Bolt Beranek and Newman Inc.

LEVEL II

12

Report No. 4371

AD A087246

Application of Symbolic Processing to Command and Control: An Advanced Information Presentation System

Annual Technical Report

Frank Zdybel, Norton Greenfeld, and Martin Yonke

April 1980

Prepared for:
Defense Advanced Research Projects Agency
Information Processing Techniques Office

80 6 2 051

(14) BBN-

Report No. 4371

(12)

(6) APPLICATION OF SYMBOLIC PROCESSING TO COMMAND AND CONTROL: AN ADVANCED INFORMATION PRESENTATION SYSTEM

(10) Frank Zdybel, Norton Greenfeld and Martin Yonke

(12) 85

(11) Apr 80

(9) Annual Technical Report,

(15) N00039-79-C-0516, VV AHPA 01.1.1-3740

Prepared by:

Bolt Beranek and Newman Inc.
50 Moulton Street
Cambridge, Massachusetts 02138

Prepared for:

Defense Advanced Research Projects Agency
Information Processing Techniques Office
1400 Wilson Boulevard
Arlington, Virginia 22209

060100

This research has been supported by the Defense Advanced Research Projects Agency under Contract N00039-79-C-0316, ARPA Order No. 3740, and monitored by the Naval Electronics System Command (Program Code 9D30).

The views and conclusions contained in this paper are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

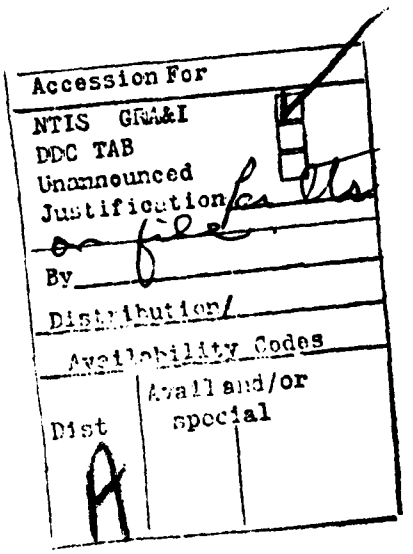

TABLE OF CONTENTS

1. OVERVIEW

Last year the Advanced Information Presentation System (AIPS) had reached the point of a first-pass implementation, which is described in [Zdybel, Yonke, and Greenfeld, 1980]. Some analysis and experimentation with that system and its internal representation structure revealed areas where the knowledge structure was inadequate -- either not epistemologically sound or not computationally reasonable. During this year we have conceptualized a new structure for the system and have begun the task of implementing our next generation AIPS system.

In this reimplementation, an effort has been made to modularize the system's knowledge structure. This has led to a totally new declarative architecture for the system, as described in [Yonke and Greenfeld, 1980]. This architecture promises to provide a system whose capabilities are more easily extended into the realm of "intelligent" behavior.

This project has been active in helping to evolve the KL-ONE knowledge representation language [Brachman et al., 1978] (in conjunction with Bill Woods' natural language effort [Woods, 1978]). In reimplementing AIPS, we have found some of our tools falling short of our needs. The result has been two particularly

interesting extensions to KL-ONE. The first extension helps with the task of building a large initial set of concepts -- from the KL-ONE programmer's point of view. Our own experiences have proven the need for such a tool. The second extension provides a control mechanism for automatically directing the derivation of fully realized KL-ONE individual descriptions.

We have also rebuilt our basic bitmap graphics package (BMG) this year, again in response to experience with the first round of implementation. Several new features were added, such as display macros and processes in the PDP-11, dual I/O streams for text and graphics, off-screen bitmap memory, event reporting for both up and down transitions on the mouse buttons, and dynamically loadable fonts. This package is documented in [Greenfeld, Zdybel, Ciccarelli, and Yonke, 1980].

We hope to be able to demonstrate the reimplemented AIPS system later this summer. While we are still running on the TOPS20 system, this situation is now almost intolerable due to memory size and processor throughput requirements. We hope to begin the conversion of the new AIPS to run on the Jericho Interlisp personal machine this fall, and to be fully implemented in that environment within the next year.

The knowledge structure for the new AIPS system is described

in the next section of this report, which also provides some insight into the behavioral properties of the system we are currently building. The third section of the report describes some of the tools which have been developed this year for implementing in KL-ONE, especially the CKLONE network building operators are described. The fourth section consists of the actual programs which initialize the KL-ONE knowledge base structure for AIPS (as of the printing of this document). Use of the CKLONE operators and profuse commenting make it one of the best available descriptions of the current system, its constructs, and its capabilities.

2. A NEW SCHEME FOR AIPS

This section overviews the declarative knowledge structure for the new AIPS system and discusses certain design decisions implicit in the revised network. The purpose is to lay the conceptual foundation for the more detailed descriptions which will follow in the fourth section of the report.

For the sake of precision, the remainder of this report uses certain typographical conventions to differentiate among references to generic KL-ONE Concepts, RoleSets, ITags, etc. and the entities that these descriptions denote. Bold capital letters are used for the names of KL-ONE concepts. Initial capitalization and initial capitalization of sub-words in a name indicates a descendant of the named concept. Thus, **DISPLAY** refers to the KL-ONE Generic Concept of that name, Display refers to KL-ONE Concepts that are generic descendants or individuals of **DISPLAY**, and display refers to something seen on the screen of a graphics terminal. Initial capitalization and bold, mixed case characters are used to highlight the names of LISP functions, and underlining are used for the names of KL-ONE RoleSets, ITags and Tags.

2.1 The Nature of Presentations

Let's begin by reflecting on AIPS' previous methods of operation. In the course of attempting to build and revise presentations for the "Middle Earth" demonstration system (described in [Zdybel, Yonke, and Greenfeld, 1980]) a number of shortcomings became apparent. The greatest of these were:

- Characterization of the information being presented (the DomainObject Role) was limited to designation of descriptions of objects that included "real world" locations.
- Characterization of the formatting process for composing a presentation (the Injection Role) was limited to simple geometric transformations among coordinate systems.
- Characterization of what presentations looked like (the Style Role) did not afford sufficient means for decomposing complex composite display entities, or for dealing with the possibility that the display behavior of the presentation might depend on further information than that specified in the DomainObject of the Presentation.

In short, we have had to depart from the view that a presentation should be described as the mapping of some real world location into some display location combined with the mapping of some real world object into some display world object. Instead, we have adopted the more general view that a Presentation describes how arbitrary information is treated by, or becomes involved in, the process of realizing arbitrary output

phenomena.

Accordingly, in the new scheme of things, the concept **PRESENTATION** has just two Roles: Application and Realization. The Application designates the specific domain world information being displayed in an individual Presentation. The Realization characterizes how the information is presented.

Note that **PRESENTATION** is not restricted to the description of graphic displays; it can be applied to other output behavior such as synthesized messages. Accordingly, a new concept: **DISPLAY** (a generic descendant of **PRESENTATION**) is used to connote graphic presentations. To the Application and Realization Roles inherited from **PRESENTATION**, it adds a third Role: Ground, which indicates the ViewSurface (a description of a continuum for drawing on) upon which the display is to be realized.

The Realization role is differentiated and modified in nearly every generic descendant of **DISPLAY**. For example, **MAP** is a generic descendant of **DISPLAY**, and its visible components (e.g.: Border, Label, CoordinateGrid, Legend, Item etc.) are all represented as differentiating sub-roles of **DISPLAY**'s Realization Role.

At **DISPLAY**, the Realization Role is value restricted to

DISPLAYITEM -- that is, to either another Display (this is the decomposition mechanism) or a DisplayForm. A DisplayForm is purely an entity of the Viewing Organization Model, a syntactic description of display behavior. The functional difference between representing some display phenomenon as a Display and representing it as a DisplayForm is mainly the difference between treating its use of information explicitly as network structure or implicitly as procedural attachments.

As before, there are two sub-species of DisplayForms: **DISPLAYATOM** (previously called **PRIMITIVEDISPLAYFORM**) and **DISPLAYCOMPOSITE** (previously called **COMPOSITEDISPLAYFORM**). For the most part, the declarative realization model (the descriptions of the various types of primitive and composite DisplayForms) continues without major change from the first AIPS implementation.

The Application Role of **PRESENTATION** partially describes the information being expressed in a given individual of some generic descendant of that concept. It does not describe all of the information involved in realizing the Presentation. Rather, it is a binding mechanism that characterizes a Presentation's application to specific information, as opposed to its inherent use of information specified in its generic "definition". Such

intensional characterization of the mandatory and optional information to be treated by a generic Presentation is achieved instead through metadescription of the Presentation.

Of course, a generic Presentation may have roles that are sub-roles of neither Realization nor Application. For example, **MAP** may have a role Injection which characterizes the scaling process used in building up the map.

2.2 On the Characterization of Information

At **PRESENTATION**, the Application Role is value restricted to zero or more individuals of **ITEMPLATE**. A Template is a means of indicating a set of inherited slots (Roles) in an SI-Net. An ITemplate (for individual-Template) is simply a Template whose application is limited to Individual Concepts. **ITEMPLATE** is the only generic descendant of **TEMPLATE**. When we say "a Template", we are referring to an Individual Concept.

TEMPLATE has two Roles: RoleGroup and ConceptGroup. Satisfiers of RoleGroup meta-indicate Roles in the AIPS Domain Model. The ConceptGroup Role restricts this specification of a set of inheritable Roles to a set of designated concepts and their descendants. Thus, a Template specifies a set of inherited slots in terms of a set of inheritable Roles and a set of

sub-lattices of the taxonomic hierarchy. Notice that this is still an "object oriented" chunk of information, with ConceptGroup being used to indicate the objects of concern. By collecting several Templates together as the values of a Role (e.g.: several Templates filling Application), we arrive finally at object independent specifications of chunks of information.

We get the necessary descriptive "grip" on Roles and Concepts through use of KL-ONE's meta-description feature. The Roles to be included in the Template are meta-hooked to Nexuses that embody Individual Concepts of **ROLE**. These Individual Concepts become the values of RoleGroup in individuals of **TEMPLATE**. Meta-hooks are used in a similar way to indicate the Concepts filling ConceptGroup.

Application is not re-described by any generic descendants of **PRESENTATION** because its value restriction cannot get any more specific than **ITEMPLATE**: otherwise, the filler of the role would be constrained to a particular Template (that is, to a particular chunk of the domain model extension). A possible approach might be to differentiate Application at various descendants of **PRESENTATION** (and then restrict only some of the sub-roles to particular Templates), but at this writing we cannot think of any sufficient reason to do this, except perhaps to provide IHook

sites to implement side effects of binding particular pieces of information.

