Agenda

- Model-based paradigm
- Case studies: UIDE, Mecano
- Architectures
- Break
- Case studies: Humanoid, ITS
- Survey of Model-Based Tools
- Conclusions
- Questions

Model-Based Paradigm: Topics

- **A New Paradigm for Interface Development**
  - » **Shortcomings of current interface development tools**
  - » **Model-based user interface development**
  - » **Success stories**

Current Tools [Myers 92b]

- **Interface builders**
  - » **NeXT IB, UIM/X, DevGuide, Prototyper, XDesigner, WindowsMAKER**
- **UIMSs**
  - » **OpenDialogue**
- **Toolkit libraries**
  - » **Macintosh Toolbox, Motif and X lib**
- **Design exploration tools**
  - » **Macromind Director, Hypercard, Visual Basic**

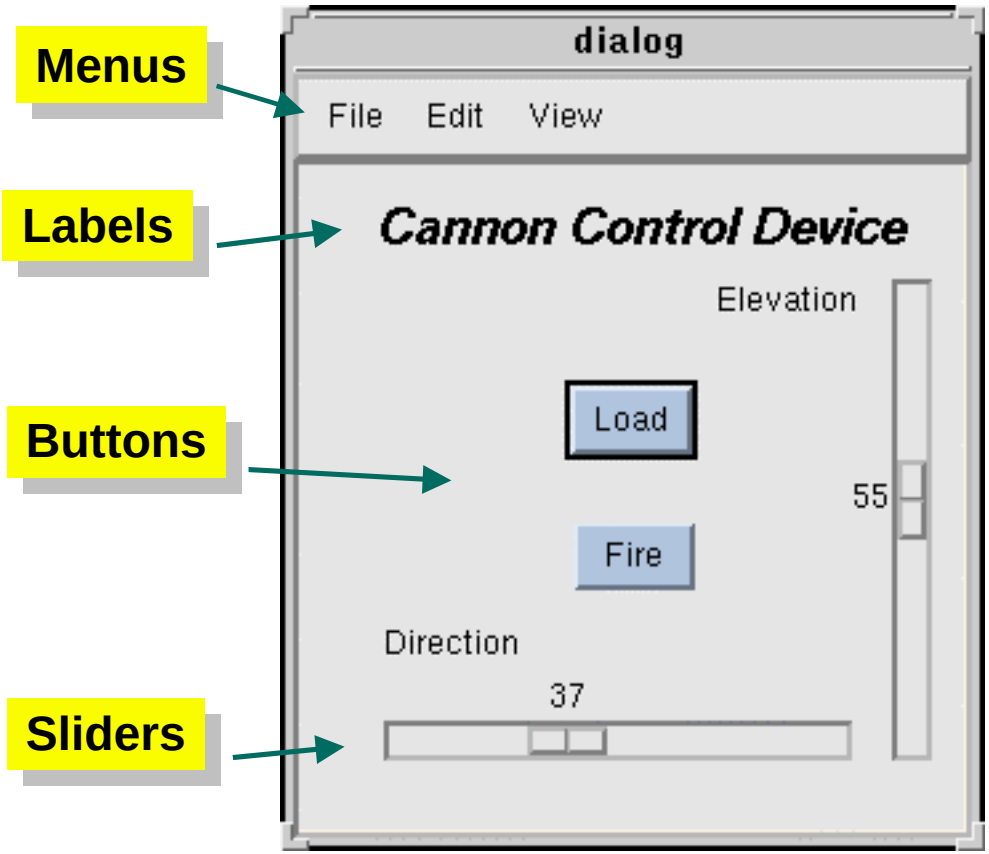
Tool Evaluation Criteria

- **Ease of use**
  - » Ease of learning and using tool to develop interfaces
- **Class of interface designs supported**
  - » Aspects of interface development targeted
- **Lifecycle support**
  - » Requirements analysis, design, implementation, maintenance
- **Performance**
  - » Performance of applications constructed with the tools

Tool Evaluation Summary Table

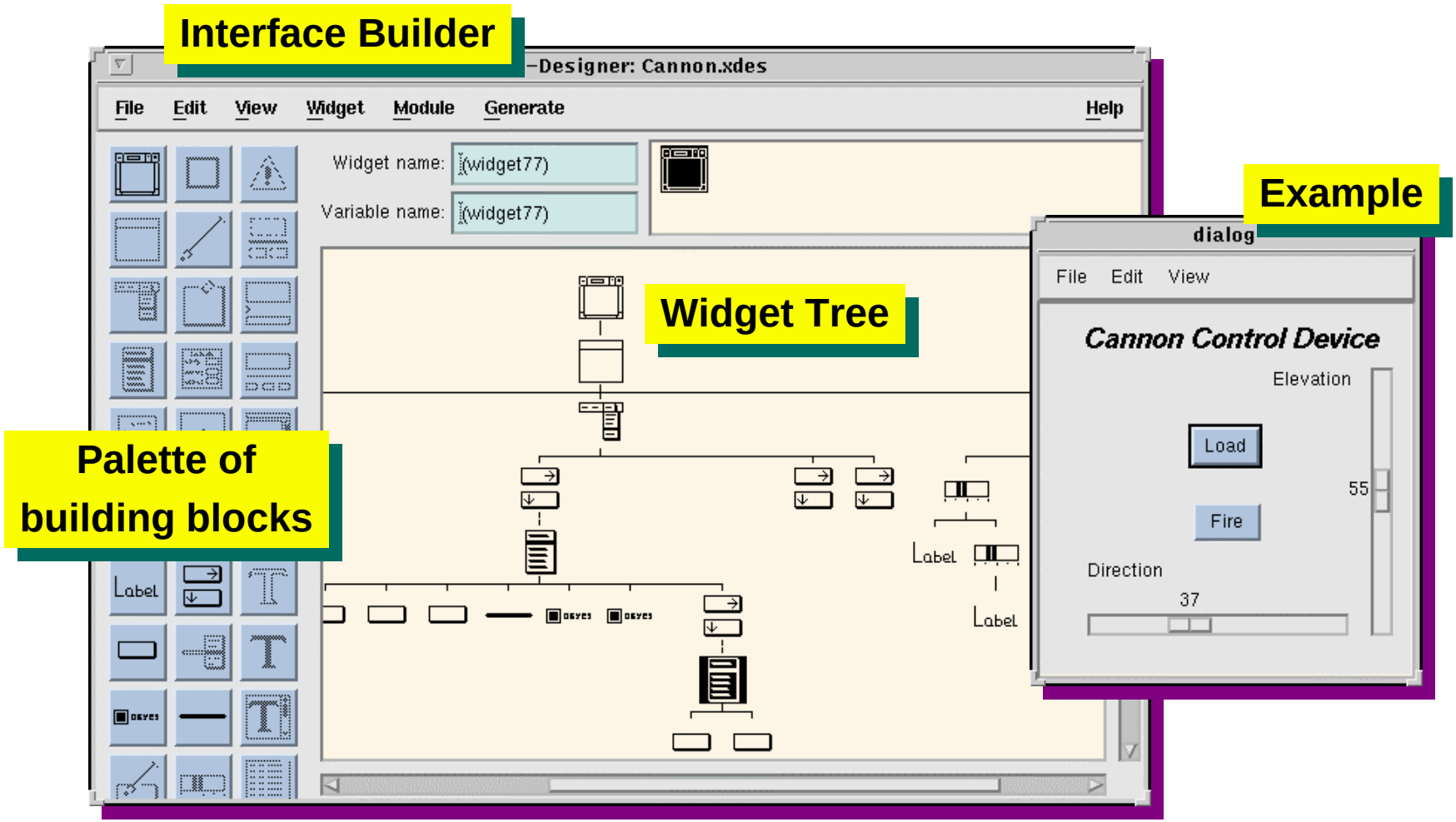
	Ease of Use	Classes of Designs		Lifecycle Support	Performance
		Supported	Not Supported		
Interface Builders	Excellent for static facets. Poor for dynamic ones	Menus, buttons, sliders, etc.	Dynamic aspects	Detailed design, implementation	Good
UIMSs	Fair	Dialogue control	Everything doable but hard to do	Implementation	Moderate
Toolkit Libraries	Poor	Everything doable, but hard to do		Implementation	Excellent
Design Exploration Tools	Excellent	Presentation, limited behavior	Complex behaviors	Early and detailed design	Poor

Interface Builders: What They Can Do



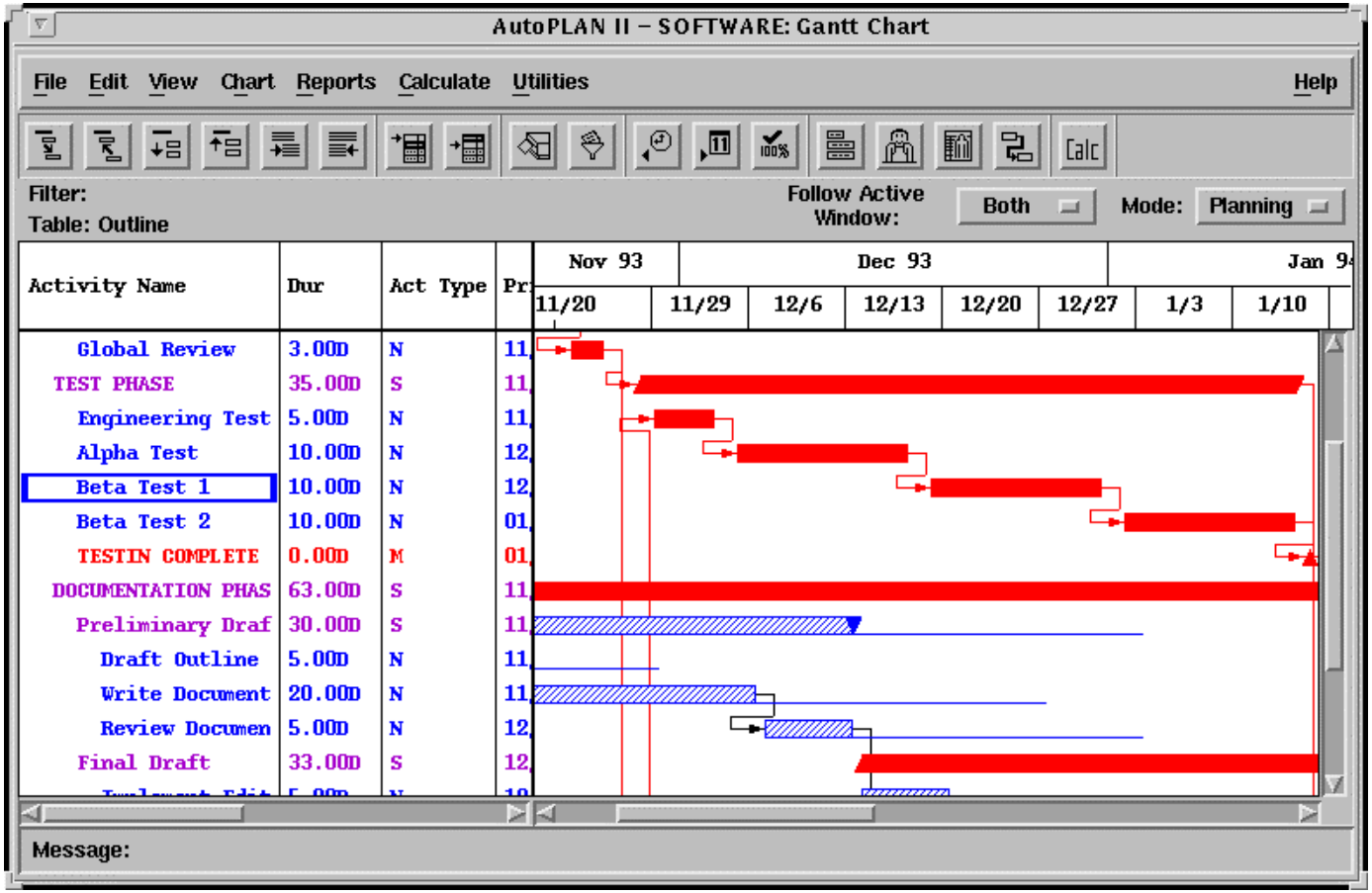
- Place building blocks
  - » Predefined set only
- Define layout
- Set characteristics
  - » Font
  - » Color
  - » etc.
- Connect to application
- Test behavior

Interface Builders: How They Do It (XDesigner)



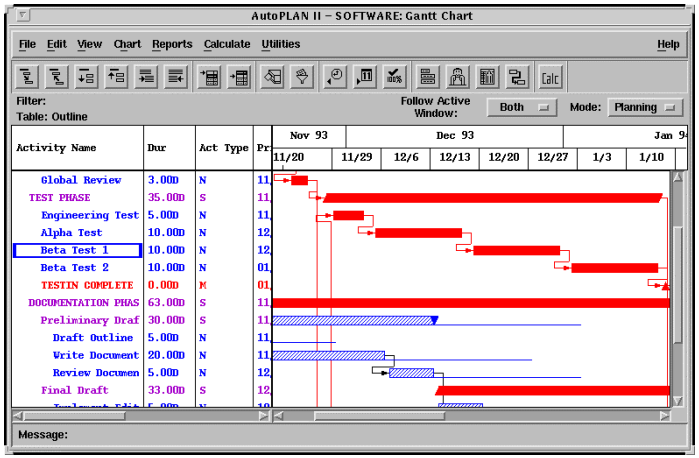
Interface Builder Limitations Example

Desired Interface: Project Planning (Auto PLAN II)



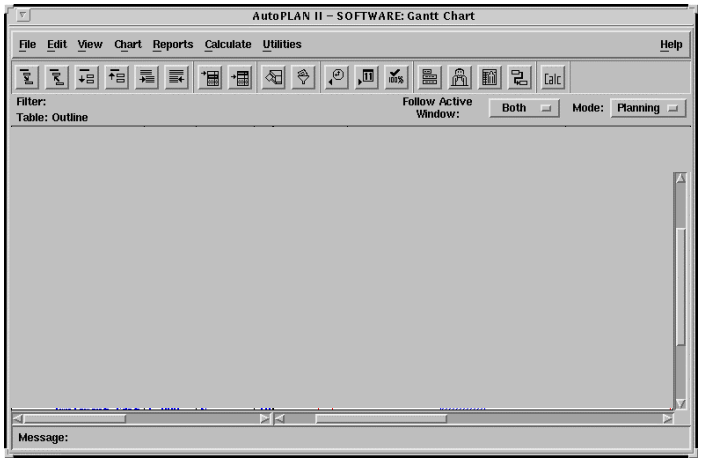
Interface Builder: Partial Solutions

Desired Interface



- Table with dynamic data
- Gantt chart
- Direct manipulation

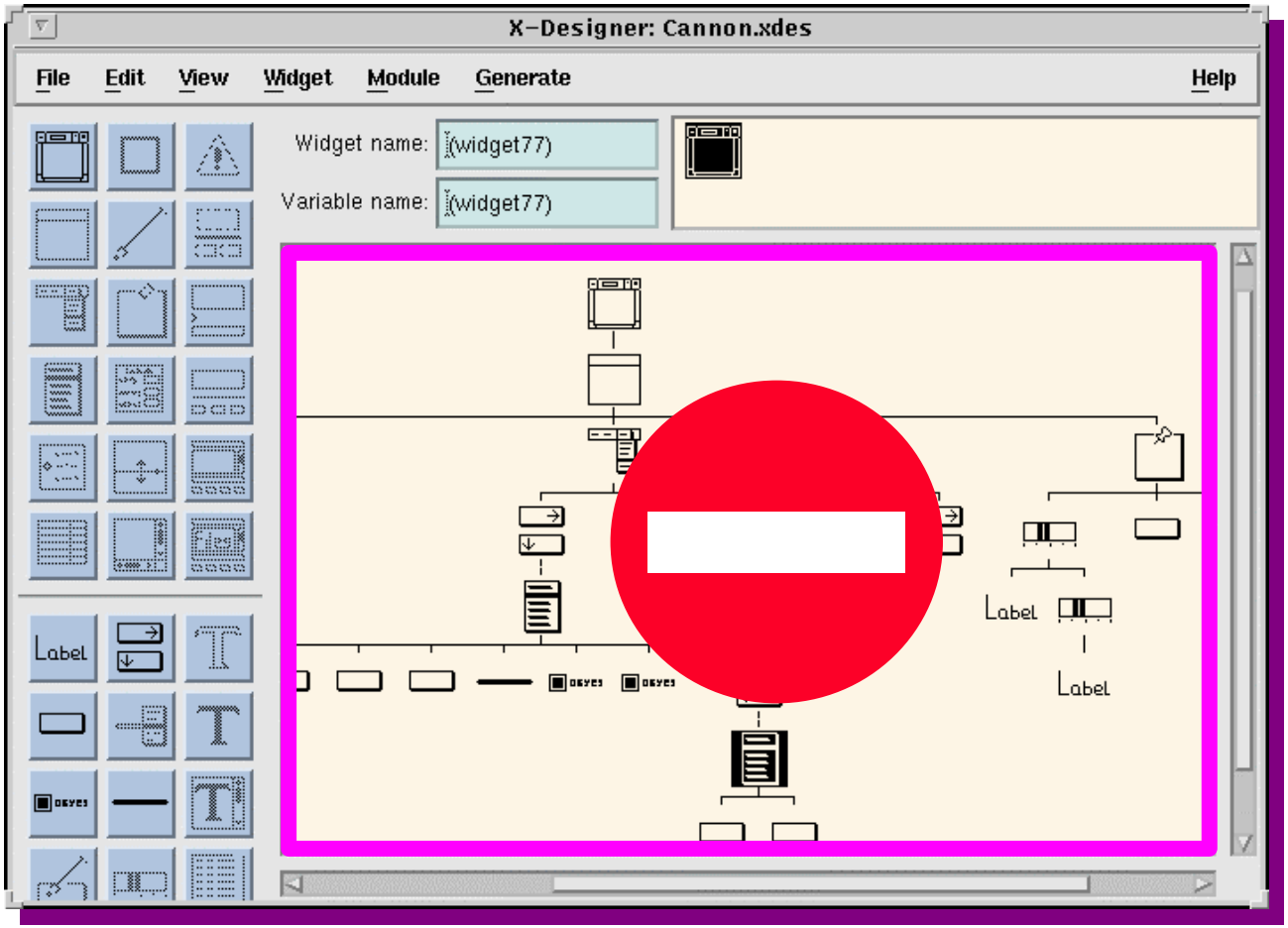
Interface Builder Solution



- Menus
- Palette (icons)
- Scrollbars

Interface Builders: Partial Solutions

Interface builders cannot build their own interface

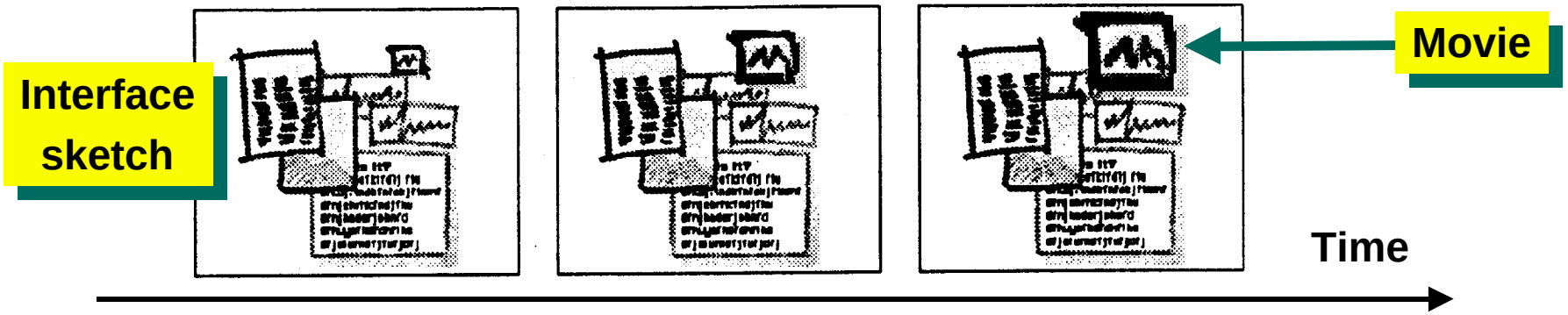


Interface Builders: Sources of Limitations

- **No support for applications involving**
  - Data with complex structures
  - Heterogeneous data
  - Variable amounts of data
  - Time-varying data
- **What You See Is All You Get**

Macromind Director: Only a Design Tool

Moving a movie while zooming it up or down



Strengths

- Rough sketches
  - » Avoid discussing details
  - » Focus on functionality
- Animation
- Easy to produce

Weakness

- Just a mockup
  - » No implementation

[Wong 92]

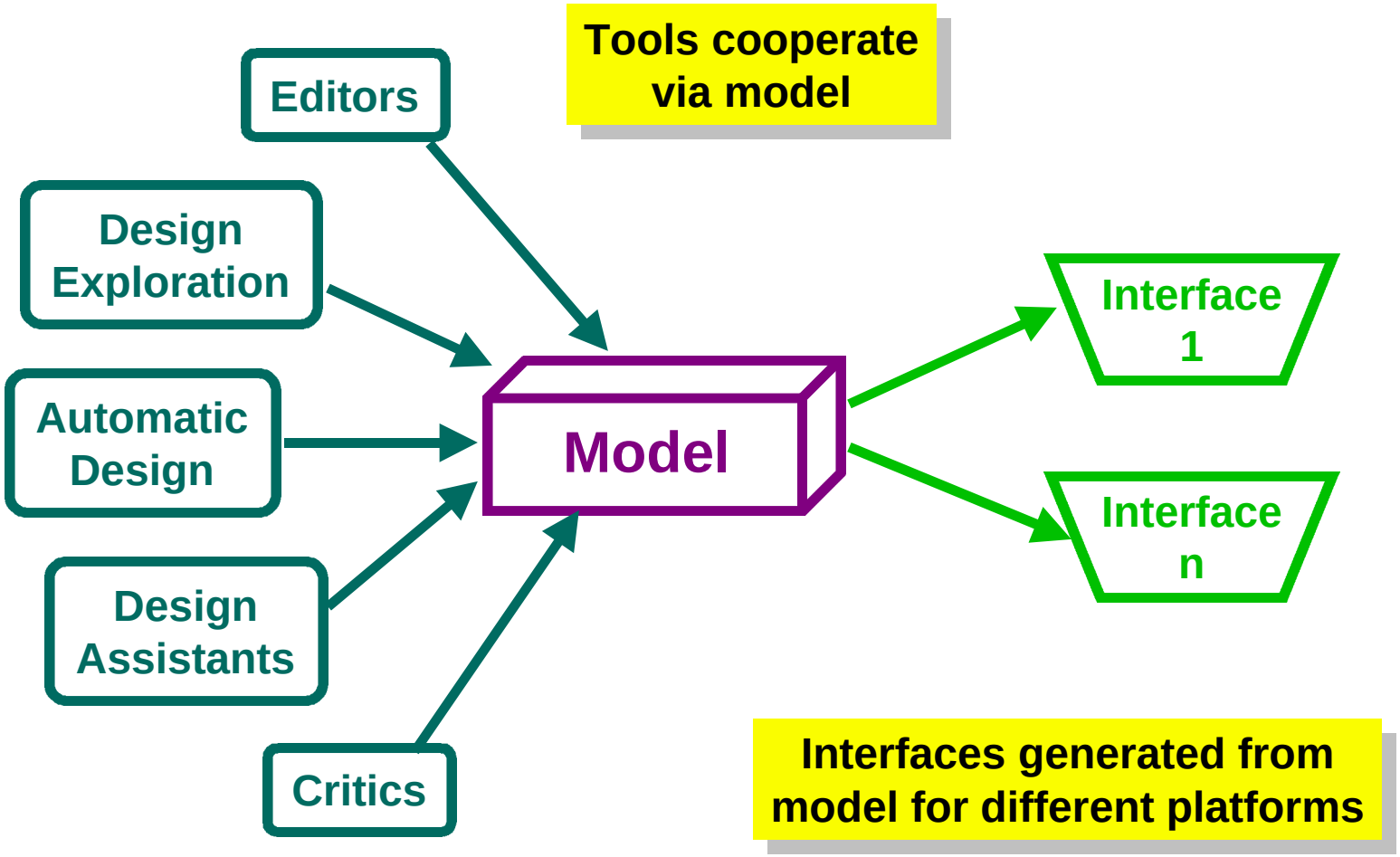
Summary of Shortcomings of Current Technology

- **Interface development is a complex process**
  - » **Tools only help with isolated portions of that process**
- **Interface development is expensive**
  - » **All windows painstakingly designed by humans**
- **Poor lifecycle support**
  - » **Changes difficult to propagate**
  - » **No tools address both design and implementation**
- **Poor support for portability & customization**

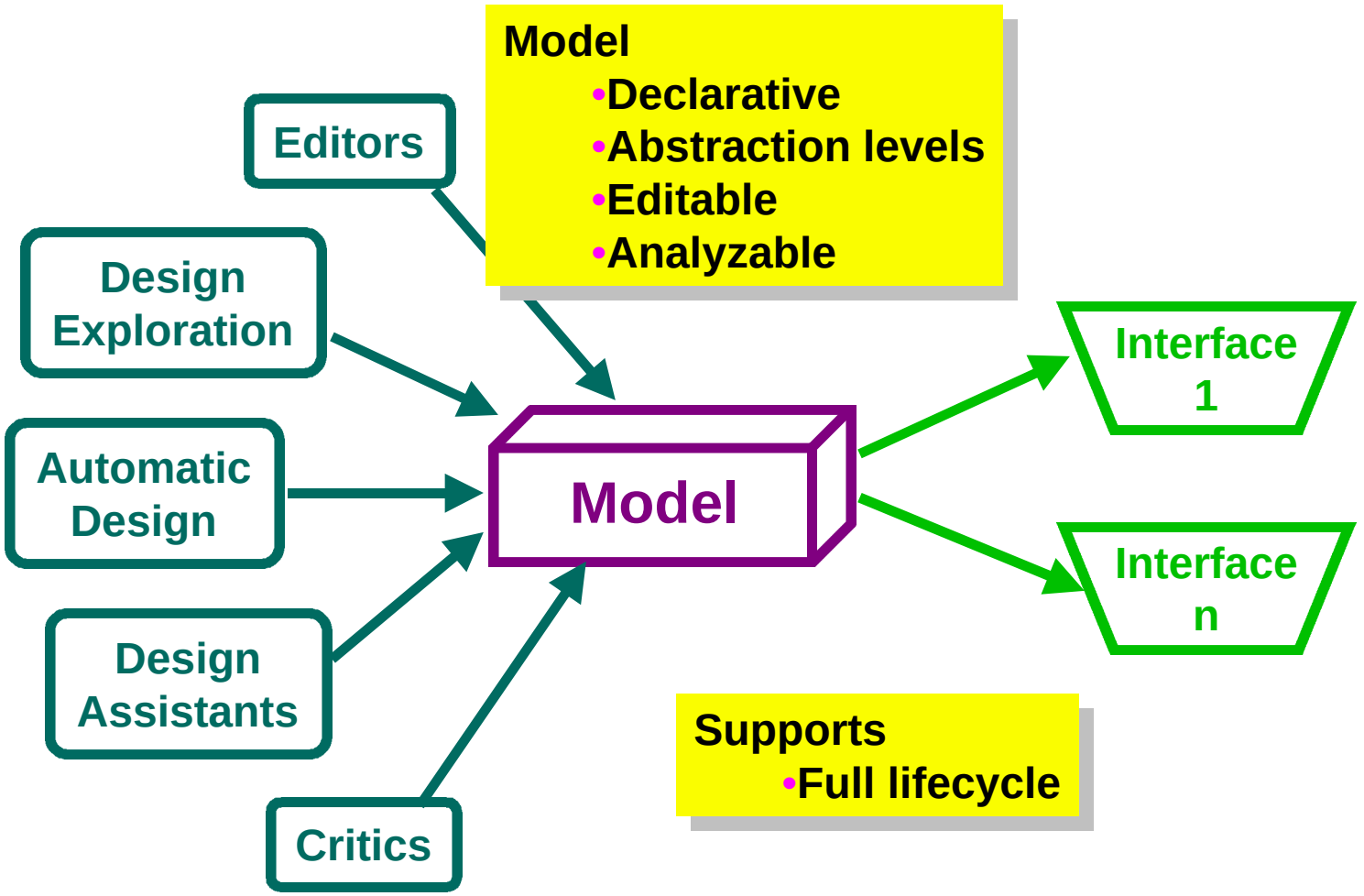
A New Paradigm: Model-Based Interface Development

