# Metaprogramming and Application Generator Approaches to Software Reuse

Grady H. Campbell, Jr.
Software Architecture & Engineering
November 8, 1988

SOFTWARE AE

# Goals of Software Reuse

productivity

reliability

# Obstacles To Software Reuse

Locating suitable components

Understanding the operation of selected components

Reliably modifying a component for current needs

# Why Straightforward Attempts At Reuse Fail

1) locating, selecting, and understanding existing code is time-consuming and modifying it is error prone.

2) assumptions
design decisions
dependencies

are

implicit

or not distinguished from functional requirements

# Making Reuse Possible

"design for reuse": generalizes "design for change" to a family of programs

"separation of concerns": domain knowledge vs software engineering knowledge

# Methods of Reuse

Explicit

- Repository
- Taxonomy/Schema-based (tailoring is automated)

Implicit

- Model-based Application Generator (selection, tailoring, and integration are automated)

# Keys To Effective, Explicit Reuse of Software

a taxonomy of abstractions (program family schemas)

differentiation criteria for the instances of each abstraction (parameters)

automatic code delivery from abstraction selection and instantiation (parameter completion)

# Abstraction

key to effective reuse: identification of "proper abstractions

"proper" abstraction: the "right" set of concrete instances are included

too narrow ➡ inflexible implementation and too little reuse

too broad ➡ implementation too inefficient for practical use or parameterization too complex

# A Partial Abstraction Taxonomy

Application (Domain) Software

Hardware Hiding
- Virtual Computer
- Virtual Device
  - Virtual Display
  - Virtual Data Storage
  - Virtual Network

System Software
- User Interface
- Data Abstraction
  - Abstract Data Type
  - Abstract Object
- Logic Abstraction
- System Generation

# "General-Purpose" vs. "Generic" Software

both enable reuse to satisfy specific needs

only generic software can be customized/optimized to those needs

# How Variation In Software Is Accommodated

alternate implementations

conditional code

parameterization

metaprograms (parameterized schemas)

# AN EXAMPLE

```
TYPE CLASSID IS NAME;

TYPE ATTRIBUTE IS
    RECORD
       ID:NAME;
       FORM:
          UNION
             VALUE:TYPE;
             RELATION:CLASSID;
          END UNION;
    END RECORD;


PROGRAM ATTRIBUTE (CLASS:CLASSID, DESCR: ATTRIBUTE) IS
    IF CURR_SLOT (DESCR.FORM) = VALUE THEN

       function g_DESCR.ID (PL : CLASS) return DESCR.FORM.VALUE;

    ELSEIF CURR_SLOT (DESCR.FORM) = RELATION THEN

       function g_DESCR.ID (pl : CLASS) return DESCR.FORM.RELATION_list;

    ENDIF


END ATTRIBUTE;


PROGRAM CLASS (ID:NAME, PARENT: CLASSID, ATTRIBUTES:
LIST OF ATTRIBUTE) IS
    type ID is obj.object;
    TABLE (ID_list, ID);

    function select () return ID_list;

    FORALL ATTR IN ATTRIBUTES

       ATTRIBUTE (ID, ATTR)

    END FORALL;
END CLASS;
```

# Elements of Implicit Reuse

A conceptual model of the problem domain

Abstractions corresponding to each domain concept

Parameterization of each abstraction determining possible instances of the concept

# The Application Generator Application Development Process

| | |
|---|---|
| Specify | (incremental and iterative with inferred defaults) |
| Validate | (for consistency and completeness) |
| Generate | (completely automated from specifications) |
| Execute | (to evaluate correctness of specifications) |

