IDE (Integrated Development Environment)

Paula S. Newman
Los Angeles Scientific Center
April 1983

DISCUSSION ORGANIZATION

PROJECT MOTIVATION

DESIGN OBJECTIVES

DESIGN OVERVIEW (partial)

RATIONALE OVERVIEW

THE LEVEL AND FRAGMENTATION PROBLEMS OF IBM DEVELOPMENT ENVIRONMENTS MAKE THEM

- DIFFICULT TO LEARN
- DIFFICULT TO USE
- PONDEROUS

A WELL KNOWN PROBLEM, YET INSUFFICIENT PROGRESS

MANY EFFORTS FOCUS ON

- TOOL IMPROVEMENTS
- SURFACE UNIFICATION

FOR REAL PROGRESS NEED FUNDAMENTAL WORK IN ALTERNATIVE SYSTEM STRUCTURES

- MANY SMALL AD-TECH PROJECTS

IDE: ONE SUCH PROJECT

ONCE AGAIN - THE FRAGMENTATION PROBLEM

DEVELOPMENT OF SINGLE APPLICATION OFTEN REQUIRES

- PROGRAMMING LANGUAGE
- MULTIPLE CONTROL LANGUAGES
  (development and productions systems and subsystems)
- APPLICATION GENERATOR LANGUAGE
- MULTIPLE DATA MANIPULATION LANGUAGES
  (for local data, files, data base, dictionary, other repos)
- MULTIPLE DATA DEFINITION LANGUAGES
  (for local data, files, data base, dict extend, ...)
- DATA UTILITY INTERFACES
- LINKAGE EDIT SPECIFICATIONS
- (+ SPECIFICATION, DESIGN, DEBUGGING, ... LANGUAGES)

MOST LANGUAGES

- HAVE UNIQUE CONCEPTS & SYNTAX
- REQUIRE MORE THAN ONE MANUAL

EFFECTS

- APPLICATION COST: COST OF SKILLS
- APPLICATION EFFECTIVENESS:
  - DOCUMENTATION COMPLEXITY (Limits Reviewers)
  - NOT FEASIBLE TO PROTOTYPE

ALTERNATIVES? SOLUTIONS?

APPLICATION PRODUCTIVITY TOOLS?
(E.G. generators, skeletons, ...)

REDUCE SOME COST
(manual work, reqd knowledge for some)

BUT SOMEONE MUST UNDERSTAND WHOLE

GENERATORS MISUSED AS SYSTEM COVER
===> as complex as general purpose languages
===> don't coexist well with other components

TOOLS ADD COMPLEXITY TO WHOLE

SUPERIMPOSED INTEGRATION?

COMMON REPOSITORY
Yet another data model
Yet another catalog mechanism

COMMON "SCREEN GENERATORS"
Possibly an area where don't want uniformity

===> DON'T ADDRESS BASIC PROBLEM -
DISJOINTED, INCONSISTENT ENVIRONMENT STRUCTURE

===> AND ADD STILL MORE COMPLEXITY

WHAT CAN BE DONE?
START BY LOOKING AT SOURCES OF FRAGMENTATION

ASSUMED DP PATTERN

N application components ===>
N sets of component support mechanisms

FACILITY ACCRETIONS

ADDED support mechanisms AND patterns

EARLY BATCH ENVIRONMENTS

APPLICATION = CTL + PGMS + DATA

CTL:	DEF(cmds)	BIND (context)	ACCESS(implicit)
PGM:	DEF(hll)	BIND(linkedit)	ACCESS(call)
DATA:	DEF(cmds, dcl)	BIND(dd..)	ACCESS (hll stmt, i/o stmt)

THEN ADDED

LANGUAGES TO ISSUE COMMANDS

TIMESHARING

DATA BASE

APPLICATION GENERATORS

DICTIONARIES

REQUIREMENTS AND DESIGN LANGUAGES

DISTRIBUTED PROCESSING

OTHER ASYNCHRONOUS

PARTIAL RESULT

DEFINITION MECHANISMS

FOR PGMS	PGMMING LANG
FOR CMDS	CMD LANG, EXEC LANG
FOR DATA	DDL's - ONE PER CATEGORY (Local data, files, dictionary extension, each kind of db.)

LINKING MECHANISMS

FOR PROGRAMS	LINKEDIT
FOR DATA	CATEGORY BASED (DD cards, PCB's)

ACCESSING MECHANISMS

FOR PGMS	CALL, SVC
FOR CMDS	EXEC, SVC
FOR DATA	DML's - ONE PER CATEGORY UTILITIES

INTEGRITY/RECOVERY MECHANISMS

FOR PGMS	ON CONDITIONS ....
FOR DATA	TRANSACTIONS ....

DIRECTORY MECHANISMS

LIBRARIES
CATALOG
DICTIONARY
CONTROL BLOCKS

OBSERVATIONS

FRAGMENTATION IS RESULT OF PROGRESS - NO SOLUTION

CAN LIMIT EFFECTS BY PERIODIC INTRODUCTION OF NEW DESIGNS

USE IN

- New applications
- Major revisions
- Prototyping

DESIGN CRITERIA

- Coherent subsumption of current accretions
- Non fragmenting pattern
- High level facilities

TO DEVELOP SUCH DESIGNS NEED STUDY PROJECTS

- IDE is one such study

IDE - CURRENT STATUS

STATUS

FOCUS ON SINGLE USER ENVIRONMENT

CONSIDERABLE SPECIFICATION

VERY LIMITED DEVELOPMENT (formal grammar, some design)

FEATURES

ONE ASSOCIATIVE DATA MODEL FOR ALL DATA
(local, file, database, directory, design specs)

ONE FULL-SCALE VHLL FOR ALL "PROCEDURAL" PURPOSES
directory access, design, implementation, command

SMOOTH INTEGRATION OF DECLARATIVE FORMS
for application generation, database definition

OBJECT-ORIENTED ENVIRONMENT - UNIFORM MECHANISMS FOR
definition, compilation, linking, cataloging, accessing

DIRECTORY AS CENTRAL FOCUS OF APPLICATION DEVELOPMENT
subsumes catalog, dictionary, binding, control