In many cases, especially with individuals of less complex Presentations, it will be desired only to designate some single Role (either a RoleSet or an IRole) as a meta-description of the Presentation's information consuming properties, or as a filler of some individual Presentation's Application Role. To accomodate this need economically, an individual of **ROLE** that meta-hooks to some Role in the Domain Model can be employed as a singleton Template. If the instance of **ROLE** designates a Generic RoleSet, then it can be assumed to embrace all concepts that inherit that Role. If it designates an IRole, of course, it could indicate only a single Individual Concept.

Finally, it may be necessary at some point to describe the information requirements of a Presentation in terms of some arbitrary predicate that can be applied to determine whether the Presentation can be used in any given situation. This escape hatch allows any presentation selection apparatus to be able to respond to arbitrary specifications for SI-Net topology without unduly complicating the information description mechanism (Templates) for the present time. A more general capability for specifying multi-concept SI-Net "patterns" may be developed

later.

2.3 Use of Meta-Description Of and In Presentations

We have alluded above to the notion that a Presentation may carry meta-desriptions of its requirements and capacities for dealing with information. These functions would be accomplished, respectively, by individuals of **INFONEED** and individuals of **INFOUSES**, which would link the generic Presentation being meta-described to some set of Templates (that is, to a characterization of an arbitrary chunk of information). The interpretation of an InfoUse would be that the Presentation being meta-described could meaningfully portray (to the human user) any object that fit one of its constituent Templates. Thus, the Templates comprising an InfoUse have the flavor of a pattern, or intensional description, and would presumably reference generic concepts (i.e., would not be ITemplates).

Notice that it is easily possible that yet more information than specified in an InfoUse could be required in order to realize the Presentation. An InfoNeed (an individual of **INFONEED**) is used to describe this information cost of a presentation. Its interpretation is that, at a minimum, the realization of the meta-described Presentation will involve the

information specified by the constituent Templates. In contrast to the disjunctive interpretation of the Templates in an InfoUse, an InfoNeed would be interpreted by taking the conjunct of its constituent Templates. Again, in an InfoNeed, the constituent Templates presumably reference Generic Concepts almost exclusively.

These meta-descriptions of Presentations are required only where it is possible to choose from among alternative Presentations (that is, mainly between major, top-level presentation forms). Since we therefor do not wish to require that all Presentations be thus meta-described, the InfoNeed description of a Presentation summarizes the requirements of any constituent Presentations that are uniquely determined by value restrictions on its Realization Roles. An alternative to this would be to derive the cumulative information characteristics for any higher-level Presentation by examining meta-descriptions of it and its constituents, but this would require meta-describing all generic Presentations, with a large attendant cost in terms of network size.

2.4 An Example

The following example illustrates how the above scheme can be used to describe a specific presentation form which I will dub **STDNTDSMAP**, for "Standard NTDS Map". This description is intended to characterize the familiar kind of polar NTDS formation map. An NTDS symbol on this type of map will be required to convey at least the general type of the entity being displayed (e.g.: "group", "CV", "surface ship"), its location, its identifier (a radio call sign or track designator), its composition (in the case of a "group" entity, the number of platforms), and its attack engagement status. In addition, it will be possible to include certain other information such as course and speed, track (previous locations), and arbitrary text.

In this case, we have a concept **MAP** which is an immediate descendant of **DISPLAY**. Among the Roles of **MAP** differentiating the Realization Role of **PRESENTATION** are:

- Border, (mandatory, restricted to an instance of **CLOSEDCURVE**, a DisplayForm);
- Legend, restricted to an instance of **MAPLEGEND**, a descendant of **ARRANGEMENT** (a description of an arbitrary arrangement of constituent DisplayItems) specialized for displaying examples of the MapItems used in the map, together with descriptive text. The derivation tag (advice attached to the RoleSet telling how to build the eventual role filler) found here sets up the Application Role of the new MapLegend to indicate the

Item, ScaleUnit, and ScaleDistance Roles of the expanding STDNTDSMap;

- ReferenceGrid, restricted to **MAPGRID**, a Presentation that knows how to depict the bounds, scale and locale of a map; the derivation tag on this role sets up the Application Role of the new mapgrid to indicate the Border, ScaleDistance and ScaleUnit Roles of the expanding STDNTDSMap;

- Label (mandatory), restricted to an instance of **TEXT** (a DisplayForm);

- Item, restricted to an instance of **MAPITEM**, a Display that specializes in depicting the identity and location of domain world objects.

The next level of description down is **NTDSMAP**, a descendant of **MAP** with the following roles:

- Border, further restricting the Border Role of **MAP** to an instance of **RECTANGLE**;

- ReferenceGrid (now mandatory), further restricting the ReferenceGrid Role of **MAP** to an instance of **POLARGRID**, which describes how to derive a "nice" set of range circles and azimuth lines to suit the bounds, scale and locale of some specific map;

- Item, further restricting the Item Role of **MAP** to an instance of **NTDSITEM**, which describes how to depict certain classes of domain objects (e.g., ships, planes, submarines) along with certain information about them (by choosing or modifying an appropriate composite DisplayForm<);

Finally, descendant from **NTDSMAP** is **STDNTDSMAP**, whose single local Role:

- Item, is nearly a copy of the Item of **NTDSMAP**, except

that its derivation tag differs, specifying additional information (optional from **NTDSITEM**'s point of view) as mandatory in the Application of the Display that will eventually satisfy this Role (e.g., radio call sign, course, speed and track).

Notice that it would be possible to achieve the same effect as this last by having a variant of **NTDSITEM**, perhaps called **STDNTDSITEM**, whose obligatory Roles and Role derivation tags would indicate the desired additional information. However, this would have the drawback of needlessly multiplying the number of conceptual entities: the Role derivation procedures for **NTDSITEM** and **STDNTDSITEM** would be nearly identical. The chosen method also has the advantage that the augmented information consuming behavior is expressed as a characteristic of **STDNTDSMAP**, without indirection through the taxonomically trivial **STDNTDSITEM**.

2.5 Distribution of Procedural Knowledge

In order for information to be graphically displayed by AIPS, three distinct processes must occur in order: derivation, location, and drawing. During the derivation process, the KL-ONE description of the display is completely realized as a network of Individual Concepts. All Roles of Obligatory Modality have their fillers derived at this time. The derivation process is driven

and guided by commentaries on the Obligatory Roles called derivation tags.

Following derivation of the descriptional structure of the display, the locations of the visible elements of the display are established. This process is driven by a special class of procedures attached to the various generic Displays with the tag ToLocate.

Finally, after the locations of all display elements have been established in the drawing continuum, the display can actually be drawn onto a bitmap array. This process is driven by another special class of procedures attached to the various generic Displays and DisplayForms with the tag ToDraw. In the case of DisplayAtoms, these attached procedures directly invoke the LISP drawing primitives of the BMG graphics package. In the case of higher level entities, the ToDraw functions recursively invoke the ToDraw functions attached to their constituents.

Several advantages accrue from dividing the information display process into these three phases. It is a considerable simplification when establishing the locations of display elements, for example, to know that all of the display elements have been identified and placed in their proper structural relationships to each other. We are also better prepared to

handle situations where it is necessary to re-locate some or all of the elements of a Display without modifying the structure of the Display's description (and similarly, situations where we must re-draw portions of a Display without disturbing its layout).

Let's now go a bit further with our example and sketch out what happens when the user creates an individual of **STDNTDSMAP**. First of all, the Application of the new STDNTDSMap will be filled either directly by the user (i.e., he will edit onto the Application IRole ITemplates that describe which platforms and what extra information about each he wishes to see) or through some agent (e.g.: an information source based on LADDER [Moore, 1978] produces a set of ITemplates in response to a question such as "Show me the friendly subs within 400 NM of Naples.")

The next step in the process would be to begin deriving the obligatory Roles of the new STDNTDSMap. This derivation process is driven by the derivation tags carried on the individual RoleSets. In this case, intermediate obligatory Roles would include Ground, CoordSys, ScaleUnit, and ScaleDistance. The derivations for filling these are all fixed at the level of **MAP**. Generally speaking, we may have to provide more than one derivation method for a given Role, because one or more of the

fillers of these interdependent Roles may be specified by the user, either directly or indirectly. We will have to use different derivation methods for the remaining Roles, depending on what information has already been specified from the outside.

For purposes of this example, let us suppose that the user has indicated only the information to be displayed, and has left us with the task of deriving fillers for all of these intermediate roles. We begin by searching through the domain objects mentioned in the Application of the Display to determine the most suitable shared coordinate system for describing their world locations. If more than one world coordinate system is involved, the appropriate transforms among these systems are established at this time.

After establishing the proper world coordinate system for the objects, we turn to the task of determining the appropriate size and scale of the Map. The simplest method for doing this is to determine the smallest inter-object distance in the world coordinate system and select a map scale that results in some idealized minimum visual separation between the objects' depictions. The result is captured as a Transform established between the chosen world coordinate system and the internal coordinate system of the filler of the Ground Role (a

ViewSurface). We then derive the size and shape of this ViewSurface by determining the maximum spreads among world object locations. Knowing the maximum spread in world locations lets us pick the filler of the ScaleUnit Role (e.g., miles, tens of miles, or hundreds of miles). Since at this point we already have the appropriate Transform in hand, it is easy to derive the filler of ScaleDistance. A deeper treatment of this problem would involve negotiation among constraints on minimum visual separation and display size, with "fall back" solutions involving either consolidation of structural elements (e.g.: collapsing ship symbols into a task force symbol) or a window [Teitelman, 1977] with scrolling capability.

Once the intermediate Roles of the STDNTDSMap have been filled, the derivation process continues into the sub-roles of Realization. As with the intermediate Roles, the bulk of the Realization Roles are filled through the guidance of attached derivation tags that individuate either further Displays or DisplayForms. In the case of Item Role of the STDNTDSMap, the filler is produced by generating an appropriate set of items according to the Application of the newly individuated STDNTDSMapItem. As the individuals of **NTDSITEM** are created, the derivation process recurses to fill their obligatory Roles.

Following derivation of the display description, the process of locating the identified components of the display commences. This process might be initiated as part of a top-level information display function called by the user, or as a side effect of some other user activity (e.g.: opening a Window of a certain size and shape onto the ViewSurface of the Display). In either case, the process itself is driven and defined by inheritable ToLocate tags attached to the various generic descendants of **DISPLAY** and **DISPLAYFORM**. These tags refer to procedures that embody the knowledge of how to locate the constituent parts and pieces of the various generic Displays.