# SPECTRUM

## A Domain-Independent Application Generation Environment

SOFTWARE AE

# SPECTRUM

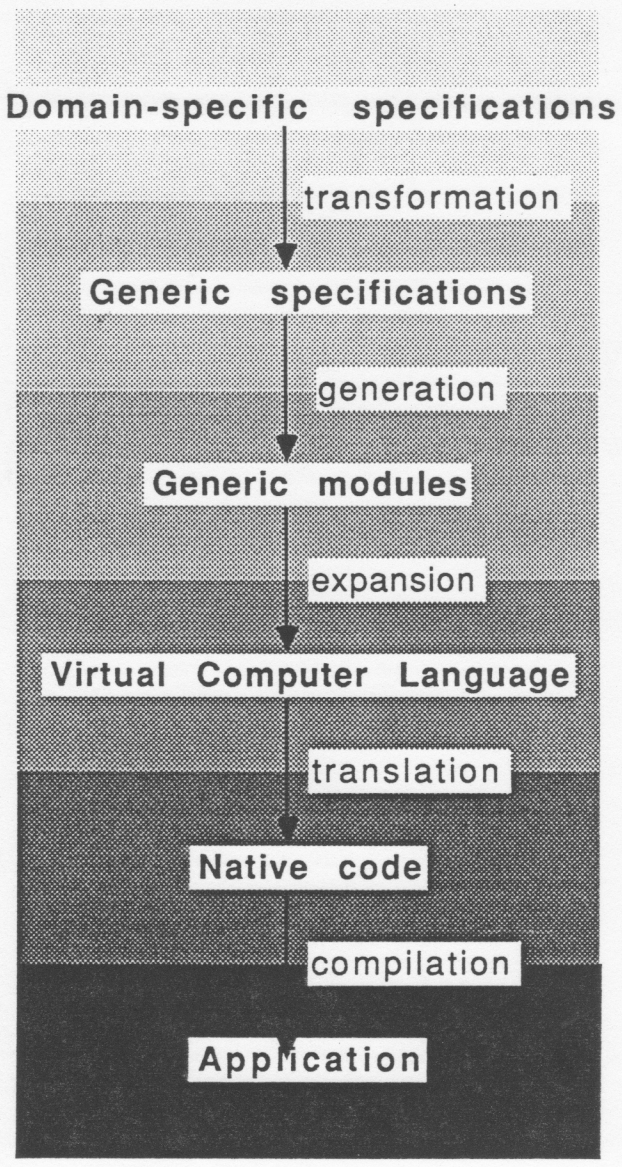
An environment for automatic software application generation from high level specifications.

- Implicit reuse of software
- Model-based specification
- Domain independent
- Supports iterative development; integrated prototyping.