APPLICABILITY

FEASIBILITY DEMONSTRATION

SOURCE OF SEPARABLE IDEAS

WITH (relatively) SMALL DEVELOPMENT EFFORT

- PROTOTYPING VEHICLE
- SINGLE-USER ENVIRONMENT

WITH MAJOR DEVELOPMENT EFFORT

- FULL ENVIRONMENT

FUNCTIONAL OBJECTIVES

ENVIRONMENT WITH

- NON-FRAGMENTING BASIC PATTERN
- NECESSARY ACCRETED FACILITIES

  WITH their convenient aspects
  WITHOUT associated fragmentation

COMMAND LANGUAGE

WITH ad-hoc aspects
WITHOUT HLL/CMD LANGUAGE split

DATABASE

WITH declarative def, query-like access, atomicity
WITHOUT HLL / DDL / DML split
WITHOUT SYSTEM/SUBSYSTEM split

APPLICATION GENERATION

WITH declarative def
WITHOUT uneasy coexistence with HLL

DICTIONARY

WITH accessibility of information
WITHOUT dictionary/catalog split

REQUIREMENTS AND DESIGN LANGUAGES

WITH function
WITHOUT separate repositories, views of application

ASYNCHRONISM WITHIN/AMONG APPLICATIONS

WITHOUT separate provisions for tasking, distribution, DB

SOURCES

ASSOCIATIVE DATA BASE MODELS (REL, E-R, ..)

Need data base in environment
Models can subsume others

VERY HIGH LEVEL LANGUAGES (SETL)

Raise environment level
Local data associative --> unification with DB

OBJECT-ORIENTED SYSTEMS (SMALLTALK ..

Provide non-fragmented system pattern
Better base for production systems:

distributed applications
control systems

APPLICATION GENERATION

Hi-level spec of generalizable processing

MODULE INTERCONNECTION LANGUAGES

Substitute for low level linkedit,
Link to design levels

"The time appears to be right for the integration of languages, operating systems, and database research on object models."

Peter Wegner, 1982

OBJECT - ORIENTED ENVIRONMENT

BASED ON PROGRAMMING LANGUAGE IDEA OF ABSTRACTION

(Simula, CLU, ...

PROGRAMMING LANGUAGE ABSTRACTIONS

DEFINED BY MODULE DEFINITION

SPECIFY ONE OR MORE OPERATIONS + IMPLEMENTATION

DEFINITION MAY HAVE MANY INSTANCES

ACCESSED AS NAME.OP

STACK --> STK1, STK2

STK1.PUSH(), STK2.POP

OBJECT-ORIENTED ENVIRONMENT

USES ABSTRACTION AS ORGANIZING PRINCIPLE

CONTAIN ONLY ABSTRACTIONS = OBJECTS

DESIGN OBJECTIVES

USE OBJECT ORIENTATION AS A BASE

One application component, one set of support mechanisms

DESIGN:

ABSTRACTION LANGUAGE (approximation)

Associative local data model
Includes asynchronous capabilities
Version for interactive commands

DECLARATIVE DEFINITION FORMS

For data base, application generation
With close ties to procedural form

DICTIONARY/CATALOG FACILITIES

Subsuming current functions
Subsuming MIL function

DATABASE-ORIENTED ATOMICITY FACILITIES

DEVELOPMENT OF DESIGN

REMOVAL OF HLL DATA ACCESS FRAGMENTATION

LOCAL DATA, FILE, DATA BASE, UTILITY

INCORPORATE RESULTS IN ABSTRACTION LANGUAGE

UNIFY PROGRAM, DATA ACCESS

UNIFY PROCEDURAL DEFINITION
WITH DECLARATIVE DEFINITION (FOR DB, APGEN)

------ (From here directional) -------

OUTLINE SINGLE-USER / SINGLE-THREAD ENVIRONMENT

DIRECTORY = CATALOG + DICT + MIL + DESIGN REPOSITORY

LANGUAGE EXTENSIONS FOR CMDS/EXECS

EXTEND FOR MULTI-THREAD PROCESSING

COHERENT INTER-OBJECT COMMUNICATION (SYNCH, ASYNCH)

COHERENT ERROR-HANDLING & RECOVERY

------ (From here little done) -------

EXTEND TO MULTI-USER ENVIRONMENT

EXTENDED NAMING, CATALOGING, DISTRIBUTION, ...

HLL DATA ACCESS FRAGMENTATION

LOCAL REFS + I/O STMTS + DB DML + UTILITIES

TO REMOVE: STEPS

1. MODEL DEVICES AS PROGRAMS (common)

2. USE SINGLE ASSOCIATIVE MODEL FOR FILES AND DATABASES

? ADD DATA MODEL TO LOCAL TYPES, EXTERNAL REF BY NAMEQUAL

3. USE SINGLE ASSOCIATIVE MODEL FOR ALL DATA

   Sufficiently expressive for DB

   Supports programming structures (VAR, ARRAY)

   Accessible in programming language style (NOT "UPDATE")

IDE DATA MODEL

FROM DB (Relational, Functional) + SETL MODELS

A DATA COLLECTION IS A COLLECTION OF SETS.

A SET CAN BE

single-member	or	multi-member
of scalars	or	of tuples
constant	or	variable.

SCALAR SET CONSTRAINTS

"BASE SET" (integer, real, string, objptr, userdefined)
RANGES
ENUMERATIONS
.....

TUPLE SET CONSTRAINTS

SOURCE SETS FOR EACH POSITION (can be tuple sets)
DEPENDENCIES (M-1, M-N, ..) (for degree 2)

CAN ALSO DECLARE SUBSETS.