When the ToLocate procedure attached to **STDNTDSMAP** is completed, the ToDraw function inherited by **STDNTDSMAP** from **PRESENTATION** is called. Again, this process might be initiated either as part of a top-level information display function, or as a result of opening a Window onto the ViewSurface of the Display. For all Displays, the method for drawing is the same: take all the fillers of the Realization Role and invoke their ToDraw functions. **DISPLAYCOMPOSITE** has a similar ToDraw function attached to it, but some generic descendants of **DISPLAYCOMPOSITE** have more particular ToDraw functions of their own. Each DisplayAtom, of course, has its characteristic ToDraw function, which invokes LISP graphics primitives of the

BMG package.

In this example we have seen that the proposed network arrangement, at least in the case of a map, provides sufficient means to evenly distribute the procedural knowledge in the structure as one moves from **PRESENTATION** down the taxonomic hierarchy, with the most specialized Displays inheriting a large share of their attached procedural knowledge.

3. ADDITIONS TO KL-ONE

Over the past two years, most of our attention has been quite properly directed at KL-ONE as a knowledge representation language, with relatively little energy spent on developing KL-ONE as a programming medium. For example, certain necessary utilities were implemented early in KL-ONE's development (e.g., network printing and reading facilities), but little attempt has been made to integrate these nicely into the Interlisp programming environment. In this section we discuss some of the extensions we have made to KL-ONE to facilitate the task of programming knowledge-based applications.

3.1 CKLONE

The purpose of the CKLONE package is to provide a simple and user-extensible tool for initializing KL-ONE networks. The package provides a set of CLISP operators that can be used to write network building functions. The advantages of CKLONE over other network building methods include:

- the translation scheme is quite simple and can easily be changed in order to track new versions of KL-ONE or in order to include user-defined network building operators;
- CKLONE forms embedded in function definitions can be edited with the Interlisp editor, and any changes will

be noticed by the Interlisp file package;

- network initialization functions written in CKLONE provide an easily read account of a KL-ONE network that can be divided up among Interlisp files and interspersed with the definitions of any procedural attachments;
- CKLONE forms compile into calls to KL-ONE functions, so no special loading apparatus or library is required;
- Interlisp and CKLONE forms can be freely intermixed to handle special cases.

This package is intended primarily for use where the resulting translations will execute only infrequently. The code generated is far from efficient because, in order to guarantee that the resulting network does not depend on the order in which forms are executed, translations must generally first check to see whether some entity already exists before doing any building. Also, the efficiency of generated forms has been sacrificed to simplicity in the translation scheme.

3.1.1 CKLONE's CLISP Operators

The CKLONE file, when loaded, defines six new CLISP operators: concept, iconcept, roleset, proleset, irole and contexts. A form having one of these operators as CAR can be dwimified, and generally translates into a PROG that binds one of its variables to the Concept or Role that is to be supplied (that is, either found or built). The CDR of the form may contain both

arbitrary LISP forms and forms whose CARs are reserved atoms (subordinate operators), which in turn translate into KL-ONE function calls that operate on the entity bound in the PROG.

The concept operator is used to supply a KL-ONE Generic Concept (which then becomes "the current Concept" -- the value of $$CONCEPT). It has the syntax:

(concept <concept-spec> --)

where <concept-spec> is expected to be either an atom that is the name of the Concept to be supplied, or else a form that returns the Concept to be supplied (such a form can be an atomic CLISP or CKLONELIBRARY construct). Thus, the three CKLONE forms:

```
(concept (KLGetNamedConcept (QUOTE FOO)) --)
(concept FOO --)
(concept `FOO --)
```

are all well-formed, and have equivalent results (assuming FOO exists already) as far as the concept retrieved is concerned. (See the next section on CKLONELIBRARY for a description of the ` operator.) The form:

(concept (QUOTE FOO) --)

on the other hand, will result in an error when the translation is evaluated, because (QUOTE FOO) does not return a Concept. The error does not occur at dwimify-time because CKLONE cannot, in

general, expect to distinguish Concept-producing and non Concept-producing expressions.

The iconcept operator is used to supply a KL-ONE Individual Concept (which then becomes "the current Concept" -- the value of $$CONCEPT). There are three possible forms:

```
(iconcept <iconcept-spec> --)
(iconcept of <concept-spec> --)
(iconcept <iconcept-spec> of <concept-spec> --)
```

where <iconcept-spec> is either an atom that is the name of the Individual Concept or a form that evaluates to an Individual Concept, and <concept-spec> is either an atom that is the name of a Generic Concept or a form that evaluates to a Generic Concept. In the third case, if <iconcept-spec> is a form, <concept-spec> is ignored.

The roleset operator is used to produce a local Generic RoleSet (which then becomes "the current Role" -- the value of $$ROLE). There are three possible forms:

```
(roleset <roleset-spec> --)
(roleset of <concept-spec> --)
(roleset <roleset-spec> of <concept-spec> --)
```

where <roleset-spec> is either an atom that is the name of a local Generic RoleSet or else a form that evaluates to a Generic

RoleSet, and <concept-spec> is either an atom that is the name of a Generic Concept or else a form that evaluates to a Generic Concept. In the first case, the Concept having the RoleSet is defaulted to "the current Concept", so the embedded roleset form in:

(concept **FOOCONCEPT** (roleset FOOROLE1 --) --)

refers to a local RoleSet of **FOOCONCEPT**. The form:

(roleset NIL --)

can be used to supply un-named local Roles. The second and third forms of the roleset operator are used to add local Generic RoleSets to (or retrieve them from) arbitrary Concepts other than the current Concept. This is done simply by rebinding $$CONCEPT, so constructs of the form:

(roleset <role-producing-form> of <concept-spec> --)

can produce situations in which the current Role is not directly attached to, or even inherited by, the current concept.

The proleset operator is used to supply a Particular RoleSet (which becomes the value of $$ROLE). Like the roleset operator, it has three possible forms:

```
(proleset <proleset-spec> --)
(proleset of <iconcept-spec> --)
(proleset <proleset-spec> of <iconcept-spec> --)
```

This operator functions analogously to the roleset operator just discussed.

The irole operator is used to supply an IRole (which then becomes the value of $$ROLE). It has two possible forms:

```
(irole <irole-spec> --)
(irole <irole-spec> of <iconcept-spec> --)
```

Here, <irole-spec> either designates the name of the "lowest accessible" inherited or local RoleSet (the name of the IRole), or is a form that evaluates to some IRole (the user is left with the responsibility of assuring that there is a set of cables connecting to the RoleSet). The second case of the irole form allows an IRole to be added to or retrieved from some Concept other than the current one.

When <irole-spec> is an atom that names an IRole, the irole operator always results in an IRole being added to the current Individual Concept. If it is desired to lay hands on some already existing IRole (say, to edit the value), the following hack should be used:

```
(irole (KLFindOneNamedInstanceRole --) --)
```

or use any alternative LISP form for <irole-spec> that will return the desired IRole.

Finally, the contexts operator can be used to change "the current contexts list" (the value of $$CONTEXTS), which is used as the context list in all CKLONE translations based on functions that take a context list as a parameter. There are two forms of use:

```
(contexts <contexts-spec>)
(contexts <contexts-spec> --)
```

where <contexts-spec> is expected to be either an atom that names a single Context, or else a list whose elements are either atoms that name Contexts or forms that evaluate to single Contexts. The first usage of the contexts operator expands into:

```
(SETQ $$CONTEXTS --)
```

while the second usage translates into a PROG that binds $$CONTEXTS as a PROG variable and goes on to include translations of any embedded forms.

3.1.2 The CKLONE Subordinate Operators

Forms that have as CAR certain "subordinate operators" may be embedded in forms based on various of the CLISP operators

discussed above. Such forms are translated at the same time and as an effect of the translation of the form in which they are embedded. They cannot be translated independently. Subordinate operators allow certain changes to the current Concept or current Role to be simply and readably specified:

DIFFBY, as in: (diffby <role-spec>+). Establishes the RoleSets specified by <role-spec>i as differentiators of the current Role ($$ROLE). Each <role-spec> is expected to be either an atom that names some subordinate RoleSet, or else a LISP form that evaluates to a RoleSet. In the case where <role-spec> is a Role name, the resulting translation will attempt to find some Role of the given name attached to one of the current Concept's subconcepts. Note that KL-ONE Role names need not be unique, so specifying a RoleSet by name is generally less preferred than specifying in terms of both the name and the concept to which the RoleSet is locally attached. If no Role of the given name can be found, an error will occur when the translation is evaluated. The user is responsible for assuring that appropriate cables connect the Concept of the current Role and the Concepts of the Roles specified by <role-spec>i. This operator can be used only under the CKLONE CLISP roleset operator.

DIFFS, as in: (diffs <role-spec>). Establishes the current Role ($$ROLE) as a differentiator of the RoleSet specified by <role-spec>. <role-spec> is expected to be either an atom that names some RoleSet inherited by a superconcept, or else a LISP form that evaluates to a RoleSet. In the case where <role-spec> is a Role name, the resulting translation will attempt to find some Role inheriting the given name and inherited by one of the current Concept's superconcepts (the translation will eschew any local Roles inheriting the name). Note that KL-ONE Role

names need not be unique, so specifying a RoleSet by name is generally less preferred than specifying in terms of both the name and the Concept to which the RoleSet is locally attached. If no Role of the given name can be found, an error will occur when the translation is evaluated. This operator can be used only under the CKLONE CLISP operators roleset and proleset.

IHOOKS, as in: (ihooks <ihook-spec>+). Establishes the specified IHooks on the current CKLONE entity (either $$CONCEPT or $$ROLE, as appropriate). Each <ihook-spec> is expected to be either a list of the form:

(<time-spec> <KLActivity>
<procedure-name>)

or of the form:

(<time-spec> <KLActivity>
<lockout> <procedure-name>)

where <time-spec> is one of PRE, POST, WHEN, etc., <KLActivity> is the usual specification of the net-transforming operation that is to trigger the IHook, <lockout> is the lockout atom for the IHook (if any) and <procedure-name> is the name of the function that is the body of the IHook. This operator can be used under any of the CKLONE CLISP operators except contexts.

INDIVIDUATES, as in: (individuates <concept-spec>+). Establishes the current Concept as an individual of the specified Generic Concepts. Each <concept-spec> is expected to be either an atom that names a Concept, or a form that evaluates to one. This operator can be used only under the CKLONE CLISP operator iconcept.

ITAGS, as in: (itags <itag-spec>+). Establishes the specified ITags on the current CKLONE entity (either

$$CONCEPT or $$ROLE, as appropriate). Each <itag-spec> is expected to be a list of the form:

(<key-spec><sub-key-spec><datum>)

where <sub-key-spec> is optional. This operator can be used under any of the CKLONE CLISP operators except contexts.

METADESCRIBES, as in: (metadescribes <entity-spec>+). Establishes the current Concept ($$CONCEPT) as a metadescription of the metadescribable entities specified by <entity-spec>i. Each <entity-spec> is expected to be either an atom that names a Concept, or a LISP form that evaluates to a meta-describable net-entity. This operator can be used only under the CKLONE CLISP operator iconcept.