Oriented to complex applications

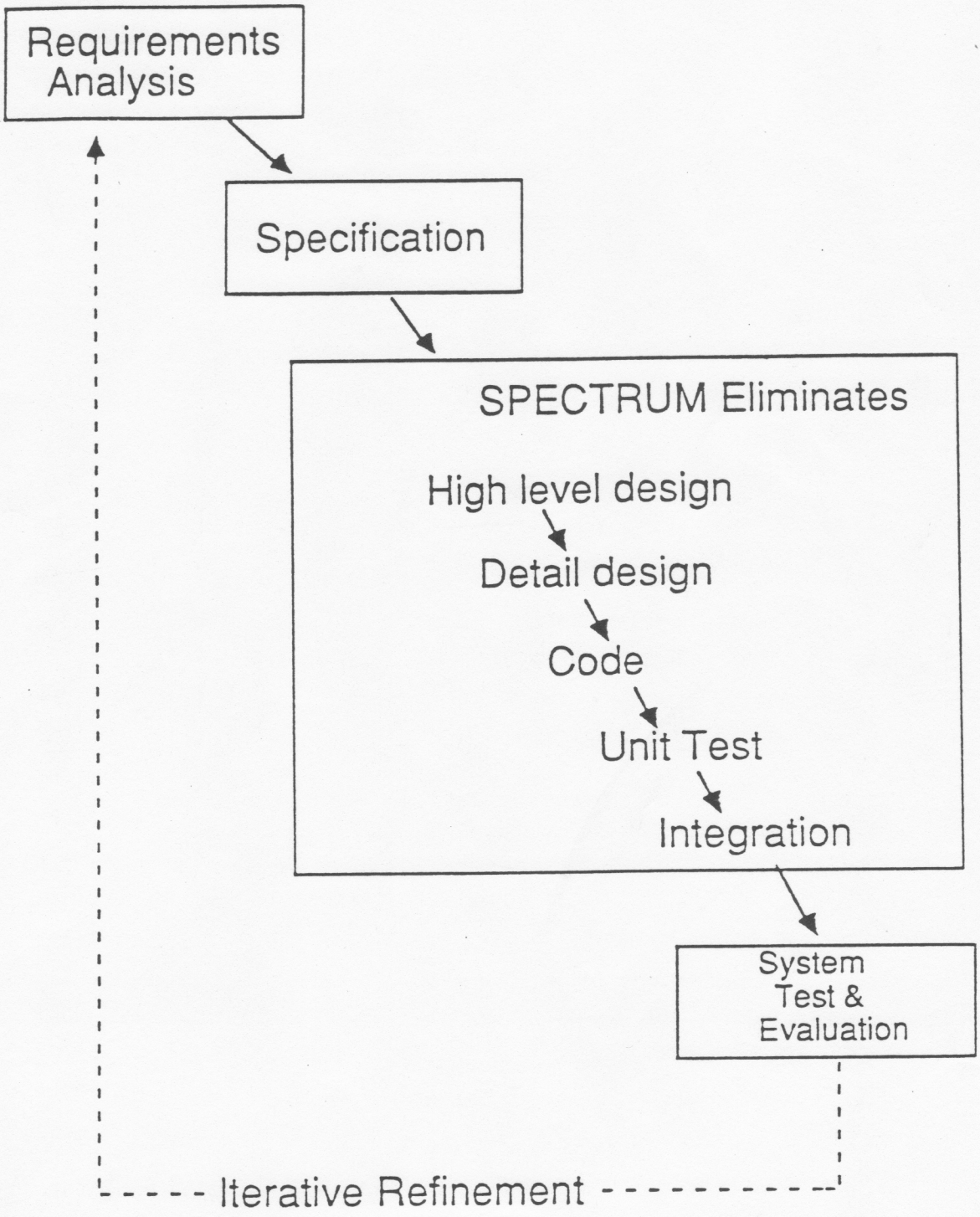
- Applications built around semantic data model.
- Combines symbolic techniques with traditional procedural approaches.
- Allows integration of external software.

