π-Resolution in Design Space Exploration

Andrew Burrow
Department of Computer Science,
The University of Adelaide SA 5005, Australia.

Robert Woodbury
School of Architecture, Landscape Architecture and Urban Design,
The University of Adelaide SA 5005, Australia.

Key words design space explorers, typed feature structures, functional decomposition, mixed initiative

Abstract In studying the phenomenon of design we use models to envision mechanisms by which computers might support design. In one such model we understand design as guided movement through a space of possibilities. *Design space explorers* embody this model as mixed-initiative environments in which designers engage in exploration via human computer interaction.

Constraint resolution provides a formal framework for interaction in design space explorers. Rather than directly providing solutions to design problems, constraint resolution provides a mechanism for organizing construction. Therefore, we are less interested in the set of solutions to a constraint problem than the process by which intermediate steps are generated.

π-resolution is one such mechanism applicable to design space explorers. It describes the solution, by recursive enumeration, of feature structure type constraints. During π-resolution, satisfiers are constructed by the application of type constraints drawn from an inheritance hierarchy. This constructive process provides a strong model for design space exploration. The constraint solver does not do the work of the designer, but rather design efforts are situated in, and organized by, constraint resolution. Therefore, the efficiency of the recursive enumeration in finding solutions is not an issue, since non-determinism in the search is resolved by the human user as design space exploration.

1. INTRODUCTION

Computer aided design environments involve a conflict between human and machine requirements. One approach is to suspend the search for an account of the human designer, and instead to accept metaphors of design that correspond well to formal systems. The intuition is that tools should provide a logic comprehensible to the user, rather than attempt to simulate sophisticated forms of human reasoning, and the hope is that users respond by improvising with the media.

The metaphor of design as exploration within a space of possibilities, suggests mechanisms by which computers might support design. *Design space explorers* are computer media that engage designers in exploration. Appropriate formal systems, on which design space explorers may be implemented, arise in the description of generative processes. One example, is boundary solid grammars in the GENESIS system (Heisserman, 1994).

One difficulty facing shape grammars, and other formal systems focused on the design artifact, is that the extant formalisms give little guidance in structuring the design space exploration. This manifests itself largely in the representation of goals and goal-directed movement in a design space. Lacking a user comprehensible logic in goals and exploration works against achieving clarity in user interaction.

Another approach is to explicitly consider goals in exploration and to structure a formal representation and exploration process using those goals. In such systems, transformations of the goal may become the focus of the interaction between human and machine. Here the user accessible logic is a form of problem decomposition. This paper argues that feature structure π-resolution is a formal mechanism which provides an excellent framework for design space exploration. This fit is improved by extensions which explicitly characterize the constructive resolution process and the intermediate states.

The aim of this paper is to provide an intuition of the formal mechanism of typed feature structures that allow us to both express and generate design representations characterised by complex functional roles. Our experience is that this intuition does not come easily, since most of us are attuned to an object-oriented view derived from object-oriented programming techniques. By contrast the typed feature structure formalism is terse and organised around a global view of the data structure. Therefore, where possible this paper omits the formal aspects of the mechanism in favour of the underlying intuitions. (Woodbury et al., 1999) provides a more detailed introduction.

1.1 Design Space Explorers

Engineering design space explorers involves representing possibilities in the form of symbol structures called *design states*, and *design spaces* of possibilities as a relation over design states. This requires a compact representation of design states with efficient algorithms, and definitions of the space-generating relation which are relevant to the exploration metaphor and which lend themselves to tractable computations.

A pervasive notion within the exploration metaphor is goal-directedness. In essence, goals exist independently of the designs that realize them and the process of exploration is guided by a set of goals that itself may be transformed within the exploration process. Computing with goals requires their representation. Since goals may change throughout an exploration process, design states record both the goals towards which they aim and the artifacts that (partially) realize those goals.

Another notion is that of non-determinism — the goal does not of itself determine the decisions enacting the exploration. An external agent, the user, must resolve the indeterminacy in seeking the goal. In design space explorers, each decision is expressed in the transformation of one design state to another. A design space is implicitly defined by a set of initial design states and a set of state transforming operators. A design space explorer assists a user to unfold this compact and implicational description into an explicit space of recorded possibilities.

Design space explorers provide a mixed initiative environment where user and computer play complementary roles. A user experiences the design space explorer, inter alia, through resolving exploration non-determinism. Support for this role is therefore critical to the external qualities of a design space explorer. Careful presentation of alternatives avoids overwhelming a user, and assists a user to perceive coherent movements in the design space.

The experience from extant spatial grammar based tools is that a necessary modification to their formalisms and thus their interpreters is a mechanism to control the availability of transformations so that construction occurs in coherent phases. The representation of operators should reflect this need to structure and organize alternatives. Alternatives may be mutually exclusive, interdependent or independent. Without a structuring principle for operators, these properties will be evident only in patterns distributed across the design space. With a structuring principle for operators, these properties may be expressed in the presentation and grouping of alternatives.

A design space explorer is then a device that represents goals, operators, and design possibilities. It organizes design states into spaces, and provides means to access states and apply operators to extend the explored space. A designer interacts with an explorer through states, spaces, operators, and the presentation of their interactions. The design space explorer is thus an active participant in a mixed initiative environment based on the exploratory unfolding of design spaces.