METAHOOKS, as in: (metahooks <meta-descr-spec>+). Establishes the specified meta-descriptions on the current CKLONE entity (either $$CONCEPT or $$ROLE, as appropriate). Each <meta-descr-spec> is expected to be either an atom that names an Individual Concept, or a LISP form that evaluates to one. This operator can be used under any of the CKLONE CLISP operators except contexts.

MODALITY, as in: (modality <modality-spec>). Changes the modality of the current Role ($$ROLE) to be that specified by <modality-spec>. <modality-spec> is expected to be a KL-ONE modality. This operator can be used only under the CKLONE CLISP operators roleset and proleset.

MODBY, as in: (modby <role-spec>+). Establishes the RoleSets specified by <role-spec>i as modifiers of the current Role ($$ROLE). Each <role-spec> is expected to be either an atom that names some subordinate RoleSet, or a LISP form that evaluates to a RoleSet. In the case where <role-spec> is a Role name, the resulting translation will attempt to find some Role of the given name attached to one of the current

Concept's subconcepts (the translation will eschew any local Roles inheriting the name). KL-ONE Role names need not be unique, so specifying a RoleSet by name is generally less preferred than specifying in terms of both the name and the Concept to which the RoleSet is locally attached. If no Role of the given name can be found, an error will occur when the translation is evaluated. The user is responsible for assuring that appropriate cables connect the Concept of the current Role and the Concepts of the Roles specified by <role-spec>i. This operator can be used only under the CKLONE CLISP roleset operator.

MODS, as in (mods <role-spec>). Establishes the current Role ($$ROLE) as a modifier of the RoleSet specified by <role-spec>. <role-spec> is expected to be either an atom that names some RoleSet inherited by a superconcept, or a LISP form that evaluates to a RoleSet. In the case where <role-spec> is a Role name, the resulting translation will attempt to find some Role inheriting the given name and inherited by one of the current Concept's superconcepts. KL-ONE Role names need not be unique, so specifying a RoleSet by name is generally less preferred than specifying in terms of both the name and the Concept to which the RoleSet is locally attached. If no Role of the given name can be found, an error will occur when the translation is evaluated. This operator can be used only under the CKLONE CLISP operators roleset and proleset.

NAME, as in: (name <name-spec>). Changes the locally designated name of the current Role ($$ROLE) to be that specified by <name-spec>. <name-spec> is expected to be an atom. This operator can be used only under the CKLONE CLISP operators roleset and proleset.

NUMBER, as in: (number <number-spec>). Changes the number facet of the current Role ($$ROLE) to be that specified by <number-spec>, which is expected to be a KL-ONE number facet. This operator can be used only under the CKLONE CLISP operators roleset and

proleset.

SATBY, as in (satby <irole-spec>+). Establishes the IRoles specified by <irole-spec>i as satisfiers of the current Role ($$ROLE). Each <irole-spec> is expected to be either an atom that names some IRole, or a LISP form that evaluates to an IRole. In the case where <irole-spec> is a Role name, the resulting translation will attempt to find some IRole of the given name attached to one of the current Concept's individuators. KL-ONE Role names need not be unique, so specifying a Role by name is generally less preferred than specifying in terms of both the name and the Concept to which the RoleSet is locally attached. If no IRole of the given name can be found, an error will occur when the translation is evaluated. The user is responsible for assuring that appropriate cables connect the Concept of the current Role and the Concepts of the IRoles specified by <irole-spec>i. This operator can be used only under the CKLONE CLISP operators roleset and proleset.

SATS, as in (sats <role-spec>). Changes the SATS relation out of the current Role ($$ROLE) to be that specified by <role-spec>. <role-spec> is expected to be either an atom that names some RoleSet inherited by a superconcept, or a LISP form that evaluates to a RoleSet. The user is responsible for assuring that appropriate cables connect the Concept of the current Role with the Concept of the RoleSet specified by <role-spec>. This operator can be used only under the CKLONE CLISP irole operator.

SUBCS, as in (subcs <concept-spec>+). Establishes the Concepts specified by <concept-spec>i as subconcepts of the current Concept ($$CONCEPT). Each <concept-spec> is expected to be either an atom that names some Generic Concept, or a LISP form that evaluates to one. This operator can be used only under the CKLONE CLISP operator concept.

SUPERCS, as in (supercs <concept-spec>+). Establishes the Concept specified by <concept-spec>i as

superconcepts of the current Concept ($$CONCEPT). Each <concept-spec> is expected to be either an atom that names some Generic Concept, or a LISP form that evaluates to one. This operator can be used only under the CKLONE CLISP concept operator.

TAGS, as in: (tags <tag-spec>+). Establishes the specified Tags on the current CKLONE entity (either $$CONCEPT or $$ROLE, as appropriate). Each <tag-spec> is expected to be a list of the form:

(<key-spec><datum-spec>).

This operator can be used under any of the CKLONE CLISP operators except contexts.

VR, as in: (vr <vr-spec>+). Establishes the things specified by <vr-spec>i as value restrictions of the current Role ($$ROLE). If a <vr-spec> is a non-numeric atom, it is interpreted as a Concept name. If it is a list, it is interpreted as a LISP form that evaluates to a Role value. If it is a string or a number, it is passed through to become a value restriction. Where the user wishes to make arbitrary atoms or lists to be Role value restrictions, he must use QUOTE.

In addition to the subordinate CKLONE operators mentioned above, there is an additional CKLONE CLISP operator, called atomval, that serves as an evaluation escape. As the above descriptions make clear, generally an atom (that does not dwimify to a form) that appears among expected operands in a CKLONE form is interpreted as the name of a Concept (an atom in the body of a CKLONE CLISP operator -- the part following the identifiers -- of course is just another LISP expression). Where the user desires

to evaluate an atom to produce an expected operand, (atomval <atom>) may be used.

3.1.3 CKLONE Errors

Since CKLONE allows arbitrary LISP forms to be intermixed with CKLONE forms, there are few errors that can be detected at dwimify-time. However, the following conditions are checked for during dwimification:

- Possible bad usage of a CKLONE CLISP operator directly under another CKLONE CLISP operator, e.g., use of proleset directly under concept. This is not per se illegal, e.g. (concept **FOO** (proleset FUM of (iconcept of --))) is a legal, albeit perhaps somewhat mis-organized, CKLONE construct. The diagnostic messages depend on the combination involved but are of the form "possible error in net-building statement, x or y used under z --".
- Inclusion of CKLONE sub-operators as atoms in the body of a CKLONE CLISP operator form, e.g.: (roleset NIL vr **FOO**), causes a warning message to be printed: "possible parenthesis error in net-building statement --". The same check looks for misplacement of the operand noise word "of" within the body of the form, unfortunately printing the same diagnostic.
- Misplaced sub-operator forms, e.g. (concept **FOO** (modality --)), causes an error to be generated with the message: "error in net-building statement, illegal operator sub-operator combination --".

3.1.4 Caveats

Realize that Role Names, by themselves, are in general a

poor way to specify a Role in the context of a network building statement (that is, when viewing the Role as a topological site within an expanding network). In the first place, such names are generally non-unique because they are inherited. For that matter, they are not required to be unique even among the Roles locally attached to the same Concept. This means that CKLONE expressions invoking name identification can mysteriously begin to result in unintended structures (without necessarily causing LISP errors) when prior code is added that makes names ambiguous. In the second place, such code is order-dependent because it does not include enough information to add RoleSets that cannot be found. For these reasons, the construction:

```
(diffs (roleset foo of fum))
```

is generally preferable to:

```
(diffs foo).
```

Remember that CKLONE sub-operator forms cannot be independently dwimified; they are translated only when the CKLONE CLISP forms containing them are dwimified. This means that they cannot appear outside of CKLONE CLISP operator forms. Also, attempts to dwimify them separately are futile: they will be treated as if they were calls on undefined functions, and will

not be translated.

$$CONTEXTS is bound to NIL at the top level of LISP when the CKLONE package is loaded, so that users who are not concerned with contexts need not bother with specifying them. However, if the user loads any of the resulting compiled code without loading CKLONE, he should make sure that this variable is bound properly. One way to do this would be to include the form: (contexts NIL) as the first CKLONE form to be executed.

$$CONCEPT is also bound to NIL at the top level of LISP when CKLONE is loaded, so that CKLONE CLISP forms with defaulted identifiers (e.g.: (irole), (iconcept)) will work even if the user's code has not established a binding for this variable. Although these cases are relatively rare, if the user loads any of the resulting compiled code without loading CKLONE, he should make sure that this variable is bound properly.

CKLONE is not intended for general use in the user's KL-ONE dependent functions: it does not make much of an attempt at optimized translations. This was a conscious design decision in favor of simplicity. This is not to say that it cannot be so used, but rather that embedding CKLONE forms in oft-called functions is probably unwise.

3.1.5 User Extensions to CKLONE

Adding to the CKLONE subordinate operators is simplicity itself. Merely define a translation function for the new form that takes the entire form as input and produces a list whose elements are LISP expressions (i.e., if the translation is a single KLONE function call, embed the translation in a list). Preferably, the translation function should not smash its input because the uniform scheme in CKLONE is to use CLISPARRAY to store translations, as opposed to destructively altering the user's code. The translation need not (should not) call CLISPTRAN, since this is already done elsewhere.

With the translation function defined, add entries of the form (<operator> <tranFnName>) to the macro lists for the CKLONE CLISP operators under which the new subordinate operator is to function. These lists all have names of the form CKL<operator-name>MACROS (e.g., for the proleset operator the macro list is named CKLPROLESETMACROS). It is only necessary to include an entry with the lower case version of the operator name. Finally, add the lower case version of the operator to the list CKLSUBOPRS. This list is used to aid in identifying misused or mis-parenthesized subordinate operators.

Of course the user is free to define his own CLISP

operators. The CKLONE CLISP operators were implemented by means of the CLISPWORD property discussed on page 23.54 of the Interlisp Reference Manual [Teitelman, 1978].

3.2 CKLONELIBRARY

This package defines additional CKLONE CLISP and Subordinate Operators. Whereas the CKLONE operators discussed in the above section are primarily intended for initializing KL-ONE networks, the operators contained in CKLONELIBRARY are optimized for use in arbitrary user functions. These operators are not included in the basic CKLONE package because they require the presence of certain library functions to support the resulting translations.