# Software Engineering Environment Levels

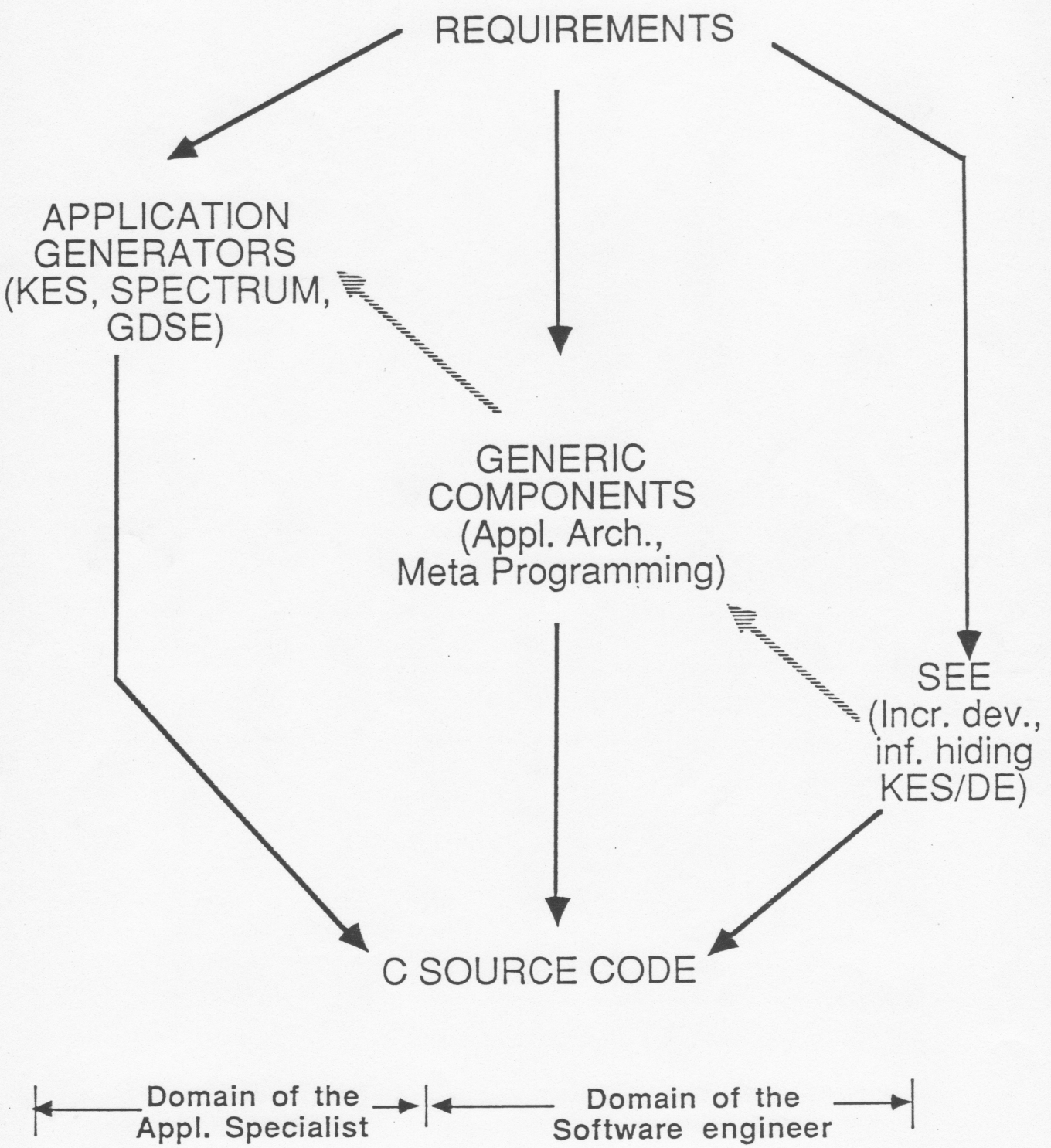



SOFTWARE AE