Tuple position sources
Basis of "Subset" Relationships

EXAMPLE - A CAVE ADVENTURE

Initial Screen

```
---------------------------------------------------------------------
 YOU ARE STANDING AT THE ENTRANCE TO A SYSTEM OF CAVES.

 FIND THE TREASURE & BRING IT BACK TO THE ENTRANCE.

 IN MOVING FROM CAVE TO CAVE, YOU WILL MEET OBSTACLES.

 TO OVERCOME OBSTACLES USE OBJECTS WHICH YOU WILL FIND.

 YOU CAN CARRY 30 POUNDS OF OBJECTS AT A TIME.

 TO GO INTO THE CAVES, PRESS ENTER
---------------------------------------------------------------------
```

Typical Screen

```
---------------------------------------------------------------------

 YOU ARE IN THE ANTEROOM.  THERE ARE EXITS TO THE
 NORTH AND WEST.  YOU CAN SEE A HATCHET AND A KEY.

 WHAT DO YOU WANT TO DO
 TELL ME BY SELECTING ONE WORD FROM EACH COLUMN

---------------------------------------------------------------------

GO        x                    NORTH
GET                            SOUTH
DROP                           EAST     x
....                           WEST
                               SPADE
                               BOOTS
                               MAGIC WAND
                               ....

---------------------------------------------------------------------

 SORRY, YOU CAN'T GO THAT WAY ---  TRY AGAIN

---------------------------------------------------------------------
```

DATA FOR CAVE ADVENTURE DESCRIPTION

Place	Direction	Object	Obstacle
'Big Cave'	'North'	'Treasure'	'LockedDoor'
'Water Cave'	'East'	'Key'	'Tunnel'
.....	....	.....	.....

Move (Place, Direction) <'Big Cave', 'North'>, ..

MoveRslt (Move, Place) <<'Big Cave','North'>,'Tunnel'>, ..

MoveObs (Move,Obstacle) <<'Big Cave','North'>,'Size'>, ..

NeedObj (Obstacle, Object) <'Size', 'Shrinking Potion'>, ..

ObjectWt (Object,Integer) <'Treasure', 10>, ..

Content (Place, Object) <'Secret Cave', 'Treasure'>, ..

LOAD CAVE DATA BASE - UTILITY EQUIVALENT

Move (Place, Direction) MoveResult (Move, Place)

MoveObs (Move,Obstacle) NeedObj(Obstacle, Object)

Content (Place, Object) ObjWt (Object, Integer)

```
Cave.[ ] +=
    [Move <'BigCave', 'South'>
                [MoveRslt 'Tunnel' MoveObs 'Size'],
            <'LastCave', 'North'>
                [MoveRslt 'Plateau' ]
            ......
```

OR

```
Cave.[  ] +=
    [*    Move          [MoveResult    MoveObs]//
        <'BigCave', 'South>  ['Tunnel'    'Size'  ],
        <'LastCave', 'North'> ['Plateau'],
        ......                                  *]
```

NOW EXAMINE, MODIFY GAME DESCRIPTION

Move (Place, Direction) MoveResult (Move, Place)

MoveObs (Move,Obstacle) NeedObj(Obstacle, Object)

Content (Place, Object) ObjWt (Object, Integer)

```
Cave.Content

Cave.Content ('Big Cave')

Cave.Content (?, 'Shrinking Potion')

{?o where Cave.ObjWt(?o) gt 10}

{?loc where Cave.Move(?loc) eq Cave.Direction}

Cave.Object -= Cave.Content ('Passage');

Cave.Content ('StepCave') +=   {* 'Wrench', 'Lamp' *};

Cave.ObjWt('Hammer') = 3 * Cave.ObjWt('MagicWand')

For <?p1, ?d, ?p2>
        where <<?p1, ?d>, ?p2> in Cave.MoveResult
        and     <<?p2, Opposite(?d)>, ?p1> notin Cave.MoveResult;
        Print ('Error from '|| ?p1 ||' going ' || ?d); End;
```

WHY THIS MODEL

GOOD DB MODEL: EASE OF ACCESS, SEMANTIC CONTENT

IN BASIC RELATIONAL MODEL,
COLUMN NAMES = DOMAINS or ASSOCIATIONS

"person | father | mother"

HERE DOMAINS, ASSOCIATIONS SEPARATE
DEFINITIONS EASIER TO UNDERSTAND

NETWORK FLAVOR ---> GOOD OPERATIONAL SEMANTICS

Add to subset ==> add to superset
Delete from set ==> Delete associations

(RELATIONAL WORK IN THIS DIRECTION

"inclusion dependencies", emphasis on domains,
hierarchic subcollections (Codd, Date, Lorie, ..)

CAN MODEL HLL STRUCTURES

HLL VARIABLE: variable, single-mbr, scalar set

X = Y + Z OK

HLL RECORD: Set mbr + some of its associations

HLL ARRAY: Binary Relation between S1 and S2

S1 - constant tuple set -
cross product of index domains
S2 - domain of array values

A (<I,J>) = 6 OK

INCORPORATE RESULTS IN ABSTRACTION-ORIENTED LANGUAGE

OBJECTIVES

UNIFY DATA AND PROGRAM REFERENCE

UNIFY PROCEDURAL AND DECLARATIVE (DB, APGEN) DEF

SUBJECTS:

ABSTRACTION REFERENCE PATTERNS: GENERAL

REFERENCE IMPLICATIONS OF "Cave.ObjWt"

DECLARATIVE AND PROCEDURAL ABSTRACTION DEFS

DEFINITION IMPLICATIONS OF "Cave.ObjWt"

ABSTRACTION REFERENCE PATTERNS: GENERAL

BASIC REQUEST STRUCTURE

```
Req object.entry (arg list);
    On exit1 (arg list) statement;
    On exit2 (arg list) statement;
    ....
    End;
```

```
Req Stack1.Pop;
    On Empty Do;.......End;
    On Ok (Top-Of-Stack) Do; .....  End;
    End;
```

READABLE CODE,
MODIFIED ARGUMENTS EXPLICIT (*)

REMAINING FORMS SHORTHAND

FOR SINGLE NORMAL EXIT ("call")

```
Object.Entry (entry args//return-args);
```

FOR SINGLE NORMAL EXIT + SINGLE EXIT ARG ("fn call")

```
Object.Op (entry args)
```

FOR CREATE + REQUEST + DESTROY

```
Object%Op (entry args)
```

OBJECT CREATION

DIR.LOCAL (Defptr // Objptr)

REFERENCE IMPLICATIONS OF "Cave.ObjWt"