3.2.1 CKLONELIBRARY CLISP Operators

Four new CLISP operators are defined: `, df, @ and the bracket operator {}. ` is an abbreviation for:

```
(KLGetNamedConcept (QUOTE <concept-name>)).
```

The df (for "descended from") operator is an infix abbreviation for KLIsConceptDescendantP, e.g.:

```
(EQ ($$CONCEPT df `WINDOW) T).
```

The {} bracket operators translate into a form that will "walk"

from a specified Concept along a path specified by either super Roles or Role names. The first element enbracketed is expected to be a form that evaluates to a Concept. Successive elements of the tuple are expected to be either atoms that name Roles or else forms that evaluate to Roles. For example, if:

```
[concept foo [roleset role1 (vr fum]]
[concept fum [roleset role2 (vr fi]]
[concept fi [roleset role3 (vr fy]]
```

then the form:

```
{|c|foo;role1;role2;role3}
```

or

```
{`foo;role1;role2;role3}
```

will result in a translation that evaluates to |C|fy. The ";" separators are optional. If the specified Role path cannot be established working away from the specified Concept, an error will result: "unable to establish role path".

The @ operator is (in effect) an abbreviation for KLFindOneNamedRole. An example of the use of the "@" operator is:

```
role2@({`foo; role1})
```

which (relative to the above example) results in |R|(role2 of fum). Note that an "extra" set of parentheses must be inserted to avoid confusing DWIM needlessly.

If the expression following "@" is an atom, it is interpreted as a Concept name if possible. Otherwise, it is evaluated. In any case, the expression preceding the operator is never evaluated.

3.2.2 CKLONELIBRARY Subordinate Operators

When loaded, CKLONELIBRARY defines two new subordinate operators: intension and extension.

INTENSION, as in: (intension <iconcept-spec>). Establishes the specified Individual Concept as the intension of the current Concept. <iconcept-spec> is expected to be either an atom that names an Individual Concept, or a form that evaluates to one. For use only under the CKLONE CLISP operators concept and iconcept.

EXTENSION, as in: (extension <concept-spec>). Establishes the specified Concept as the extension of the current Concept. <concept-spec> is expected to be either an atom that names a Concept (either generic or individual), or a form that evaluates to one. For use only under the CKLONE CLISP operator iconcept.

3.3 The Derivation Interpreter

Part of the process of creating a KL-ONE Individual Concept is to derive the fillers of its inherited RoleSets. The purpose of the derivation interpreter is to provide a greater measure of control over the Role filler derivation process than is afforded by the standard KLDerive IHook. The derivation interpreter provides the means to explicitly and declaratively specify the order in which RoleSets will be filled, and permits the programmer to designate more than one filler derivation method for a given RoleSet.

The derivation interpreter is invoked via the function **Make**, an NLAMBDA NOSPREAD, CAR of whose argument list is the name of the Generic Concept that is to be individuated, and CDR of which is a list whose elements alternate between atoms that name inherited RoleSets of the Generic Concept and forms that evaluate to Role fillers of the indicated RoleSets at the new individual. **Make** creates the new Individual Concept, together with IRoles to implement the fillers specified in the argument list. **Make** then attempts to derive fillers for each inherited RoleSet of Obligatory Modality that has not already been filled.

The activities of the derivation interpreter are controlled by derivation ITags found on the Generic RoleSets to be filled.

These "derivation tags" allow the user to:

- specify that other inherited RoleSets of the expanding Individual Concept must be filled before deriving the filler of the RoleSet to which the tag is attached;
- specify that variables be bound dynamically for the benefit of all the derivation procedures indicated in the tag;
- specify multiple derivation procedures for finding fillers of the RoleSet to which the tag is attached;
- specify that other inherited RoleSets of the Individual Concept should be filled following derivation of the filler of the RoleSet to which the tag is attached.

A derivation tag is an ITag whose major key is the LISP atom **derivation**, and whose minor keys are either the LISP atoms **Prerequisites**, **Binding**, or **Consequents**, or else the names of Role filler derivation procedures.

Entries against the minor keys **Prerequisites** and **Consequents** are expected to be the names of RoleSets inherited by the Individual Concept whose Role fillers are to be derived. Entries against the minor key **Binding** are expected to be doubleton lists whose CARs are the variables to be bound during the derivation of the Role fillers and whose CADRS are LISP forms that evaluate to the desired bindings.

Where the minor key is the name of a derivation procedure, the entries give further information pertinent to that particular

derivation method. Entries are expected to be lists whose CARs are one of the LISP atoms **Prerequisites**, **Binding** or **Arguments**. Entries of the form **(Prerequisites --)** allow the user to specify other RoleSets of the expanding Individual Concept that must be filled before the particular derivation procedure can be applied. Successive elements of the list are expected to be the names of RoleSets inherited by the Individual Concept. Entries of the form **(Binding --)** can be used to cause further dynamic variable bindings that are in force only for the particular derivation procedure. Again, successive elements of the list are expected to be doubleton lists whose CARs are variable names and whose CADRS are forms that evaluate to the desired bindings. Finally, entries of the form **(Arguments --)** specify the bindings for the arguments of the derivation function. The successive elements of the list are simply forms that are evaluated prior to application of the derivation function.

The derivation function supplied by the user is expected to return a single Role filler (Role Value for an IRole) if the Number Facet of the RoleSet being filled is 1. Otherwise (and this includes the case of RoleSets with Number Facets (1 1)), the user's derivation function is expected to return a list of Role fillers.

In order to simplify the problem of composing derivation tags, an additional CKLONE subordinate operator: **derivation** is provided. This allows the derivation tag to be treated by the programmer as a single multi-level list rather than as a number of ITags with various minor keys. The following are two examples taken from the AIPS initialization that show the use of the CKLONE **derivation** operator:

```
(derivation (MakeWindowBorder
               (Prerequisites Aperture
                  Label Context)
               (Binding (aperture {$$CONCEPT;Aperture})
                  (label {$$CONCEPT;Label}))
               (Arguments {$$CONCEPT;Context}
                  {aperture;LowerLeft}
                  {aperture;UpperRight}
                  {label;Width}
                  {label;Height}]

(derivation (Copy (Prerequisites Aperture)
               (Arguments {$$CONCEPT;Aperture}))
            (SeekVisibleSubstrate
               (Prerequisites Application)
               (Arguments {(Make DISPLAY
                              Application {$$CONCEPT;Application
                           ;Ground}))
            (Consequents Aperture]
```

In addition to the derivation ITags, the derivation interpreter is sensitive to two Tags: DefaultValue and DefaultPrototype. If no derivation for a Role Filler can be found, or if none succeeds, any existing DefaultValue Tag on the Generic Roleset will be used to provide the filler (the item of

the tag is evaluated and the result is used as the filler or fillers of the Role.)

A DefaultPrototype tag attached to a Generic Concept being individuated informs **Make** that a different Generic Concept than that named in the call to **Make** should be used. E.g., a DefaultPrototype tag on **WINDOW** indicates that an instance should be made of the more specific concept **NONSCROLLWINDOW**.

3.3.1 Control Structure of the Derivation Interpreter

Derivation tags and user-written derivation functions can fail for a variety of reasons. The interpreter will abandon a derivation method, for example, if it finds that it has been referred by **Prerequisites** specifications back to a RoleSet whose filler it is already attempting to derive. If a user written derivation function fails (by calling or causing a call to **ERROR**), the derivation interpreter proceeds to the next derivation method for the RoleSet currently having its filler derived.

If all derivation methods for a RoleSet fail (or none are provided), and the form tagged to the RoleSet as DefaultValue also fails (or no such tag is provided), the interpreter will ask the user interactively to supply the Role

Value, if the global flag ASKFILLERFLG is non-null. If all of the above methods fail to provide the role filler, the Individual Concept being created is deleted from the network and a call to **ERROR** results. To aid in debugging derivation functions and derivation tags, the user can assure an immediate break on any internal error generated by the derivation interpreter by resetting **DERIVENOBREAKFLG** to NIL.

If all attempts to fill a RoleSet named in the **Consequents** portion of a derivation tag fail, no special action is taken except to abandon those efforts and move on to the next RoleSet mentioned in the **Consequents**.

4. THE CURRENT AIPS KNOWLEDGE BASE

The following is a snapshot of the code which initializes the AIPS KL-ONE knowledge base. It is a set of Interlisp programs, but utilizes the CKLONE package described in the previous section of this report. It should be relatively self-explanatory for those already generally familiar with KL-ONE structures. For those not familiar with KL-ONE, skimming through this section will provide an overview of the structures involved; a thorough study of the KL-ONE literature will be needed for any deeper understanding.

```
(DEFINEQ

(InitMiddleEarth
  [LAMBDA NIL                                    (* Edited by
                                                 F.Zdybel on
                                                 31-Jan-80.)

    (* Initializes AIPS and the necessary Domain World
    Model SI-Net fragments for the Middle Earth
    demonstration system.)


    (InitAIPS)
    (InitGeographyModel])

(InitAIPS
  [LAMBDA NIL                                    (* Edited by Zdybel
                                                 on 1-Mar-80.)
    (* Initializes the Presentation, Viewing Organization
     and Realization Models of AIPS.)

    (concept DISPLAYITEM

    (* A Display Item can be either a Presentation or a
```

```
        Display Form. Thus decomposition of Presentations
        occurs until the level of Display Forms is
        reached.)

                (roleset Location (vr LISTP))

        (* The position inside the Display Item that
        locates it relative to anything else will probably
        be the computed center. This presents a difficulty
        only in the case of Text. There a rectangular
        envelope must be computed and the center of this
        envelope determined.)

                (roleset Width (vr NUMBERP))
                (roleset Height (vr NUMBERP))

        (* These Roles are used by layout processes that
        operate at the level of the Viewing Organization
        Model.)

                (roleset Envelope (vr CLOSEDCURVE))

        (* This role is useful for characterizing the area
        subsumed by a Display Item.)

                )
        (InitRealizationModel)
        (InitPresentationModel)
        (InitViewingOrganizationModel])

(InitPresentationModel
  [LAMBDA NIL                                    (* Edited by Zdybel
                                                 on 1-Mar-80.)

        (* Initializes the Presentation Model for AIPS.
        These concepts describe the linkage between
        information being output and the structure of the
        output.)

        (concept PRESENTATION
```

```
(* The top-level concept of the Presentation Model
for AIPS. This description embraces all forms of
information output, including synthesized natural
language messages and synthesized speech, as well
as graphic displays.)

        (roleset Application

(* This Roleset is intended to express the binding
of some particular (Individual) Presentation to a
set of information which is to be displayed.
Other Roles of the generic description of the
Presentation may imply that further information may
be involved in all Presentations of a particular
type.)

                (number (0 NIL))
                (vr ITEMPLATE))
        (roleset Realization

(* This Roleset characterizes the internal
structure of the Presentation as a group of
consitituents, which might be anything.
This Role is differentiated and modified by generic
descriptions descendant from Presentation.)

                (number (0 NIL))
                (vr **ANYTHING**)))

(concept DISPLAY
                (* A Graphically realized Presentation.)
        (supercs PRESENTATION DISPLAYITEM)
        (roleset NIL (mods Realization@PRESENTATION)
                (vr DISPLAYITEM)

(* A Display Item is either a Presentation or a
Display Form. Expansion of the description of any
individual Display terminates at the level of
Display Forms (the Viewing Organization and
Realization Models of AIPS.))

                )
```

```
        (roleset Ground (modality Obligatory)
                 (vr VIEWSURFACE))

(* This Roleset describes the coordinate system and
any boundaries of the bit map surface on which the
Display will be realized.)


        )

(concept TEMPLATE

(* A Template is a meta-description that
characterizes a set of inherited Rolesets in an
SI-Net hierarchy. Its function is to indicate a
chunk of information, as for example in
characterizing the Domain World extension that is
involved in an individual of Presentation.)

        (roleset ConceptGroup

(* All Templates are individual descriptions.
The satisfiers of the Concept Group Role
meta-indicate Concepts in the Domain World as a way
of specifying sub-lattices of the Domain Model
within which inheritance of the specified Rolesets
(specified by the Role Group Role of Template) is
to be considered as part of the set of information
being described.)

                 (number (0 NIL))
                 (vr CONCEPT))
        (roleset RoleGroup

(* The satisfiers of the Role Group Role
meta-indicate Rolesets in the Domain World Model.
Because of the inheritance of Rolesets, this is an
intensional characterization of a set of inherited
Roles. This characterization is limited in scope by
the fillers of the Concept Group Role, which
specify subsets of the SI-Net within which
inheritance of the specified Roles is to be
considered as generating the extension.)
```

```
                    (modality Obligatory)
                    (number (1 NIL))
                    (vr ROLE)))

    (concept ITEMPLATE

    (* A Template whose Concept Group Role restricts it
    to the description of slots inherited by Individual
    Concepts.)


            (supercs TEMPLATE)
            (roleset NIL (mods ConceptGroup@TEMPLATE)
                    (vr ICONCEPT)))

    (concept ICONCEPT (supercs CONCEPT)))

    (* These initialization functions build SI-Net
    descriptions of the principal Display types, such
    as Map, Table, Menu, etc.)


    (InitTableConcepts)
    (InitMapConcepts])

(InitMapConcepts
  [LAMBDA NIL                                    (* Edited by Zdybel
                                                 on 29-Feb-80.)

    (* Initializes that portion of the AIPS
    Presentation Model having to do with Maps.)


    (concept MAP
            (* Map is a major Presentation form.)
            (supercs DISPLAY)
            (roleset Name (vr STRINGP))

    (* This is the string that gets made into a Label
    for the Map.)


            (roleset CoordSys (modality Obligatory)
                    (vr 2DCOORDINATESYSTEM))
```