- **Idea:**
  - » To use a declarative interface model to drive development
- **Goals:**
  - » To provide comprehensive development environments (i.e., design and implementation phases)
  - » To deliver robust lifecycle support
  - » To improve portability of interfaces
  - » To integrate usability studies with interface development

Model-Based Approach



Model-Based Approach



Model Definition

- **Webster's definition of model**
  - » **"One who is employed to display clothes or other merchandise"**
  - » **"A set of plans for a building"**
  - » **"A system of postulates, data and inferences presented as mathematical description of an entity or state of affairs"**

Interface model

A set of plans for a user interface

A system of postulates, data and inferences presented as a declarative description of a user interface

What Is in a Model

- Tasks
  - » Tasks users are expected to perform using an application
- Application
  - » Objects and commands an application provides
- Presentation and behavior
  - » Screen appearance and input responses
- Platform characteristics
  - » I/O devices available, device characteristics
- Workplace characteristics
  - » Ambience noise, organizational chart, stress level
- User preferences

Why Models Help

- **Single repository for interface specification**
  - » **Supports tool integration**
- **Declarative representation**
  - » **Supports automated analysis**
  - » **Facilitates understanding of designs**
- **Multiple levels of abstraction**
  - » **Support smarter tools**
  - » **Provide leeway for interface reconfiguration**

Success Stories (1)

- **Prototyping from partial specifications**
  - » **Humanoid [Szekely 92, Szekely 93]**
- **Support conceptual design**
  - » **Humanoid [Luo 93]**
- **Automated interface generation**
  - » **Mecano [Puerta 94], UIDE/DON [Kim 90], GENIUS [Janssen 93]**
- **Automated design critics**
  - » **UIDE [Braudes 90, Foley 91, Byrne 94]**

Success Stories (2)

- Support for reconfigurable interfaces
  - » ITS [Gould 92, Wiecha 90]
- Context-sensitive presentations
  - » Humanoid [Szekely 90, Szekely 92]
- Animated guidance and tutorials
  - » UIDE/Cartoonist [Sukaviriya 90], [Moore 90]
- Hypertext “balloon” help
  - » Humanoid/H3 [Moriyon 94]

Summary
Model-Based Paradigm

- Addresses limitations of current commercial technology
- Enables comprehensive user interface development environment
- Mechanizes user interface theories

Blank Page

Agenda

- Model-based paradigm
- Case studies: UIDE, Mecano
- Architectures
- Break
- Case studies: Humanoid, ITS
- Survey of Model-Based Tools
- Conclusions
- Questions

Case Studies: Objectives

- Highlights of system
- Target interfaces
- Model components
- Architecture
- Examples
- Benefits and shortcomings

Case Study: UIDE

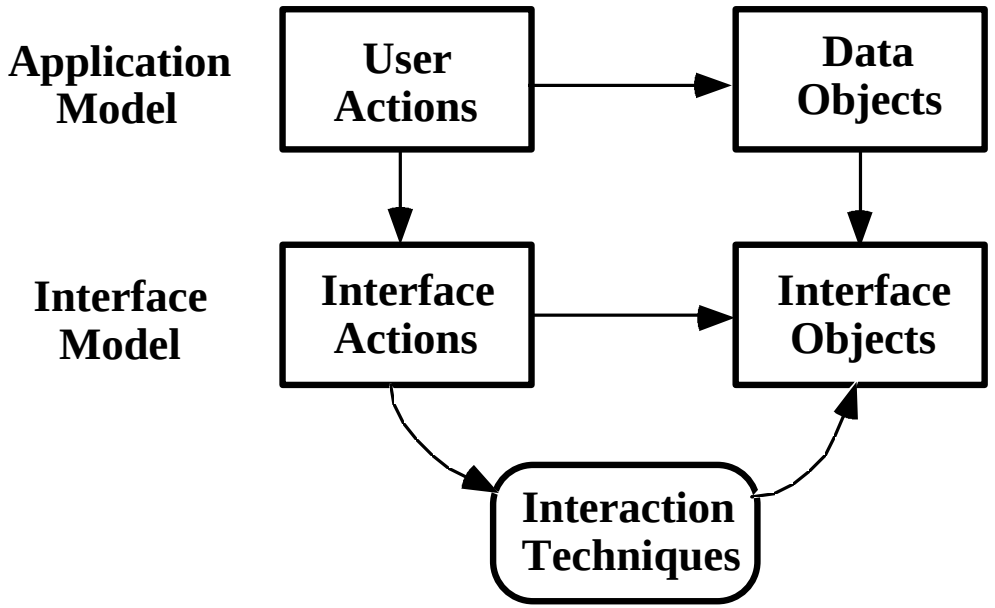
- User Interface Design Environment [Foley 88]
- Software environment supporting all facets of user interface development
  - » Designer support with design-time tools
  - » End-user support with run-time tools
- Interfaces are designed and run using a declarative specification of the interface
- Target interfaces:
  - » Small-scale prototypes

UIDE Target Interfaces

Digital Circuit Layout Editor

UIDE Paradigm

Models define actions and how they are applied to data and interface objects



UIDE Model Components (1)

- The application model defines declaratively the allowed user actions
- Each user action may affect one or more of the data objects defined in the application's data model (e.g., a string constant)

```
Action Create-NAND-gate
{
        Parameter object : NAND
        Parameter location : Position
        Pre-condition : "exist(x,DESIGN)"
        Post-condition : "exist(object,NAND)"
}
```

UIDE Model Components (2)

- The interface model defines declaratively the interface behavior that accomplishes the allowed user actions
- Each interface action is carried out by interface objects (e.g., a text field) through appropriate interaction techniques
- End users access each data object through one or more interface objects (e.g., a text field for a string constant)

UIDE Model Components (3)

Interface action

```
Action select-graphical-object
{
        Parameter graphicalObj: PresentationObject
        Parameter applicationObj: ApplicationObject
        Pre-condition:
                "exist(graphicalObj, PRES-OBJ)&
                status(graphicalObj, VISIBLE)"
        Post-condition:
                "status(graphicalObj, HIGHLIGHTED)"
}
```

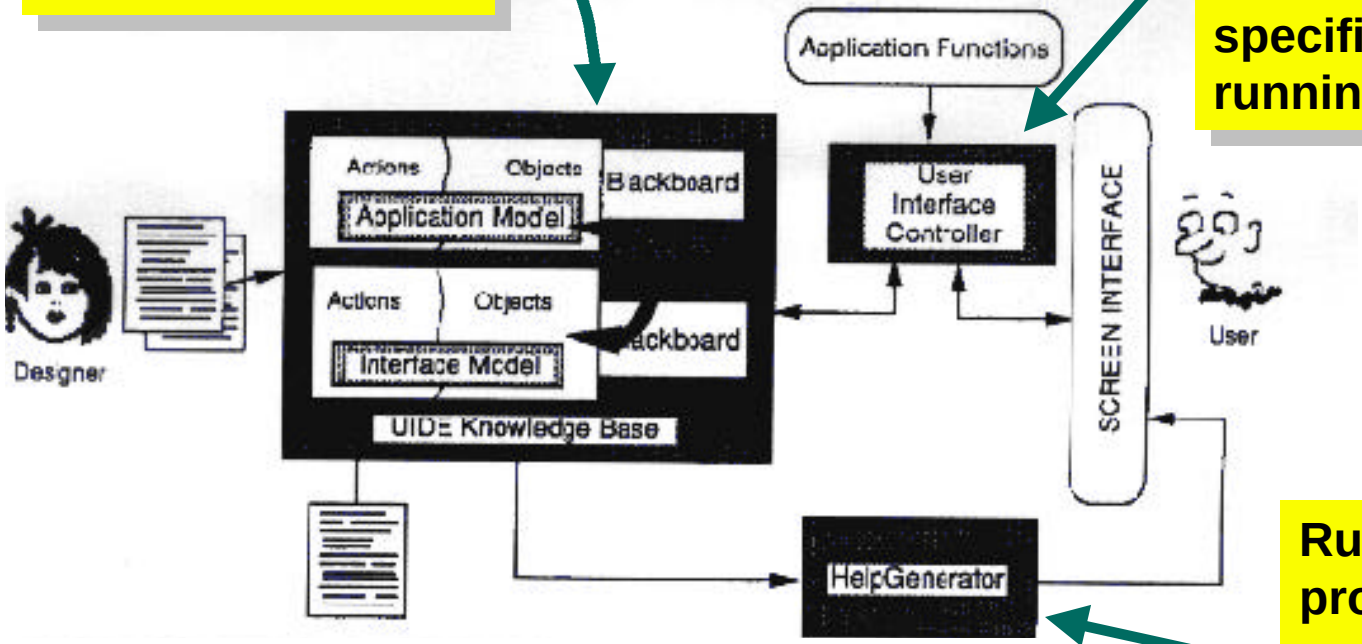
Interface object

```
Class Button {
        name : string
        location : position
        parent : window
        applicable actions : selectCommand
}
```

UIDE Architecture

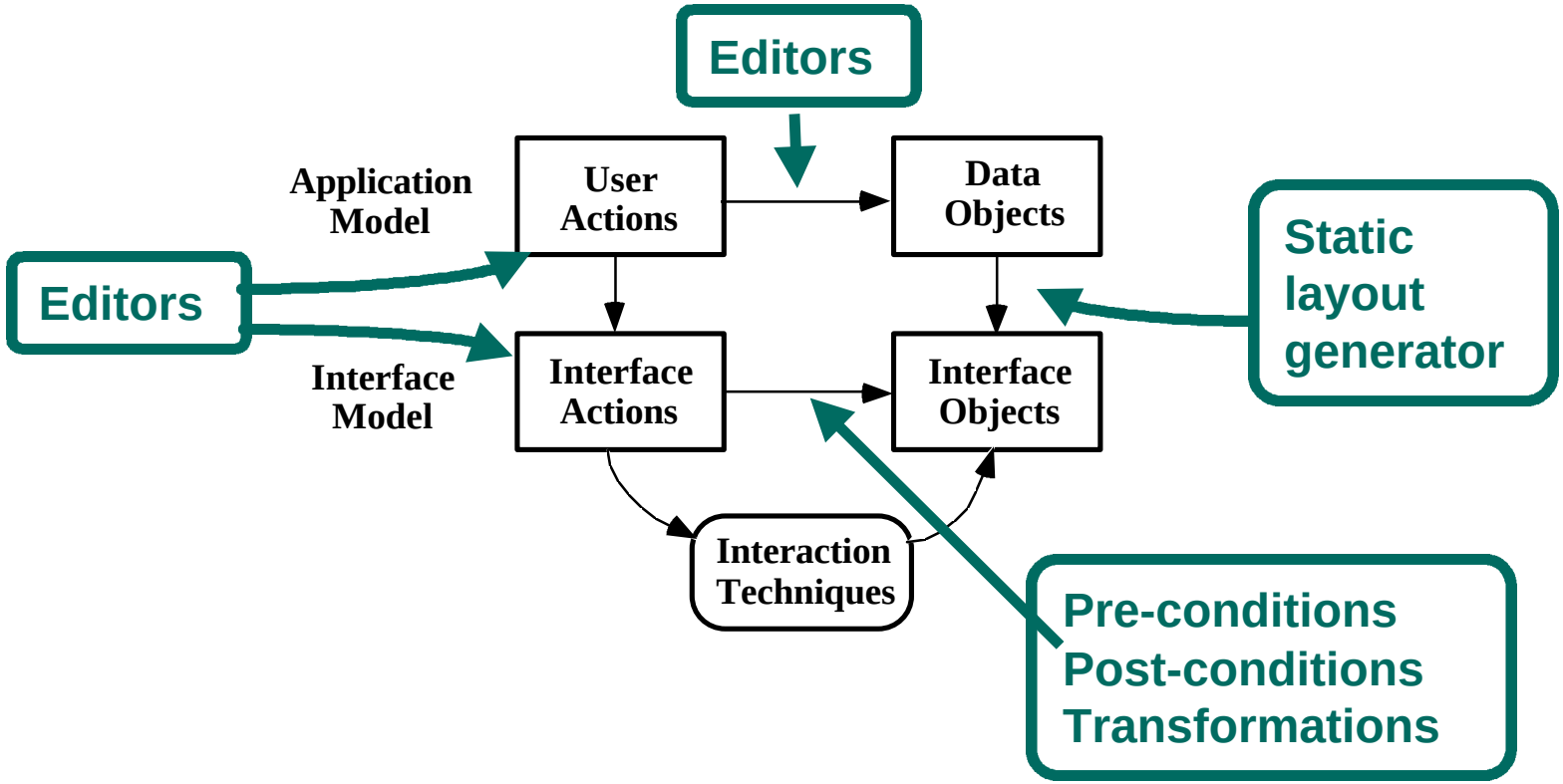
Design tools allow model editing and refinement

Run-time system transforms declarative specification into running interface



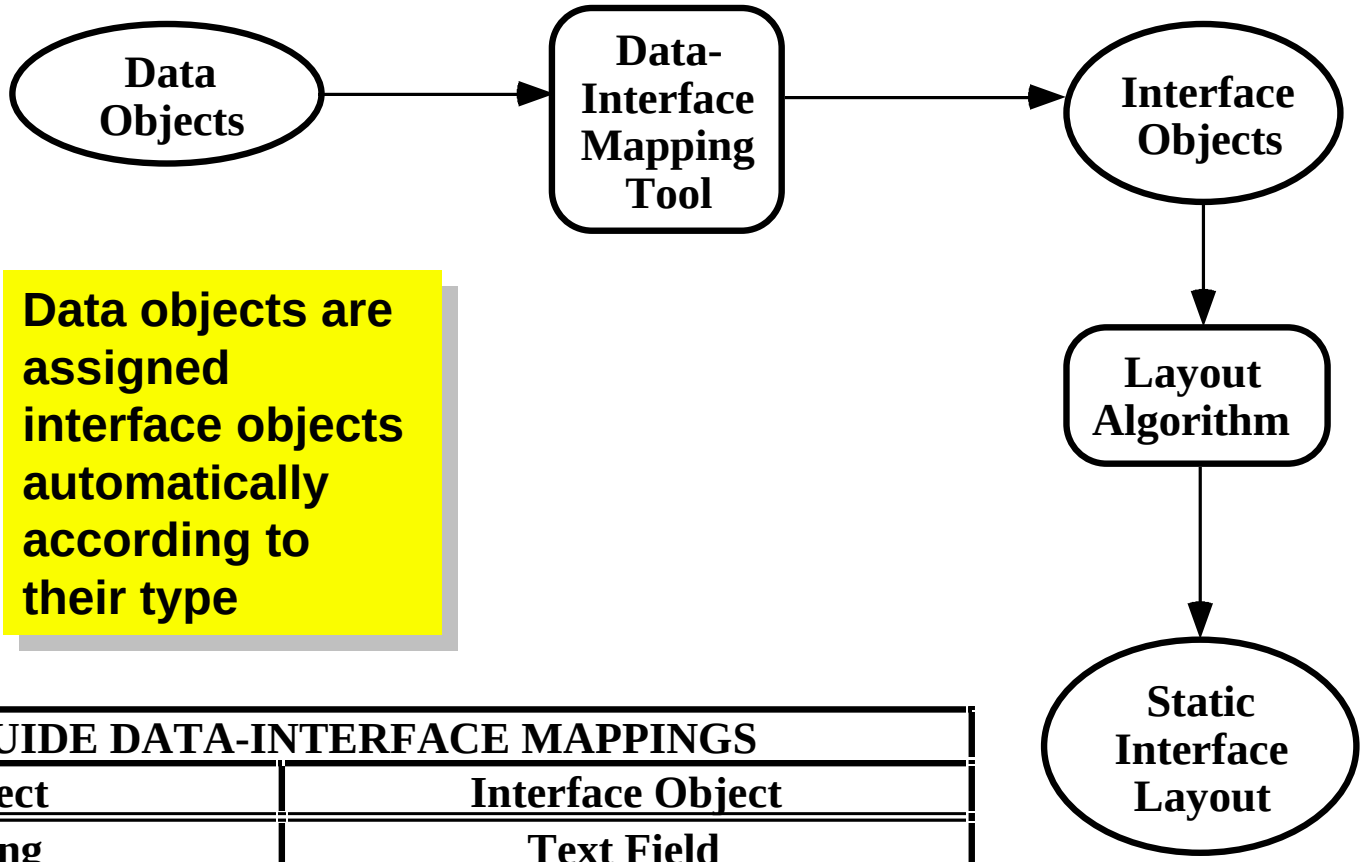
Run-time tools provide user services

Building Interfaces with UIDE: Design-Time Assistance



UIDE: Automatic Layout Generation

[de Baar 92]



Data objects are assigned interface objects automatically according to their type

SAMPLE UIDE DATA-INTERFACE MAPPINGS	
Data Object	**Interface Object**
Text String	Text Field
Boolean	Check Box
Integer	Number Field

UIDE: Specifying Interface Behavior (1)

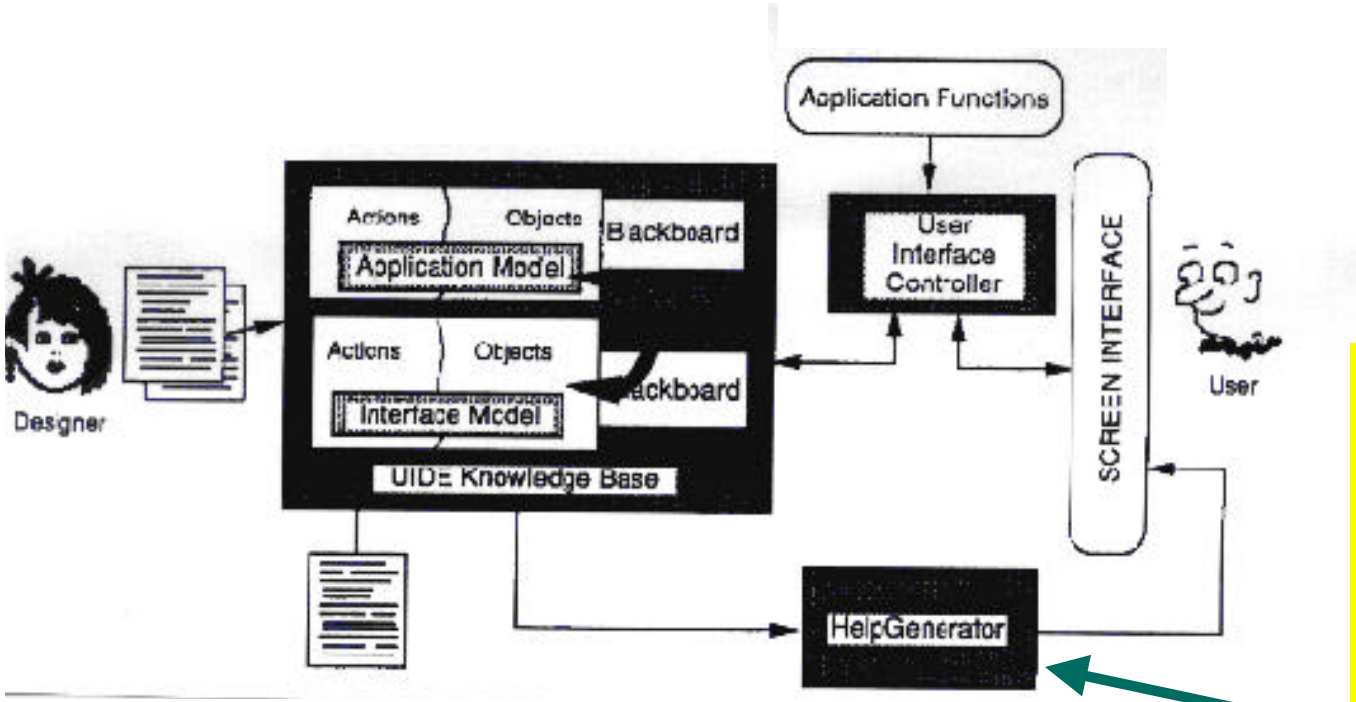
- **Behavior specification Actions specify what to do**
  - » **Actions are applied to interface objects**
  - » **Pre-conditions must be true for actions to be applied**
  - » **Post-conditions are true after actions are applied**
  - » **Constraints limit the ability of objects to execute actions**

```
Action select-graphical-object
{
        Parameter graphicalObj: PresentationObject
        Parameter applicationObj: ApplicationObject
        Pre-condition:
                "exist(graphicalObj, PRES-OBJ)&
                status(graphicalObj, VISIBLE)"
        Post-condition:
                "status(graphicalObj, HIGHLIGHTED)"
}
```

UIDE: Specifying Interface Behavior (2)

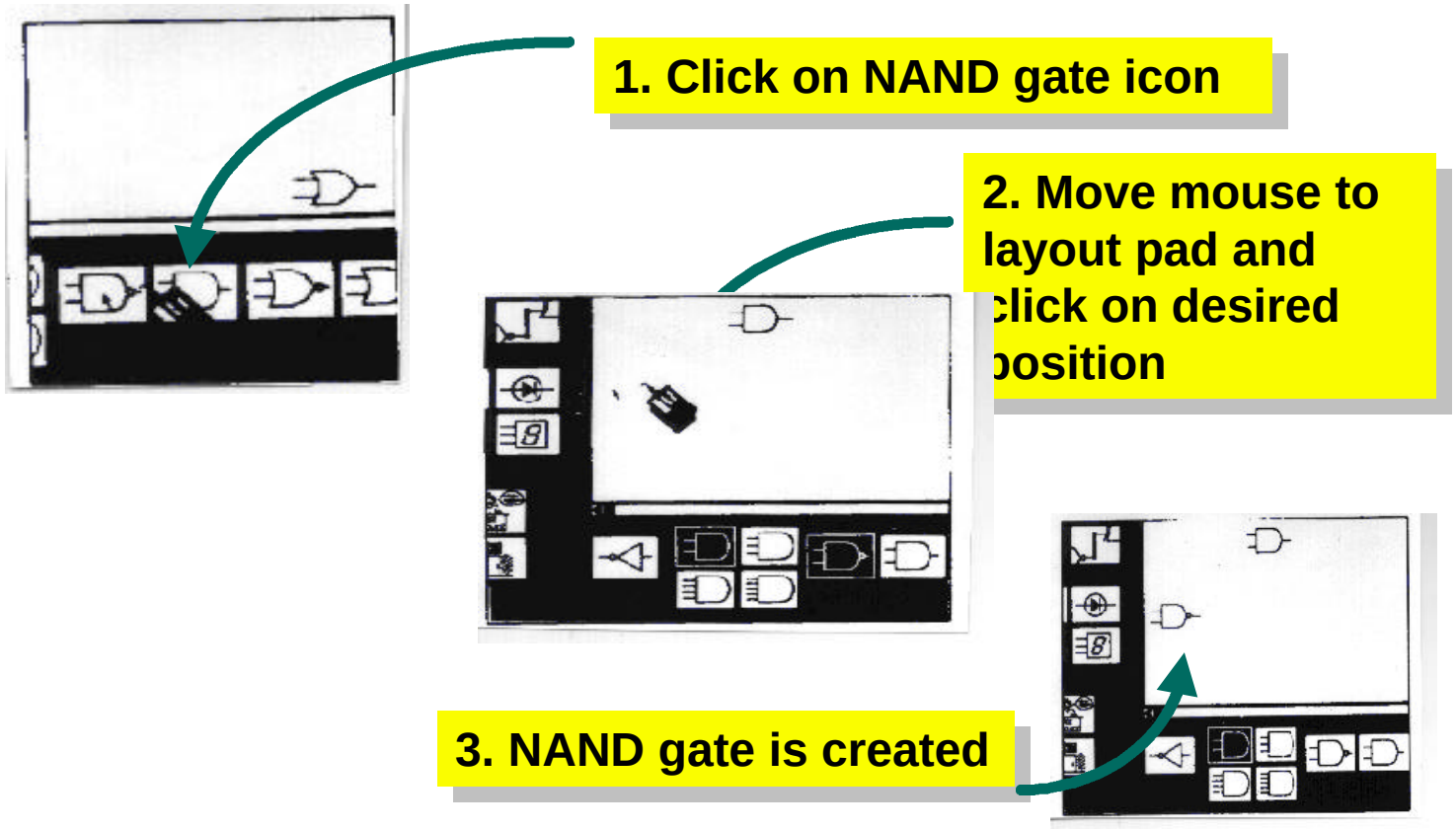
- **Interaction techniques realize actions**
  - Example: select-graphical-object by mouse click
- **Transformations provide *packaged* behavior**
  - A transformation is an algorithm that can be applied automatically to an object when requested by a designer
  - Transformations are defined in terms of actions, pre-conditions, and post-conditions
  - Transformations constitute a library of high-level, generic design paradigms

Using UIDE Interfaces: Run-Time Assistance



Help system monitors the state of the interface and builds and animates help screens by examining the interface model

UIDE: Help Animation (Create NAND gate sequence)



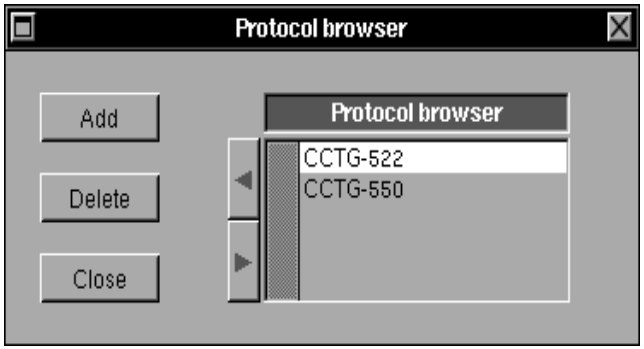
UIDE: Review

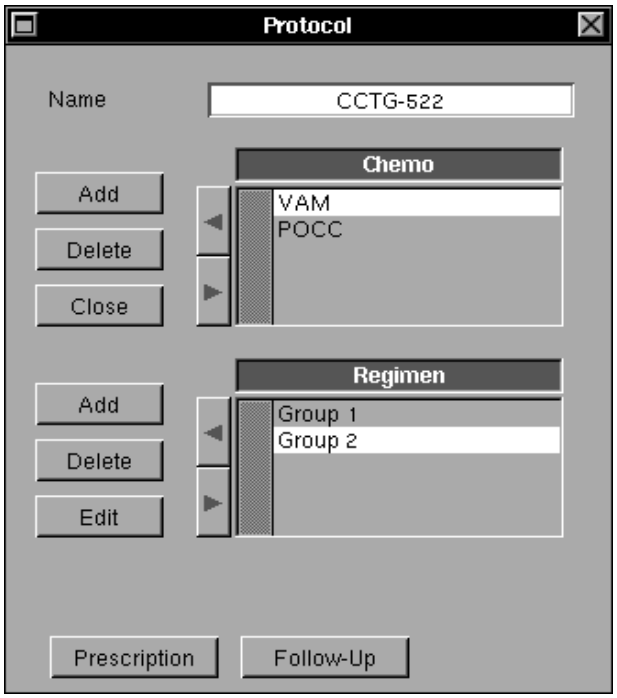
- **Benefits**
  - » Automatic interface sequencing control
  - » Automatic generation of animated help
  - » Automatic dialogue box generation
- **Shortcomings**
  - » Ease of use
  - » Scale up to large applications

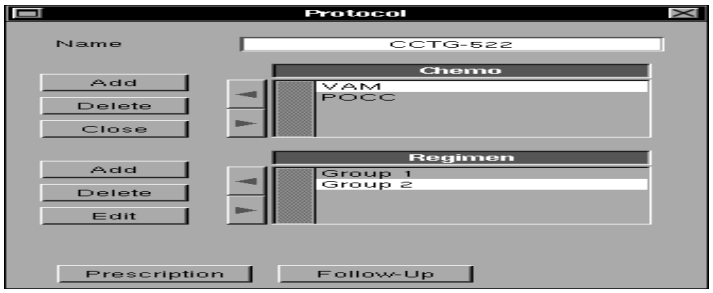
Case Study: Mecano [Puerta 94]

- **Environment to *automate* interface design**
  - » **Tools generate layout *and* behavior specifications**
  - » **Designers custom-tailor generated designs**
- **Interface design process involves**
  - » **Developer specifies application domain model**
  - » **Tools generate automatically interface specifications**
- **Target interfaces:**
  - » **Form and graph-based interfaces**

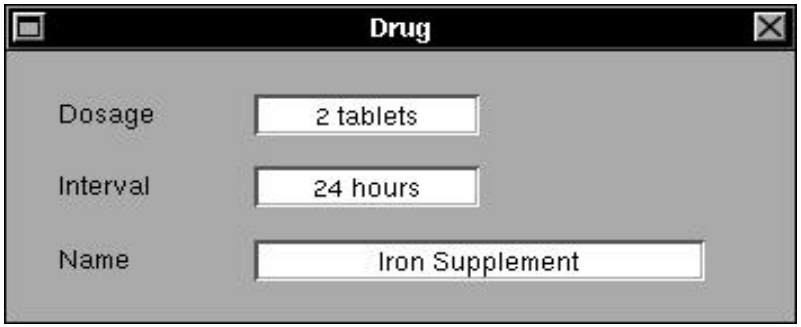
Mecano Target Interfaces (1)