1. REQUIRE

```
Temp = Cave.Objwt;
Temp = Cave.ObjWt ('MagicWand')
```

IMPLIES

- ALLOW ARGS TO ENTRIES WHICH DO NOT ACCEPT ARGS
- DO REQUEST, APPLY RETURNED TUPLE-SET TO ARGS*

2. REQUIRE

```
Cave.ObjWt += <'TreasureChest', 15.0>
Cave.ObjWt ('Hammer') = 3* Cave.ObjWt ('MagicWand');
```

IMPLIES

- LEFT-OF-EQUAL REFERENCES ACCEPTED.
- UNDERSTOOD DIFFERENTLY FROM RIGHT-OF-EQUALS

```
A.S += expression    ===>    A.E$S(expression//);
A.S -= expression    ===>    A.D$S(expression//);
A.S =  expression    ===>    A.R$S(exp1,exp2//);
```

THESE CONVENTIONS APPLY TO ALL REFERENCES

LOCAL DATA AND ABSTRACTIONS

EACH MODULE HAS "LOCAL DATA COLLECTION"

VIEW ENTIRE COLLECTION AS ABSTRACTION

- Referenced with invisible pointers

IMPLICATION: NEITHER SCALARS NOR SETS ARE OBJECTS

- Cannot define generic scalar operations
- Other problems

ADVANTAGES

- ALLOWS SET CONSTRUCTION (QUERY) FORMS IN LANGUAGE
- PROVIDES NETWORK SEMANTICS AS PRIMITIVE

MODULE DEFINITION: PROCEDURAL AND DECLARATIVE

DATA DEFINITION LANGUAGES AND GENERATOR INPUTS CURRENTLY

a) INCONSISTENT WITH PROCEDURE DEFINITIONS

b) INCONSISTENT WITH EACH OTHER

COALESCE USING RELATIONSHIPS

DECLARATIVE INFO : DATA COLLECTION
DATA COLLECTION : AGGREGATE EXPRESSION

PROCEDURAL MODULE DEF

MODULE name PROCEDURE.....

DCL;.....
aggregate expression.....
(defining entries, local data).....
END;.....

Main program

Nested module definitions.....

END;.....

DCL MODULE DEF

MODULE name generator

DCL MODULE DEF ---> GENERATOR ---> PROC or DCL MODULE DEF

TWO TYPES OF MODULE DEFINITION

- BUT SYNTACTICALLY, PHYSICALLY RELATED
- EASY COMMUNICATION BETWEEN INSTANCES OF BOTH
- EASY CREATION OF SIMPLE, TAILORED GENERATORS

USE IN CAVE ADVENTURE

Initial Screen

```
---------------------------------------------------------------
 YOU ARE STANDING AT THE ENTRANCE TO A SYSTEM OF CAVES.

 FIND THE TREASURE & BRING IT BACK TO THE ENTRANCE.

 IN MOVING FROM CAVE TO CAVE, YOU WILL MEET OBSTACLES.

 TO OVERCOME OBSTACLES USE OBJECTS WHICH YOU WILL FIND.

 YOU CAN CARRY 30 POUNDS OF OBJECTS AT A TIME.

 TO GO INTO THE CAVES, PRESS ENTER
---------------------------------------------------------------
```

Typical Screen

```
---------------------------------------------------------------

 YOU ARE IN THE ANTEROOM.  THERE ARE EXITS TO THE
 NORTH AND WEST.  YOU CAN SEE A HATCHET AND A KEY.

 WHAT DO YOU WANT TO DO
 TELL ME BY SELECTING ONE WORD FROM EACH COLUMN

---------------------------------------------------------------

GO       x                    NORTH
GET                           SOUTH
DROP                          EAST     x
....                          WEST
                              SPADE
                              BOOTS
                              MAGIC WAND
                              ....

---------------------------------------------------------------

 SORRY, YOU CAN'T GO THAT WAY ---  TRY AGAIN

---------------------------------------------------------------
```

STRUCTURE OF ADVENTURE GAME

CAVEDEF	Declarative Def of Cave Structure Objects
CAVE	Instance of CaveDef
MYSCREEN	Declarative Def of Screen Interface Object
ADVENTURE	Procedural Def of Game Control

ADVENTURE -------> CAVE, MYSCREEN

MYSCREENS: DECLARATIVE DEF OF SCREEN INTERFACE MODULE
(Input to hypothetical, non-built-in generator)

```
Module Myscreens GenScreens to Procedure;
  Dcl;
    Screens 'S1'  [ CtnsWindows 'W1' ],
            'S2'  [ CtnsWindows 'W2', 'W3', 'W4', 'W5']
            .....
    Windows 'W1'   [Wpos < 1, 1>, <24,80>   WClass 'Message'],
            'W2'   [Wpos < 1, 1>, < 5,80>   WClass 'Message'],
            'W3'   [Wpos <12, 1>, <20,40>   WClass 'Menu'],
            'W4'   [Wpos <12,41>, <20,80>   WClass 'Menu'],
            'W5'   [Wpos <21, 1>, <25,80>   WClass 'Message']
  End;
Endmod;
```

COMPILER

PARSES DCL BLOCK
PLACES INFO IN DATAGROUP OBJECT
PASSES TO GENERATOR
ACCEPTS GENERATED MODULE DEFINITION

USE BY

Myscreens%S1 (BgnMsg)
--- requests output of one part screen

Myscreens%S2 (LocMsg, Word1, Word2, EMsg//TWd1, TWd2)
--- requests output of four part screen

CAVEDEF: DECLARATIVE DEF OF DATAGROUP

```
Module CaveDef Datagroup To Procedure;

Dcl;
  Var      'BgnMsg'     [Base     'String']
  Set      {* 'Place', 'Direction', 'Object',
              'Obstacle', 'Word1', 'Word2' *}
                        [Base     'String']
  Rel      'Move'       [Roles <* 'Place', 'Direction' *>],
           'Content'    [Roles <* 'Place', 'Object' *>]
  ARel     'MoveRslt'   [Roles <* 'Move', 'Place'*>],
           'MoveObs'    [Roles <* 'Move', 'Obstacle'*>],
           'NeedObj'    [Roles <* 'Obstacle', 'Object'*>],
           'ObjtWt'     [Roles <* 'Object', 'Integer' *>]
 End;

EndMod;
```

DECLARATION SCHEMA SAME AS FOR LOCAL DATA

GENERATED ENTRIES: Place, E$Place, D$Place, R$Place,.....

ALSO DERIVED DATA,....