(* This is the Coordinate System that is assumed to be shared by all of the items being depicted. It is not the Coordinate System of the depictions.)

(roleset ScaleUnit (vr **DISTANCEUNIT**))

(* The filler of this role should be selected according to the user's convenience. It is used in constructing the reference grid and the scale ikon of the legend.)

(roleset ScaleDistance (vr **NUMBERP**))

(* This is the distance equivalent to the filler of the Scale Unit Role in the Coordinate System of the Presentation.)

(roleset Label (diffs Realization@**DISPLAY**)
(vr **TEXT**))
(roleset Legend (diffs Realization@**DISPLAY**)
(vr **MAPLEGEND**))

(* This may be one of the places where one desires to have a part of a Presentation viewed through one window while other parts of the Presentation are viewed through another (i.e. when scrolling over a Map with a window one may wish that the Legend is always in view.))

(roleset Border (diffs Realization@**DISPLAY**)
(vr **CLOSEDCURVE**))

(* For some types of projections, the border of the Map is not necessarily a Rectangle, or even a Polygon.)

(roleset ReferenceGrid (diffs Realization@**DISPLAY**)
(vr **MAPGRID**))
(roleset Item (diffs Realization@**DISPLAY**)
(vr **MAPITEM**)) (* The things making up the map.)

```
        )

    (concept MAPLEGEND (supercs ARRANGEMENT))
            (roleset ScaleIkon (diffs Item@ARRANGEMENT)
                    (vr MAPSCALE))

    (* This is the funny little bar with the colored
    stripes and the numbers under it.
    Note that this part of the Legend is sensitive to
    the current scale of the map, which may change due
    to Window-driven scaling.)


            (roleset Table (diffs Item@ARRANGEMENT)
                    (number (0 NIL))
                    (vr LEGENDTABLE))       (* This is the
                                            little
                                            correspondence Table
                                            between map symbols
                                            and explanatory
                                            text.)
            (roleset Label (diffs Item@ARRANGEMENT)
                    (vr TEXT))              (* Probably says
                                            something brilliant
                                            like "Legend".)
            )

    (concept MAPGRID (supercs DISPLAY))
            (roleset ReferenceLine (diffs Realization@DISPLAY)
                    (modality Obligatory)
                    (number (4 NIL))
                    (vr CURVE))
            (roleset ReferenceLabel (diffs Realization@DISPLAY)
                    (modality Obligatory)
                    (number (4 NIL))
                    (vr TEXT))              (* Attached
                                            procedures know how
                                            to realize the
                                            particular types of
                                            reference grids.)
            ])

(InitTableConcepts
  [LAMBDA NIL                               (* Edited by Zdybel
                                            on 1-Mar-80.)
    (* Initializes Concepts having to do with Tables.)
```

```
(concept ARRANGEMENT

(* An Arrangement is any collection of Display
Items whether or not their arrangement conveys
information. Rows, Columns, Legends are examples of
Arrangements.)

        (supercs DISPLAY)
        (roleset Item (diffs Realization@DISPLAY)
                 (modality Obligatory)
                 (number (2 NIL)))      (* If there aren't
                                        at least two items,
                                        why bother?)
        (roleset ItemOrder (modality Obligatory)
                 (vr LISTP))

(* Imposes an ordering on the fillers of the Item
Role. The elements of the list point directly at
fillers.)

        (roleset Border (diffs Realization@DISPLAY)
                 (vr CLOSEDCURVE))
        (roleset Separator (diffs Realization@DISPLAY)
                 (modality Optional)
                 (number (0 NIL))
                 (vr DISPLAYFORM))      (* Could be anything
                                        but is probably a
                                        Line.)
        )

(concept TABLE

        (* A two-dimensional table is conceived
        of as an Arrangement of Arrangements.)

        (supercs ARRANGEMENT)
        (roleset Group (mods Item@ARRANGEMENT)
                 (vr ARRANGEMENT))

(* Groups are either the columns or rows of the
Table. Just which depends on how one wishes to
describe the Table: as a set of columns or a set of
rows.)
```

```
        (roleset GroupOrder (mods ItemOrder@ARRANGEMENT)
                 (vr LISTP))          (* Imposes an
                                      ordering on the
                                      fillers of the Group
                                      Role.)
        (roleset Theme (vr TEMPLATE))

(* Template identifies the slots in the Domain
Model that correspond to elements of each group in
the Table.)


        (roleset ThemeOrder (vr LISTP))

(* Imposes an ordering on the slots comprising the
Template filling the Theme Role.)
        )

[concept ROWTABLE

(* A Table whose Theme is spread along its rows.
Hence, a column of rows.)


        (supercs TABLE COLUMN)
        (roleset RowOrder (mods GroupOrder@TABLE))
        (PROG [(row (roleset Row (mods Group@TABLE)
                             (vr ROW]
              (roleset Header (diffs (atomval row)))
                       (modality Obligatory)
                       (number 1))

(* The entries in the Header Row will be
constructed out of the slot names of the slots
indicated in the Theme of the Table.)


              (roleset Foot (diffs (atomval row))
                       (number (0 1)))

(* A very long table might want another index
running along the bottom edge.)


              (roleset Entry (diffs (atomval row))
```

```
                        (modality Obligatory)
                        (number (1 NIL]

(concept ROW

(* Note that a Row in a Table may have a non-linear
structure (e.g., in order to "compress" the width
of the table by putting two interleaved rows
together to make a single group in the table.))


        (supercs ARRANGEMENT)
        (roleset NIL (mods Item@ARRANGEMENT)
                 (modality Obligatory)
                 (number (2 NIL)))
        (roleset Separation (vr NUMBERP))
                                        (* Horizontal
                                        separation,
                                        naturally.)
        )

(concept COLUMNTABLE

(* A Table whose Theme is spread along its columns.
Hence, a row of columns.)


        (supercs TABLE ROW)
        (roleset ColumnOrder (mods GroupOrder@TABLE))
        (PROG [(column (roleset Column (mods Group@TABLE)
                                (vr COLUMN]
              (roleset Entry (diffs (atomval column)))
                       (modality Obligatory)
                       (number (1 NIL))
                       (vr COLUMN))
              (roleset LeftIndex (diffs (atomval column))
                       (modality Obligatory))

(* The items of the Left Index will be constructed
out of the Role names of the slots indicated by the
Table's Theme.)


              (roleset RightIndex (diffs (atomval column))
                       (number (0 1)))
```

```
    (* A very wide Table might also want an index along
    its left edge.)


              ))

    (concept COLUMN

              (* In contrast to Rows, Columns are
              expected to have a linear structure.)

              (supercs ARRANGEMENT)
              (roleset NIL (mods Item@ARRANGEMENT)
                       (modality Obligatory)
                       (number (2 NIL)))
              (roleset Separation (vr NUMBERP))
                                              (* Vertical
                                              separation,
                                              naturally.)
              ])

(InitViewingOrganizationModel
  [LAMBDA NIL                                 (* Edited by Zdybel
                                              on 1-Mar-80.)

    (* Initializes the Viewing Organization Model of
    the AIPS SI-Net. These concepts have to do with the
    syntactic organization of display elements, and
    include such as Window, View Surface, Display Form,
    etc.)


    (concept MOTILEDISPLAYITEM

    (* The purpose of this concept is to mark which
    Display Items can be re-located
    (those whose locations are not significant to their
    semantics.))


              (supercs DISPLAYITEM))

    (concept STABILEDISPLAYITEM

    (* The purpose of this concept is to mark which
    display items cannot be re-located
```

```
(those whose locations are significant to their
semantics.))

        (supercs DISPLAYITEM))
(InitCoordinateSystemConcepts)

(* Set up all the necessary concepts for describing
positions on View Surfaces, Display Surfaces, etc.)

(concept VIEWSURFACE

(* A View Surface is a medium for the arrangement
of viewable objects. It is a Region with an
internal coordinate system.)

        (supercs RECTANGULARREGION)
        (roleset CoordSys (modality Obligatory)
                 (vr CARTESIANSYSTEM))
        (roleset NIL (mods Continuum@RECTANGULARREGION)
                 (vr CARTESIANSYSTEM))

(* Both the Continuum (the containing coordinate
system) and the internal coordinate system must be
Cartesian.)

        (roleset Window (number (0 NIL))
                 (vr WINDOW))          (* An aspect of View
                                       Surfaces is that
                                       Windows open onto
                                       them.)
        )

(concept DISPLAYSURFACE

(* A Viewsurface that represents an actual chunk of
bitmap memory, either visible or invisible.)

        (supercs VIEWSURFACE)
        (roleset Plane (modality Obligatory)
                 (vr PLANE))
```

```
(* Indicates which of the primal Display Surfaces
the Display Surface is located on.)

        (roleset Location (mods LowerLeft@RECTANGULARREGION)
                (vr LISTP))

(* A Tuple describing the location of the origin
(lower left corner) of the internal coordinate
system in the coordinate system of the Continuum
(Screen.))

        (roleset RegionNumber (modality Obligatory)
                (vr NUMBERP))

(* Establishes the correspondence with a BMG
Display Region more directly than via Nexii.)

        (roleset NIL (mods CoordSys@VIEWSURFACE)
                (vr VIEWSYSTEM))
        (roleset NIL (mods Continuum@VIEWSURFACE)
                (vr VIEWSYSTEM))

(* Because a DISPLAYSURFACE represents actual map
memory, the internal and external coordinate
systems must be integer Cartesian.)
        )

(concept VISIBLESURFACE

(* Here the medium is constrained to be a Display
Region onto one of the visible planes.
This difference is reflected in a different
derivation of the Display Region filling the Medium
Role.)

        (supercs DISPLAYSURFACE))

[concept INVISIBLESURFACE

(* Here the medium is constrained to be a Display
Region onto one of the off-screen planes of bitmap
memory.)
```

```
            (supercs DISPLAYSURFACE)
            (roleset NIL (mods Plane@DISPLAYSURFACE)
                     (vr (iconcept HIDDENPLANE of PLANE])

    (concept PLANE
            (* Represents a plane of BMG-11 bitmap memory.)
            (roleset PlaneNumber (modality Obligatory)
                     (vr NUMBERP))
            (roleset NIL (mods Window@DISPLAYSURFACE)
                     (vr SCREENWINDOW))

    (* Note that the role filler of CoordSys should be
    the same as the role filler of Continuum
    (the containing and internal coordinate systems are
    the same.))
            )
    (InitWindowConcepts)                    (* Sets up the
                                            generic descriptions
                                            of the different
                                            types of Windows.)

    NIL])

(InitWindowConcepts
  [LAMBDA NIL                               (* Edited by Zdybel
                                            on 6-Mar-80.)
    (concept WINDOW

    (* Window is a concept that serves many purposes.
    In the first place, a Window is a convenient way to
    organize and mobilize the use of the limited
    available display area. In the second place, it can
    serve as a context for the interpretation of input.
    Finally, a Window constitutes an important part of
    the environment for the realization of a display.
    This explains how it is possible to implement some
    scrolling in a way that eliminates undesirable edge
    effects: the "position" of the Window over the
    substrate can influence the re-layout and
    re-realization of the underlying Presentations.)


            (supercs DISPLAY)
            (roleset Context (mods Ground@DISPLAY)
                     (vr VISIBLESURFACE)
```

```
                    (tags (DefaultFiller `CONRACSCREEN)))

    (* Basically, the context of the Window must always
    be specified from outside. If not specified, it can
    only be defaulted.)


            (roleset Substrate (modality Obligatory)
                    (vr VIEWSURFACE))

    (* Note that at the current time a Window can open
    onto at most one View Surface.
    The derivation by SeekVisibleSubstrate assumes that
    DISPLAY has procedural attachments for selecting
    displays given their eventual content.)


            [roleset Border (diffs Realization@DISPLAY)
                    (vr RECTANGLE)
                    (derivation (MakeWindowBorder
                                (Prerequisites Aperture
                                               Label Context)
                                (Binding (aperture
                                       {$$CONCEPT;Aperture})
                                         (label
                                       {$$CONCEPT;Label}))
                                (Arguments
                                     {$$CONCEPT;Context}
                                    {aperture;LowerLeft}
                                   {aperture;UpperRight}
                                           {label;Width}
                                          {label;Height}]
            [roleset Label (diffs Realization@DISPLAY)
                    (vr WINDOWLABEL)
                    (derivation (MakeWindowLabel
                                (Prerequisites Aperture
                                               Context)
                                (Binding (aperture
                                    {$$CONCEPT;Aperture}))
                                (Arguments
                                     {$$CONCEPT;Context}
                                    {aperture;LowerLeft}
                                   {aperture;UpperRight}))
                               (Consequents Border]
            [PROG (servant)
                  (servant_(roleset Servant (number (0 NIL))
```

```
                                        (vr WINDOW)))

    (* A Window may have other windows closely
    associated with it. The "Servant" relationship
    indicates other windows whose locations must be
    attended to when a window is moved.)

                (roleset OuterServant
                        (diffs (atomval servant)))
                (roleset InnerServant
                        (diffs (atomval servant]

    (* The role Servant is differentiated into
    InnerServant and OuterServant because the rules for
    relocating these two cases are likely to differ.
    Note that it is necessary to use a circumlocution
    in order to set up the differentiation properly
    (because of name inheritance.))

            (roleset Master (vr WINDOW))
            (roleset Aperture (modality Obligatory)
                    (vr DISPLAYSURFACE))
            [roleset NIL (mods Location@DISPLAY)
                    (derivation (Copy (Prerequisites Border)
                                      (Arguments
                                {$$CONCEPT;Border;LowerLeft}]
            [roleset NIL (mods Width@DISPLAY)
                    (derivation (Copy (Prerequisites Border)