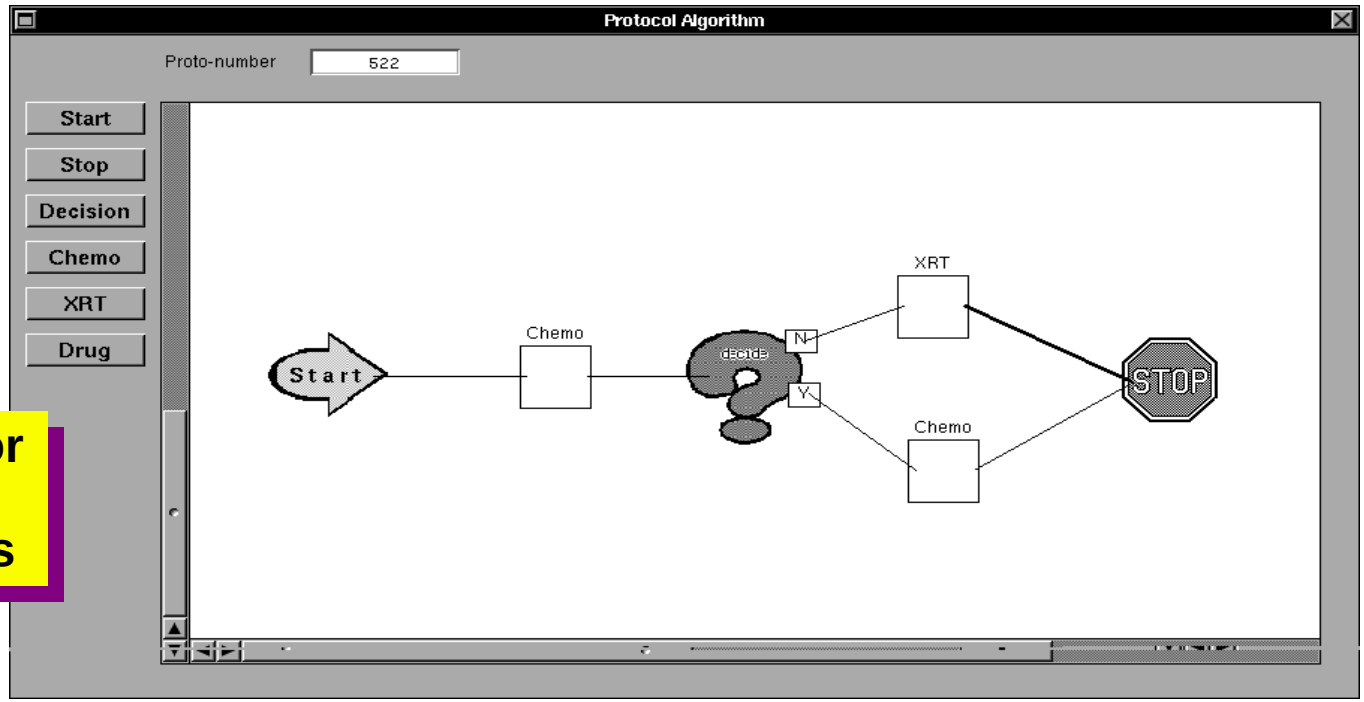


Entry forms for medical treatment specification interface (over 60 windows total)



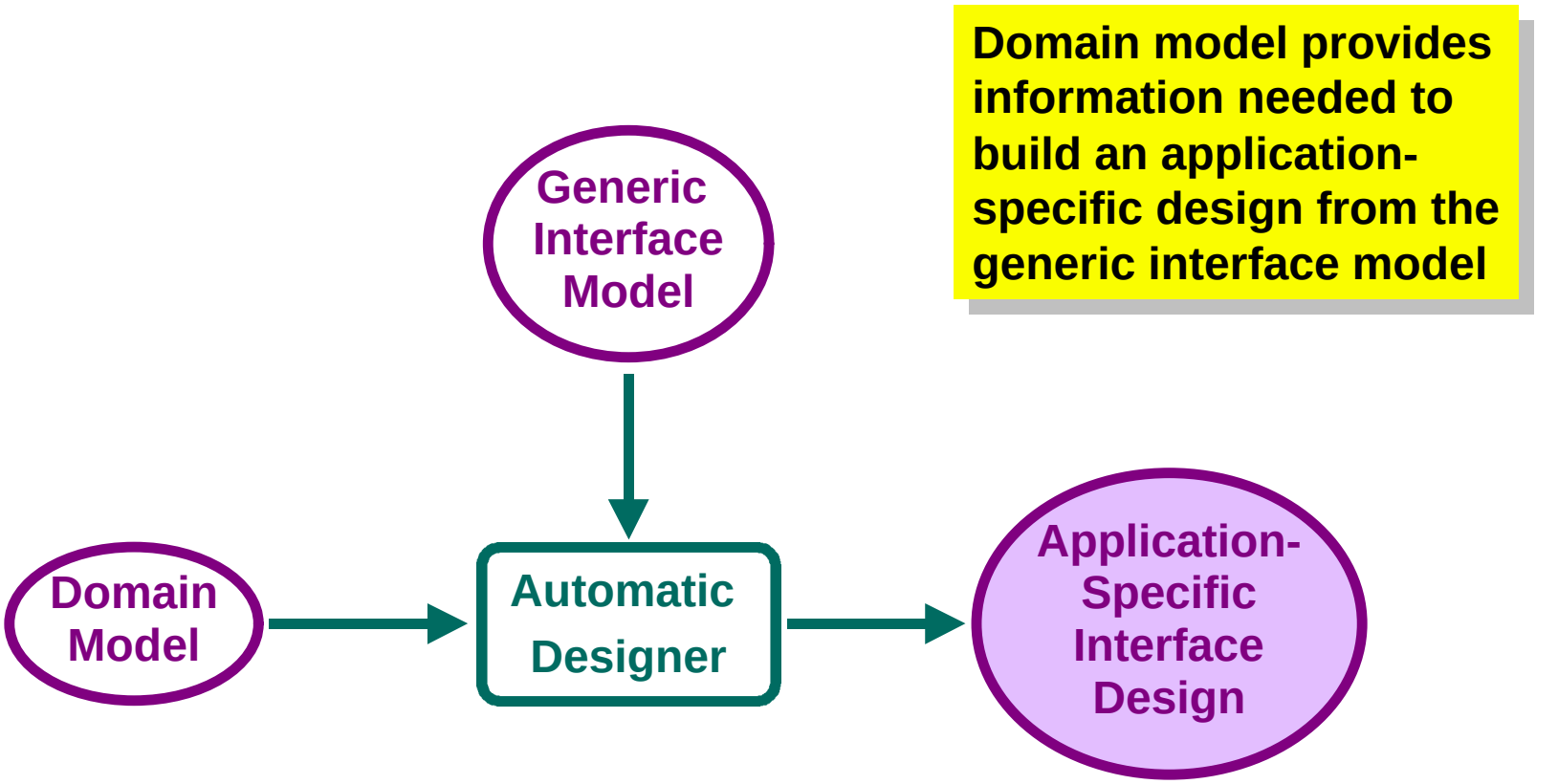
Mecano Target Interfaces (2)

- Graphical editors are domain-specific
- Users can only connect graphical objects as allowed in the domain



Graphical editor for specification of medical treatments

Mecano Paradigm



Domain model provides information needed to build an application-specific design from the generic interface model

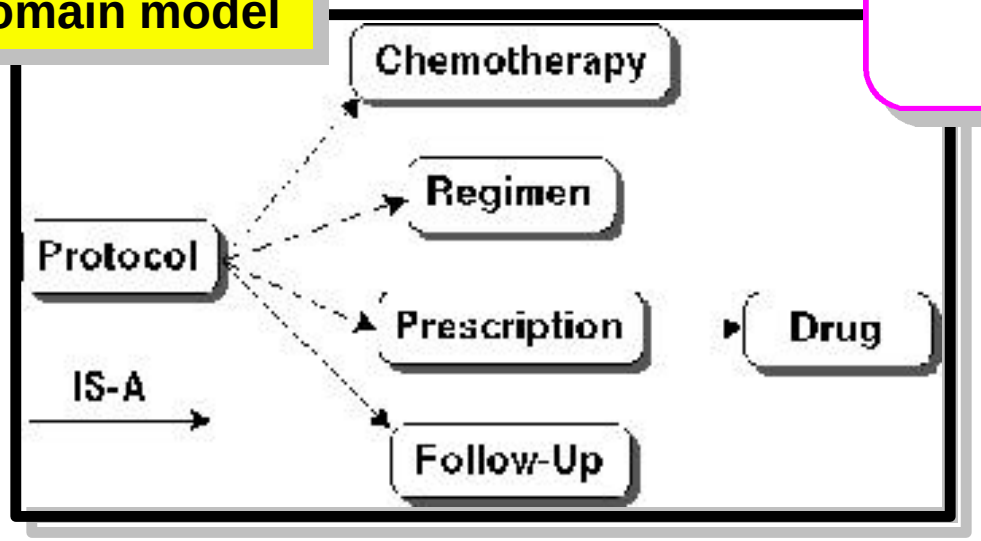
Domain Models

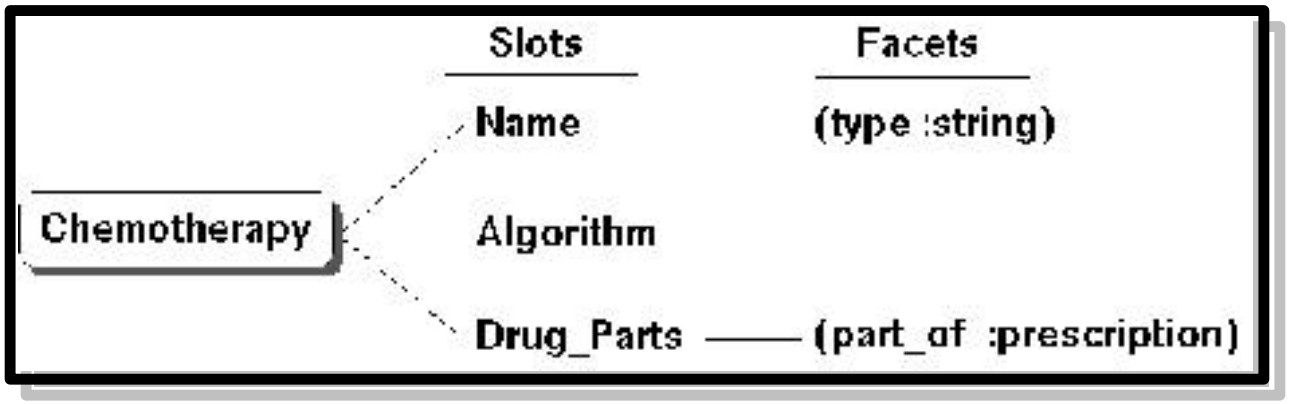
Domain model

Declarative representation of

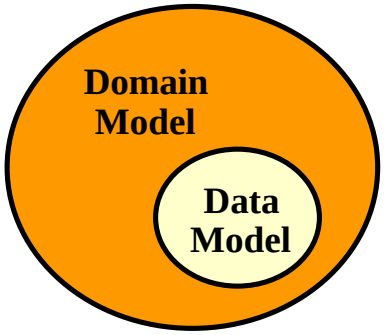
- **objects in a domain**
- **their relationships**

Clinical trial domain model



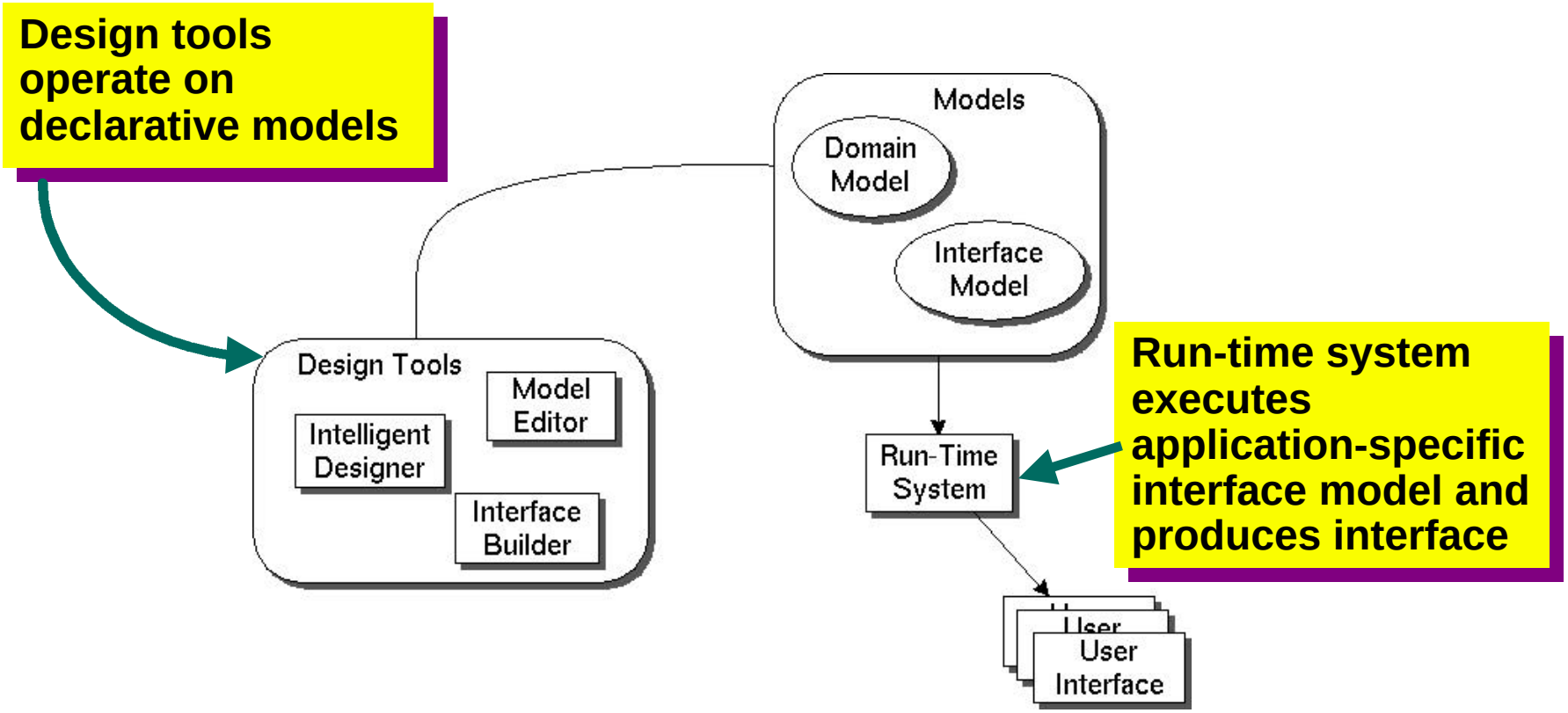


Domain Models vs. Data Models

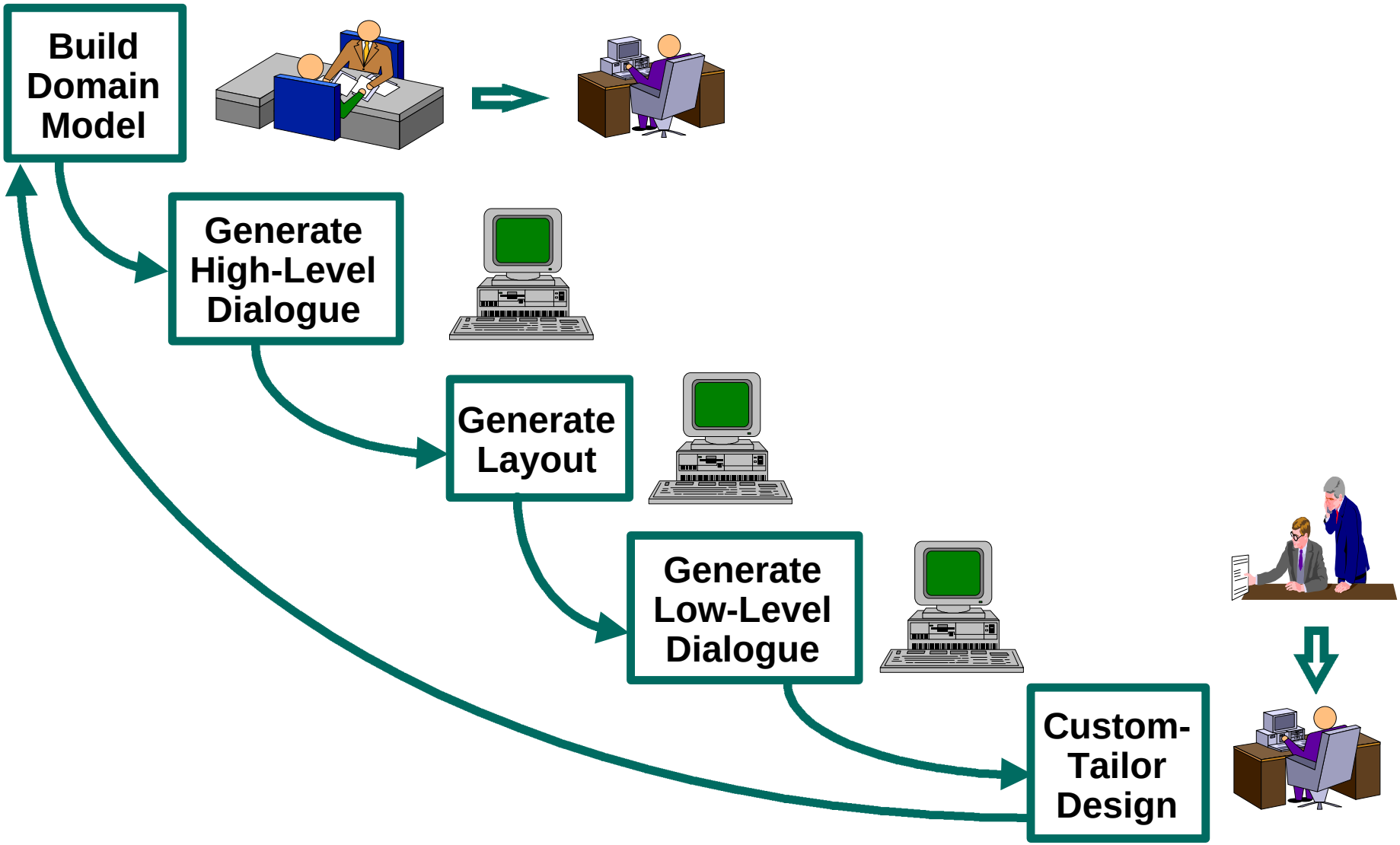


- Domain models extend data models
- Relationships among objects are made explicit and declarative
- Data models are useful only for automatic layout generation
- Domain models are useful for automatic layout and behavior generation

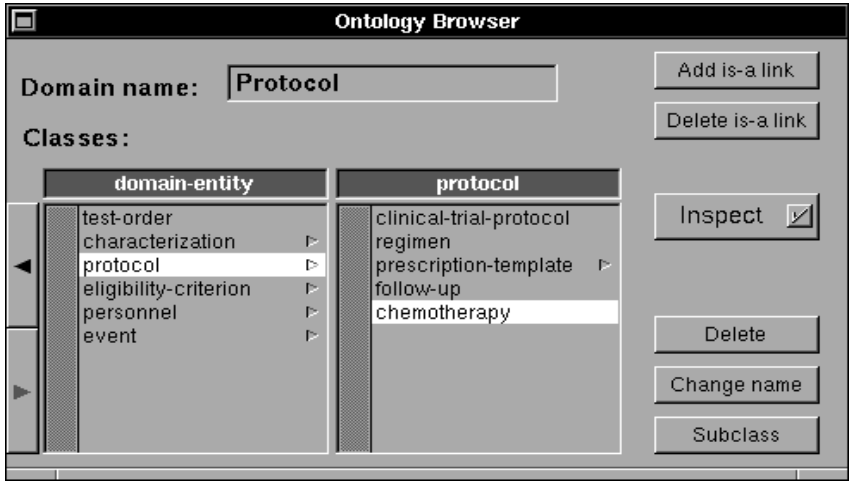
Mecano Architecture

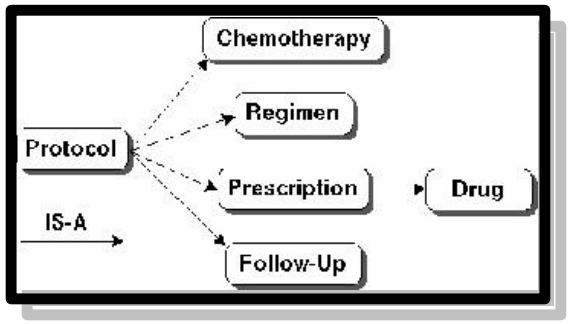


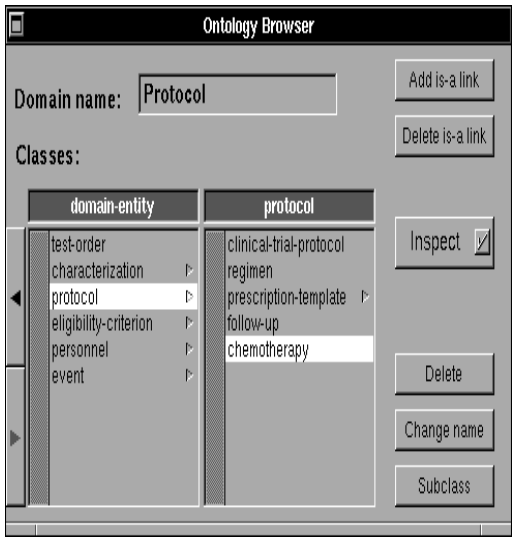
Designing Interfaces with Mecano

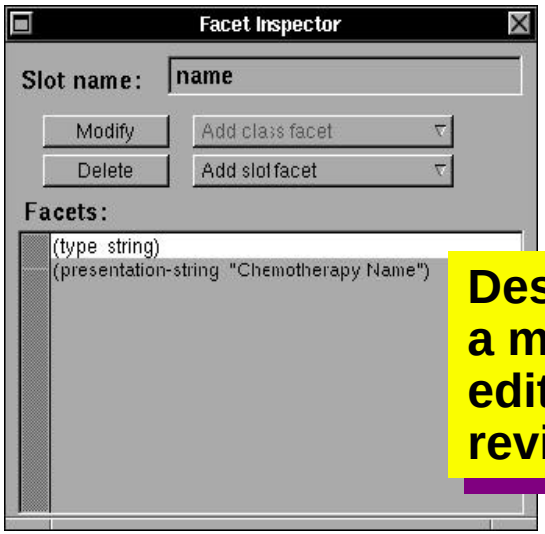


Mecano:
Building Domain Models







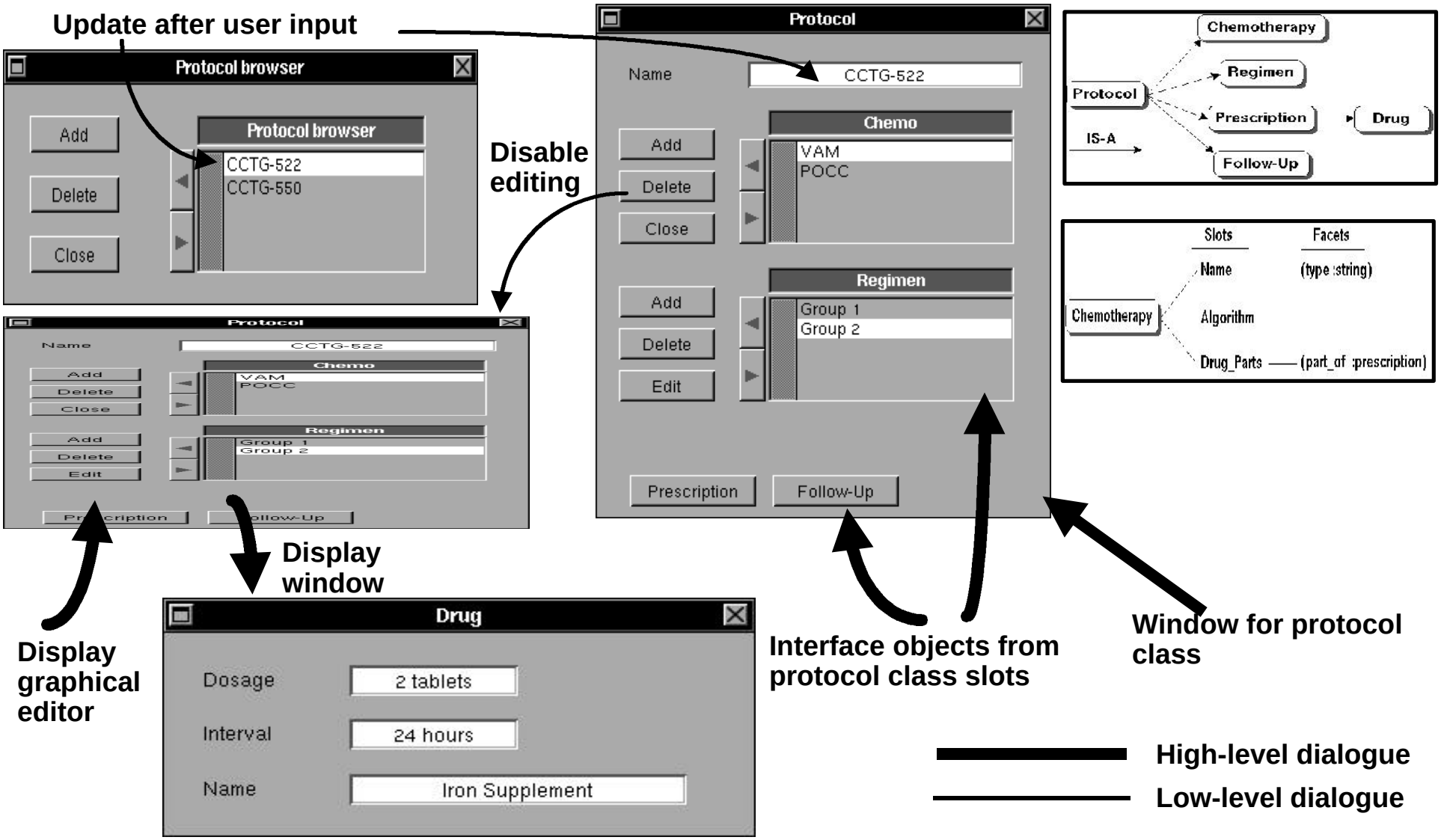


Designers employ a model browser editor to build and review models

Mecano: Generating Dialogue [Eriksson 94]

- Relationships in the domain model determine dialogue
- High-level dialogue
  - » Display decomposition into windows
  - » Mapping of data objects to interface components
  - » Window navigation
  - » Example: using IS-A relationships to determine window navigation
- Low-level dialogue
  - » Side effects of user actions
  - » Domain-specific constraints on user input
  - » Example: using PART-OF relationships to determine graphical object connectivity