DECLARATIVE DEFINITION: DATAGROUP FORM
DERIVED DATA BY USER DEFINED ENTRIES

```
Module CaveDef Datagroup

Dcl;
  Entry
        'ObjKg'    [Returns 'KgRtn' [Args <* 'ObjWt' *>]],
        'R$ObjKg' [Args <* 'ObjWt', 'ObjWt'*> ]
        .......

  Var     'BgnMsg'    [Base    'String']
  Set     {* 'Place', 'Direction', 'Object',
             'Obstacle', 'Word1', 'Word2' *}
                      [Base    'String']
  Rel     'Move'       [Roles <* 'Place', 'Direction' *>],
        'Content'    [Roles <* 'Place', 'Object' *>]
  ARel    'MoveRslt'   [Roles <* 'Move', 'Place'*>],
        'MoveObs'    [Roles <* 'Move', 'Obstacle'*>],
        'NeedObj'    [Roles <* 'Obstacle', 'Object'*>],
        'ObjWt'   [Roles <* 'Object', 'Real' *>]
  End;

  Module ObjKg Internal;
  Ercv;
  Return ({<?o, ?w> where ?w = Cvt(ObjWt(?o)) });
  EndMod;

  Module R$ObjKg Internal;
  .....

EndMod;



Cave.ObjKg ('Hammer') = 3 * Cave.ObjKg ('MagicWand');
```

INTERCHANGE STORED, DERIVED VALUES
WITH NO EFFECT ON ACCESSING PROGRAMS

ADVENTURE CONTROL

```
Module Adventure Procedure;

 Dcl;

   Set    {* 'Object',   'Place' *}  [Base 'String']
   Rel    'Carries'     [Roles <* 'Object', 'Bool' *>],
          'Content'     [Roles <* 'Place', 'Object' *>],
          'Tmove'       [Roles <* 'Place', 'String' *>]
  Const   'Game'        [Base 'ObjPtr'  PtrDef  'CaveDef']
   Var    'TWt'         [Base 'Integer'],
          {* 'TPlace', 'Tneed', 'TObs', 'TrsIt', 'TWd1', 'TWd2', 'EMsg'
                        [Base String]
   End;

 Ercv ();
 Myscreens%S1(Game.BgnMsg);
 TPlace = 'AnteRoom';     TWt = 0;
 Content = Game.Content;
 EMsg = '';
 While True;
    Myscreens%S2(LocMsg,Game.Word1,Game.Word2,EMsg//TWd1,TWd2);
    Case TWd1, TWd2;
     Of < eq 'Go', in Game.Direction >
                Do;   Tmove = <TPlace, TWd2>;
                      Tobs = Game.MoveObs(Tmove);
                      Tneed = Game.NeedObj (Tobs);
                      TrsIt = Game.MoveRsIt(Tmove);
                      Case TrsIt, Tneed, Carries(Tneed);
                         Of <eq Null, * , *>
                            EMsg = 'You can't go in that direction';
                         Of <neq Null, neq Null, False>
                            EMsg = ObsMsg (Tobs, Tneed);
                         Otherwise Do; TPlace = TrsIt;
                            EMsg = ''; End;
                       End;
               End;
      Of < eq 'Take', in Game.Object>
      ......
      Otherwise EMsg = 'That doesn't make sense';
      End;
   End;

Defs of Myscreen, LocMsg, ObsMsg, other
....
Endmod;
```

DEVELOPMENT OF DESIGN

REMOVAL OF HLL DATA ACCESS FRAGMENTATION

LOCAL DATA, FILE, DATA BASE, UTILITY

INCORPORATE RESULTS IN ABSTRACTION LANGUAGE

UNIFY PROGRAM, DATA ACCESS

UNIFY PROCEDURAL DEFINITION
WITH DECLARATIVE DEFINITION (FOR DB, APGEN)

------ (From here sketch only) -------

OUTLINE SINGLE-USER / SINGLE-THREAD ENVIRONMENT

DIRECTORY = CATALOG + DICT + MIL + DESIGN REPOSITORY

LANGUAGE EXTENSIONS FOR CMDS/EXECS

EXTEND FOR MULTI-THREAD PROCESSING

COHERENT INTER-OBJECT COMMUNICATION (SYNCH, ASYNCH)

COHERENT ERROR-HANDLING & RECOVERY

------ (From here little done) ------

EXTEND TO MULTI-USER ENVIRONMENT (little work done here)

EXTENDED NAMING, CATALOGING, DISTRIBUTION, ...

OUTLINE SINGLE-USER / SINGLE-THREAD ENVIRONMENT

OBJECTIVES

DIRECTORY INTEGRATING FUNCTIONS OF

CATALOG, MIL, DICTIONARY/REPOSITORY

ONLINE CMDS & EXEC EQUIVALENTS

BY LANGUAGE EXTENSION

DIRECTORY: DESIRED FUNCTIONS

CATALOG

- OBJECT CREATION AND NAMING

BASIC OBJECT PREPARATION SUPPORT

- SEPARATE COMPILATION
- BINDING

STRUCTURED APPLICATION DEVELOPMENT SUPPORT (MIL)

- SEPARATE INTERFACE / EXTREF SPECIFICATION
- PARALLEL COMPONENT DEVELOPMENT

DICTIONARY/REPOSITORY

- ACCESSIBLE IMPLEMENTATION LEVEL DEFS
- HIGHER-LEVEL DESCRIPTIVE / EXECUTABLE DEFS
- GROUPING / VERSIONING OF DESCRIPTIONS

DIRECTORY: BASIC CATALOG FUNCTION

READ OPERATIONS - AS FOR DATAGROUP OBJECT

OBJECT names of persistent objects

TYPE (OBJECT, OBJECT) links objects to their defs

CLASS (OBJECT, "procedure" | "process" | ....)