                                      (Arguments
                                 {$$CONCEPT;Border;Width}]
            [roleset NIL (mods Height@DISPLAY)
                    (derivation (Copy (Prerequisites Border)
                                      (Arguments
                                {$$CONCEPT;Border;Height}]
            [roleset NIL (mods Envelope@DISPLAY)
                    (derivation (Copy (Prerequisites Border)
                                      (Arguments
                                        {$$CONCEPT;Border}]
            (tags (DefaultPrototype `NONSCROLLWINDOW)))

    (concept NONSCROLLWINDOW (supercs WINDOW)
            [roleset NIL (mods Aperture@WINDOW)
                    (derivation (Copy (Prerequisites Substrate)
                                      (Arguments
```

```
                                        {$$CONCEPT;Substrate}))
                                    (Consequents Border
                                                 Label]
            [roleset NIL (mods Substrate@WINDOW)
                         (vr VISIBLESURFACE)
                         (derivation (Copy (Prerequisites Aperture)
                                           (Arguments
                                           {$$CONCEPT;Aperture}))
                                     (SeekVisibleSubstrate
                                       (Prerequisites Application)
                                       (Arguments {
                                                   (Make DISPLAY
                                                     Application
                                       {$$CONCEPT;Application})
                                                   ;Ground}))
                                     (Consequents Aperture]

(* In the case of a StaticWindow, the Display
Region that is the Implementation Role Filler is
also the Display Region that is the Medium of the
Substrate.)
        )

(concept SCROLLWINDOW (supercs WINDOW)
         (roleset ScrollBar (diffs Realization@DISPLAY)
                  (number (0 2))
                  (vr CONTROLBAR))

(* Designations on the Scroll Bar of a Window can
be used to modify the description filling the
Window's Location Role.)

         (roleset SubstrateLocation (vr LISTP))

(* The location of the Window's origin
(the origin of its display region) on the
coordinate system of the Substrate.)

         (tags (DefaultPrototype `FASTWINDOW)))

(concept SLOWWINDOW

(* A Window which scales and scrolls by causing the
stuff being viewed through it to be re-Presented.)
```

```
            (supercs SCROLLWINDOW)
            (roleset ScaleBar (diffs Realization@DISPLAY)
                     (vr CONTROLBAR))

        (* Designations in the Scale Bar of a Window can be
        used to modify the descriptions filling the Entry
        and Exit Roles of the CoordSys of Window's
        Substrate (i.e., the scale of the projection
        through the Window). Since there is no hardware
        level stuff for doing scaling, it must be
        accomplished by re-presentation, hence can only be
        accomplished by SlowWindows.)
                )

        (concept FASTWINDOW

        (* Any Window whose substrate is an actual chunk of
        map memory can scroll very quickly via
        BMGMoveRegion. Hence the name FastWindow.)

          (supercs SCROLLWINDOW)
          [roleset NIL (mods Substrate@SCROLLWINDOW)
                   (vr INVISIBLESURFACE)
                   (derivation
                     (SeekInvisibleSubstrate
                       (Prerequisites Application)
                       (Arguments {(Make DISPLAY Application
                                         ({$$CONCEPT;Application}))
                                  ;Ground}]
          ])

(InitCoordinateSystemConcepts
  [LAMBDA NIL                                    (* Edited by Zdybel
                                                 on 29-Feb-80.)

        (* Initializes those concepts having to do with
        Coordinate Systems and mappings between them.
        Some major changes from the previous paradigm occur
        among these concepts. For example, positions are
        now simply LISTP, and must be interpreted relative
        to some Coordinate System found in the context of
        the description referencing the position.)
```

```
(concept COORDINATESYSTEM
        (roleset Name (vr STRINGP))

(* This Role must be explicitly included because it
is necessary for distinguishing among Coordinate
Systems that do not as yet have Entries and Exits.
It is not desirable to use a Concept Name as other
than an indexing mechanism at the implementational
level.)

        (roleset Exit (number (0 NIL))
                (vr MAPPING))          (* These are the
                                       transforms for
                                       leaving the
                                       Coordinate System.)
        (roleset Entry (number (0 NIL))
                (vr MAPPING))          (* Transforms for
                                       entering the
                                       Coordinate System.)
        (roleset Dimensionality (modality Obligatory)
                (vr NUMBERP)))

(concept INTEGERSYSTEM (supercs COORDINATESYSTEM)

(* The purpose of this distinction is to signal
that certain transformations must involve rounding.
Bit maps are integer systems.)
        )

(concept 2DCOORDINATESYSTEM (supercs COORDINATESYSTEM)
        (roleset NIL (mods Dimensionality@COORDINATESYSTEM)
                (vr 2)))

(concept ORTHOGONALSYSTEM

(* This Concept should have an SD that describes
the constraint of Orthogonality on the system's
axes.)
        (supercs COORDINATESYSTEM))

(concept CARTESIANSYSTEM (supercs 2DCOORDINATESYSTEM
                                  ORTHOGONALSYSTEM))

(concept VIEWSYSTEM
        (* This kind of coordinate system
```

```
                can be used to describe display regions.)
              (supercs INTEGERSYSTEM CARTESIANSYSTEM))

    (concept VIEWSYSTEM (supercs CARTESIANSYSTEM INTEGERSYSTEM))

    (* This rather specialized description applies to
    the coordinate systems of bit map terminals.)
            )

    (concept MAPPING (roleset From (modality Obligatory)
                                   (number (1 NIL))
                                   (vr COORDINATESYSTEM))
              (roleset To (modality Obligatory)
                          (number (1 NIL))
                          (vr COORDINATESYSTEM))

    (* Should have an ITag whose referent is a function
    that can be applied to a list of positions and
    smash them with transformed coordinates.)
            )

    (concept LINEARTRANSFORM (supercs MAPPING)
              (roleset NIL (mods From@MAPPING)
                           (vr ORTHOGONALSYSTEM))
              (roleset NIL (mods To@MAPPING)
                           (vr ORTHOGONALSYSTEM))
              (roleset Origin (vr LISTP))
              (roleset UnitVector (vr LISTP)))

    (concept 2DLINEARTRANSFORM (supercs LINEARTRANSFORM)
              (roleset NIL (mods From@LINEARTRANSFORM)
                           (vr CARTESIANSYSTEM))
              (roleset NIL (mods To@LINEARTRANSFORM)
                           (vr CARTESIANSYSTEM)))

    (concept TRANSLATION (supercs 2DLINEARTRANSFORM)
              (roleset NIL (mods UnitVector@2DLINEARTRANSFORM)
                           (vr <1 1>))          (* Most of our
                                                transforms will be
                                                purely
                                                translations.)
              ])

(InitRealizationModel
  [LAMBDA NIL                                   (* Edited by Zdybel
                                                on 29-Feb-80.)
```

```
(* Initializes the AIPS Realization Model.
Concepts at this level have to do with geometric
shapes (which are isomorphic to descriptions of
line drawings.))


(concept DISPLAYFORM

(* Display Items come in two flavors: Displays
(graphic Presentations) and Display Forms
(graphic phenomena.))


        (supercs DISPLAYITEM))

(concept DISPLAYATOM

(* A Display Atom is the most primitive type of
Display Form. It has no constituents and is the
final descriptive level before descent into LISP
drawing procedures.)


        (supercs DISPLAYFORM))

(concept POINT (supercs DISPLAYATOM)     (* The Location of a
                                            Point is defined in
                                            the simplest terms
                                            possible.)
        )

(concept CURVE                           (* Any line,
                                            including straight
                                            lines.)
        (supercs DISPLAYATOM)
        (roleset Approximation (number (0 NIL))
                 (vr EDGESET))

(* For the moment, we will be approximating curved
lines with straight line segments.)


        )

(concept LINE                            (* The straight
```

```
                                                    variety.)
            (supercs CURVE)
            (roleset NIL (mods Approximation@CURVE)
                     (number 0))
            (roleset EndPoint (modality Obligatory)
                     (number 2)
                     (vr LISTP)))

    (concept EDGESET                                (* A cheap way of
                                                    representing a bunch
                                                    of little contiguous
                                                    lines.)
            (supercs DISPLAYATOM)
            (roleset VertexList (modality Obligatory)
                     (vr LISTP)))

    (concept CLOSEDCURVE (supercs CURVE)

    (* May differ from CURVE by the addition of tagged
    procedures for determining whether a given point is
    within the Approximation (which is expected to be
    an edgeset that closes on itself.))
            )

    (concept ELLIPSE (supercs CLOSEDCURVE)
            (roleset SemiMinorAxis (vr NUMBERP))
            (roleset SemiMajorAxis (vr NUMBERP))
            (roleset Inclination (vr NUMBERP)))

    (concept CIRCLE (supercs ELLIPSE)
            (roleset Radius (vr NUMBERP)))

    (concept DISPLAYCOMPOSITE (supercs DISPLAYFORM)
            (roleset Component (modality Obligatory)
                     (number (0 NIL))
                     (vr DISPLAYFORM))          (* The Argument role
                                                for the old Display
                                                Composite has been
                                                eliminated here.)
            (roleset CoordSys (vr CARTESIANSYSTEM))

    (* The coordinate system by which the locations of
    the component Display Forms are interpreted.)
            )

    (concept POLYGON (supercs DISPLAYCOMPOSITE CLOSEDCURVE)
```

```
            (roleset Order (vr NUMBERP))
            (roleset Side (mods Component@DISPLAYCOMPOSITE)
                      (number (3 NIL))
                      (vr LINE))

    (* Note that there is a possible redundancy in how
    the positions of the vertices are expressed.
    There may be an Approximation Edgeset, or the
    positions may be roles of the Sides.)
            )

    (concept REGULARPOLYGON (supercs POLYGON)
            (roleset Center (mods Location@DISPLAYITEM)
                      (vr LISTP))
            (roleset EdgeLength (vr NUMBERP)))

    (concept RECTANGLE (supercs POLYGON)
            (roleset NIL (mods Order@POLYGON)
                      (vr 4))
            (roleset NIL (mods Side@POLYGON)
                      (number 4))
            (roleset LowerLeft (vr LISTP))
            (roleset UpperRight (vr LISTP)))

    (concept REGION

    (* A Multi-purpose Concept.
    ITags on this concept should refer to functions
    that can decide whether a position is "inside" the
    region and can compute the distance to a boundary
    of the region from a given position.)


            (roleset Continuum (modality Obligatory)
                      (vr CARTESIANSYSTEM))
                                        (* The Continuum is
                                        the coordinate
                                        system over which
                                        the region is
                                        defined.)
            (roleset Boundary (modality Obligatory)
                      (vr CLOSEDCURVE))     (* A Region never
                                        has more than a
                                        single boundary.)
            (roleset Aperture (number (0 NIL))
                      (vr REGION))          (* A Region may have
```

```
                                                    holes in it.)
            )

    (concept RECTANGULARREGION (supercs REGION))
             (roleset NIL (mods Boundary@REGION)
                      (vr RECTANGLE))
             (roleset NIL (mods Continuum@REGION)
                      (vr 2DCOORDINATESYSTEM))
             (roleset NIL (mods Aperture@REGION)
                      (number 0))
             (roleset LowerLeft (vr LISTP))
             (roleset UpperRight (vr LISTP))

    (* LowerLeft and UpperRight are the positions of
    the corresponding corners of the region in the
    Continuum coordinate system.)
            ])

(InitGeographyModel
  [LAMBDA NIL                                       (* Edited by Zdybel
                                                    on 29-Feb-80.)

    (* This initialization function grows the
    descriptions for Geography Model.
    This portion of the Domain Model is generally
    necessary for complete description of a certain
    class of Maps which include outlines of
    land masses, bodies of water, etc.)

    (concept NATION

    (* One of the top level concepts of the Geography
    Model. Further expansion of this Concept will make
    possible specified depiction of national attributes
    such as Capital, Seaport, IndustrialCenter,
    MilitaryInstallation, etc. The Geography model is
    actually a small Domain World in and of itself.)


             (roleset Name (vr STRINGP))
             (roleset Territory (vr GEOREGION)))

    (concept GEOREGION

    (* A region on surface of the earth that is
    modelled as having a bounded area.
```

```
This might describe cites, for example, but not
villages. It might describe some rivers, but not
others.)

        (supercs REGION GEOFEATURE)
        (roleset Name (vr STRINGP))
        (roleset NIL (mods Continuum@REGION)
                (vr (iconcept WORLDCARTESIAN of
                        CARTESIANSYSTEM))))

(* Notice that we are assuming for the moment a
single world coordinate system that is not
spherical.)

        (roleset Feature (number (0 NIL))
                (vr GEOFEATURE)

(* Notice that a feature may also be a region, so
this allows us to cope with islands in the sea,
cities on the land, etc.)
                ))

(concept WATERBODY                          (* One of two
                                            general classes of
                                            Geo Regions ...)
        (supercs GEOREGION GEOFEATURE)
        (subcs OCEAN SEA LAKE)
        (roleset Land (diffs Feature@GEOREGION)
                (vr LANDMASS)               (* The Other of the
                                            two general classes
                                            ...)
                )
        (roleset NIL (diffs Feature@GEOREGION)
                (vr WATERFEATURE))

(* Note that a Water Body is allowed to contain
land masses but not land features.
Thus a city on an island in the sea is not
necessarily retrieved as a Feature of the sea.)
        )

(concept LANDMASS (supercs GEOREGION GEOFEATURE))
        (subcs CONTINENT ISLAND CITY)
        (roleset Water (diffs Feature@GEOREGION)
```

```
                    (vr WATERBODY))
               (roleset NIL (diffs Feature@GEOREGION)
                    (vr LANDFEATURE)))))

     (concept WATERFEATURE (supercs GEOFEATURE)
               (subcs REEF SEA STRAIT CHANNEL GULF))

     (concept LANDFEATURE (supercs GEOFEATURE)
               (subcs RIVER MOUNTAIN CITY PENINSULA ISTHMUS])
)
```