Mecano Example: Dialogue Generation

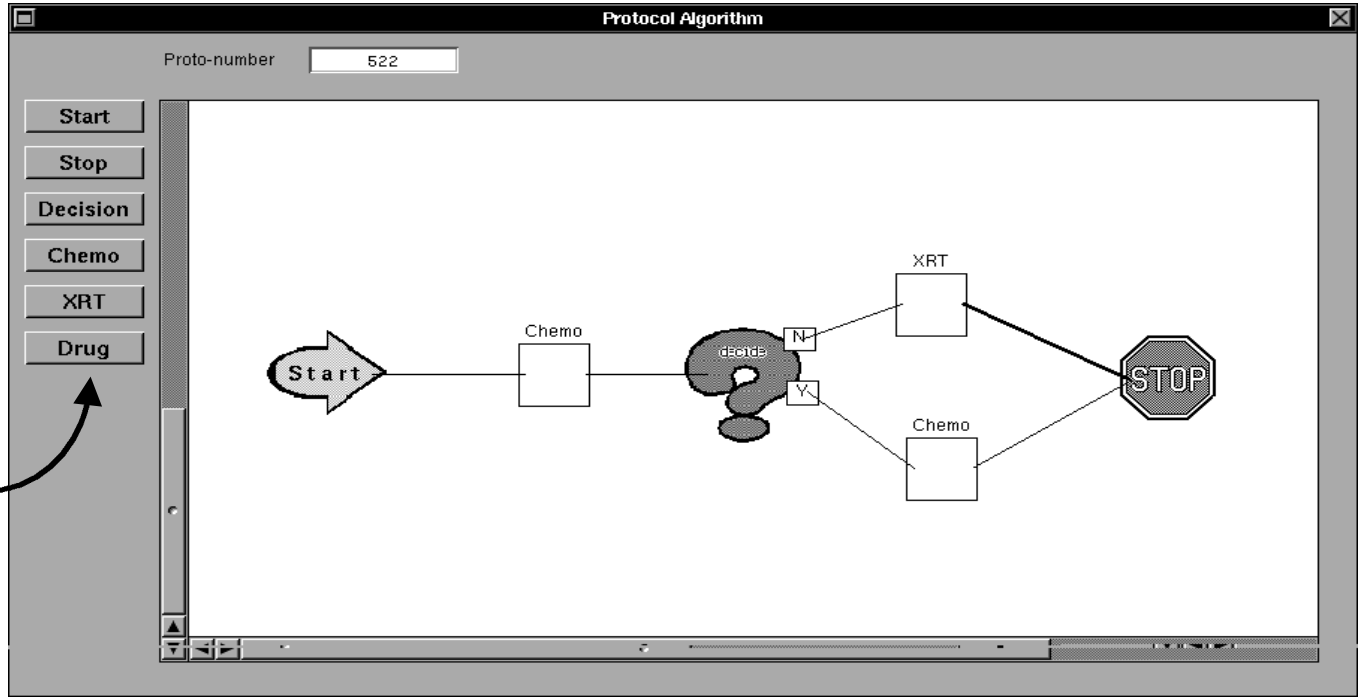


Mecano Example: Domain-Specific Graphical Editor Generation

```
(slot algorithm
      (type :procedure)
      (allowed-classes :xrt :chemotherapy :drug))
```

Automatic designer maps type "procedure" to a graphical editor

Automatic designer creates canonical graphical objects for allowed classes and defines corresponding push buttons



Mecano Review

- **Benefits**
  - » Highly automated design environment
  - » Allows automatic generation of dynamic behavior
  - » Couples application design and interface design
  - » Supports development of large-scale interfaces as well as prototypes
- **Shortcomings**
  - » Automatic nature reduces design space
  - » Lacks a task model component (under development)
  - » Dynamic behavior generation is limited to certain types

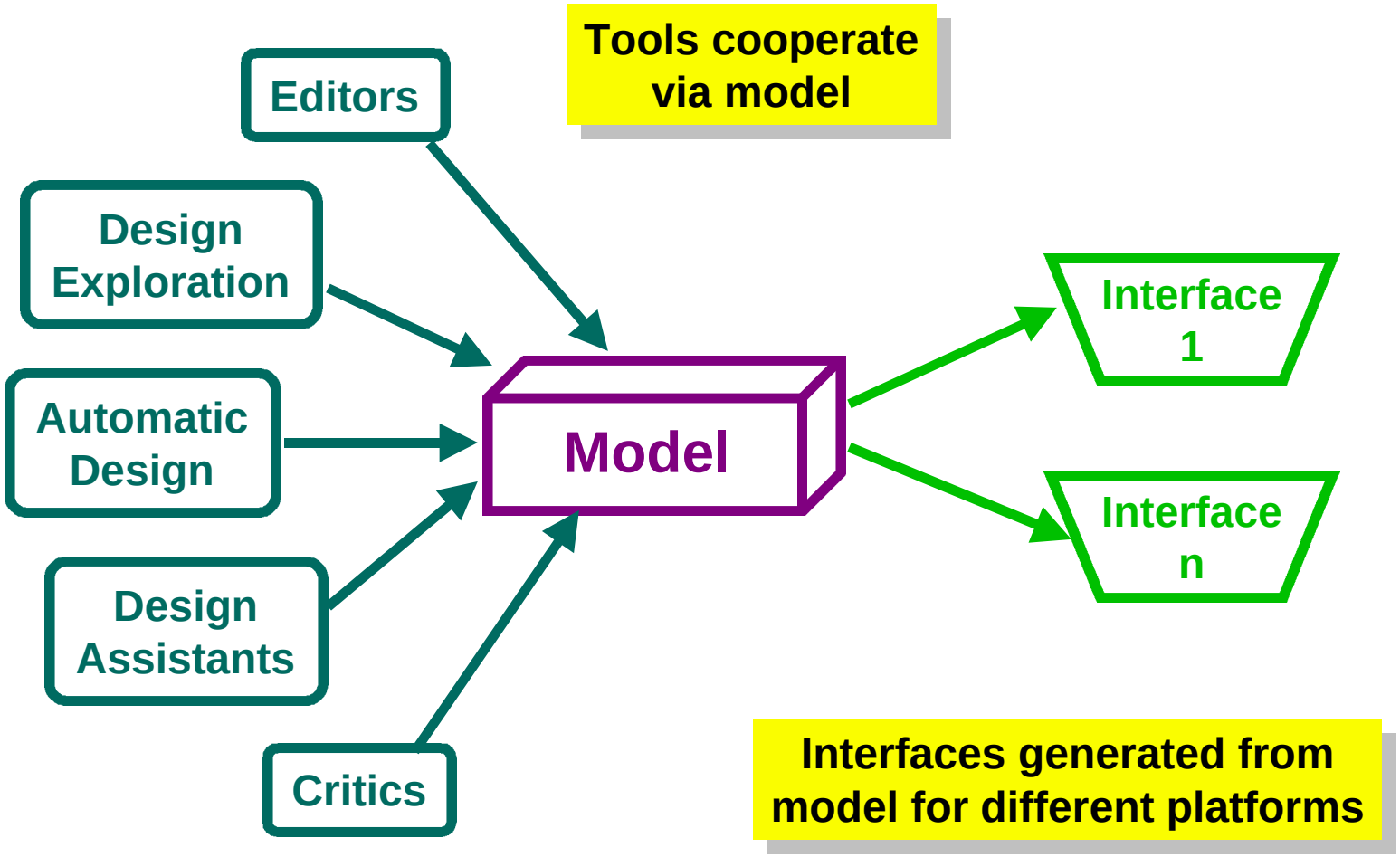
Agenda

- Model-based paradigm
- Case studies: UIDE, Mecano
- Architectures
- Break
- Case studies: Humanoid, ITS
- Survey of Model-Based Tools
- Conclusions
- Questions

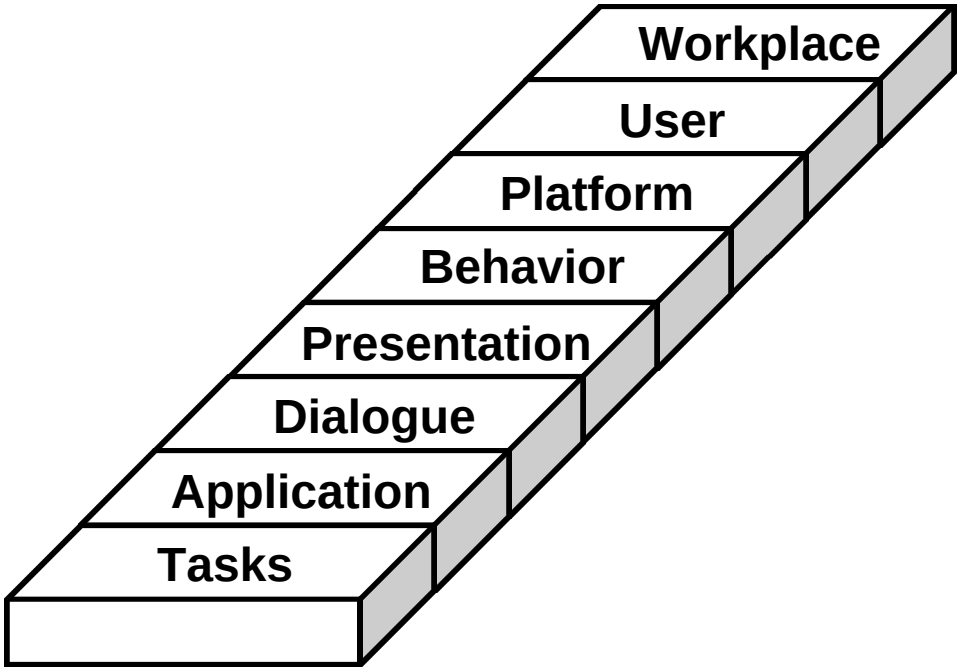
Architectures for Model-Based Interface Development

- Basic architecture
- Model contents
- Full architecture

Model-Based Approach



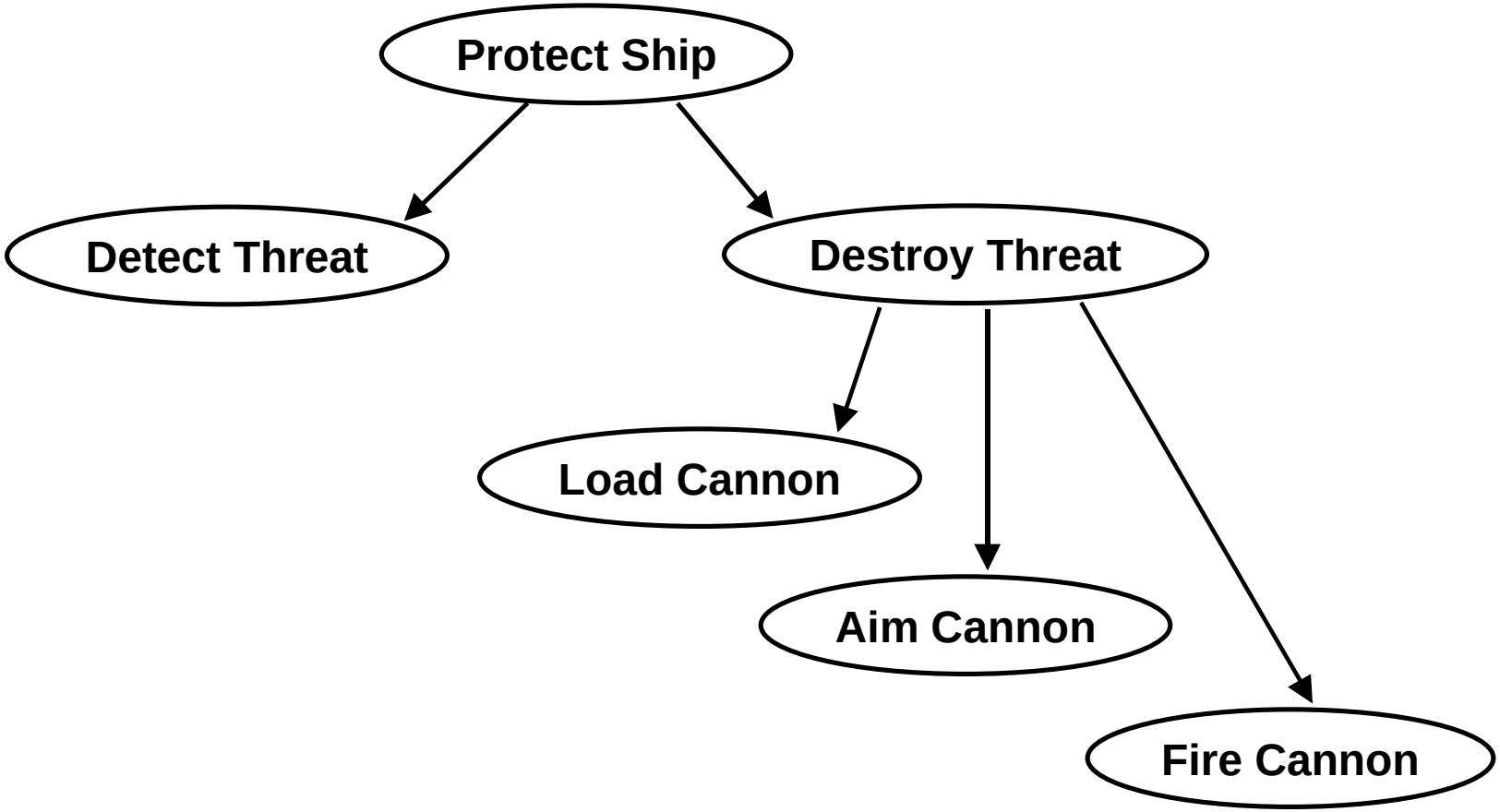
Model


Workplace
User
Platform
Behavior
Presentation
Dialogue
Application
Tasks

Task Model

- **Specification of tasks users perform**
  - » **Goals: specifies when a desired state is met**
  - » **Methods: procedures to achieve a goal**
    - – **Atomic methods achieve goals in one step**
    - – **Composite methods decompose goals into subgoals**
- **Task models result from task analyses**
  - » **GOMS [Card 83], TKS [Johnson 94]**
- **Task models research**
  - » **ADEPT [Johnson 94]**
  - » **GOMS [Kieras 85] [John 92]**

Task Model Contents Example



Why Model Tasks?

- **User centered design:**
  - » understand what users want to do
  - » understand how they do it
- **Benefits of task models**
  - » Enable automatic help generation
    - – Animations showing how to complete tasks [Sukaviriya 90]
  - » Enable automated design critics
    - – Execution and learning time estimates
  - » Lay foundation for design of an application

Application Model

- **Specification of services applications provide**
  - » **Objects: capture state of world entities**
  - » **Operations: change the state of objects**
- **Operations = primitive methods of task model**

Application Model Contents

- **Objects**
  - » Type
  - » Slots

```
Cannon: Device
        loaded: Boolean
        aim: Coordinate
```

- **Operations**
  - » Preconditions
  - » Inputs
  - » Actions
  - » Postconditions

```
Load (c: Cannon)
        precondition: not (loaded (c))
        postcondition: loaded (c)
        actions:
                c.loaded = true
```

```
Fire (c: Cannon)
        .....
```

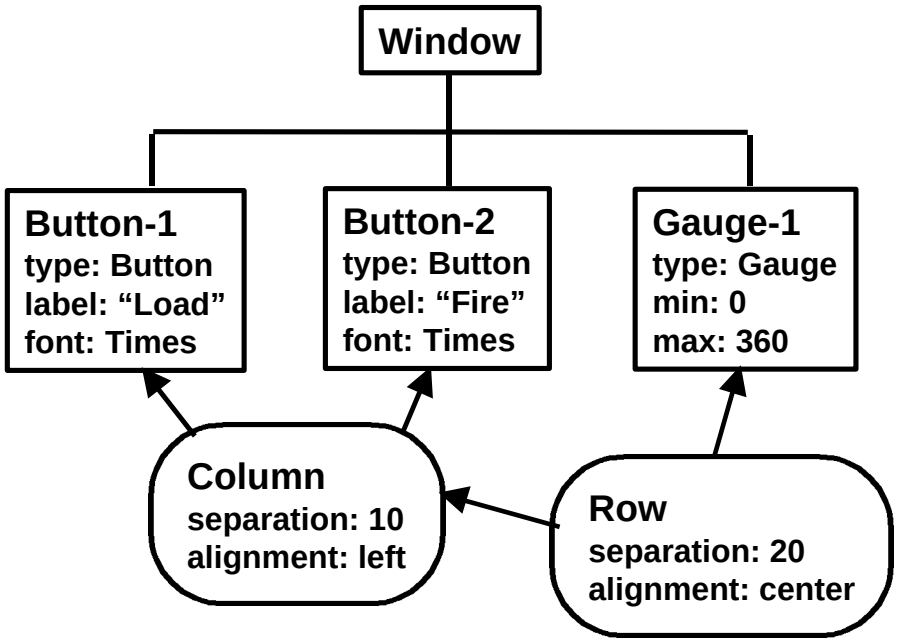
Dialogue Model

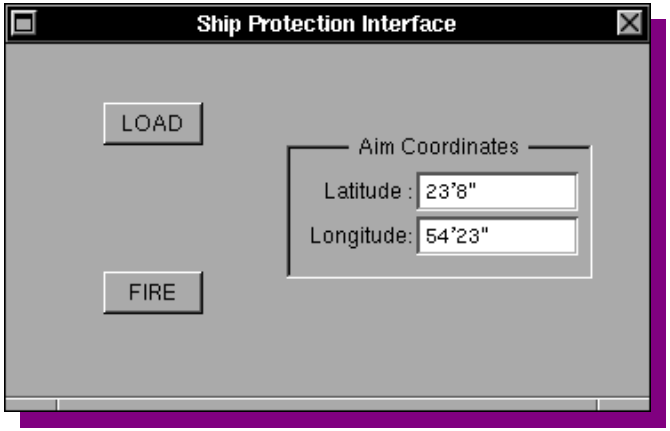
- Specification of human-computer conversation
- Specification of when computer can
  - » Query user
  - » Present information
- Specification of when user can
  - » Invoke commands
  - » Select or specify inputs
- Example: Load, Fire Cannon
  - » “Fire” button disabled until “Load” button is pressed

Presentation Model

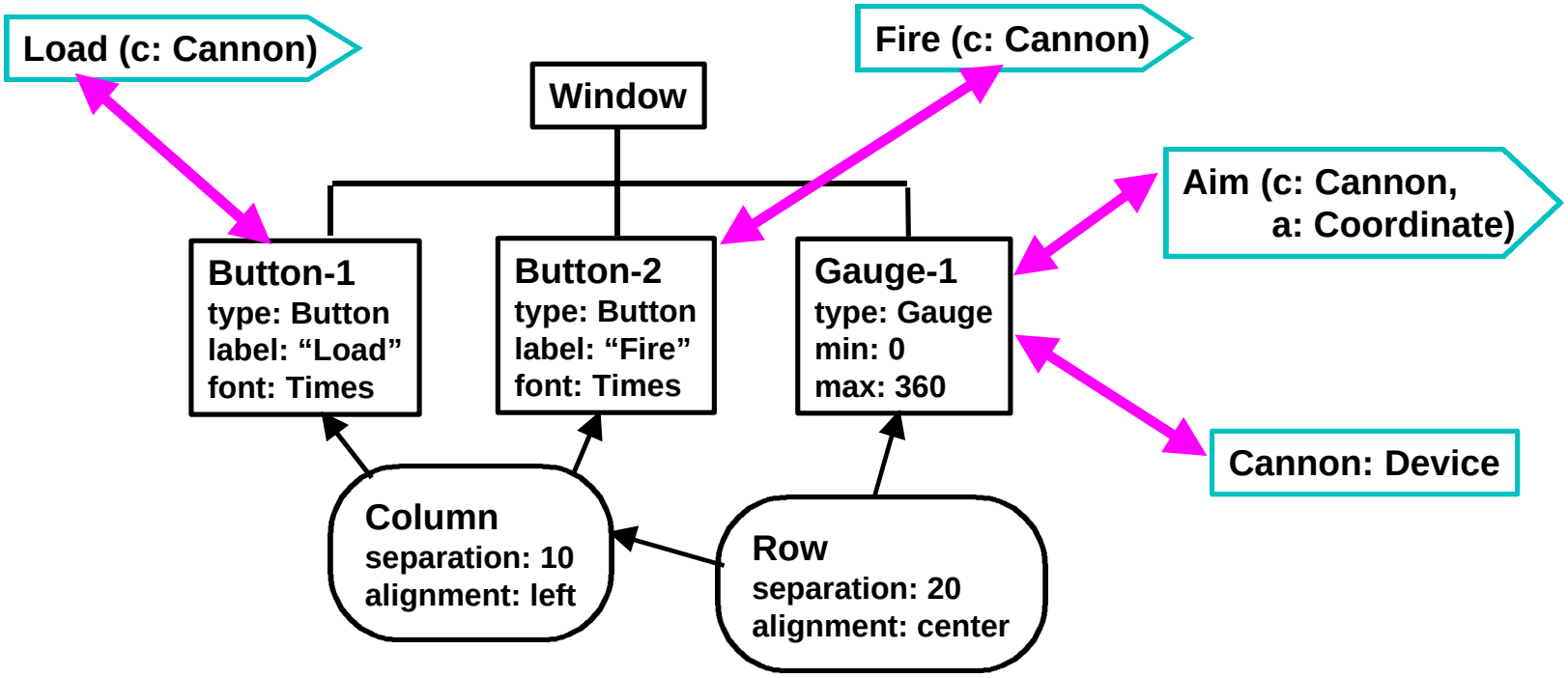
- **Specification of object & operation appearance**
  - » **Hierarchical decomposition of display into components**
  - » **Presentation medium of components**
    - – **Screen, speech output, sound, ...**
  - » **Component attributes**
    - – **Type (text, icon, graphic, etc), color, size, font, etc.**
  - » **Layout of components**
    - – **Row, column, graph, table, application-specific**

Presentation Model Contents





Presentation/Application Model Relationship

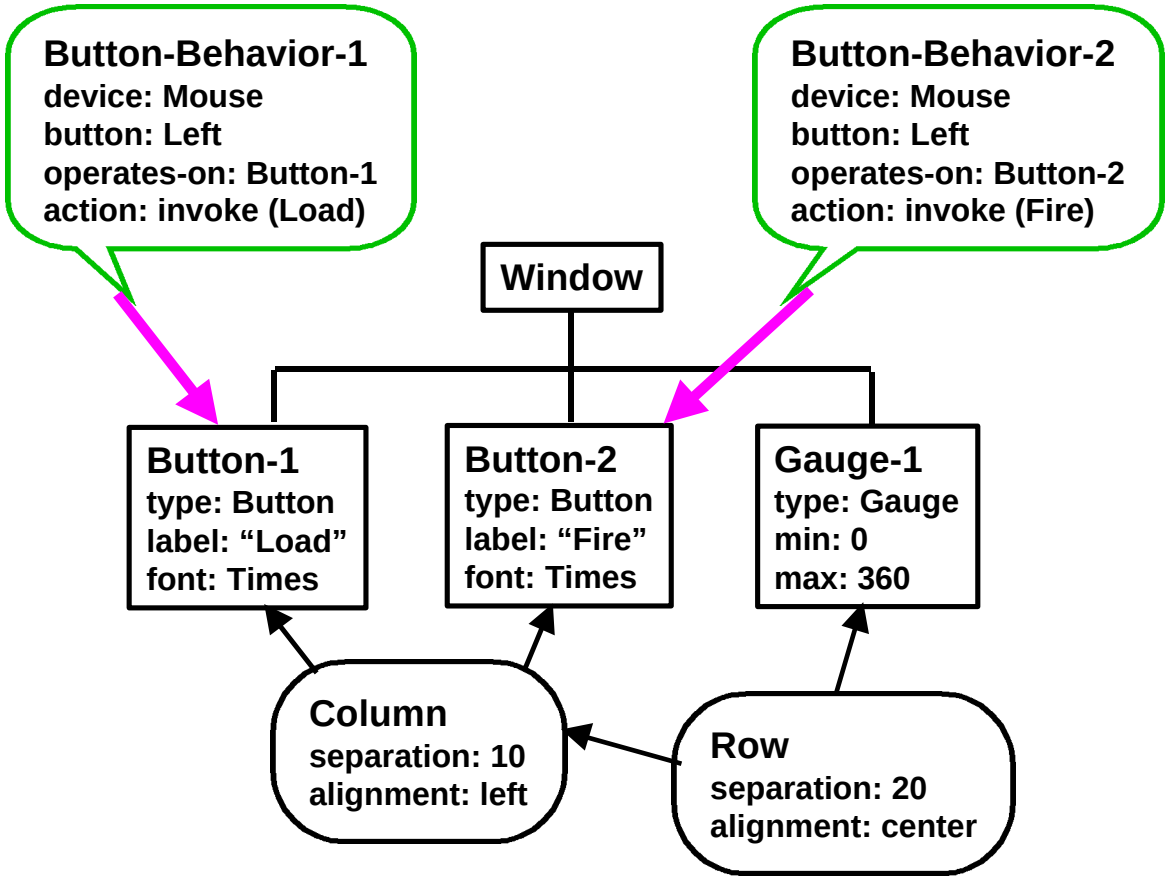


Behavior Model

- **Specification of input behavior**
  - » **Presentation components where applicable**
  - » **Behavior medium/device**
    - – **Keyboard, mouse, pen, voice, ...**
  - » **Behavior attributes**
    - – **Mouse: which button, gesture kind (click, drag), ...**
    - – **Keyboard: which key, modifiers**
    - – **...**
  - » **Behavior actions**
    - – **Invoke operation**
    - – **Set operation input**
    - – **...**

Button-Behavior-1
device: Mouse
button: Left
operates-on: Button-1
action: invoke (Load)

Behavior/Presentation Model Relationship



Platform

- Specification of platform characteristics
  - Input devices
    - Mouse: No. buttons, speed, ...
    - Keyboard: keys, modifiers, function keys, ...
    - Pen: buttons, pressure, ...
    - Glove: degrees of freedom, ...
  - Output devices
    - Screen: resolution, colors, speed
    - Speaker: quality, stereo
  - CPU
    - Speed, memory & disk size
  - Networking
    - Latency, bandwidth

User Model

- **Specification of user characteristics**
  - **task experience**
  - **application experience**
  - **system experience**
  - **use of other systems**
  - **typing skills**
  - **motivation**
  - **computer literacy**
  - **frequency of use**
  - **....**

[Wilson 93]

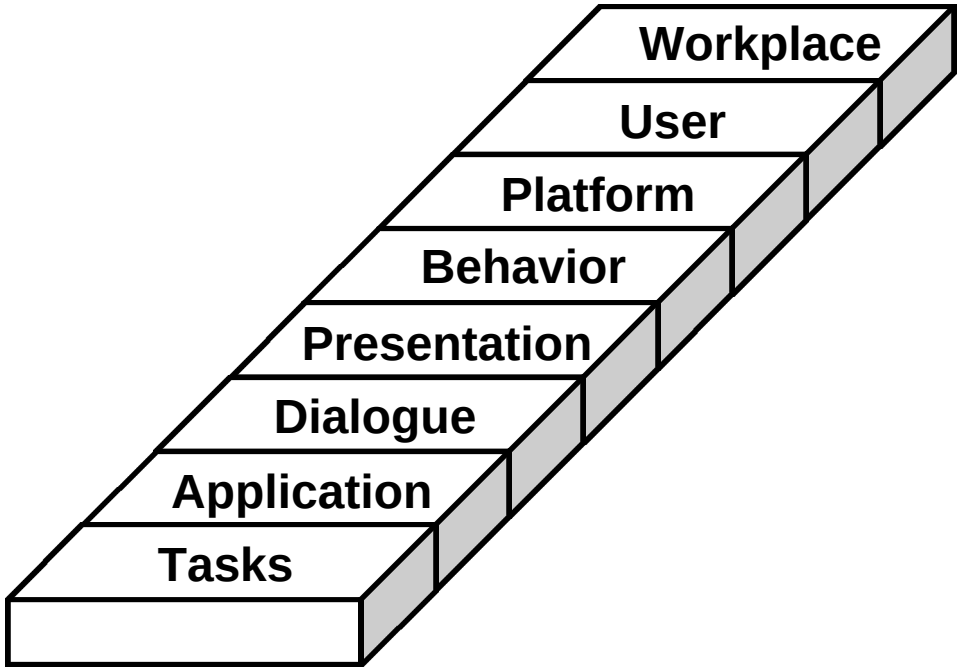
Why Model Users?