REF (OBJECT, STRING) external refs

BIND (REF, OBJECT) binding of external refs

STATE (OBJECT, string)

DESC (OBJECT, string) arbitrary text*

RELOBJ (OBJECT, OBJECT) related objects*

RELDESC (RELOBJ, string) explains relationships*

UPDATE OPERATIONS - INFORMATION GROUPED INTO "ENTRIES"

SOME UPDATE OPERATIONS

```
Dir.Entry  ===>  create directory entry
Dir.Ready  ===>  create represented object
Dir.Ext    ===>  create entry and object
```

BASIC OBJECT PREPARATION: LINKAGE REQUIREMENTS

```
Module Adventure Procedure;

 Dcl;

    Set    {* 'Object',   'Place' *}  [Base 'String']
    Rel    'Carries'     [Roles <* 'Object', 'Bool' *>],
           'Content'     [Roles <* 'Place', 'Object' *>],
           'Tmove'       [Roles <* 'Place', 'String' *>
    Const   'Game'       [Base 'ObjPtr'  PtrDef  'CaveDef']
    Var     'TWt'       [Base 'Integer'],
           {* 'TPlace', 'Tneed', 'TObs', 'TrsIt', 'TWd1', 'TWd2', 'EMsg' *}

                        [Base String]
    End;

    ....
  End;
```

- "Game.Content" = DEREFERENCE LOCAL PTR 'Game'
- "Const 'Game' [Base 'ObjPtr' PtrDef 'CaveDef'] = DCL OF 'Game'
- PTRS (usually) "TYPED" BY SYMBOLIC DEF NAME (CaveDef)
- BIND SYMBOLIC DEF NAMES BEFORE COMPILATION
- BIND CONSTANT POINTERS BEFORE EXECUTION

BASIC OBJECT PREPARATION: PRIMITIVE STEPS

1. CREATE SOURCE CODE

   Create object of type "text" and load (with editor)

2. PREPARE FOR COMPILATION

   Create directory entry for object of type "Def"

   Include:
   - Name of source code object
   - Binding of Definition References
   - ....

3. COMPILE (Reference "Def" entry for parameters)

   Results:
   - Represented Object = Datagroup with interface and extref information
   - Hidden object with compiled code

4. CREATE INSTANCE

   Create Dir Entry With:
   - Identification of Def
   - Binding of remaining extrefs

   Create Represented Object

ALLOWS SEPARATE COMPILATION, INTERFACE CHECKING

BASIC OBJECT PREPARATION - EXAMPLE

DIRECTORY CONTENT ADDITIONS, BY STEP

	Object	Type	Class	State	LocSrc	DefRef	DefBind
1.	CaveSrc	'Text'	'Proced'	'Ready'			
	AdvSrc	'Text'	'Proced'	'Ready'			
2.	CaveDef	'Def'	'Proced'	'Ready'	'CaveSrc'		
	AdvDef	'Def'	'Proced'	'Ready'	'AdvSrc'	'CaveDef'	<,'CavDef'>
						Ref	Bind
3.	'Cave'	'CaveDef'	'Proced'	'Ready'			
	'AdvCtl'	'AdvDef'	'Proced'	'Ready'	'Game'		<,'Cave'>

1. CREATE TEXT OBJECTS

```
Dir.Ext ('CaveSrc', 'Text');   Edit ('CaveSrc')
Dir.Ext ('AdvSrc',  'Text');   Edit ('AdvSrc')
```

2. CREATE DEF OBJECTS FOR COMPILER OUTPUT

```
Dir.Entry ('CaveDef', [Type 'Def' Class 'Proced'
          LocSrc 'CaveSrc' ]])

Dir.Entry ('AdvDef', [ Type 'Def' Class 'Proced'
         LocSrc 'AdvSrc' DefRef 'CaveDef' [DefBind 'Cavedef']])

Compile ('CaveDef')                    Compile ('AdvDef')
```

3. CREATE OBJECTS

```
Dir.Ext ('Cave', 'CaveDef'// Ptr1)

Dir.Ext ('AdvCtl', [ Type 'AdvDef' Class 'Proced'
                          Ref 'Game' [Bind 'Cave']])

Dir.Ready ('AdvCtl')
```

STRUCTURED APPLICATION PREPARATION

MIL FACILITIES FOR MORE FORMAL DEVELOPMENT

PARALLEL DEVELOPMENT / COMPILATION OF COMPONENTS
APPLICATION STRUCTURE DESCRIPTION

ADDITIONAL BUILT-IN OBJECT TYPES

INTFDEF: REPRESENTS COMPILED INTERFACE DEFINITION
SPECIFIES INTERFACE SOURCE,
OPTIONAL EXTREFS AND BINDINGS

GROUP: REPRESENTS OBJECT GROUP (e.g. application)
SPECIFIES COMPONENTS AND FUNCTIONS

E.G. SPECIFY ADVENTURE

```
Dir.Entry ('Adventure',
            [Type 'Group'
             Desc 'Structure of Cave Adventure'
             Ctns 'AdvIntf' [Fun 'Interface to ctl'],
                  'AdvDef'  [Fun 'Impl of ctl'],
                  'CaveDef' [Fun 'Cave Schema' ]]

Dir.Entry ('AdvIntf', [ Type 'IntfDef'  DefSrc 'AdvIntfSrc'
                        IntfRef 'Cave' ]

Dir.Entry ('CaveDef', [Type 'Def' Class 'Proced'
            LocSrc 'CaveSrc' ]])

Compile ('AdvIntf');  Compile ('CaveDef');
```

THEN IMPLEMENT

```
Dir.Entry('AdvDef', [Type 'Def' LocSrc 'AdvSrc' LocIntf 'AdvIntf'
                DefRef 'Game' [DefBind 'Cavedef']])

Compile ('AdvDef')

Dir.Ext ('Cave', 'CaveDef')

Dir.Entry ('AdvCtl',...);     Dir.Ready ('AdvCtl');
```

LIFECYCLE SUPPORT - EXAMPLE

BEGIN

CREATE GROUP "CAVEAPP" WITH DESCRIPTION

DESIGN

CREATE OBJECT "CAVEDSN1" ; PSL TYPE SCHEMA (e.g.)

INDEX IN "CAVEAPP", FUNCTION = DESIGN

ALTERNATIVE DESIGN

CREATE OBJECT "CAVEDSN2" ; VERSION OF CAVEDSN1

INDEX IN "CAVEAPP", FUNCTION = DESIGN

PROTOTYPE

CREATE GROUP "CAVEPROT";

INDEX IN "CAVEAPP", FUNCTION = PROTOTYPE

INCLUDE ALL RELATED MODULES.

```
CAVEAPP  [CTNS
                CAVEDSN1
                CAVEDSN2,
                CAVEPROT [CTNS
                                CAVESRC,...
```

PROVIDE FOR INTERACTIVE CMDS, EXECS, OTHER NON-COMPILED

REQUIREMENTS

FOR INTERACTIVE LANGUAGE (COMMANDS, QUERIES)

NO DECLARATIONS (---> UNTYPED LOCAL SETS)
PLACEMENT IN OBJECT FRAMEWORK

FOR 'EXEC' EQUIVALENTS

COMMAND GENERATION
UNTYPED SETS
.....

