work.

6 REFERENCES

Allen, J. F. (1999) Mixed Initiative Interaction, *Proc. IEEE Intelligent Systems* **14**(6), pp.14-23.

Amant, R. S. and P. R. Cohen (1997) Interaction with a mixed-initiative system for exploratory data analysis, *Proceedings of Intelligent User Interfaces*, pp. 15-22.

Burrow, A. L. (1999) Computational Design and Formal Languages. Ph.D. thesis, Department of Computer Science, Adelaide University, Australia.

Burstein, M and D. McDermott (1996) Issues in the development of human-computer mixed-initiative planning, in Gorayska B. and J. L. May (eds), *Cognitive Technology: In Search of a Humane Interface*, Elsevier Science. Amsterdam, pp. 285-303.

Carpenter, B(1992) *The Logic of Typed Feature Structures with applications to unification grammars, logic programs and constraint resolution*, Cambridge Tracts in Theoretical Computer Science. Cambridge University Press.

Chien, S. F. (1998) Supporting information navigation in generative design systems. Ph.D. thesis, School of Architecture, Carnegie-Mellon University.

Chien, S. F. and U. Flemming (1996) Design Space Navigation: An Annotated Bibliography. technical report EDRC 48-37-96, Engineering Design Research Center, Carnegie-Mellon University, Pittsburgh, PA, USA.

Cohen, P.R. and M. Johnston, and D. McGee, et al. (1997) QuickSet: Multimodal interaction for distributed applications, *Proceedings of the fifth international multimedia conference,*ACM Press, pp. 31--40.

Ferguson, G. and J. F. Allen (1994) Arguing About Plans: Plan Representation and Reasoning for Mixed-Initiative Planning,*Proceedings of the 2nd International Conference on AI Planning Systems,* Chicago, IL, pp. 43--48.

Ferguson, G and J. Allen and B. Miller (1996) TRAINS-95: Towards a mixed-initiative planning assistant, *Proceedings of the 3rd International Conference on AI Planning Systems,*Edinburgh, Scotland, pp. 70--77.

Guinn, C. (1993) A computational model of dialogue initiative in collaborative discourse, *AAAI93 Fall Symposium on Human-Computer Collaboration,* Raleigh, NC, pp 32-39.

Hartrum, T. and S. A. DeLoach (1999) Design Issues for Mixed-Initiative Agent Systems,*Proceedings of the AAAI-99 Workshop on Mixed-Initiative Intelligence*. Orlando Fl.

Johnston, M. and P. Cohen and D. McGee et al. (1997) Unification-based Multimodal Integration, *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, Madrid, Spain, pp. 281--288.

Kiefer, B. and T. Fettig (1995) FEGRAMED : An Interactive Graphics Editor for Feature Structures. Research Report RR-95-06, German Research Center for Artificial Intelligence (DFKI), Saarbrucken, Germany.

Rich, C. and C. L. Sidner (1997) COLLAGEN: When agents collaborate with people, Johnson W. L. and B. Hayes-Roth (eds), *Proceedings of the first international conference on autonomous agents* (Agents'97), ACM Press, New York, pp. 284-291.

Rich, C. and C. L. Sidner (1998) COLLAGEN: A collaboration manager for software interface agents, *User Modeling and User-Adapted Interaction,* **8**(3-4), 315--350.

Schulte, C. (1997) Oz Explorer: A Visual Constraint Programming Tool, *in* L. Naish (ed), *Proceedings of the Fourteenth International Conference on Logic Programming*, Leuven, Belgium, The MIT Press, pp. 286--300.

Tecuci, G. M. and K. W. Boicu and S. Lee (1999) Mixed-Initiative Development of Knowledge Bases, *Proceedings of the AAAI-99 Workshop on Mixed-Initiative Intelligence*. Orlando Fl.

Woodbury, R. F. and A. Burrow and S. Datta et al. (1999) Typed feature structures and design space exploration, *Special Issue on generative systems, Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, **13**(4), pp. 287-302.

Woodbury, R. F. and A. L. Burrow (1999) Pi-resolution and design space exploration,in C Eastman and G Augenbroe (eds*), Proceedings of CAAD Futures'99, in, Computers in Building*, Kluwer Academic Publishers, pp. 291-308.

Zeller, A. (1997) Configuration Management with Version Sets - A Unified Software Versioning Model and its Applications. Ph.D. thesis, TU Braunschweig. Germany.

Zeller, A. and G. Snelting (1995) Handling version sets through feature logic, in W. Schfer and P. Botella (eds.), *Proc. 5th European Software Engineering Conference (ESEC), Vol. Vol. 989 of Lecture Notes in Computer Science*. pp. 191--204.