- Reconfigure presentation & behavior to user
  - ADEPT [Johnson 93]
- Provide appropriate level of help
  - [Moore 90]
- Actively tutor user during interaction
  - Guidon [Clancey 79]
  - West [Burton 81]
  - Sophie [Brown 75]

Workplace Model

- **Specification of workplace characteristics**
  - » system use: mandatory, optional
  - » turnover rate: high, moderate, low
  - » organization role: manager, clerical
  - » Environment factors
    - – Noise level
    - – Light level
  - » Cultural characteristics
    - – Meaning of colors, words, icons
- **Example**
  - » CPU Connectix Powerbook Utilities
    - – Settings for home, office, travel

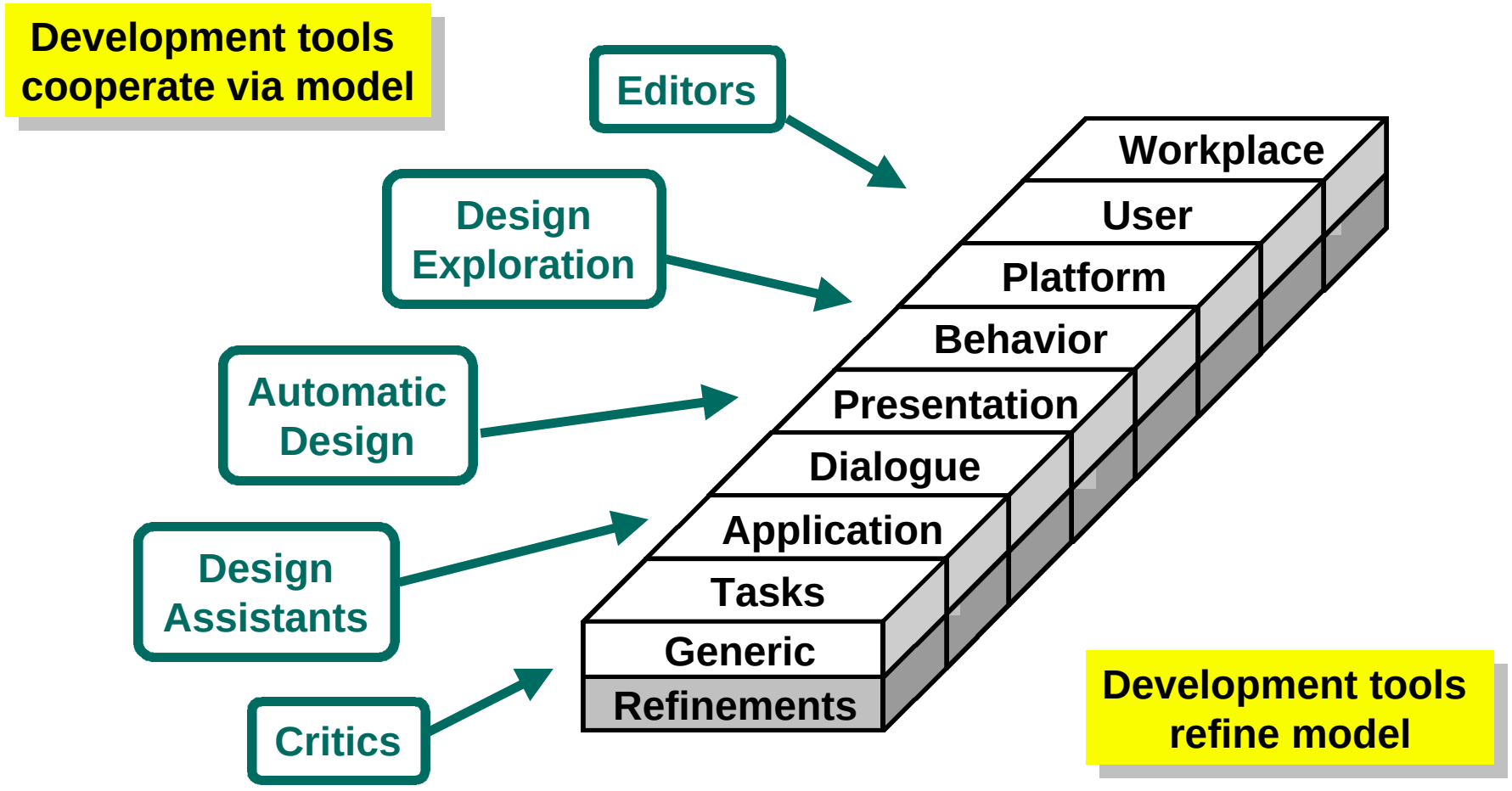
Model Components Summary



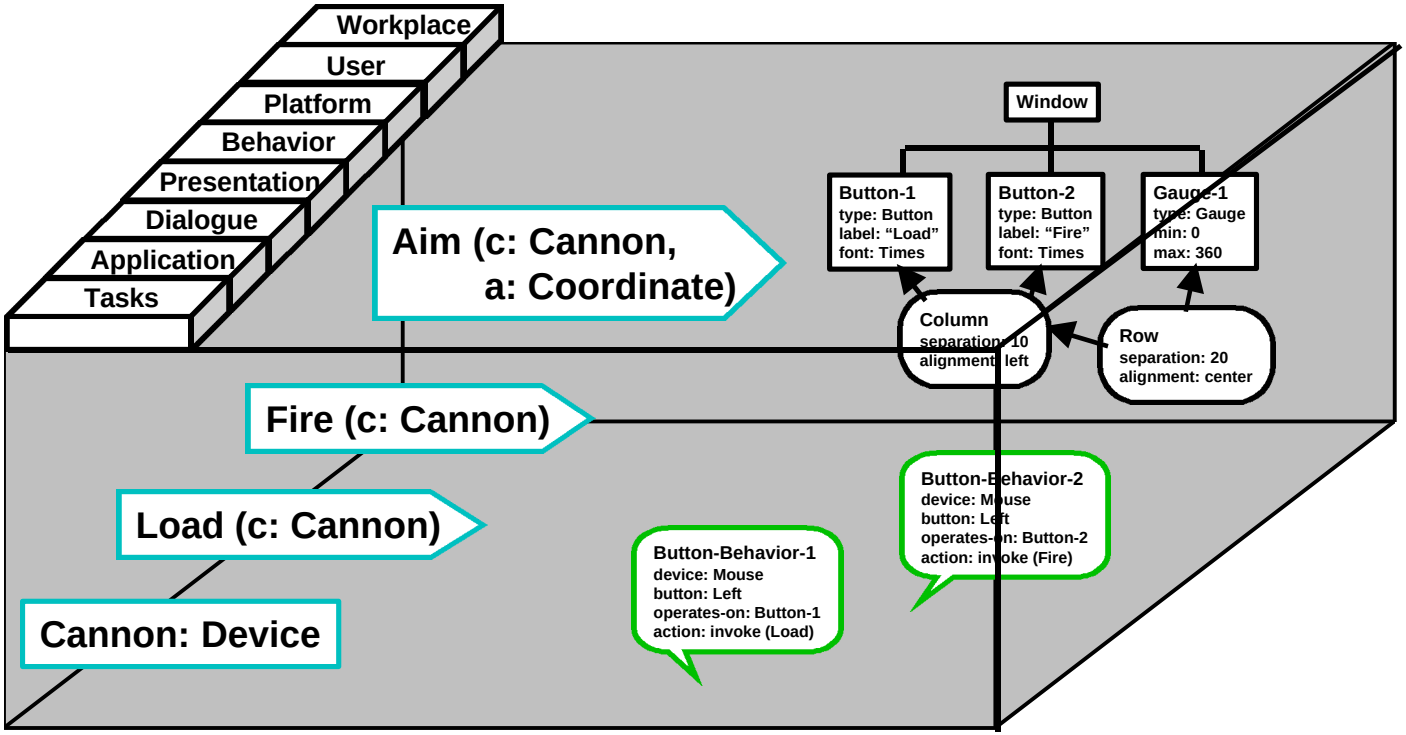
Model components:

- **define a vocabulary for specifying arbitrary interfaces**
- **provide a reusable framework for developing interfaces**

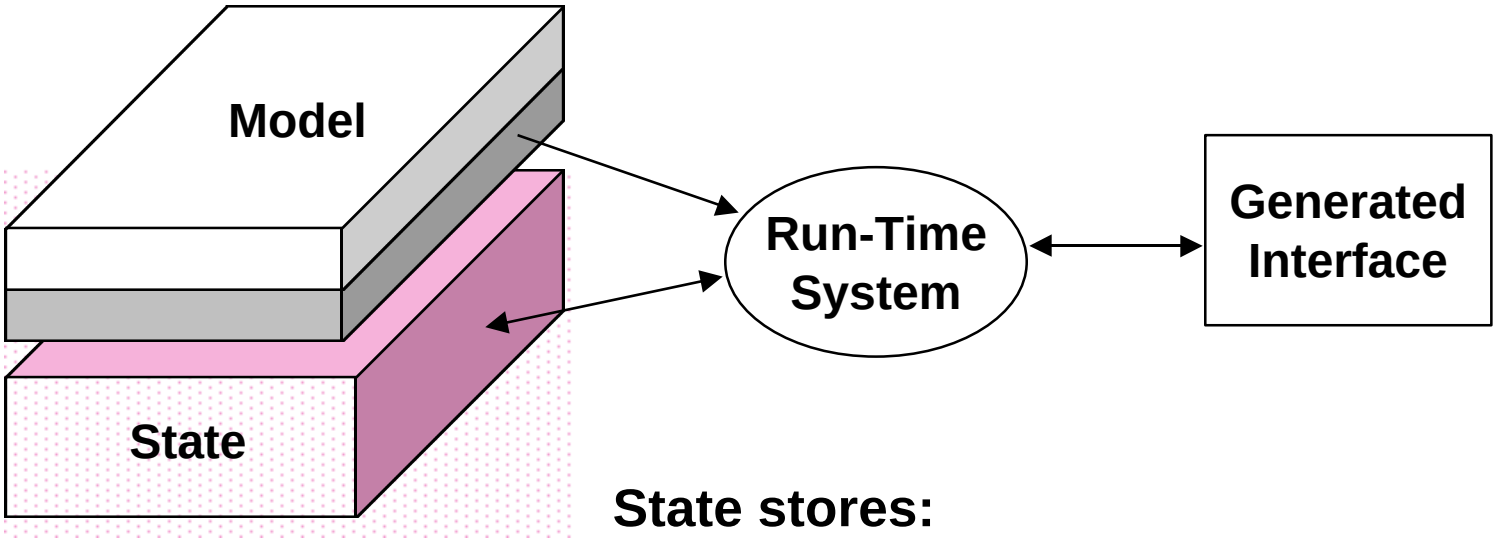
Development Architecture: Model Refinement



Model Refinement Example



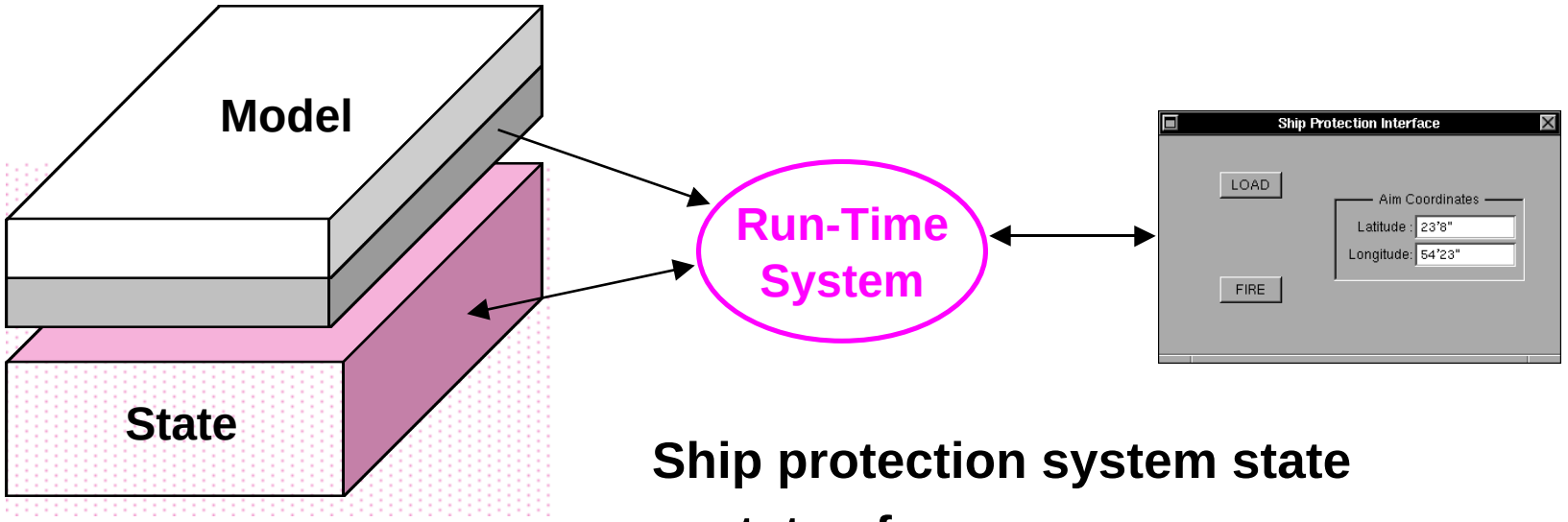
Application Architecture



State stores:

- object instances
- instances of presentation components
- dialogue state
- current user characteristics
- interaction history

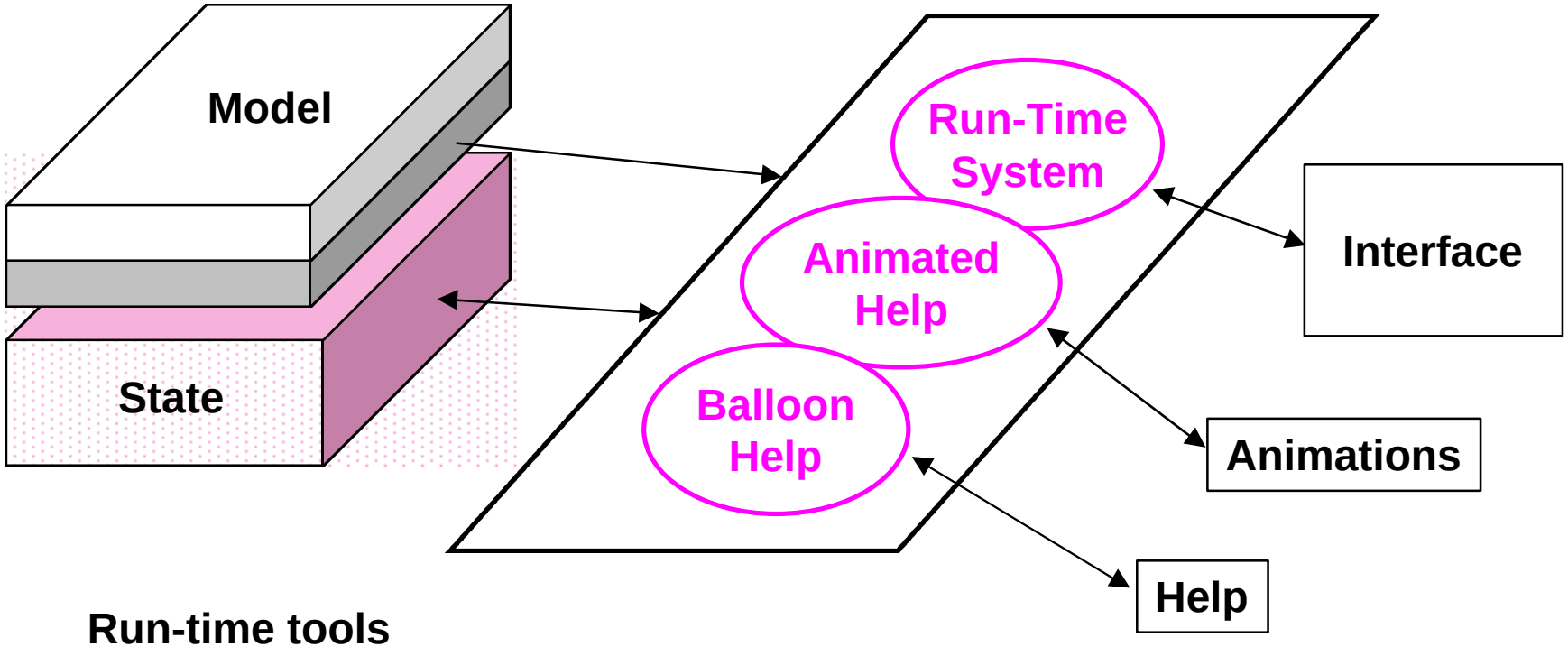
Application Architecture Example



Ship protection system state

- **state of cannon**
  - » **loaded, aim**
- **state of buttons and type-ins**
  - » **dimmed, value, location, size**
- **state of behaviors**
  - » **enabled, running**

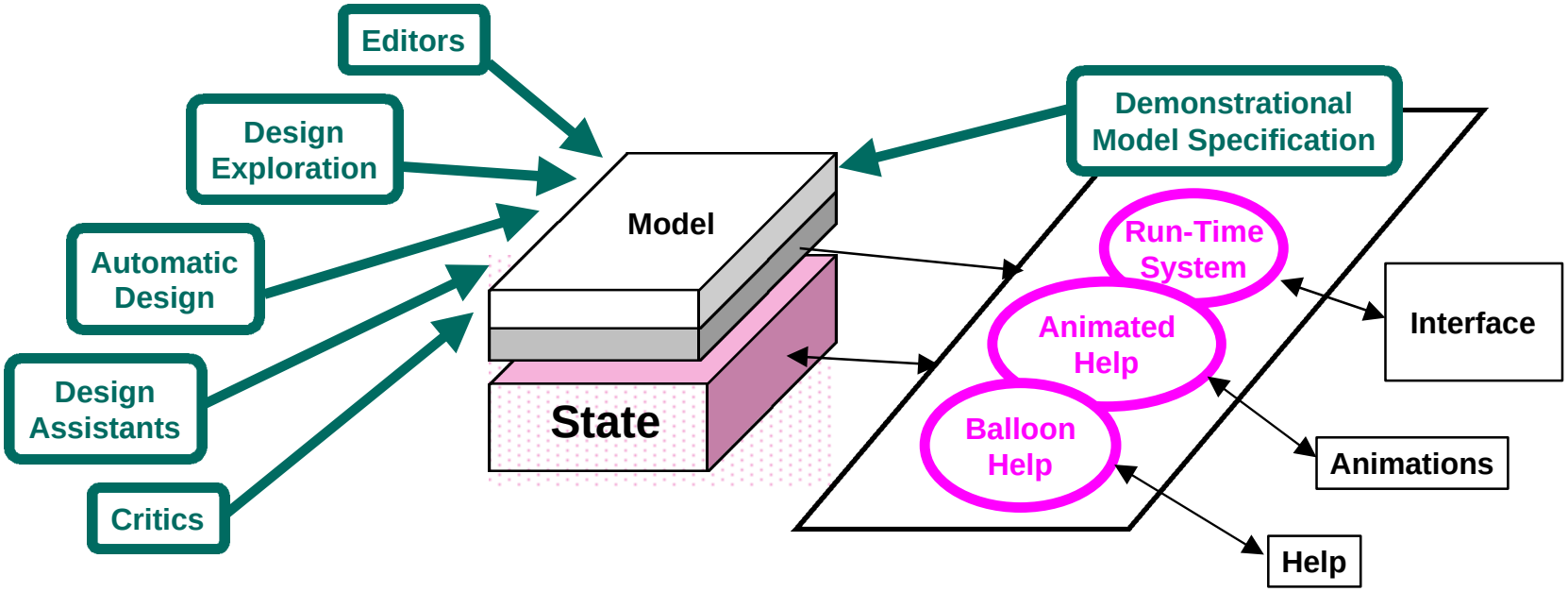
Application Architecture With Run-Time Tools



Run-time tools

- **Provide services to users**
- **Use model to analyze state**
- **Change state**

Full Development Architecture



- Iterative model refinement assisted with tools
- Immediate feedback after model changes
- Services automatically reconfigured when model is refined

Summary
Architecture

- Model specifies all aspects of interfaces
- Reusable, extensible & portable models
- Modeling tools refine models
- Run-time system generates interface
- Run-time services based on model & state

Framework for comprehensive interface execution and development environments

Agenda

- Model-based paradigm
- Case studies: UIDE, Mecano
- Architectures
- Break
- Case studies: Humanoid, ITS
- Survey of Model-Based Tools
- Conclusions
- Questions

Blank Page

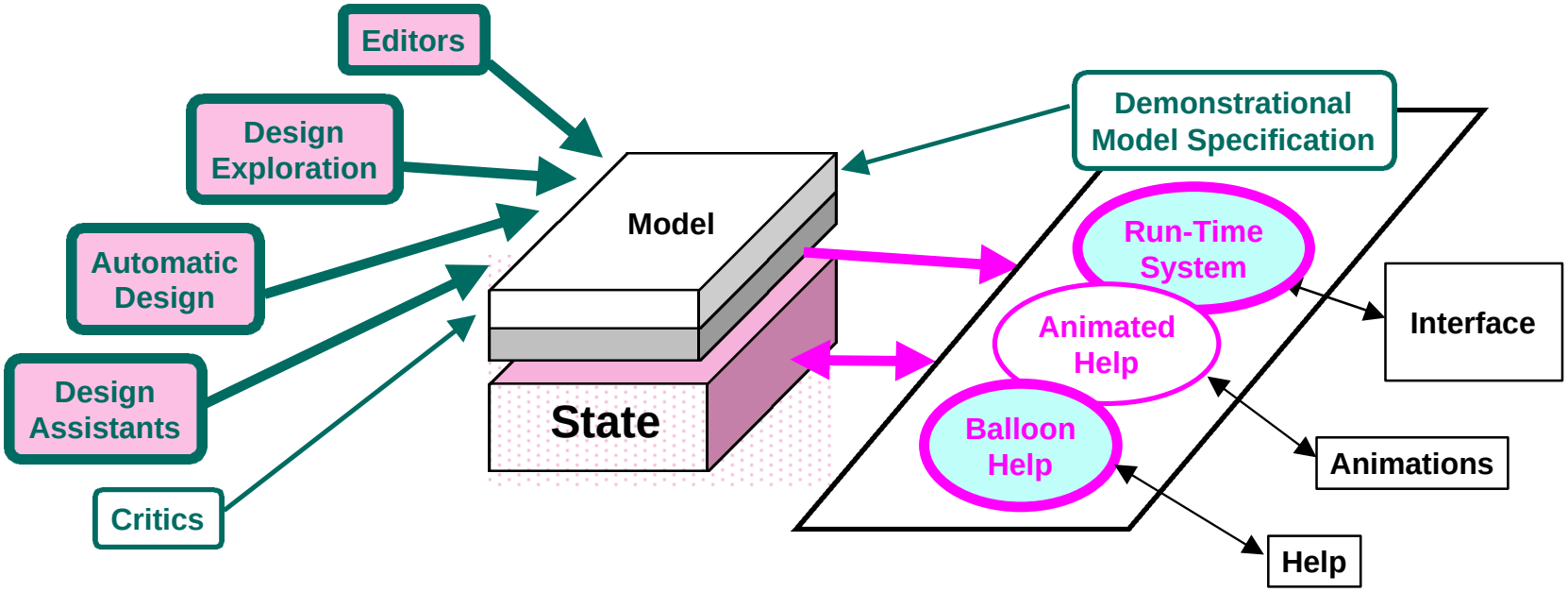
Agenda

- Model-based paradigm
- Case studies: UIDE, Mecano
- Architectures
- Break
- Case studies: Humanoid, ITS
- Survey of Model-Based Tools
- Conclusions
- Questions

Humanoid: User Interface Development Environment

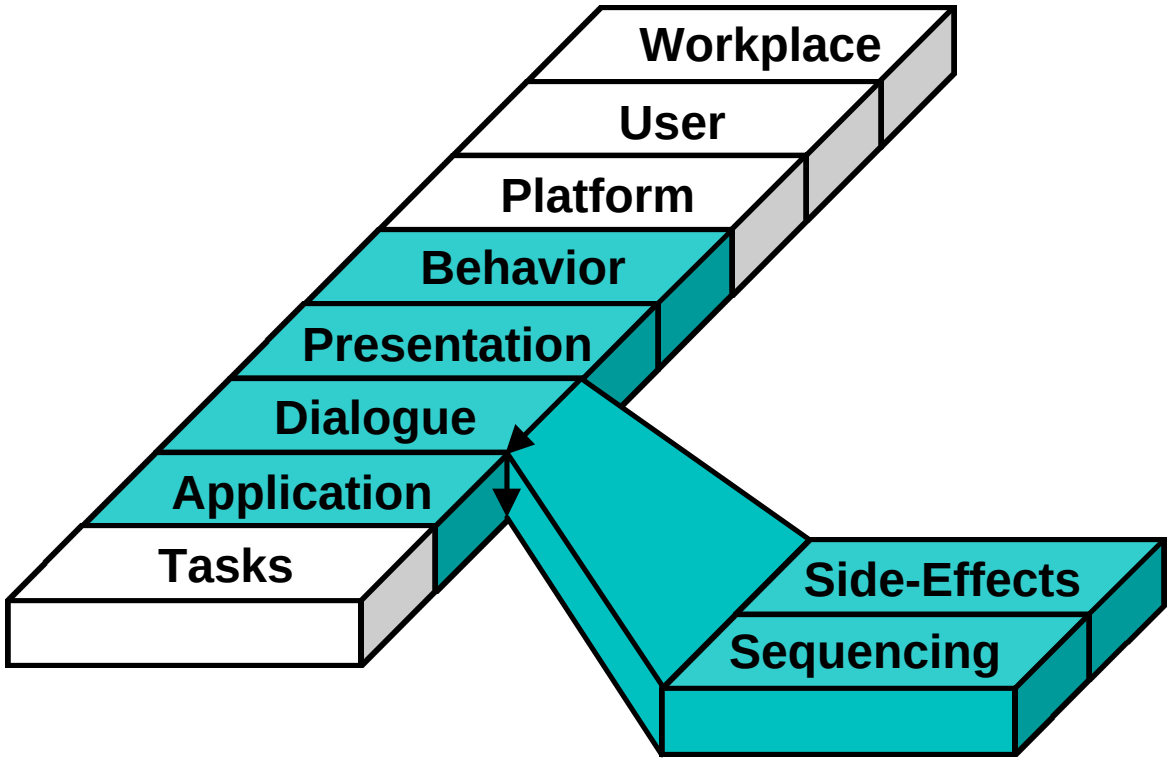
- Supports early conceptual design on line
- Supports refinement into finished products
- Assists with design exploration
- Facilitates construction of all interface features

Humanoid Architecture



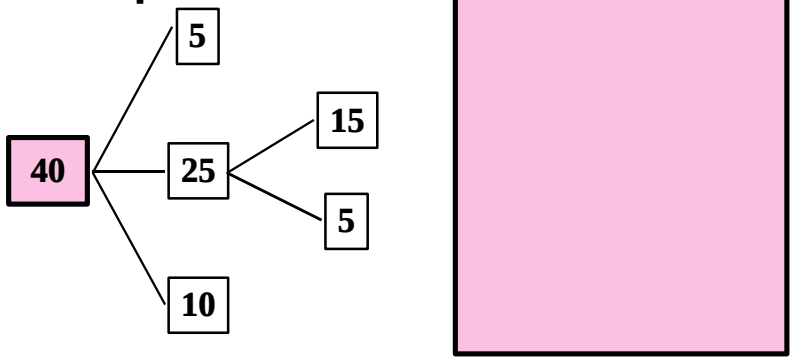
Humanoid's architecture vs. Generic architecture

Model

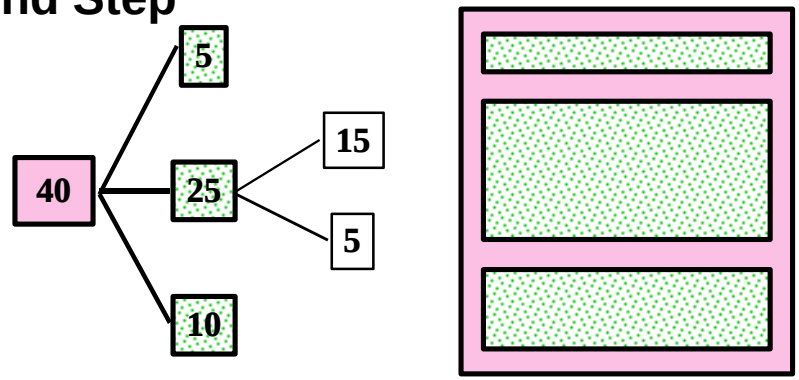

Workplace
User
Platform
Behavior
Presentation
Dialogue
Application
Tasks
Side-Effects
Sequencing