NON-COMPILED GENERIC FACILITIES

DISPLAY (Ptr) ===>

"DISPLAY" ACCESSES DEF(ptr) FOR SET NAMES

+.....

APPROACH

ONLINE OBJECTS (Interactive Language)

INTERPRETED OBJECTS (Execs, Other non-compiled)

INTERACTIVE LANGUAGE (COMMANDS, QUERIES, ...

ONLINE OBJECT

OBJECT WHOSE MAIN PROGRAM FROM TERMINAL

ONE BUILT-IN - NO DECLARATIONS OR NESTED MODULES

(POSSIBILITY OF OTHERS - ACCESS BY "ENTER"/"LEAVE")

LANGUAGE VARIATIONS

SHOW expression(Query Language)

UNDECLARED LOCAL SETS (.A = .B)

SYMBOL RESOLUTION VARIATIONS

TRANSACTION SPECIFICATION VARIATIONS (further on)

.....

EXECS, OTHER NON-COMPILED

INTERPRETED OBJECTS

INSTANCES HAVE TEXT OBJECTS AS TYPES

FEWER DECLARATIVE REQUIREMENTS

LANGUAGE VARIATIONS

UNTYPED LOCAL SETS ALLOWED (.A = .B)

UNTYPED POINTERS CAN BE DEREFERENCED

```
P = Dir.Ptr('objname')          Rcv; On A (P)....; End;

                ....

            Q = Dir.Type (P)

                ....

              = P.(exp referencing Q) (args)
```

COMMAND GENERATION

EXECUTE string;

EXECOBJ objectname (type TEXT, local or external)

(= DO; statements of referenced object END;)

PROBLEM: REFERENCES FROM COMPILED OBJECTS
(SEMANTICS OF LITERAL FUNCTION REFERENCES,....)

(PARTIALLY RESOLVED BY REQUIRING
INTERFACE OBJECTS IN SOME CASES)

EXTEND FOR ASYNCHRONOUS OPERATION

OBJECTIVE - COHERENT SUPPORT FOR VARIOUS INTER-OBJECT RELS

- LOCAL ABSTRACTION (as before)
- CO-ROUTINE
- TASKING
- SERIALLY ACCESSIBLE DATA BASE
- CONCURRENTLY ACCESSIBLE DATA BASE
- OTHER CLIENT / SERVER RELATIONSHIPS
- COOPERATIVE ASYNCHRONOUS - SIMULATION

OBJECTIVES: CONSISTENCY IN

- COMMUNICATION MECHANISMS (SYNCH/ASYNCH)
- ERROR-HANDLING & RECOVERY (INTRA/INTER OBJECT)

ASYNCHRONOUS EXECUTION: COMMUNICATION

P1

```
REQ P3.R1(args);
  ON X1 (args) ...
  ON X2 (args) ...
  END;

REQ P4.R1 (args)
  ON X1 (args)
  ON X2 (args)
  ON NOTFOUND
  END;
```

PROCEDURE P3

```
SRCV;
  ON R1 (args) ...
  ON R2 (args)...
  END;

RETURN X1 (args) REENTER..
```

P2

```
SENDR P3.R1 SET A1;

SENDR P4.R1(args) SET A2;

RCV
  ONF DefP3.X1 (args)
      WHERE $RESP(?i) = A1
      .....
  ONF DEFP3.X2 (args) ....
  ON  NOTFOUND
  END;
```

PROCESS P4

```
RCV ?i
 ONR R1 (args) guard DO;
     REQ = $REQ(?i);..
       .....
 END;

RESPOND X2 (args) FOR REQ;
```

- PROCESS = 'MODULE xxxx PROCESS'
- PROCEDURES "LOCAL" OR "SHARED", PROCESSES SHARED
- SHARED PROCESS MANAGES OWN Q, RESPONDS OUT OF SEQ
- GUARD: PREDICATE IN USER DATA, BUILT-IN LOCAL DATA
  Q CONTENT - ORDER, SENDER, MTYPE, REQID,..
  SYSTEM VAR - TIME, DATE, ...
- DISTINGUISH REQUEST/RESPONSE PAIRS FROM UNLINKED MSGS

ERROR HANDLING AND RECOVERY: LOCAL

```
Module X Procedure

  Lbl1: Trans Onfail Lbl3 Do;

        Lbl2: Trans Do;
              ....
              End;

        If .... Then Cancel Lbl1 DueTo 'Weather';
        End;

  Lbl3; Trans Do;
        ....
        Case $Backout
            Of Eq 'Arith' Then
            Of Eq 'PointerRef' Then
            Of Eq 'Weather' Then
            End;
        End;

  End;
```

WHEN FAIL

BACKOUT DATA TO STATE AT TRANSACTION BEGIN

CONTROL PASSES TO ONFAIL DESTINATION

BUILT-IN DATA CONTAINS REASON FOR FAILURE

INCLUDES LOCAL OBJECT BACKOUT

===> ATOMICITY PROVISIONS IN LANGUAGE
===> IMPROVES OPTIMIZATION POTENTIAL

ERROR HANDLING AND RECOVERY: INTER-OBJECT

GOAL: PROVIDE VARIOUS LEVELS OF STATE COUPLING AMONG OBJECTS

NONE - EACH REQUEST INDEPENDENT (but execution atomic)

"LOCKING" - RESERVE/RELEASE OF SHARED PROCEDURE

IF RESERVING TRANSACTION FAILS WITHIN RESERVE
THEN RESET SHARED, ELSE NO RESET

IF SHARED PROCEDURE FAILS, BACKOUT RESERVER

VARIANT - RESERVE COPY

"DB TRANSACTION" - RESERVE/RELEASE OF SHARED PROCESS

RESERVATION EXTENDS TO EXTERNAL TRANSACTION

ADDED TO TRANSACTION HEADER

RESERVED PROCESS MUST BE "XPROCESS"

ACCEPTS RESPONSIBILITY FOR INTEGRITY

AIDED BY SYSTEM

RESERVE ---> RESERVE MSG WITH TRANS-ID

REFERENCE ---> Q INFO HAS TRANS ID

FAIL OF RESERVER ---> NOTIFY MSG

ACCESSIBLE ONLY BY SYNCH REQUEST?