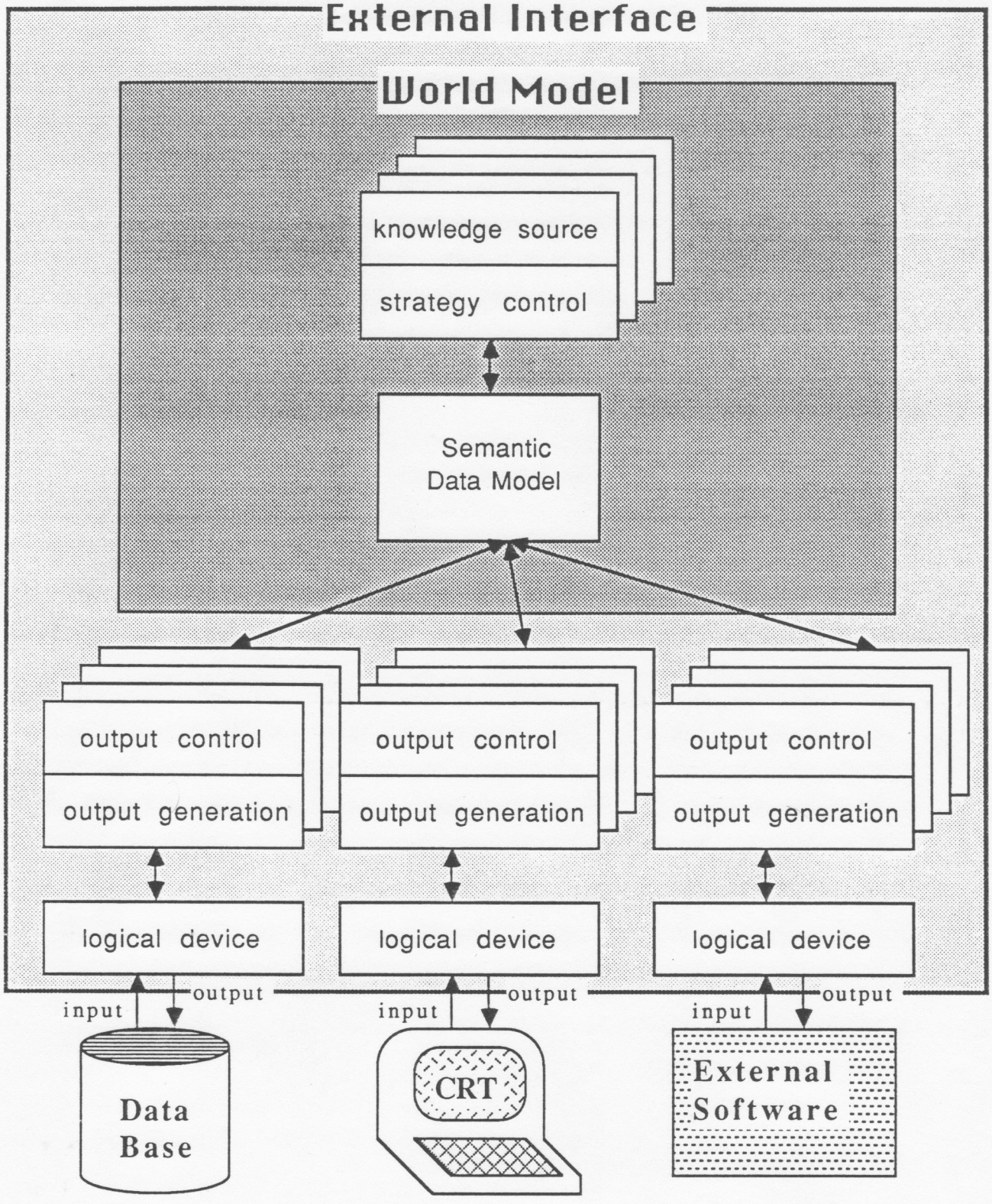
# SPECTRUM CHANGES THE SOFTWARE LIFE CYCLE MODEL