1.2 π-Resolution and Design Space Explorers

Feature structures generalize record-like data structures. They are a form of directed graph, where nodes represent domain objects and edges represent associative connections. Object identity is modeled by structure sharing and the structure as a whole models partial information. Since feature structures model partial information there exists an informational ordering with an efficient decision procedure, and an operation to combine information where consistent. In addition, *descriptions* provide a means for calling out collections of feature structures based on *satisfaction*. Satisfaction has the property that, if a feature structure satisfies a description, then so too does every feature structure which extends the information in it.

These properties of feature structures are attractive in design space explorers. As *representatives* for design states, explicit models for partial information lend legitimacy to intermediate exploration states. In particular, we represent functional decompositions of design states by feature structures, and functional roles in these decompositions by features. In exploration terms, extending a feature structure corresponds to answering "How?" questions which transform the problem to include finer levels of subproblem. This is an attractive process as the surface logic of a design space explorer, because it captures a notion of goal directedness in terms of the problem at hand.

π-resolution is a non-deterministic search for solutions to a query description. A solution is a feature structure, which satisfies the query and the *constraint system*. It is the result of a sequence of extension steps corresponding to the satisfaction of constraints, which are organized into an inheritance hierarchy of types. Since π-resolution proceeds by extension, the resultant search space can be order embedded into the informational ordering over feature structures. Since constraints are drawn from an inheritance hierarchy, alternatives may be organized according to notions of abstraction. If feature structures represent functional decompositions, π-resolution non-determinism allows exploration in terms of alternative functional decompositions.

User input accounts for human judgment in the design process, but the intervention associated with this input is also important. Human judgment could be applied without intervention by enumerating the design space and having the user select from the solutions. One issue is computational intractability — in exploration, the user allocates computer resources by guiding exploration. More important is the cognitive load on the user. Design environments should leverage, rather than squander, human skills. Well informed partitioning of solutions, via exploration, depends on exploration choices comprehensible in terms of user-defined distinctions.

This brings us to π-resolution as a model for design space exploration and to the search space of feature structures as a record of exploration. Given that feature structure types and their constraints capture useful distinctions, π-resolution monotonically refines those distinctions, and an element in the search space records the distinctions made locally. Navigation in the space of alternatives then proceeds by asking "which of the decisions important to me was made here?" Since the search space order embeds into the informational ordering the classification and indexing techniques in the knowledge representation literature are available for content based indexing and querying.

1.3 Related Work

Design space explorers have their genesis in the shape grammar literature (Stiny, 1980; Stiny and March, 1981). Later researchers focused on the formal system and human-computer interfaces required to create effective design space explorers (Heisserman, 1994; Carlson, 1993; Coyne, 1991; Carlson and Woodbury, 1994; Woodbury et al., 1992; Harada et al., 1995; Woodbury and Chang, 1995).

Feature structure theory has its provenance in computational linguistics as described by Carpenter (Carpenter, 1992), but owes much to its application to logic programming by Aït-Kaci(Aït-Kaci and Podelski, 1993), who was also the first to realise the potential optimisations for reasoning operations on bounded complete partial orders (Aït-Kaci et al., 1989). The breadth first π-resolution techniques for recursive type constraints in feature structures also grew out of work on feature structures as a term language for logic programming (Aït-Kaci et al., 1993).

The use of order theory for information management and presentation is based on techniques for searching in partial orders (Ellis, 1995) and on the notion of lattices as a media for philosophical discourse as proposed in Formal Concept Analysis (Wille, 1992).

There are numerous knowledge level characterisations of design information stressing the role of function. For example, the SEED knowledge level (Flemming and Woodbury, 1995) and the SHARED object model(Gorti et al., 1998). Some of these are sufficiently explict and parsimonious to admit a symbol level interpretation.

2. FEATURE STRUCTURES

In this section we define a feature structure language(Carpenter, 1992) in sufficient detail to discuss π-resolution. Since the resolution of type constraints may be described independently of the class of feature structures(Carpenter, 1992, p 228), we do not consider type inference, inequations, or extensional types. Furthermore, we do not consider descriptions in detail except to note their use in representing queries and type constraints. Rather than provide formal definitions for subsumption and unification we consider these mechanisms in terms of functional decomposition.

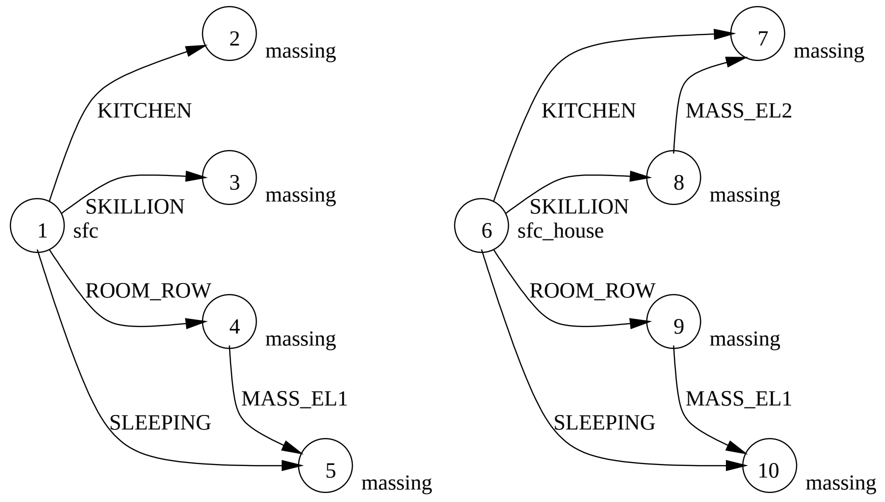

Figure 1: Feature structures in graph notation.