INTERLOCKS CAUSE RESERVER TRANSACTION BACKOUT (detection??)

FUNCTIONAL OBJECTIVES

ENVIRONMENT WITH

- NON-FRAGMENTING BASIC PATTERN
- NECESSARY ACCRETED FACILITIES

  WITH their convenient aspects
  WITHOUT associated fragmentation

COMMAND LANGUAGE

WITH ad-hoc aspects
WITHOUT HLL/CMD LANGUAGE split

DATABASE

WITH declarative def, query-like access, atomicity
WITHOUT HLL / DDL / DML split
WITHOUT SYSTEM/SUBSYSTEM split

APPLICATION GENERATION

WITH declarative def
WITHOUT uneasy coexistence with HLL

DICTIONARY

WITH accessibility of information
WITHOUT dictionary/catalog split

REQUIREMENTS AND DESIGN LANGUAGES

WITH function
WITHOUT separate repositories, views of application

ASYNCHRONISM WITHIN/AMONG APPLICATIONS

WITHOUT separate provisions for tasking, distribution, DB

SOURCES

ASSOCIATIVE DATA BASE MODELS (REL, E-R, ..)

Need data base in environment
Models can subsume others

VERY HIGH LEVEL LANGUAGES (SETL)

Raise environment level
Local data associative --> unification with DB

OBJECT-ORIENTED SYSTEMS (SMALLTALK ..

Provide non-fragmented system pattern
Better base than existing for
distributed applications
control systems

APPLICATION GENERATION

Hi-level spec of generalizable processing

MODULE INTERCONNECTION LANGUAGES

Substitute for low level linkedit,
Link to design levels

"The time appears to be right for the integration of languages, operating systems, and database research on object models."

Peter Wegner, 1982

DESIGN OBJECTIVES

USE OBJECT ORIENTATION AS A BASE

One application component, one set of support mechanisms

DESIGN:

ABSTRACTION LANGUAGE (approximation)

Associative local data model
Includes asynchronous capabilities
Version for interactive commands

DECLARATIVE DEFINITION FORMS

For data base, application generation
With close ties to procedural form

DICTIONARY/CATALOG FACILITIES

Subsuming current functions
Subsuming MIL function

DATABASE-ORIENTED ATOMICITY FACILITIES

DEVELOPMENT OF DESIGN

REMOVAL OF HLL DATA ACCESS FRAGMENTATION

LOCAL DATA, FILE, DATA BASE, UTILITY

INCORPORATE RESULTS IN ABSTRACTION LANGUAGE

UNIFY PROGRAM, DATA ACCESS

UNIFY PROCEDURAL DEFINITION
WITH DECLARATIVE DEFINITION (FOR DB, APGEN)

------ (From here directional) ------

OUTLINE SINGLE-USER / SINGLE-THREAD ENVIRONMENT

DIRECTORY = CATALOG + DICT + MIL + DESIGN REPOSITORY

LANGUAGE EXTENSIONS FOR CMDS/EXECS

EXTEND FOR MULTI-THREAD PROCESSING

COHERENT INTER-OBJECT COMMUNICATION (SYNCH, ASYNCH)

COHERENT ERROR-HANDLING & RECOVERY

------ (From here little work) ------

EXTEND TO MULTI-USER ENVIRONMENT (little work done here)

EXTENDED NAMING, CATALOGING, DISTRIBUTION, ...

SOME RELATED WORK

DATA BASE STRUCTURES IN PROGRAMMING LANGUAGES

EAS/E, PLAIN, RIGEL, PASCAL-R, DAPLEX (ADAPLEX)

SINGLE-MODEL LANGUAGES

APL, LISP: with consistent environments; SETL, TAXIS

RESOLVING DATA BASE WITH DATA ABSTRACTION

Leavenworth, Weller: fundamental
DAPLEX, RIGEL: less rigorous, hidden

OTHER

Consistent, full function PSEs: DOD xAPSEs (Intermetrics)
Integrated commercial DB access packages: NOMAD, FOCUS ...
Procedural/executable specification languages: FST, ...
Programming language extension for display support (Gries)

POTENTIALLY SEPARABLE ELEMENTS

DATA MODEL

More expressive than basic relational
More accessible than E/R, functional

DATA ACCESS LANGUAGE

More powerful than SQL, as user-friendly

DECLARATIVE OBJECT DEFINITION

For general application generation
Allow expansible set of simple generators
For data base definition
Replacement for 'subsystem' approach

'AGGREGATE OPERATIONS' (on data base subsets)

Utility Functions within DML
Applications in Engineering & Distributed DB

INTER-OBJECT COMMUNICATION CONCEPTS

Consistent syntax, semantics supporting
message passing, abstraction, co-routine, function call.

IDE - CURRENT STATUS

STATUS

FOCUS ON SINGLE USER ENVIRONMENT

CONSIDERABLE SPECIFICATION

VERY LIMITED DEVELOPMENT (formal grammar, some design)

FEATURES

ONE ASSOCIATIVE DATA MODEL FOR ALL DATA
(local, file, database, directory, design specs)

ONE FULL-SCALE VHLL FOR ALL "PROCEDURAL" PURPOSES
directory access, design, implementation, command

SMOOTH INTEGRATION OF DECLARATIVE FORMS
for application generation, database definition

OBJECT-ORIENTED ENVIRONMENT - UNIFORM MECHANISMS FOR
definition, compilation, linking, cataloging, accessing

DIRECTORY AS CENTRAL FOCUS OF APPLICATION DEVELOPMENT
subsumes catalog, dictionary, binding, control

APPLICABILITY

FEASIBILITY DEMONSTRATION

SOURCE OF SEPARABLE IDEAS

WITH (relatively) SMALL DEVELOPMENT EFFORT

- PROTOTYPING VEHICLE
- SINGLE-USER ENVIRONMENT

WITH MAJOR DEVELOPMENT EFFORT

- FULL ENVIRONMENT