Example: TreeViz Visualization of Hierarchies

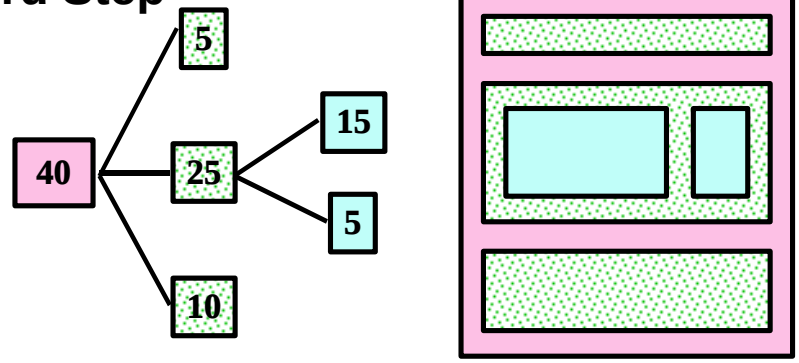
1st Step



2nd Step

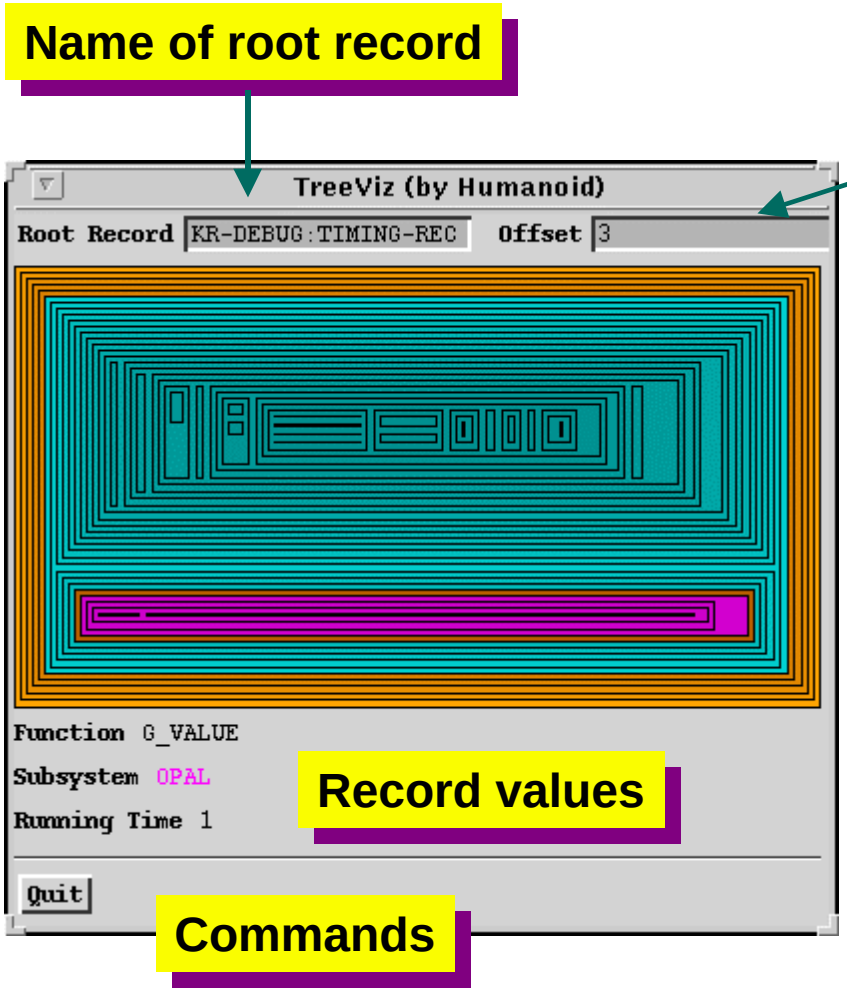


3rd Step



- Column/row space partitioning
- Rectangle size ~ node value
- Recursive: all tree levels

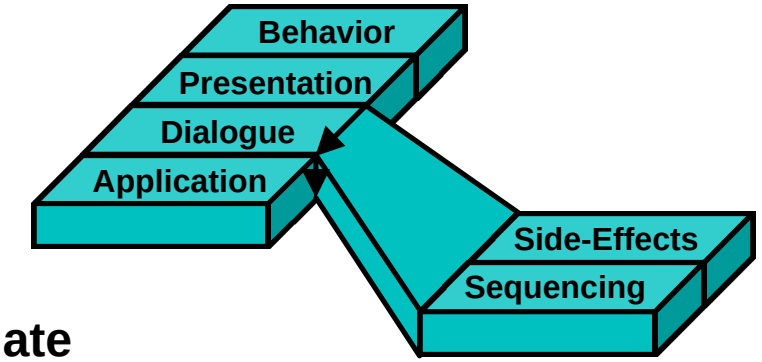
TreeViz
Implemented in Humanoid



- **Hierarchy**
  - » Tree of subroutine calls
- **Record values**
  - » Subroutine execution time
- **Color**
  - » Subsystem where subroutine is defined

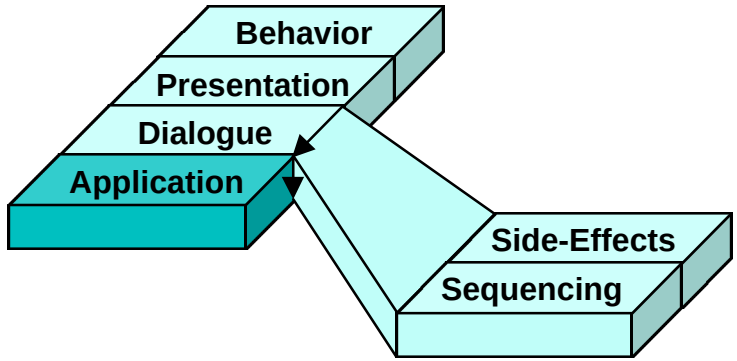
Modeling Language Features

- **Abstraction levels**
  - » Support design tools
- **Constraints**
  - » Support automatic screen update
  - » Support enabling/disabling behaviors
- **Iteration**
  - » Supports variable amounts of data
- **Conditionals**
  - » Support heterogeneous data
  - » Support context-sensitive displays

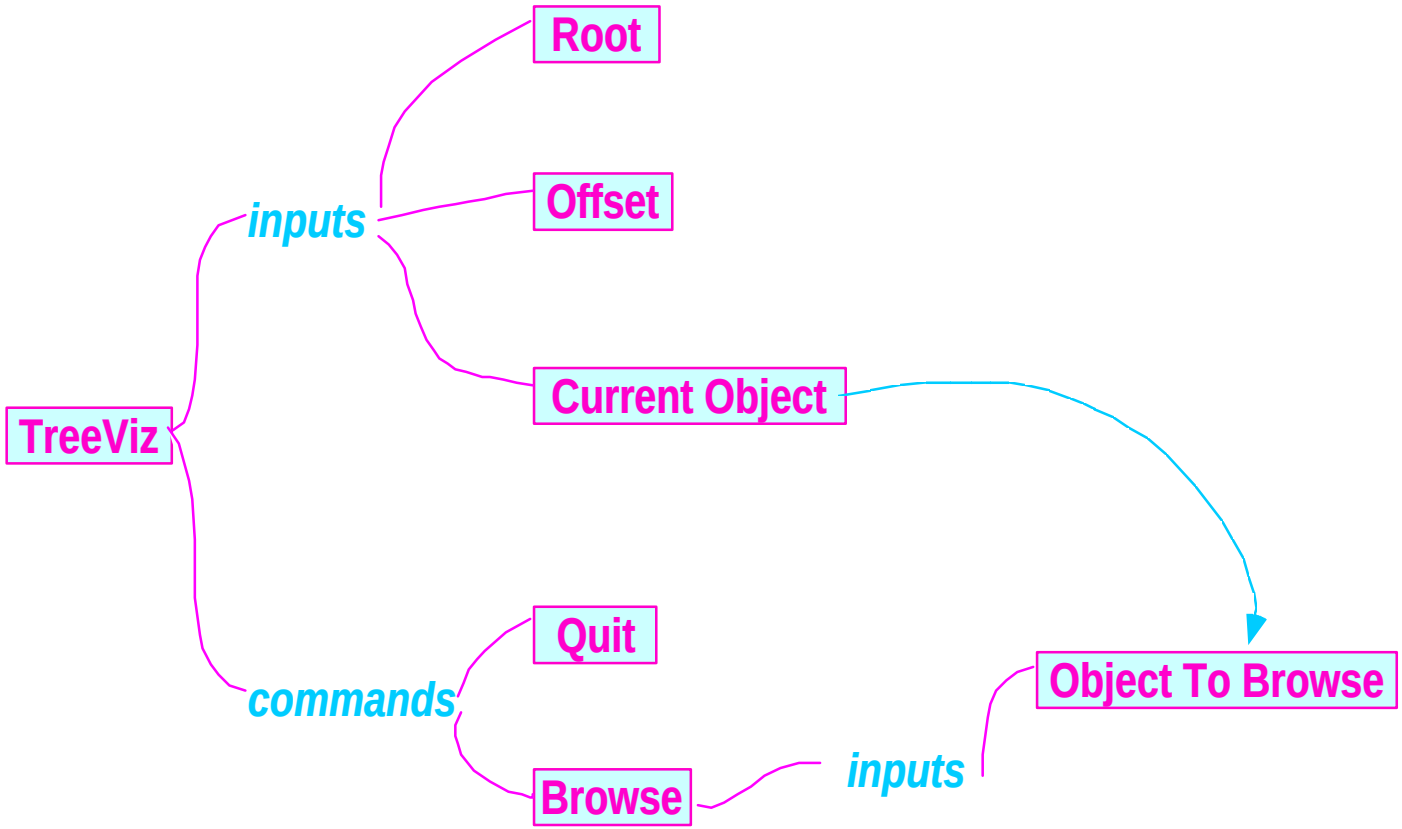


Application Semantics Model

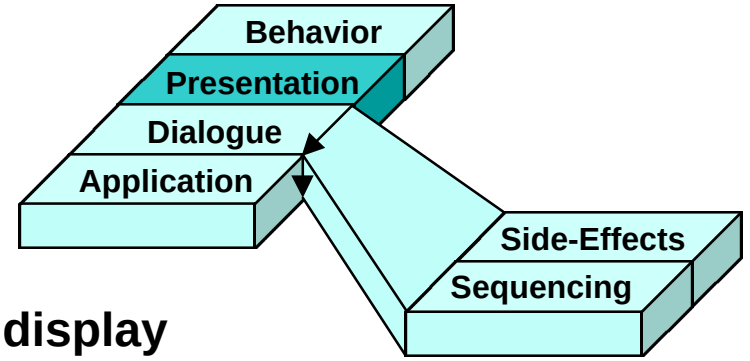
- Global inputs
  - » Value
  - » Type
  - » Validation predicate
- Commands
  - » Action
  - » Inputs
  - » Preconditions
  - » Exceptions
- Command and input groups



Application Model: TreeViz

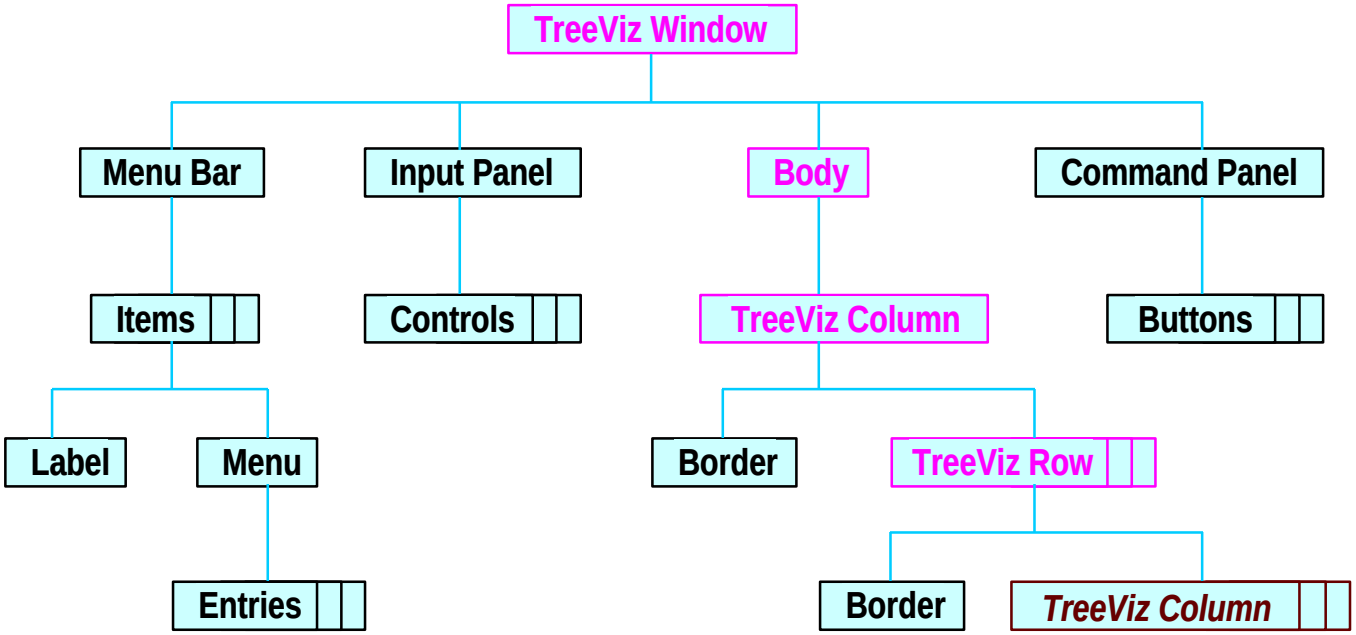


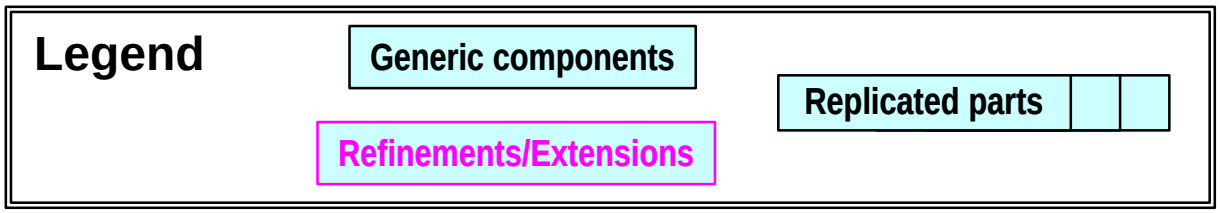
Presentation Model



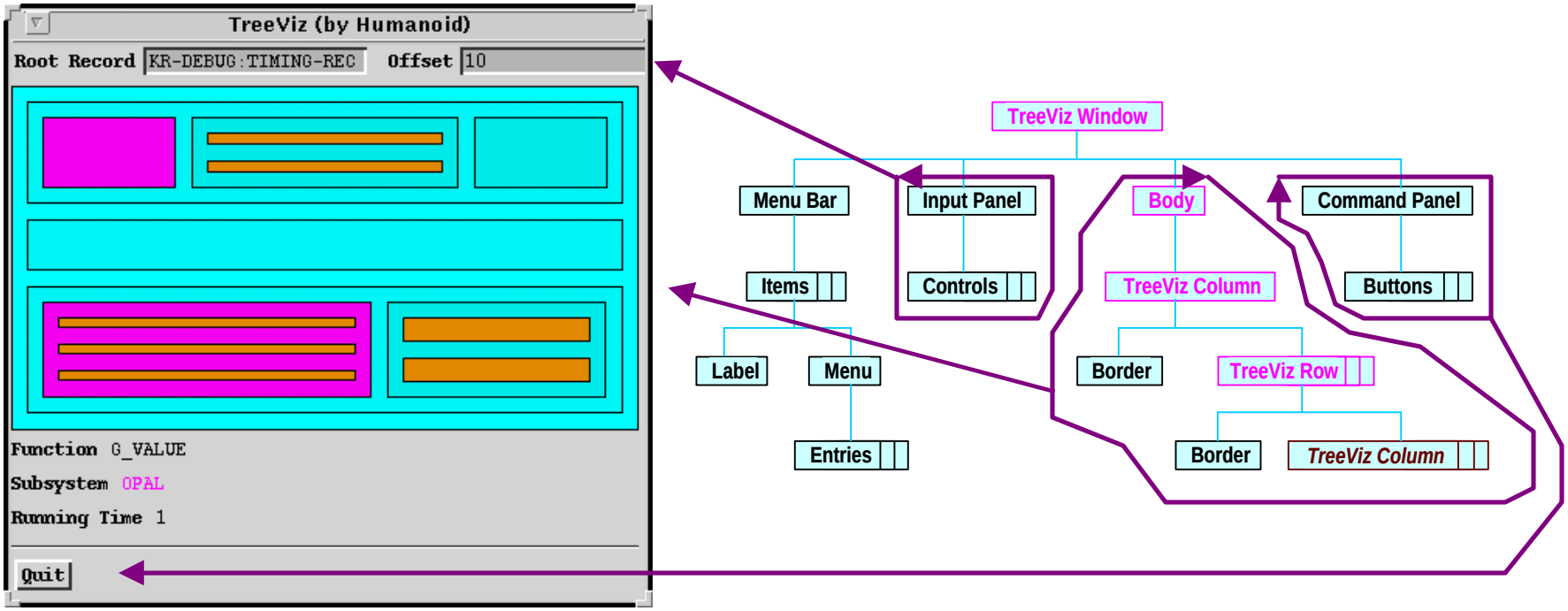
- **Templates**
  - » **Hierarchical decomposition of display**
  - » **Pluggable components**
  - » **Replication (iteration)**
  - » **Choice based on data properties (conditionals)**

Presentation Model: TreeViz



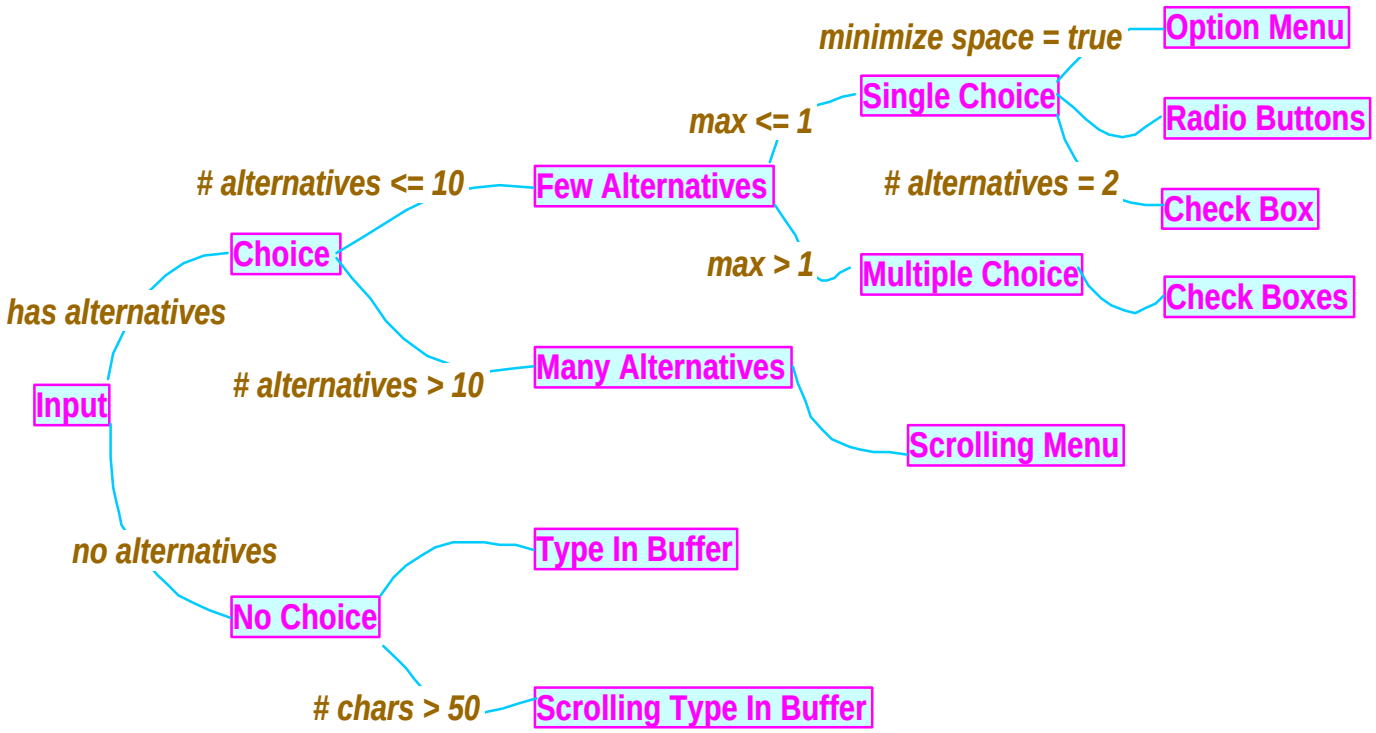


Example Display

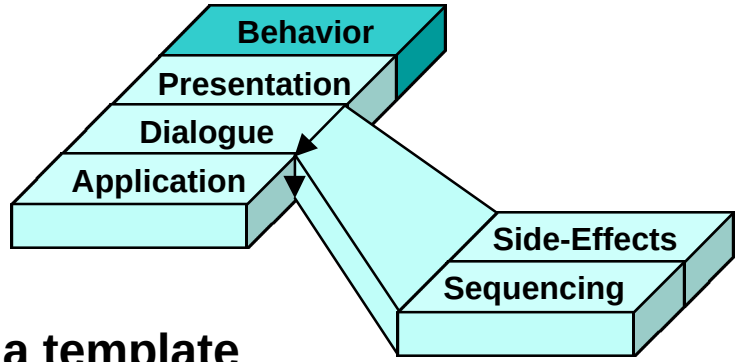


Presentation Choice: Example

- **Decision trees select presentation methods**
  - » **Example: selecting dialogue box building blocks**

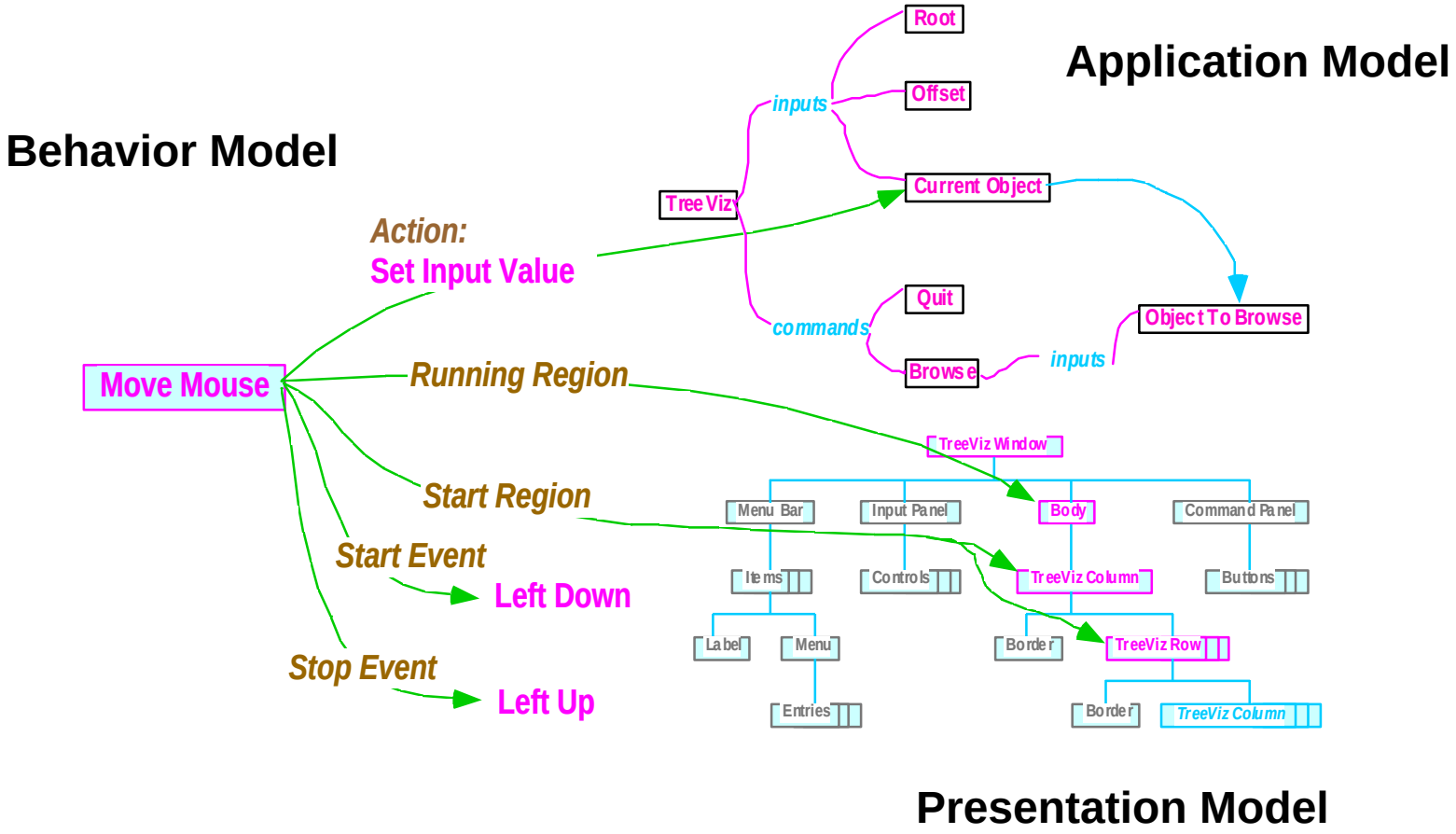


Behavior Model

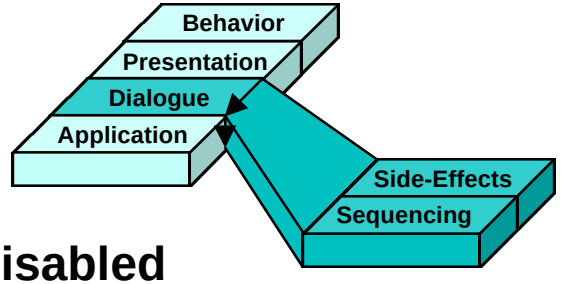


- **Interactors**
  - » Start and stop events
  - » Start and running region
    - – Example: all instances of a template
  - » Start, running and stop action
    - – Examples:
      - set input to value
      - invoke command
  - » Active, inactive

Behavior Model: TreeViz



Sequencing & Side-Effects Model

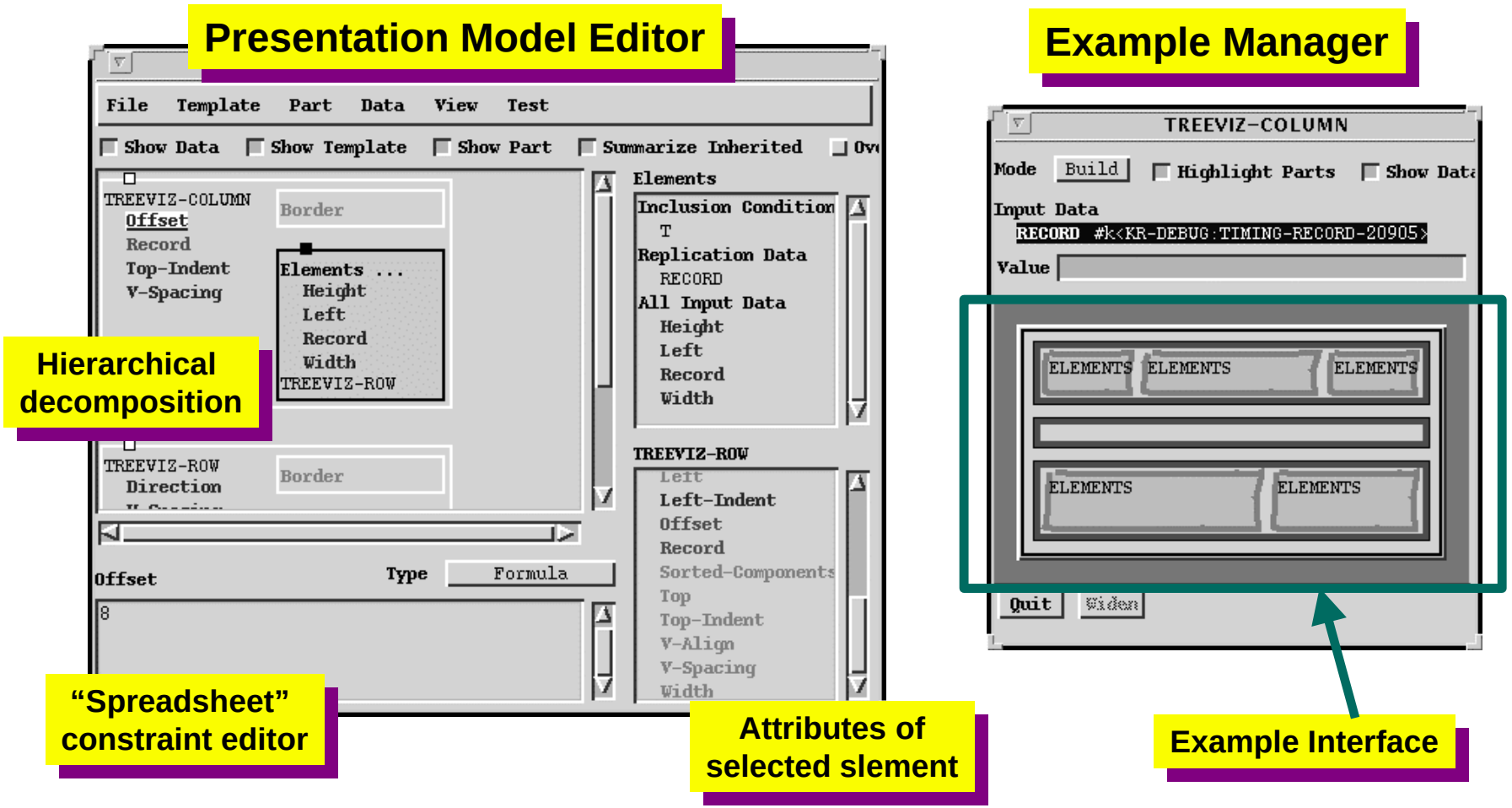


- **Sequencing specified implicitly**
  - » Derived from application model
    - – Invalid preconditions --> command disabled
    - – Invalid inputs --> E.g., “OK” button disabled
  - » Specified as attributes of command and input groups
    - – E.g., only one command in group active
    - – E.g., inputs in group prompted for in sequence
- **Access to lower level status information**
  - » Command and input states
  - » Demons
    - – E.g., No longer-active, became-active, active-to-running