Feature structures generalize record-like data structures and represent partial information. They may be depicted as rooted, directed, finite, node and edge labeled graphs, where nodes represent objects, edges represent functional relationships, and structure sharing is taken to model identity. Therefore, cycles represent reflexive relationships, and informational equivalence does not assert identity. Each node in a feature structure may be interpreted as the root of a feature structure, termed a *substructure*. At nodes, labels are interpreted as types, and on edges, labels name the functional role that the target fills wrt the source.

Types are exploited as a means of efficiently encoding information. They are organized into a multiple inheritance hierarchy, in which information associated with a type is extended in inheriting types, i.e., an informational ordering. It is mandated that this ordering is a bounded complete partial order (BCPO), meaning that for every set of types with a common subtype there is most general common subtype or join. Thus, certain reasoning operations over types are available as the lattice operations $\sqcup$ (join, or most general common specialization) and $\sqcap$ (meet, or most specific common generalization).

Figure 1 depicts a pair of simple feature structures in a subsumption relation. The example represents the decomposition of a building design, the nodes denote building entities and the edges denote functional roles in the design. By characterising feature structures as representations of partial information, it is possible to consider both examples as partial representations of some final design. The second feature structure provides additional information about the final design as recognised by the subsumption relation.

The example demonstrates three sources of ordering in the information. The simplest sources of ordering are the elision of feature paths and the ordering over the type labels. In this case, the first feature structure does not define the feature path SKILLION MASS_EL2, and the type of the first root node is a supertype of the second's. However, since structure sharing is taken to model identity, information on the identity of the represented objects is encoded in the feature paths. Each node is interpreted as an existential proposition that there exists some object conforming to the node's type. The existentially quantified node cannot be interpreted as stating the identity of the represented object, i.e., two distinct nodes may represent the same object or separate objects. Instead object identity is not specified except wrt the feature paths extending to a node. The second feature structure identifies a single object as fulfilling the roles of both KITCHEN and SKILLION MASS_EL2, which is a piece of information not present in the first feature structure.

Since designs are inherently existential representations that are made more specific by proceeding stages of the design process, the notion of partial information captured in typed feature structures is a useful one. This is especially true of the functional decomposition of a design. Monotonic satisfaction between *descriptions* and feature structures allows us to describe a constructive mechanism for functional decomposition. For each description, either an information minimal feature structure exists that satisfies the description, or the description is unsatisfiable. Since every specialisation also satisfies the original description, the union of feature structures satisfying a collection of descriptions must also satisfy every description in the collection and thus satisfy their conjunction. Hence we are able to express information about designs by descriptions, and construct feature structures by *unification* operation as a means of directly processing the descriptions.

2.1 Feature Structures

Given an inheritance hierarchy, a BCPO of types $\langle \mathsf{Type}, \leq_{\mathsf{Type}} \rangle$ and a set of features Feat, a typed feature structure is formally defined as a rooted, directed, finite, node and edge labeled graph.

Definition 1 (Feature Structures(Carpenter, 1992)) *A* feature structure *is a tuple* $F = \langle Q, \bar{q}, \theta, \delta \rangle$, *where*

- Q *is a finite set of* nodes,
- $\bar{q} \in Q$ *is the* root node,
- $\theta : Q \mapsto \mathsf{Type}$ *is the node* typing *function,*
- $\delta : \mathsf{Feat} \times Q \rightarrow Q$ *is the partial* feature value *function, and*
- Q *is the smallest set such that* $\bar{q} \in Q$ *and* $q \in Q \wedge q' = \delta(f, q) \rightarrow q' \in Q$.

A shorthand for composition of the feature value function is useful to denote traversal along a sequence of features. Following Carpenter(Carpenter, 1992), we refer to a sequence of features as a *path*, the collection of paths as $\mathsf{Path} = \mathsf{Feat}^*$, and the *empty path* as ϵ. We then extend the feature value function to paths, such that $\delta(\epsilon, q) = q$ and $\delta(f\pi, q) = \delta(\pi, \delta(f, q))$.

Now, feature structures can be interpreted as structures assigning values to paths. The feature value function assigns nodes to paths. The typing function assigns types to these nodes. A node may also be interpreted as the root of a feature structure. Hence, types and feature structures are also assigned to paths.