# MULTI-LEVEL DEVELOPMENT ENVIRONMENT

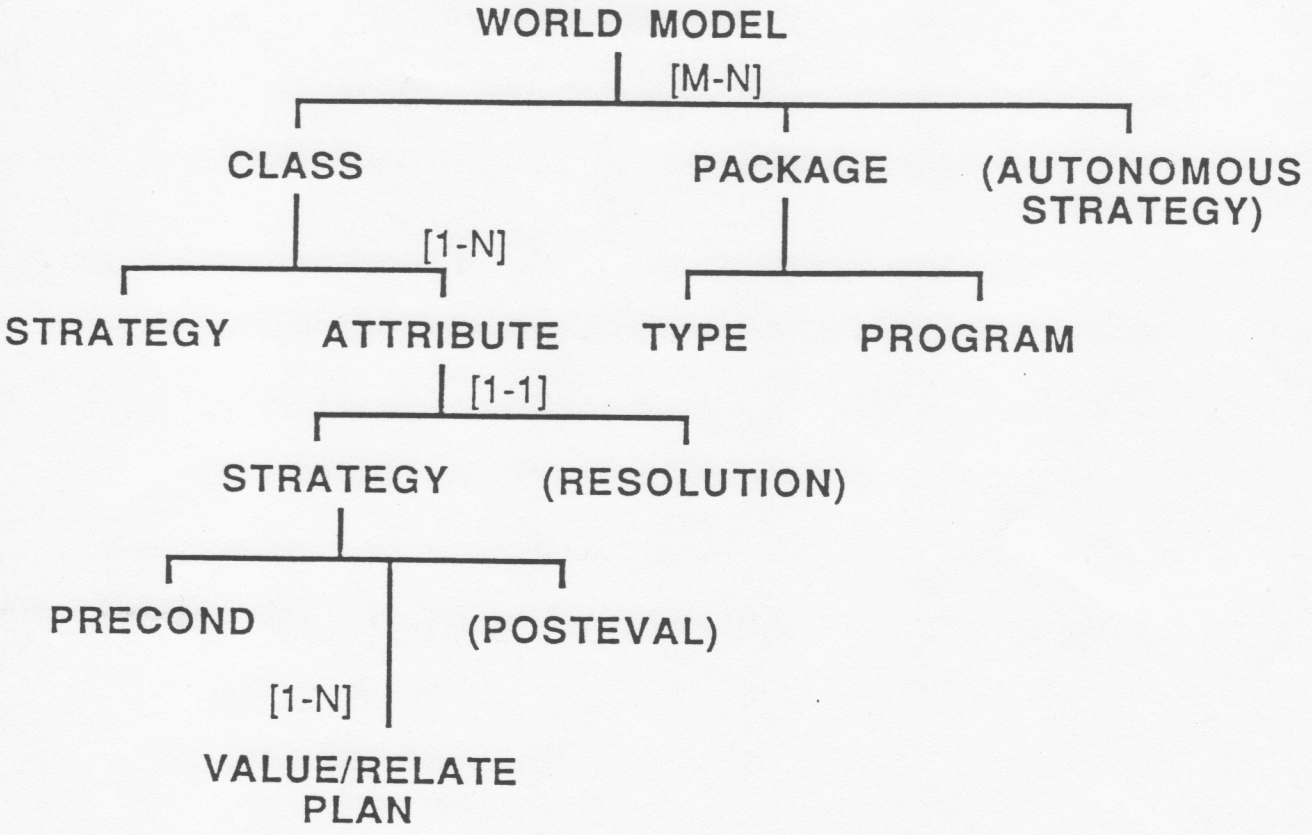


# Application Model




Software AE

# WORLD MODEL CONCEPTUAL STRUCTURE




---

CLASS STRATEGY
- INSTANTIATE/GENERATE
- INITIALIZE
- REFINE/FORGET

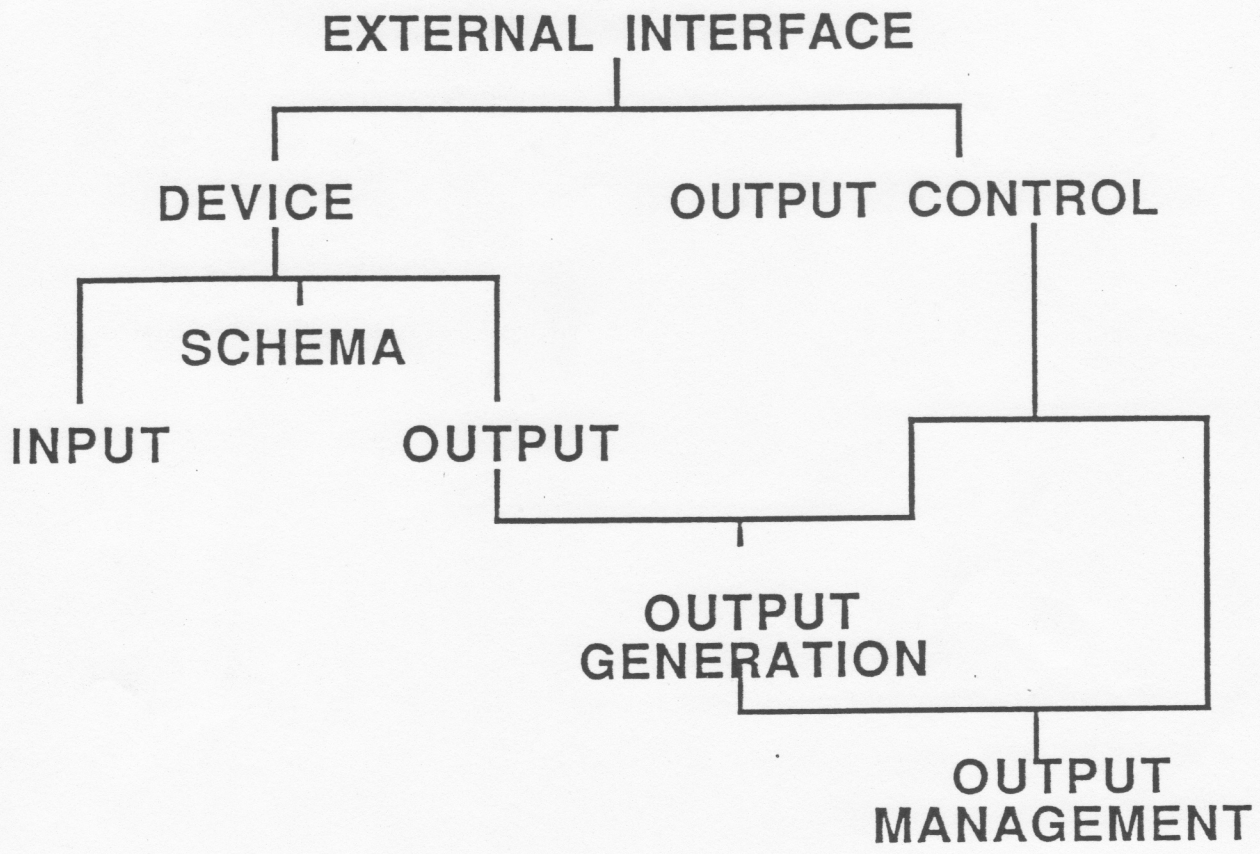
ATTRIBUTE
- VALUE/RELATE
- INHERITED

STRATEGY
- DEMAND
- EVENT

VALUE PLAN
- USER INPUT
- RULE KS
- PROCEDURAL KS
- DEVICE INPUT
- INHERITANCE
- SUBSTRATEGY

RELATE PLAN
- USER SELECTION
- PREDICATE SELECTION
- GENERATE
- DEVICE INPUT
- INHERITANCE
- SUBSTRATEGY

# EXTERNAL INTERFACE CONCEPTUAL STRUCTURE




---

DEVICE
- STANDARD HARDWARE
- EXTERNAL SOFTWARE
- RELATIONAL DATABASE

INPUT
- OUTPUT SYNCHRONOUS
- PROMPTED SYNCHRONOUS
- (ASYNCHRONOUS)

OUTPUT CONTROL
- SEQUENTIAL
- SELECTION
- CONCURRENT
- SUBORDINATE

OUTPUT GENERATION
- -OBJECT-ORIENTED
- RELATION-ORIENTED
- TEXT/DOCUMENT
- HYBRID/NESTED

OUTPUT MANAGEMENT
- ACTIVATION
- OBJECT SELECTION
- ENABLED INPUTS

# APPLICATION SPECIFIC ENVIRONMENTS

An Application Specific Environment (ASE) is a software development and life cycle support tool specialized to a single application class.

- Generation from high level specifications
- Interface reflects semantics of application domain
- Rapid prototyping is an integral concept
- Programming not involved

# What Can Be Done Now

4GL application generators

Object-oriented languages

Manual schema-based metaprogramming

# Summary

Reusable Components must be designed for reuse

Proper abstraction is essential to effective reuse

Metaprogramming provides a practical representation of abstraction supporting explicit reuse of derivable variations

Mechanisms for explicit reuse are a basis for implicit reuse in the form of application generators