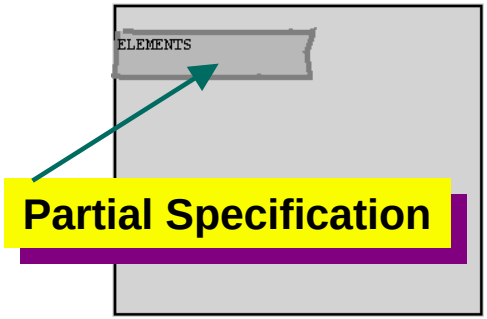
Modeling Environment

- All features of designs visible & changeable
- Example interface updated when model updated
- All views of designs are linked together
- Spreadsheet paradigm for entering constraints

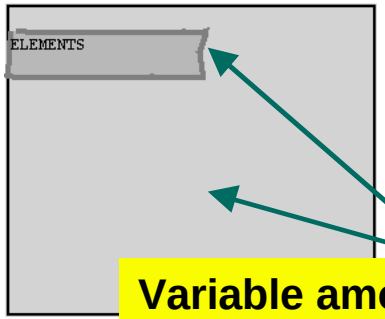
Design Tool: Model Editor and Tester



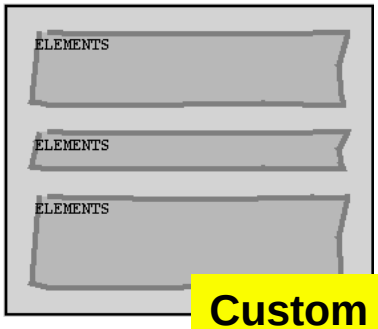
Prototyping from Partial Specs



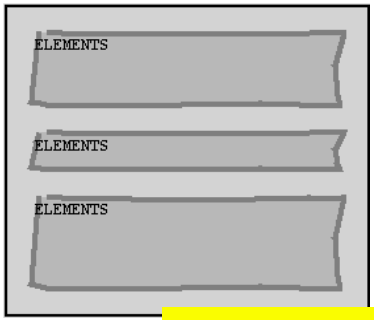
Partial Specification



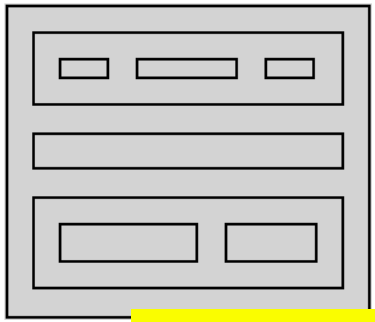
Variable amounts of data



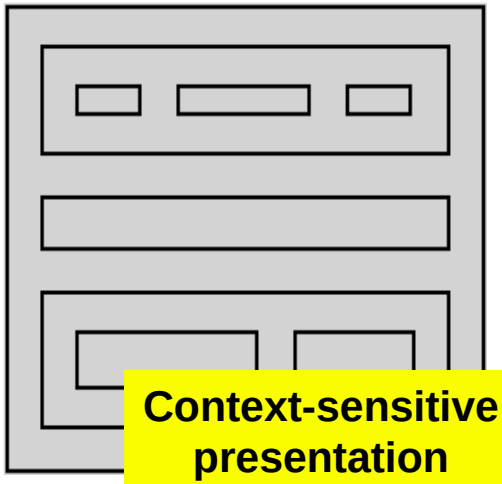
Custom layouts



Complex structured data



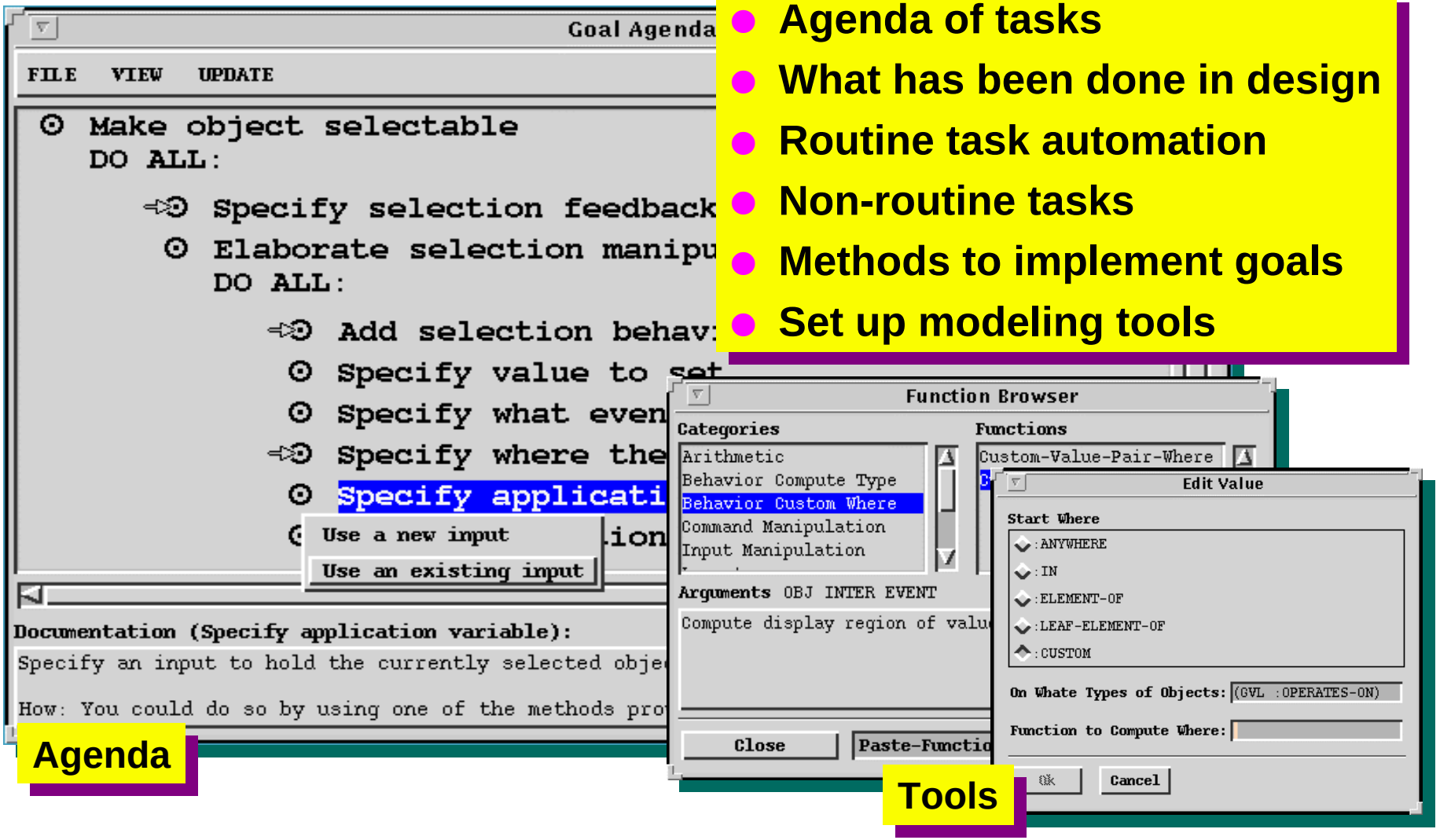
Recursive presentation



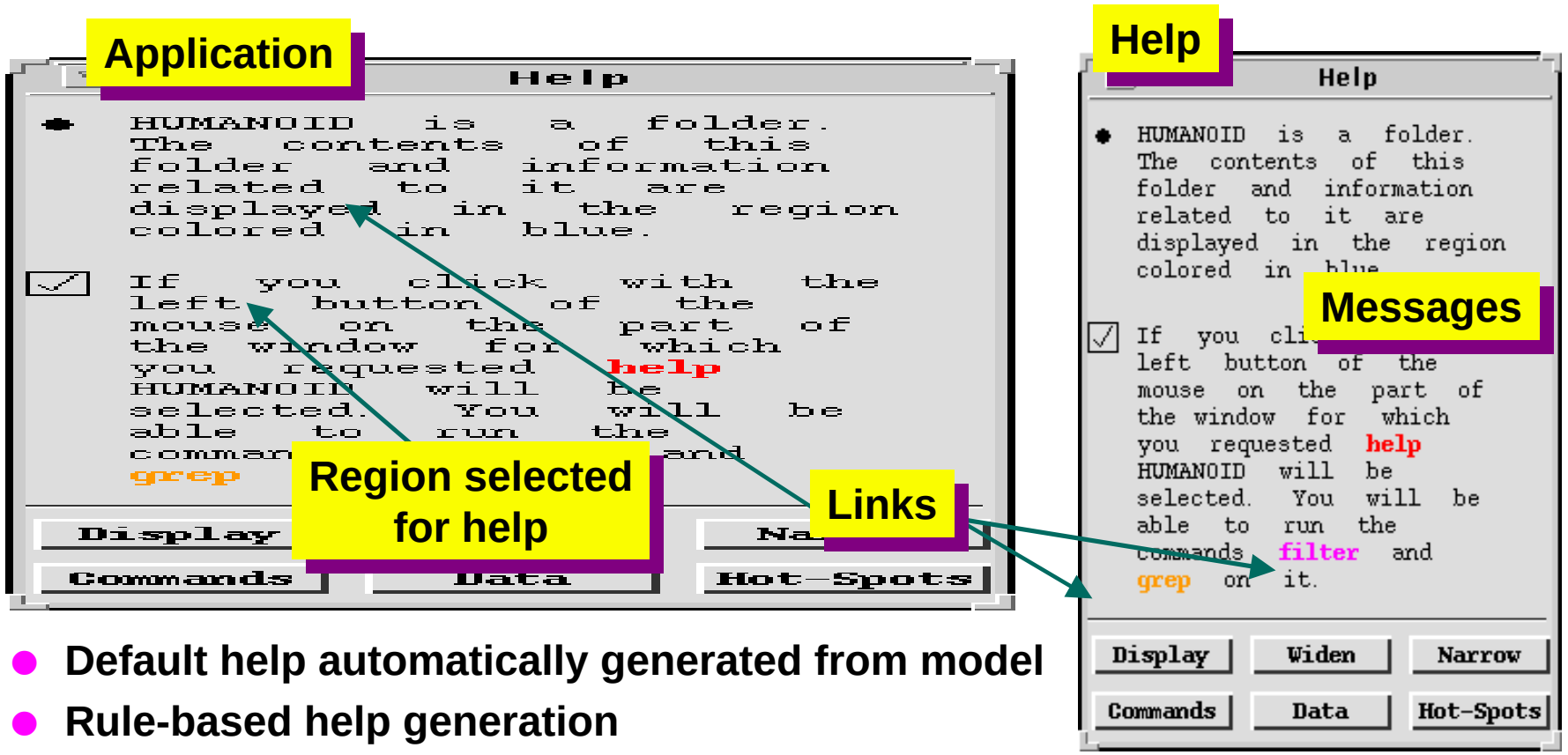
Context-sensitive presentation

Design Tool:
Design Assistants

- Agenda of tasks
- What has been done in design
- Routine task automation
- Non-routine tasks
- Methods to implement goals
- Set up modeling tools

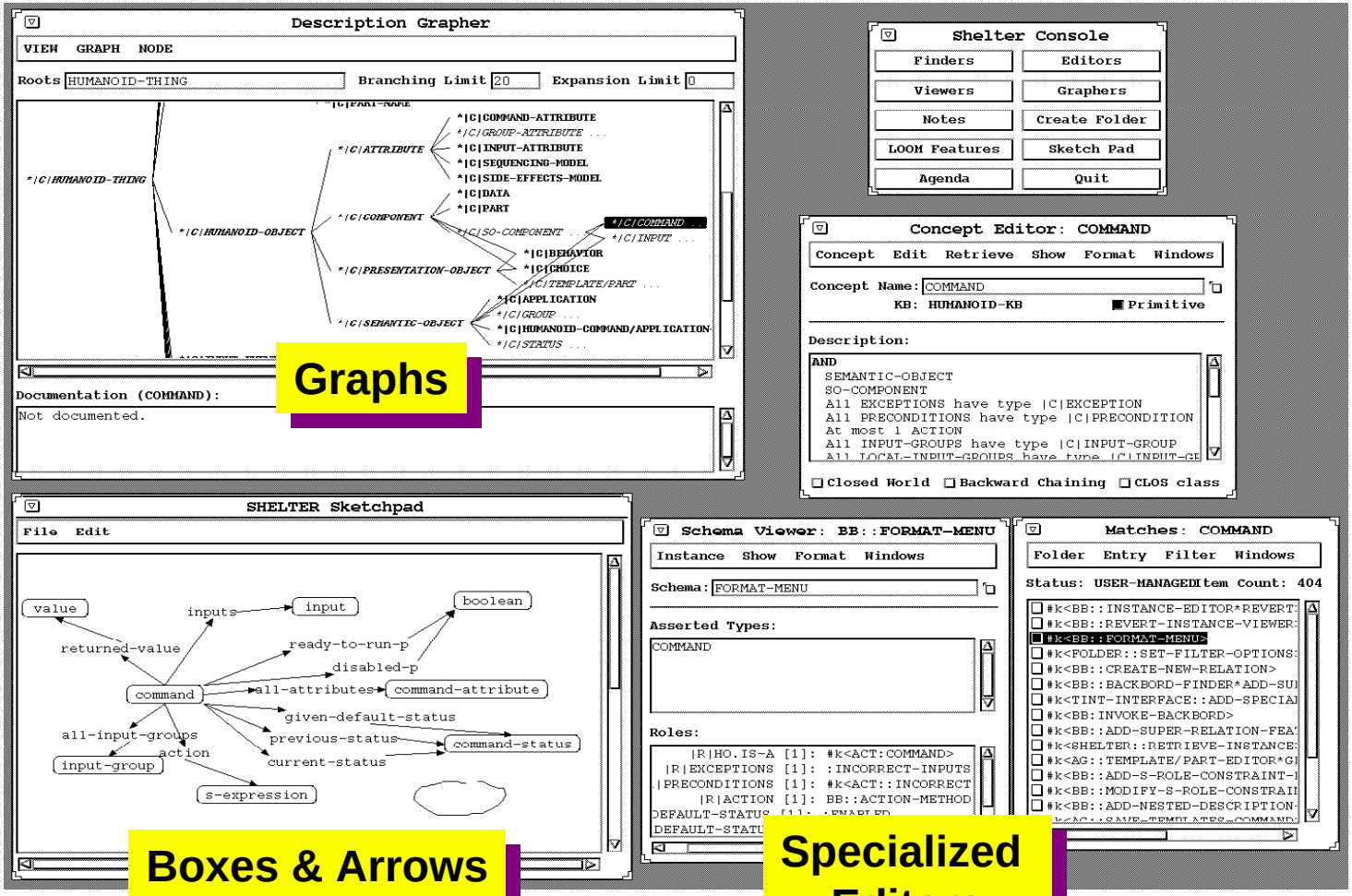


Run-Time Tool: Hypertext “Balloon” Help



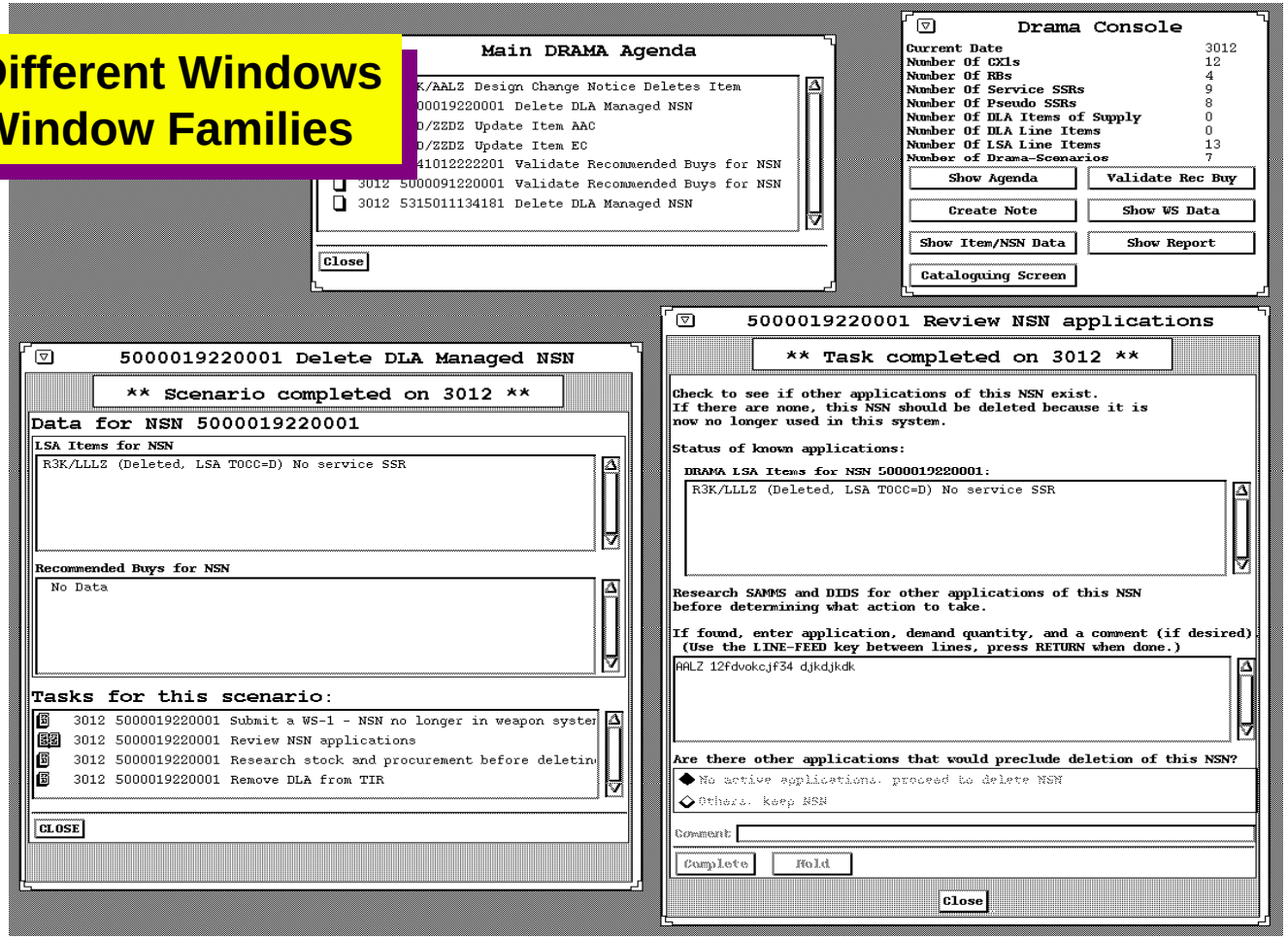
- Default help automatically generated from model
- Rule-based help generation
- Levels of help message customization
  - » Editing examples of messages, changing rule conditions, defining new rules

Applications: SHELTER Knowledge-Base Development



Applications: DRAMA Logistics Management

75+ Different Windows
13 Window Families



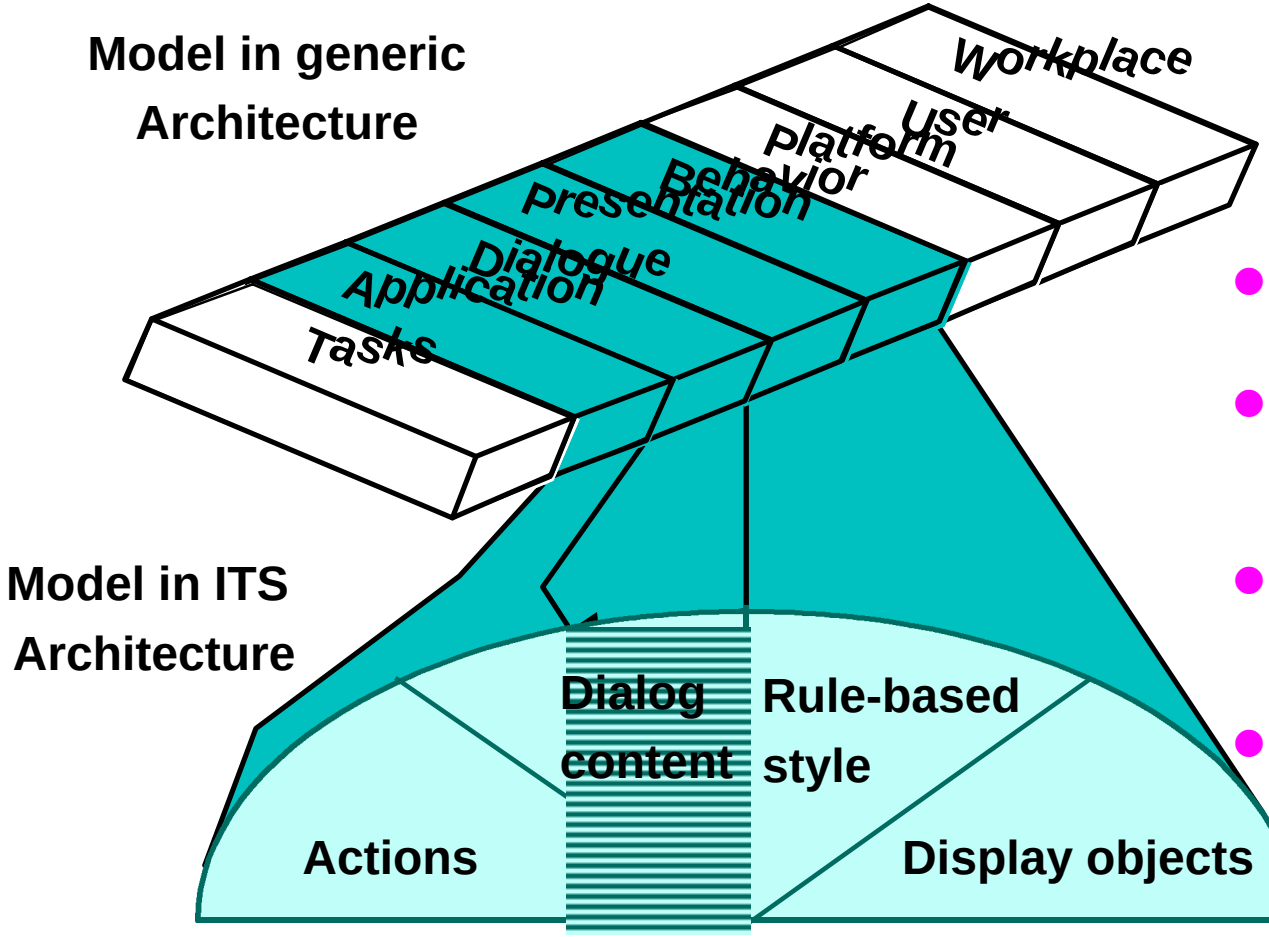
Humanoid: Review

- **Benefits**
  - » Supports wide range of interfaces
  - » Complex interfaces developed without programming
  - » Immediate visualization of consequences of model changes
  - » Allows delay design commitments
  - » Framework for incorporating support tools
- **Shortcomings**
  - » Interactive development environment is hard to use
  - » Performance

ITS: A Tool for Rapidly Developing Interactive Applications

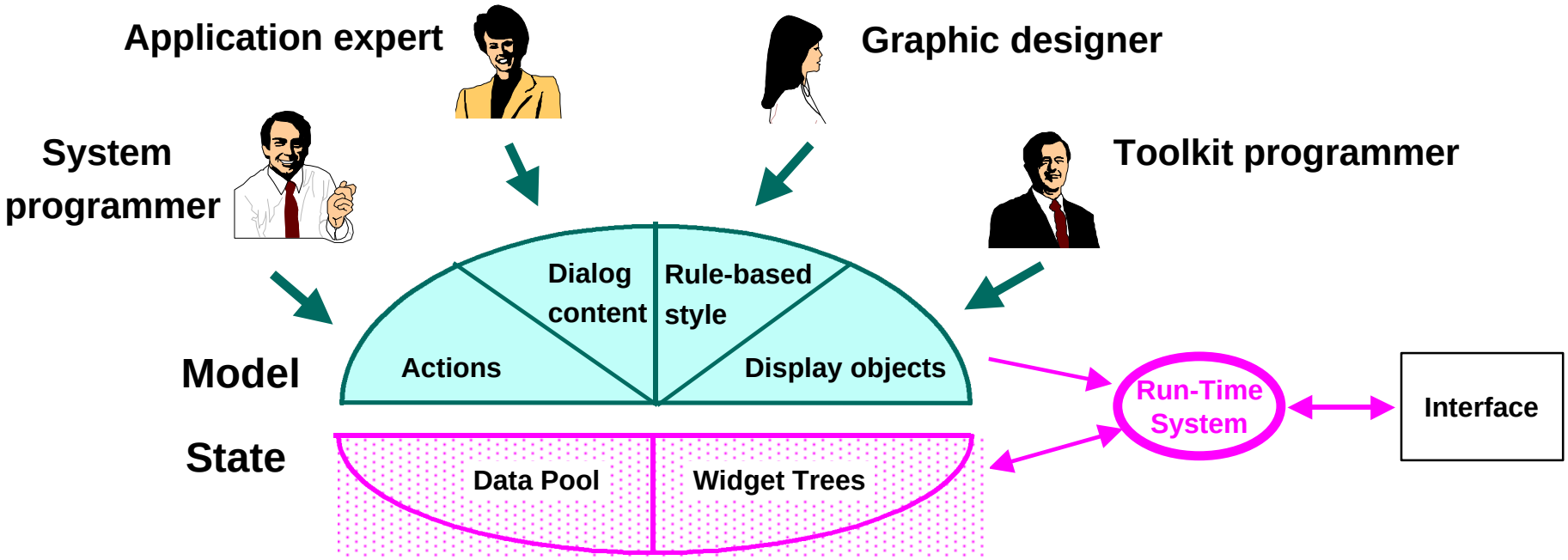
- **Reconfigurable interfaces**
  - » **Different interaction devices, users, countries**
- **Direct involvement by different specialists**
  - » **Domain experts, graphic artists, system and toolkit programmers**
- **Four layer architecture separates concerns**
- **Production quality tool**
- **Widely used applications**
  - » **Information kiosks (multimedia)**
  - » **Business applications**

ITS
4 Layer Architecture



- **Actions**
  - » Semantic routines
- **Dialog content**
  - » Application objects
  - » Contents of display
- **Rule-based style**
  - » Rules for presenting objects on the screen
- **Display objects**
  - » Primitive toolkit building blocks

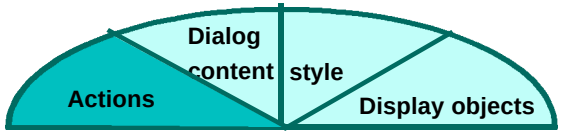
ITS Architecture



- Model elements specified by different specialists
- Data pool: shared data between application and interface
- Widget trees: representation of display
- On the fly generation of interfaces based on model and data pool

Action Layer

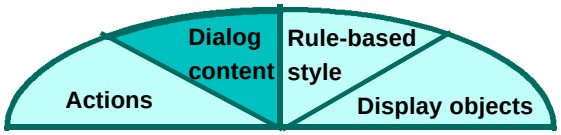
- Procedures that perform computation
- Communicate with interface by
  - » Storing values in data pool
  - » Many dialogue objects can refer to data pool elements
  - » Notification mechanism to trigger display updates
- Run-time system calls actions
  - » In response to input events



System programmer

Dialog Content Application Model

- **Declaration of structure of application data**
  - » **Forms: set of fields (a record)**
  - » **Lists: a set of forms**
    - – **Form fields can contain lists**
- **Independent of display information**
  - » **Views can show only subset of data**
  - » **Multiple views on same data**



Application expert

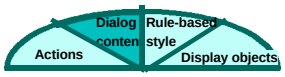
Dialog Content (Application Model) Example

Data definition of airline reservation system

```
list listname = flights, numrecords = 10
    field destination, rangename = cities, size = 20
    field departure_time, size = 10
    field departure_date, size = 20
    field airline, rangename = airlines, size = 20
    field number_stops, size = 5
```

Legend:

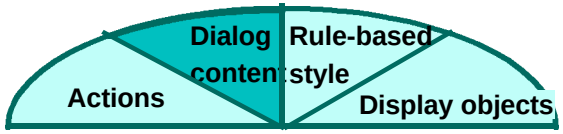
- list: declaration of a list
- listname: name of the list
- numrecords: number of records in list
- field: name of field in record
- rangename: type of value stored in field
- size: estimate of number of character



Application expert

Dialog Content
Dialog/Presentation Model

- **Frames**
  - » Corresponds to a window or region of the display
  - » Definition of data to be displayed
  - » Does not define the appearance of the display
- **Dialogue control**
  - » Defines when frames are shown on the screen
    - – Frame “activation”



Application expert

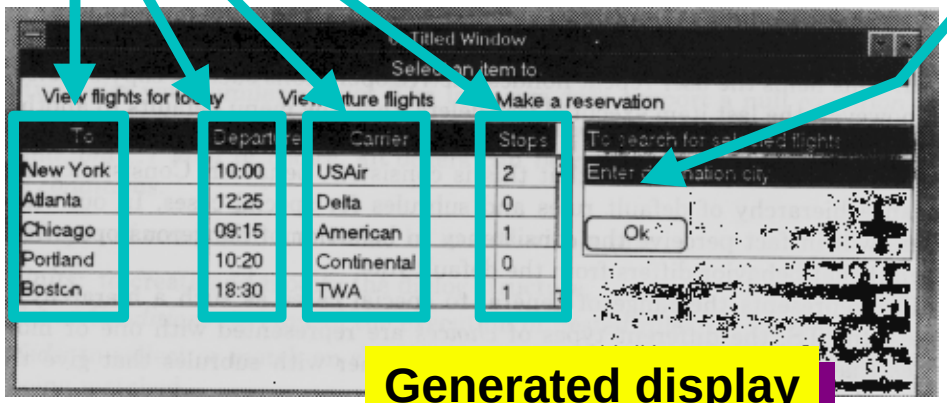
Dialog Content (Dialog/Presentation Model) Example

frame id = check_today, action = getlist, listname = flights, value = flights.data
list listname = flights, number = 5
list-item field = destination, message = “To”
list-item field = departure_time, message = “Departure”
list-item field = departure_date, size = 20
list-item field = airline, message = “Carrier”
frame message = “To search for selected flights”
...