Definition 2 (Path Value(Carpenter, 1992)) *Given a feature structure $F = \langle Q, \bar{q}, \theta, \delta \rangle$ and a path π such that $\delta(\pi, \bar{q})$ is defined, the* substructure *at π is the feature structure $F@\pi = \langle Q', \bar{q}', \theta', \delta' \rangle$, where*

$$Q' = \{\delta(\pi', \bar{q}') \mid \pi' \in \mathsf{Path}\} \ , \tag{1}$$

$$\bar{q}' = \delta(\pi, \bar{q}) \ , \tag{2}$$

$$\theta'(q') = \begin{cases} \theta(q') & : \ q' \in Q' \\ \mathit{undefined} & : \ \mathit{otherwise} \end{cases} \ , \mathit{and} \tag{3}$$

$$\delta'(f, q') = \begin{cases} \delta(f, q') & : \ q' \in Q' \\ \mathit{undefined} & : \ \mathit{otherwise} \end{cases} \ . \tag{4}$$

2.2 Subsumption

The informational ordering over types is extended to feature structures by the *subsumption* relation. A feature structure's information is represented by the values it assigns to paths. Subsumption is an inclusion ordering over this information. A feature structure F *subsumes* another F', written $F \sqsubseteq F'$, if and only if: for every path π defined in F, π is defined in F' and the type at π in F subsumes the type at π in F'; and for every pair of paths π and π' defined in F, if π and π' identify a single substructure in F then π and π' identify a single substructure in F'. This is expressed formally via a class of maps between feature structure node sets, such that every node in q in F is mapped into a node q' in F', the type at q subsumes the type at q', and q' occurs at a superset of the paths at which q occurs. If $F \sqsubseteq F'$ and $F' \sqsubseteq F$ then F and F' are information equivalent *alphabetic variants*, written $F \sim F'$.

Returning to the example in Fig. 1, the first feature structure subsumes the second because every functional role in the first is present in the second, every object fulfilling multiple functional roles in the first fulfills a superset of these roles in the second, and for every functional role identified in the first the object fulfilling this role in the second is of a matching or more specific type. Each piece of information represented in a feature structure relates to the existence of a feature path or the type label assigned to a substructure at a feature path.

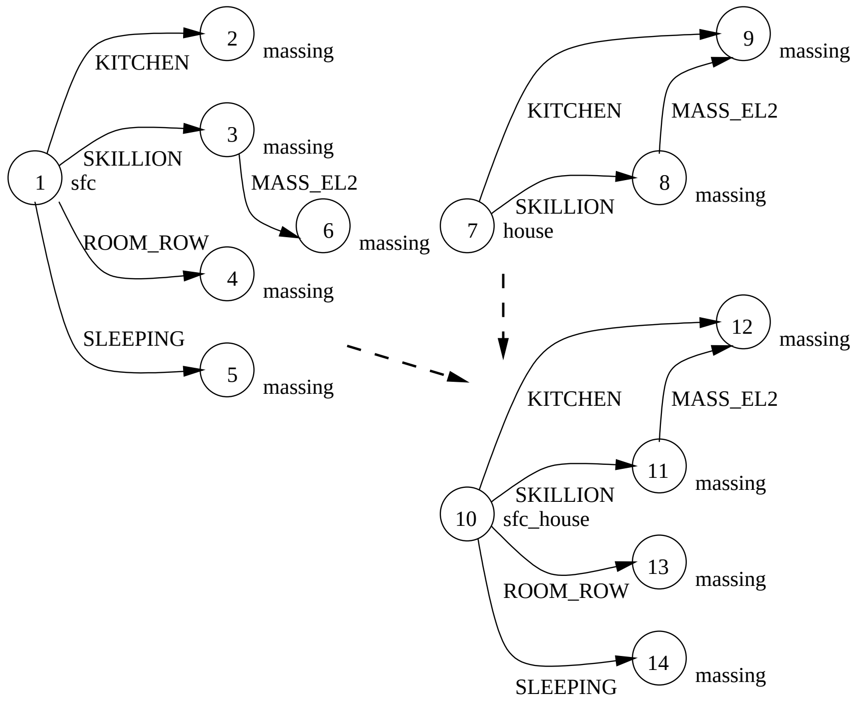

Figure 2: Feature structure unification: the bottom feature structure is the unification of the left and right feature structures.

Subsumption makes it clear that the information associated with a feature structure derives from the collection of feature paths and the type labels associated with substructures. However, subsumption can also be related to abstraction. On the structural side abstraction relates to the elision of functional roles and the uncertainty in their identification, while on the type labelling side it relates to the classification of the represented object. The recursive type constraint system provides a bridge relating these two sources of information.

2.3 Unification

Given the feature structure subsumption order, we seek a reasoning operation to compute the conjunction of two feature structures. Intuitively, this operation seeks the most general feature structure that is more specific than either operand. Unification is such a procedure, that either fails if the two feature structures represent inconsistent information, or constructs the most general specialization otherwise.

The *unification* of feature structures F and F', written $F \sqcup F'$, is a conjunction of the information considered by subsumption. Unification incorporates the collections of feature paths in F and F' as well as their type labelling. It must group feature paths so that sharing is replicated and reassign type labelling. For every pair of paths π and π' in F, if π and π' identify a shared substructure in F, then they also identify a shared structure in $F \sqcup F'$ so that it is subsumed by F. Likewise for F'. This is formalised by a partitioning of the nodes in F and F' into equivalence classes. $F \sqcup F'$ exists only if there exists a common specialisation of the types in each equivalence class.

In terms of functional decomposition, the unification of two design representations simultaneously decides whether some object may exist which is consistent with the two functional decompositions and, if so, provides an information minimal representation true only of such an object. Unification is the single constructive mechanism used to reconcile information from a type constraint system with a query description during design space exploration.