REFERENCES

Brachman, R.J., Ciccarelli, E., Greenfeld, N.R., and Yonke, M.D. KLONE reference manual. BBN Report No. 3848, Bolt Beranek and Newman Inc., July 1978.

Greenfeld, N.R., Zdybel, F., Ciccarelli, E., and Yonke, M.D. BMG Reference Manual. BBN Report No. 4368, Bolt Beranek and Newman Inc., April 1980.

Moore, R.C., Haas, N., Konolige, K., Robinson, A.E., Sacerdoti, E., and Sagalowicz, D. Mechanical intelligence: research and applications, Final Technical Report. Technical Report Project 6891, SRI International, August 1978.

Teitelman, W. A Display Oriented Programmer's Assistant, pages 905-915. IJCAI, Cambridge, MA, 1977.

Teitelman, W., et al. INTERLISP Reference Manual. Revised October 1978 edition, Xerox Palo Alto Research Center, Palo Alto, CA, 1978.

Woods, W.A. Research in natural language understanding, Quarterly Technical Progress Report No. 2. BBN Report No. 3797, Bolt Beranek and Newman Inc., March 1978.

Yonke, M.D., and Greenfeld, N.R. AIPS: An Information Presentation System for Decision Makers, pages 48-56. the University of Hawaii and the Association for Computing Machinery, January, 1980. A revised version of this paper is also available in BBN Report No. 4228, Bolt Beranek and Newman Inc., December 1979.

Zdybel, F., Yonke, M.D., and Greenfeld, N.R. Application of real-time symbolic processing to command and control, Final Technical Report. BBN Report No. 3849, Bolt Beranek and Newman Inc., February 1980.

INDEX