The key intuition to understanding unification is the collection of equivalence classes. During the unification of two feature structures, nodes are collected into equivalence classes according to feature paths. This relates directly to the information derived from feature paths in subsumption. In the simplest case a pair of nodes, one from each operand, are placed into an equivalence class if the same feature path leads to each node. Where there is already structure sharing there are already equivalence classes of paths, and therefore unification leads to more complicated aggregations of equivalence classes.

Figure 2 depicts three simple feature structures which are the operands and result of a unification operation. The example demonstrates a type join at the root node, the inclusion of new feature path value from both operands and the preservation of structure sharing.

2.4 Descriptions

Descriptions serve to call out feature structures through the *satisfaction* relation. Satisfaction is monotonic: if a feature structure satisfies a description, then so too does every feature structure which it subsumes(Carpenter, 1992, p 55). Therefore, a description may be interpreted as a reference to its most general satisfiers. In fact, every feature structure is the most general satisfier of a disjunction free description(Carpenter, 1992, p 56). The relationship between descriptions and feature structures is stated formally in terms of the functions MGSat and MGSats. Given the set of all feature structures $\mathcal{F}$, for the disjunction free descriptions $\mathsf{NonDisjDesc}$ there is a surjection $\mathrm{MGSat} : \mathsf{NonDisjDesc} \to \mathcal{F}$, and for the set of descriptions Desc, sets of pairwise incomparable feature structures are identified by the function $\mathrm{MGSats} : \mathsf{Desc} \to 2^{\mathcal{F}}$.

In a design space explorer, descriptions act as compact textual representations of functional decompositions. Since description satisfaction is monotonic wrt feature structure specialisation, descriptions are statements providing a lower bound of specificity on the represented object. By the inclusion of disjunction into descriptions a conjunction of these statements is satisfied by an arbitrarily large collection of functional decompositions. In this paper we are proposing that a form of π-resolution provides a formal framework for the implementation of a design space explorer. The key feature is the organisation of knowledge about functional decompositions into a system of recursive type constraints.

2.5 Recursive Type Constraints

Without additional type information, the type t of a feature structure F asserts only that the represented object is an instance of t or some subtype. By stating restrictions on the feature structures of each type, we assert additional properties on the feature structures and therefore about represented objects. The *constraint system* expresses restrictions on a substructure, according to its type, which determine legal labellings over a finite but arbitrary collection of paths. *Constraint resolution* determines whether a feature structure satisfies a constraint system.

Definition 3 (Constraint System(Carpenter, 1992)) *A* constraint system *is a total function* $\mathrm{Cons} : \mathsf{Type} \mapsto \mathsf{Desc}$.

A feature structure is *resolved* if it satisfies the constraint system at every substructure. A substructure satisfies the constraint system if it satisfies the type constraint for its type and all supertypes.

Definition 4 (Resolved Feature Structure(Carpenter, 1992)) *A feature structure* $F = \langle Q, \bar{q}, \theta, \delta \rangle$ *is* resolved *if and only if*

$$\forall \pi \in \mathsf{Path}, \forall t \in \mathsf{Type} \quad : \quad t \leq_{\mathsf{Type}} \theta(\delta(\pi, \bar{q})) \rightarrow F@\pi \textit{ satisfies } \mathrm{Cons}(t) \qquad (5)$$

Satisfaction monotonicity ensures a direct relationship between constraint resolution and unification. A type constraint description is satisfied if a most general satisfier subsumes the constrained substructure. Therefore, constraint resolution is the search for the most general feature structure subsumed by both a constraint satisfier and the current substructure. This is decided by unification. In this paper we make no distinction between a description and its set of most general satisfiers, i.e., we equate constraints with feature structures, and consider satisfaction in terms of subsumption by a most general satisfier, and resolution in terms of unification with a most general satisfier.

2.6 π-Resolution

Given a query description D, π*-resolution* is the search across a sequence of feature structures $P_0 \sqsubseteq P_1 \sqsubseteq P_2 \sqsubseteq \ldots \sqsubseteq P_k$. The initial feature structure in each sequence is a most general satisfier of the query description. The sequence represents the inclusion of type information in the form of constraints — each element extends its predecessor by unification with a type constraint. Since most general satisfiers may occur as collections and unification may fail, the search for resolved feature structures involves a collection of sequences.

Definition 5 (π-Resolution) *Given a feature structure $F = \langle Q, \bar{q}, \theta, \delta\rangle$ and a path π such that $\delta(\pi, \bar{q})$ is defined, let the sequence $t_1, t_2, \ldots t_k$ be an enumeration of the types $\{t \mid t \leq_{\mathsf{Type}} \theta(\delta(\pi, \bar{q}))\}$. We take $F \overset{\pi}{\Rightarrow} F'$ if and only if there exists a sequence $F_0 \sqsubseteq F_1 \sqsubseteq \ldots \sqsubseteq F_k$ such that $F \sim F_0$ and $F' \sim F_k$ and*

$$F_{i+1} \sim F_i \sqcup (\pi \,|\, C_{i+1}) \;\wedge\; C_{i+1} \in \mathrm{MGSats}(\mathrm{Cons}(t_{i+1})) \qquad (6)$$

The operation in Definition 5 is one step in a π-resolution search. During the search process the current partial satisfier evolves via numerous such resolution steps with the addition of information from the constraint system. Each step acts on a substructure of the current partial satisfier. The notation $\pi \mid F$ denotes the most general feature structure whose value at the path π is F. It is guaranteed to add no information to the partial satisfier P except at the substructure $P@\pi$, so the process can be thought of as acting directly on substructures. However, missing from the definition is a procedure to select π. The π-resolution process recursively enumerates the most general satisfiers of the query description. The process is complete since it can be proven that a feature structure is a most general solution only if it is the result of some sequence of π-resolution steps(Carpenter, 1992).

3. INCREMENTAL Π-RESOLUTION

In the proposed mixed initiative environment the user is to provide analytical talents in the comprehension of the design's balance of qualities, within a discourse based on the functional decomposition of the design problem. The machine is to provide feedback on the computable attributes of the current design state, the most fundamental being the current design artifact's satisfaction of the functional decomposition. In this context, design space explorers must transform both the functional decomposition and the corresponding model.

Since model transformation is external to π-resolution, we are interested in identifying possible synchronization points. It is important that these synchronization points define legitimate intermediate states, because they will be parameters to model transformation operations. In particular, their may be further choice points associated with the model transformation. Choice points are another issue in the algorithm. By definition π-resolution generates resolved feature structures, and therefore the solution set is unaffected by the organization of choice points. However, choice points determine the trajectories in feature structure space that resolution may trace. This determines the exploration freedom available to the design space explorers.

As it stands, π-resolution is incremental in the sense that it appears to solve type constraints one substructure at a time. However, it is difficult to formalize this property. A complete search strategy is ensured by a breadth first strategy that selects the unresolved path of minimal length(Carpenter, 1992, p 244). However, a substructure in a cycle can still undergo multiple resolution steps.

Definition 5 suggests a finer level of incremental construction: independently resolving each type constraint introduced at a supertype, rather than resolving the conjunction of constraints on supertypes. By recording the types against which a substructure has been resolved, it is possible to ensure that a substructure is resolved against each type at most once. This is an important synchronization property. It also suggests a useful intermediate state invariant. Note that, irredundant enumeration of types in Definition 5 controls the order in which type constraints are resolved at a substructure. While unification is commutative, this may not be true of the associated model transformations. Therefore, an important class of orderings are the linear extensions of the inheritance hierarchy. In terms of the model transformation, this order is analogous to the order in which supertype constructors are called in C++(Ellis and Stroustrup, 1990) i.e., inherited types are processed before derived types.

Another problem for synchronization is substructure unification. During π-resolution existing substructures may be unified. In terms of functional decomposition, this corresponds to a decision to share an object between functional roles. Clearly, this operation requires model transformations and therefore synchronization, especially since the substructures may be at different levels of resolution.

We propose *incremental π-resolution* as an algorithm with minimal incremental steps and coincident synchronization and choice points. In this paper we characterize the intermediate states in the algorithm. That is, each substructure records a *down-set* of types (a set closed under the addition of supertypes) against which it has been resolved, and may be resolved against any type which extends this down-set and which is also a supertype of the current label on the substructure. The first restriction ensures that each type is resolved once at a substructure, and only after all supertypes have been resolved. The second restriction on type resolution ensures that incremental π-resolution does not perform resolutions outside those in Definition 5. When a substructure is resolved against a type, the type constraint is unified at the substructure.

In incremental π-resolution, synchronization points coincide with choice-points where the user selects a substructure and a type. The down-set of types to which the substructure has been resolved is extended by the addition of the type and a satisfier of the type's constraint is unified at the substructure. This corresponds to one of the intermediate steps in Definition 5. The user experience of this control point is mediated by the model in the design space explorer. For example, having selected a building entity to work on the incremental π-resolution state is consulted to determine the sound functional decompositions available wrt the current state of the building entity. When constraints conflict, the disagreement between the attempted type constraint and the partial satisfier are available to the design space explorer in order to inform the user.

Minimal changes simplify model transformations. The model must be transformed to account for a single substructure which has been incremented by a type, all of whose supertypes have already been handled. New structure entering the partial satisfier from the constraint has minimal impact on the model, since it has not yet been resolved to even the most general type and therefore would not be expected to be represented in the model. Another synchronisation task occurs when existing substructures are unified in the updated substructure. For example, SKILLION MASS_EL2 and KITCHEN from the first feature structure in Fig. 2. The model transformation corresponding to the new structure sharing is informed by the equivalence classes generated in the unification and the previous resolution state of the unified substructures. For example, a certain type resolution may signal a resource allocation in the model so that if two substructures resolved to this resource sensitive type are unified the the design space explorer must relinquish one instance of the resource in the resulting entity.

The trajectories possible in incremental π-resolution capture a notion of goal directedness that is both formal and open. In addition, incremental π-resolution contains a component of shortest unresolved path strategy, since a path does not occur until all substructures on the path have been resolved to include the feature they carry in the path. While the algorithm is sophisticated by the need to track multiple open choice points, this is achieved by the addition of an extra type function that maps nodes to type down-sets identifiers. This information is also useful to the design space explorer in generating model transformations for unified substructures. In this sense incremental π-resolution departs from logic programming in removing the left to right ordering on rule application and throwing the construction of the search space into a completely non-deterministic process.

3.1 Incremental π-Resolution State

An incremental π-resolution state is an annotated partial satisfier. It comprises: the query description, a partial satisfier of the query, and a representation of resolved types. Its purpose is to provide for the computation of available refinements at an incremental π-resolution choice-point.

Definition 6 (Incremental π-Resolution State) *Given a query description D, a feature structure $P = \langle Q_P, \bar{q}_P, \theta_P, \delta_P \rangle$, and a function* $\mathrm{TState} : Q_P \mapsto 2^{\mathsf{Type}}$*, the tuple* $\Pi = \langle D, P, \mathrm{TState} \rangle$ *is an* incremental π-resolution state *if and only if*

$$\begin{array}{rcll} & & P \textit{ satisfies } D & (7) \\ \forall q \in Q_P & : & \mathrm{TState}(q) \subseteq \{t \mid t \leq_{\mathsf{Type}} \theta_P(q)\} & (8) \\ \forall q \in Q_P, \forall t \in \mathrm{TState}(q) & : & P@q \textit{ satisfies } \mathrm{Cons}(t) & (9) \end{array}$$

Because paths do not uniquely define substructures, TState is defined over nodes rather than over paths. This makes (9) awkward so we write $P@q$ for the substructure rooted at q. Otherwise, Definition 6 is straightforward, simply asserting that TState records the state of a partial satisfier in terms of the types resolved at each substructure.

3.2 Incremental π-Resolution

Given a query description D, the algorithm is initialized by having the user select a most general satisfier P_0 from the query description and setting TState to the empty set for all nodes in P_0. Each step in the algorithm requires the user to select a substructure for resolution, a type extending the substructure by resolution, and a most general satisfier of the extending type's constraint.

Definition 7 (Incremental π-Resolution) *Given an incremental π-resolution state* $\Pi = \langle D, P, \mathrm{TState} \rangle$*, a path π_i such that $q_i = \delta_P(\pi_i, \bar{q}_P)$ defined a type t_i such that*

$$t_i \in \{t \mid t \leq_{\mathsf{Type}} \theta_P(q_i) \wedge t \notin \mathrm{TState}(q_i) \wedge t' \leq_{\mathsf{Type}} t \longrightarrow t' \in \mathrm{TState}(q_i)\}$$

and a most general satisfier $F_i \in \mathrm{MGSats}(\mathrm{Cons}(t_i))$.

We take $\Pi \overset{\pi_i, t_i, F_i}{\Longrightarrow} \Pi'$ *if and only if* $P \sqcup (\pi_i \mid F_i)$ *exists, and set*

$$\begin{array}{rcll} \Pi' & = & \langle D, P', \mathrm{TState}' \rangle & (10) \\ P' & \sim & P \sqcup (\pi_i \mid F_i) & (11) \\ \mathrm{TState}'(q') & = & \bigcup\{\mathrm{TState}(\delta_P(\pi, \bar{q}_P)) \mid \delta_{P'}(\pi, \bar{q}_{P'}) = q'\} & (12) \end{array}$$

The set of types from which an extending type t_i may be drawn is determined to ensure that the new TState' value is a down-set. Intuitively, a type cannot be selected unless it makes a contribution to the target type on the partial satisfier and all its supertypes have already been resolved, i.e, it subsumes the type on the node and all its supertypes are in $\mathrm{TState}(q_i)$. Equation 12 mirrors the unification operation computation of types for equivalence classes. It is awkward in this definition because the equivalence relation is not available. Intutively, it states that where substructures unify the new resolution state is the union of their resolution states. This effectively prevents type constraints from being repeatedly applied even where this occurs indirectly via unification. This is an important property supporting the model synchronisation process.

Initially there is one possible move for each substructure in the most general satisfier selected for the query description. Since TState is set to $\emptyset$ for all initial substructures, the extending type is restricted to the most general type, which is also guaranteed to subsume any type. It is unlikely that the constraint on the most general type is non-empty. Therefore, each such initial move finishes with the most general type inserted into TState for the selected substructure.

For subsequent moves, the algorithm acts on a user selection at π_i extending to type t_i and with satisfier F_i, by inserting t_i into $\mathrm{TState}(\delta_P(\pi, \bar{q}_P))$ and unifying F_i at $P@\pi$. Each such move introduces new structure or refines existing substructures. The active choice points in an incremental π-resolution state rapidly grow downwards from the root node. This is the intention of the characterisation — minimal incremental changes supporting synchronisation with the model, and a completely open process which may be tailored by the design space explorer with heuristics to suit the complete range of problem decomposition styles.

4. CONCLUSION

Functional decomposition is suggested as a framework for exploration without claiming that functional decompositions equate to designs. Therefore, separate actions are required to ensure a design artifact consistent with the functional decomposition. The proposed algorithm is a modification to π-resolution which makes additional choice points available to the resolution strategy. The advantage of these choice points lies in their relationship to the synchronization needs of an external model transforming process.

In the case of the design space explorer, this external mechanism may involve a hybrid system perhaps including shape grammar fragments. Functional decomposition by incremental π-resolution provides a context for these operations that may be considered a sophisticated labelling strategy. However, additional benefits are realised in exploiting the partial ordering as a system for indexing both whole designs and design fragments. Retrieval is then possible on the basis of design requirements.

One benefit of this approach is that a single object represents the design state's functional decomposition rather than a computational state spread across a database. This can be coupled with a characterisation of feature structures that eliminates alphabetic variance by the generation of persistent identifiers for path equivalences. Since the computational state is characterised by the type hierarchy and path equivalences, persistent storage then extends to computation states. Each such computation state is a table mapping a collection of path equivalences to a target type and a resolution state. Under these conditions design spaces may be stored beyond the execution of the program and their indexing extended to incomplete designs.

This paper introduces the notion of incremental construction by providing a characterization of the intermediate states. The algorithm is not prescriptive in form. Instead it provides a formal mechanism upon which a variety of heuristics can be grafted. This paper does not consider this area, except to note that many very interesting strategies are directly realised as partitionings and traversals of the type hierarchy especially where additional appropriateness constraints are introduced on the combinations of features and types.

The described mechanism has been implemented, as part of a project to engineer a design space explorer based on properties of typed feature structures. A major engineering task has been the management of identifiers for the type down-sets used to represent resolution state. However, this itself has enjoyed benefits arising from the formalism that were unexpected. The experience suggests that even the most formal symbol representations have surprising and useful properties that arise as non trivial consequences of the theory.

5. ACKNOWLEDGEMENTS

The authors would like to acknowledge the support given by the Australian Research Council Large Grants Scheme, The Department of Industry, Science and Tourism Major Grant Scheme, the Australian Postgraduate Award Scheme, The Key Centre for the Social Applications of Geographic Information Systems and The University of Adelaide School of Architecture, Landscape Architecture and Urban Design.

References

Aït-Kaci, H., Boyer, R., Lincoln, P., and Nasr, R. (1989). Efficient implementation of latice operations. *ACM Transactions on Programming Languages and Systems*, 11(1):115–146.

Aït-Kaci, H. and Podelski, A. (1993). Towards a meaning of LIFE. Technical Report 11, Paris Research Laboratory of Digital Equipment Centre Technique Europe, Rueil-Malmaison, France.

Aït-Kaci, H., Podelski, A., and Goldstein, S. C. (1993). Order-sorted feature theory unification.

Technical Report 32, Paris Research Laboratory of Digital Equipment Centre Technique Europe, Rueil-Malmaison, France.

Carlson, C. (1993). *An Algebraic Approach to the Description of Design Spaces*. PhD thesis, Department of Architecture,Carnegie-Mellon University.

Carlson, C. and Woodbury, R. (1994). Hands-on exploration of recursive patterns. *Languages of Design*, 2:121–142.

Carpenter, R. (1992). *The Logic of Typed Feature Structures : With Applications to Unification Grammars, Logic Programs and Constraint Resolution*. Number 32 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge.

Coyne, R. (1991). *ABLOOS: An Evolving Hierarchical Design Framework*. PhD thesis, Department of Architecture, Carnegie Mellon University, Pittsburgh, PA. also available as a TechReport, (EDRC-02-15-91), Engineering Design Research Center, CMU.

Ellis, G. (1995). *Managing Complex Objects*. PhD thesis, Department of Computer Science, University of Queensland.

Ellis, M. A. and Stroustrup, B. (1990). *The annotated C++ reference manual*. Addison-Wesley, Reading, MA, USA.

Flemming, U. and Woodbury, R. (1995). Software environment to support early phases in building design (seed): Overview. *Journal of Architectural Engineering*, 1:147–152.

Gorti, S. R., Kim, G. J., and Siriam, R. D. (1998). An object-oriented representation for product and design processes. *Computer-Aided Design*, 30(7):489–501.

Harada, M., Witkin, A., and Baraff., D. (1995). Interactive physically-based manipulation of discrete/ continuous models. In *SIGGRAPH '95 Conference Proceedings*, volume 29, pages 199–208. ACM Siggraph, ACM.

Heisserman, J. (1994). Generative geometric design. *IEEE Computer Graphics and Applications*, 14(2):37–45.

Stiny, G. (1980). Introduction to shape and shape grammars. *Environment and Planning B: Planning and Design*, 7(3):343–352.

Stiny, G. and March, L. (1981). Design machines. *Environment and Planning B: Planning and Design*, 8(3):241–244.

Wille, R. (1992). Concept lattices and conceptual knowledge systems. *Computers Math. Applic.*, 23(6–9):493–515.

Woodbury, R. F., Burrow, A. L., Datta, S., and Chang, T.-W. (1999). Typed feature structures in design space exploration. *AIEDAM Special Issue on Generative Systems in Design*. manuscript available at http://www.arch.adelaide.edu.au/ rw/publications/progress.html.

Woodbury, R. F. and Chang, T.-W. (1995). Massing and enclosure design with SEED-Config. *ASCE Journal of Architectural Engineering*, 1(4):170–178.

Woodbury, R. F., Radford, A. D., Taplin, P. N., and Coppins, S. A. (1992). Tartan worlds: A generative symbol grammar system. In Noble, D. and Kensek, K., editors, *ACADIA 92*, pages 211–220, Charleston, SC.