Frame definition

Application expert 

5 Elements shown



Generated display

Data definition

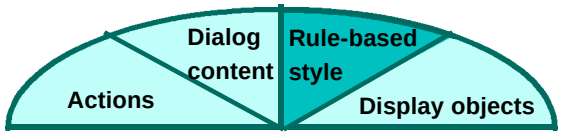
list listname = flights, numrecords = 10
field destination, rangename = cities, size = 20
field departure_time, size = 10
field departure_date, size = 20
field airline, rangename = airlines, size = 20

Rule-Based Style Presentation & Behavior Model

Style

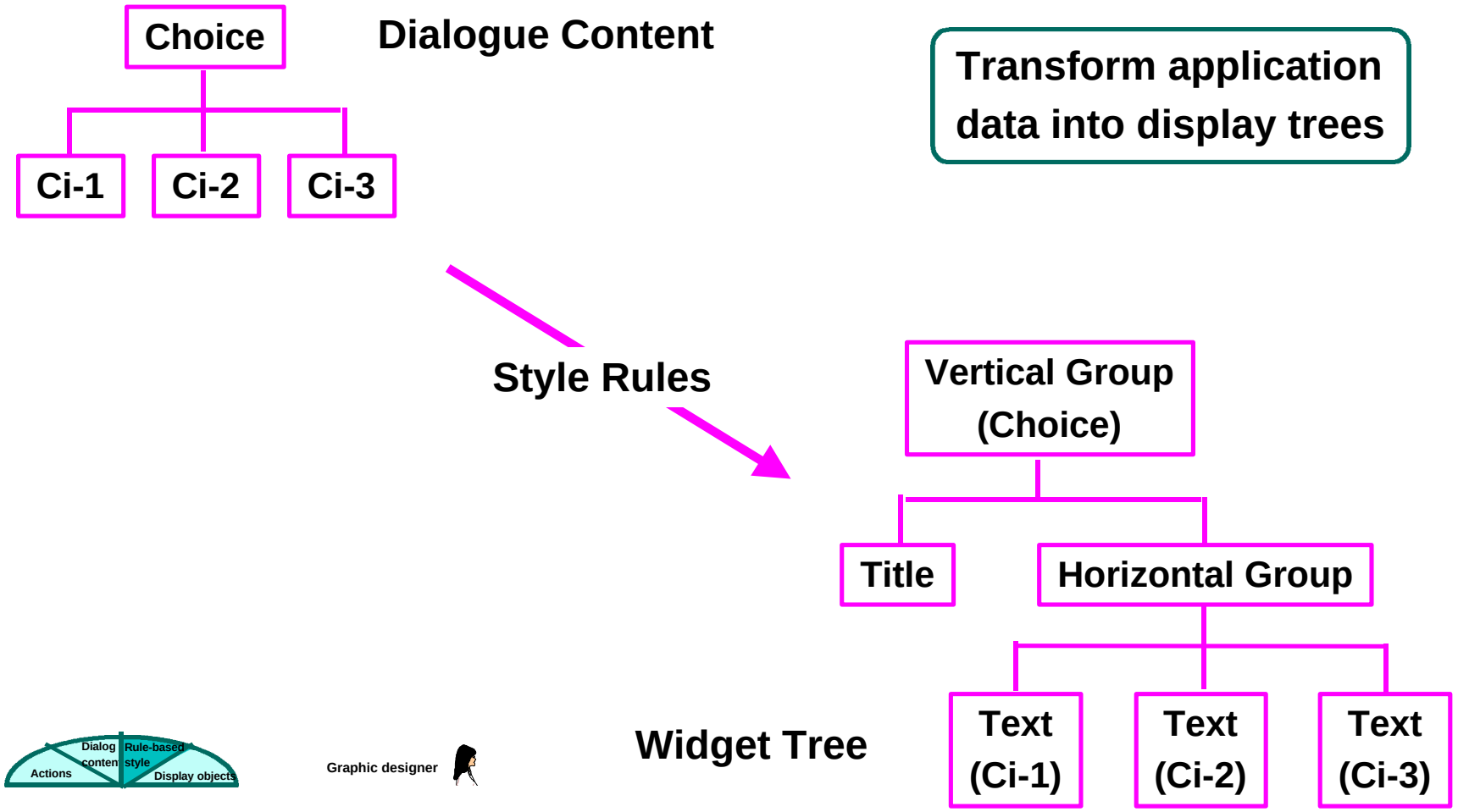
Coordinated set of decisions on appearance and behavior used in a family of applications

- Refers to both input and output
- Style in the small and in the large
- Applies to more than one application



Graphic designer

Display Generation



Style Rules

Style rules determine display format

Conditions

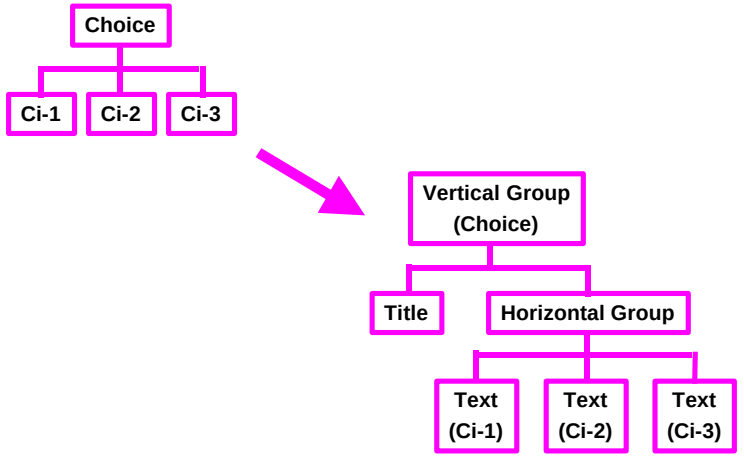
- When rule is executed
  - » Names of dialogue frames
  - » Fields in dialogue frames
  - » Field attributes
    - – rangename
    - – size
  - » No. of choices in choice fields

Results

- Formatting instructions
  - » Create display elements
  - » Set attributes of display elements
  - » Control execution of nested rules
  - » Group display elements together

Style Rules: Example

Rule for displaying a set of choices as a menu with a title



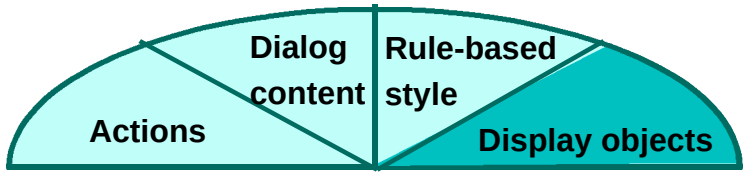
when type = choice
build
unit type = Vertical-Group
unit type = Title
unit type = Horizontal-Group
unit type = message, replicate = all

When asked to display a set of choices
then build a
a vertical arrangement of
a title
and a horizontal arrangement
of a message for each choice

Graphic designer 

Display Objects

- **Implementation of the display building blocks**
  - » **Request screen space from parent**
  - » **Respond to space allocation**
  - » **Paint object on the screen**
  - » **Respond to input events**
    - – **activate frames**
    - – **select object**
    - – **execute actions**



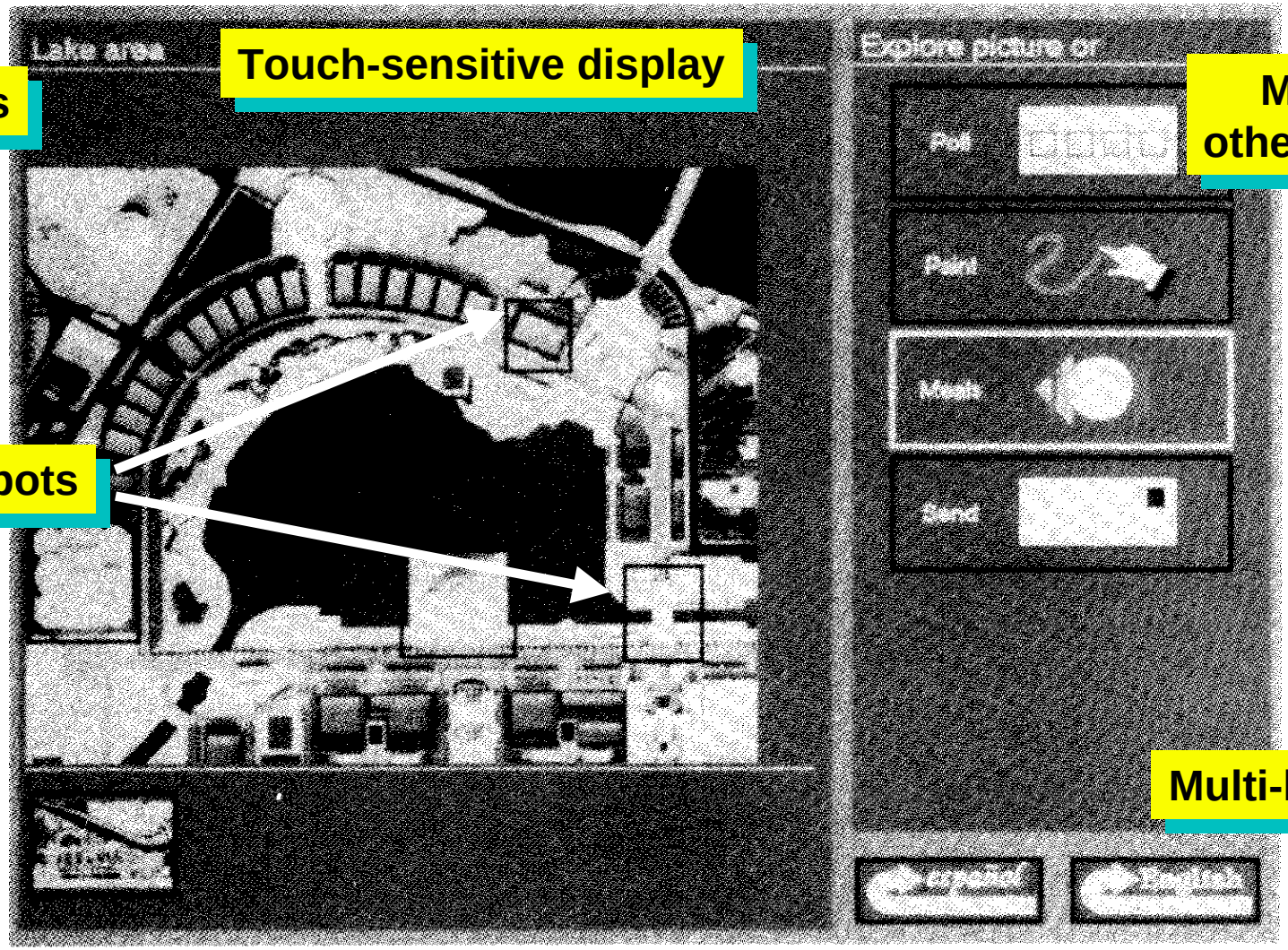
Toolkit programmer

Applications
1992 EXPO in Seville

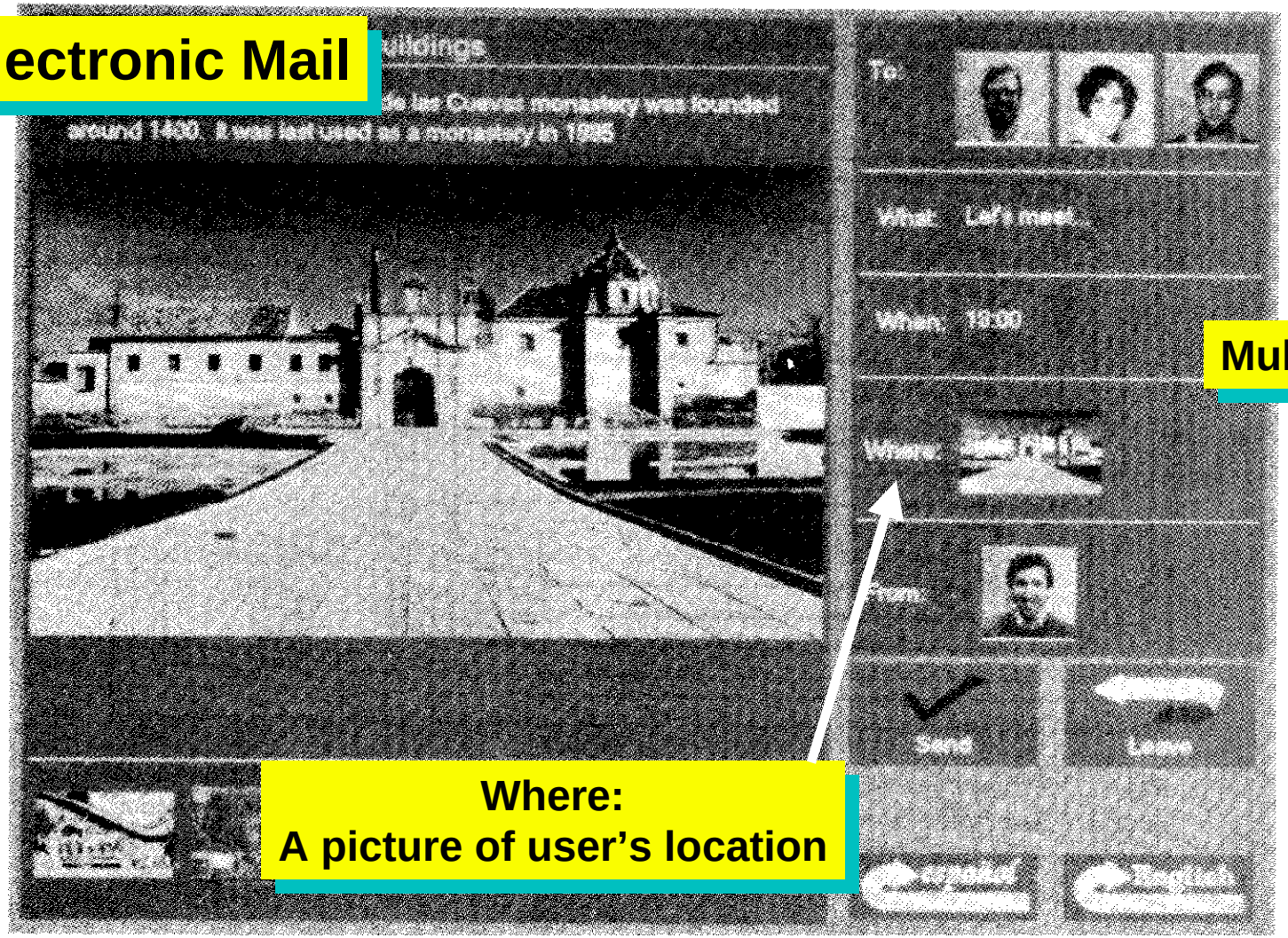
- **Visitor information system**
  - » Maps and directions to pavillions
  - » Person to person and group electronic mail
  - » Automated restaurant reservations
  - » Public opinion polling
  - » Finger painting and picture taking
- **Used by millions of people for several months**
  - » Network of IBM 486 computers
  - » Touch screen interface




ITS: Information Kiosk
Expo 92, Seville, Spain



ITS: Information Kiosk
Expo 92, Seville, Spain



ITS: Review

- **Benefits**
  - » Clean separation of design concerns
    - – Allow involvement by different specialists
  - » Easy to tailor interfaces to multiple platforms
  - » Production-quality system
- **Shortcomings**
  - » Limited set of tools
  - » Relatively long learning curve

Blank Page

Agenda

- Model-based paradigm
- Case studies: UIDE, Mecano
- Architectures
- Break
- Case studies: Humanoid, ITS
- Survey of Model-Based Tools
- Conclusions
- Questions

Survey of Model-Based Tools

- Current model-based work
- Comparison along key dimensions
  - Model components covered
  - Range of design and run-time tools
  - Practicality

Current Model-Based Work

Development Environments

- UIDE
- Mecano
- ITS
- Humanoid
- ADEPT
- GENIUS
- Trident

Usability

- GOMS

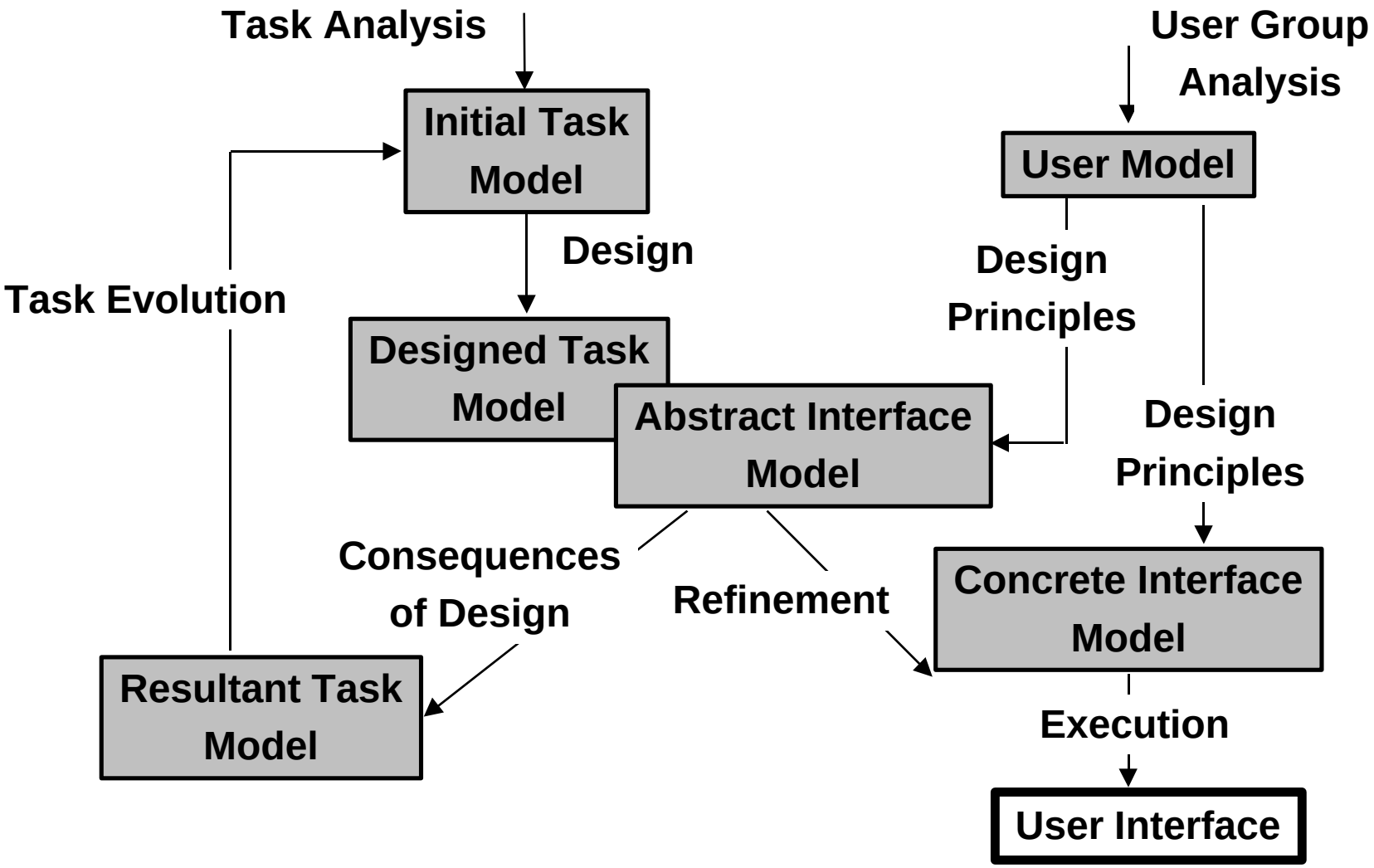
Notations

- EDICM
- E-R Models

ADEPT [Johnson 93]

- Development environment for interface prototyping
- User-Task centered design
  - » Task model *evolves* throughout design
- Multi-step refinement process
  - » Task model
  - » Abstract interface model
  - » Concrete interface model
  - » Executable code
- Platform independence

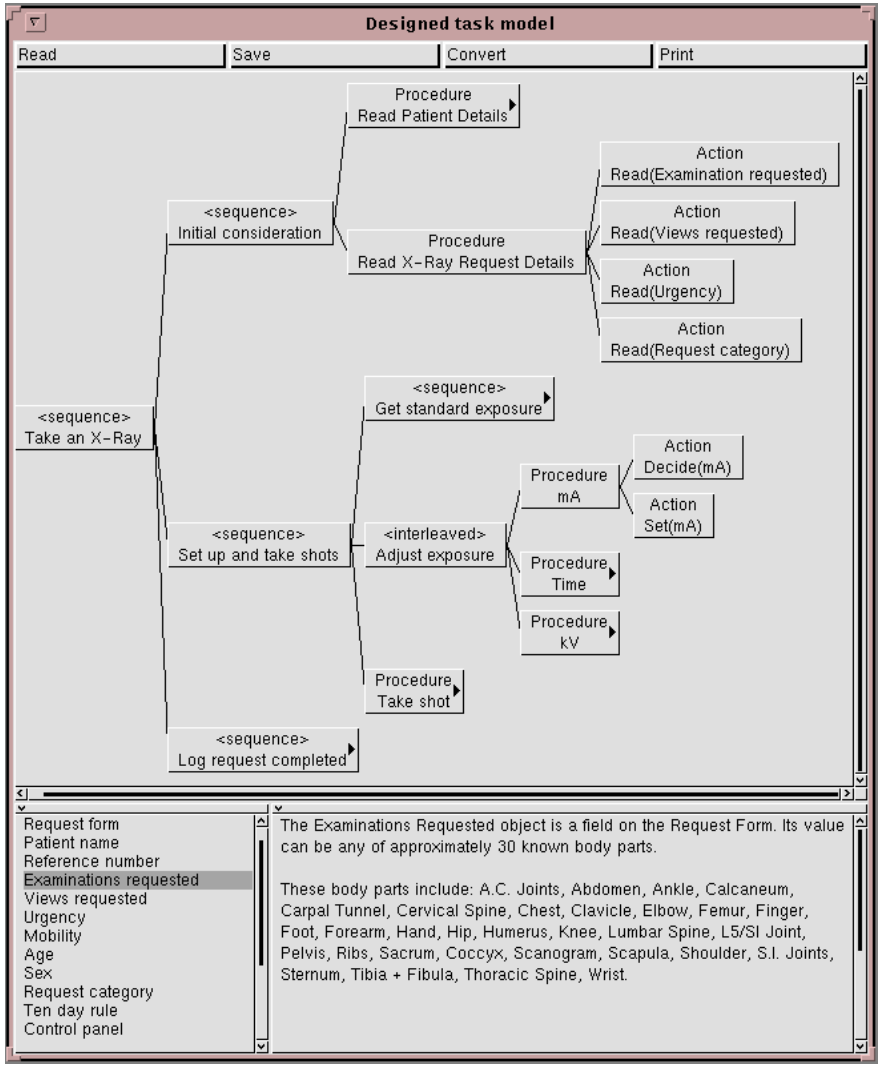
ADEPT Model & Processes



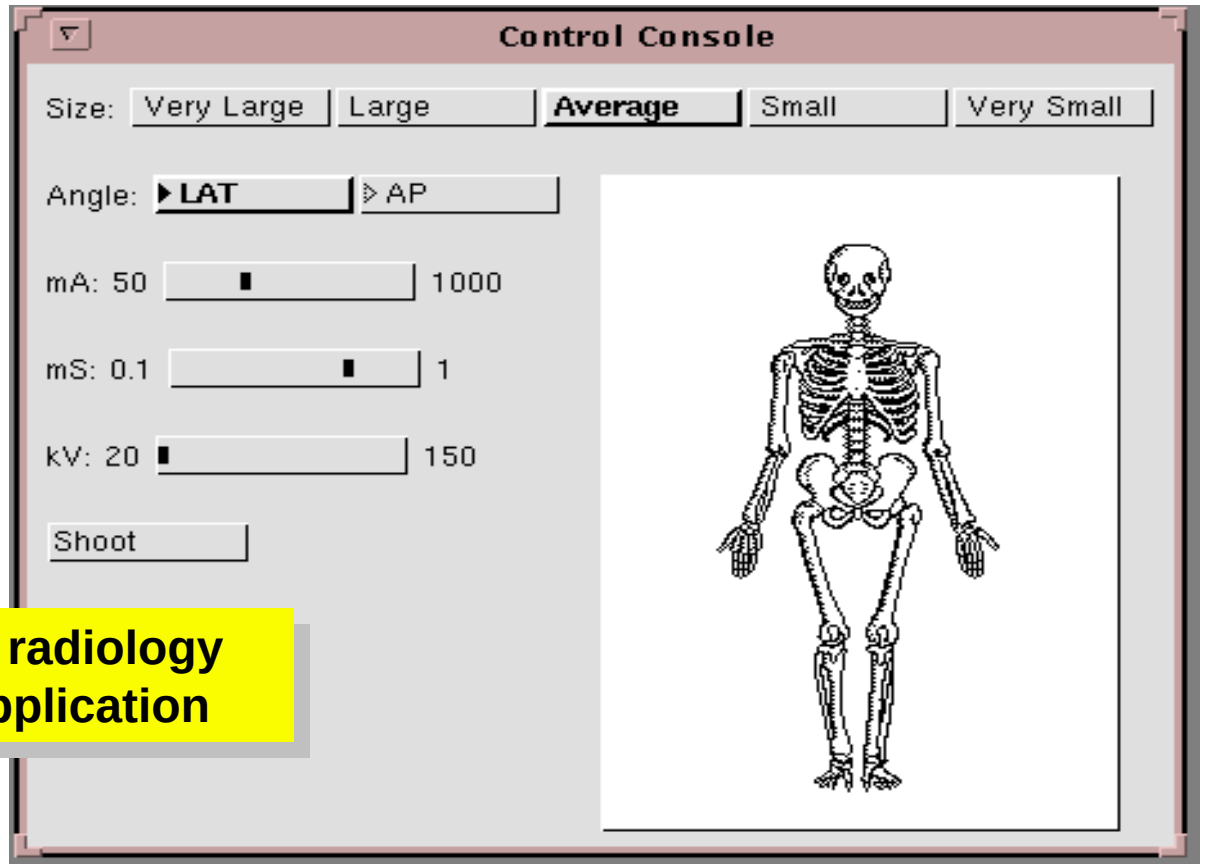
ADEPT Example: Task Model

Task model for a radiology workstation application interface

Designers build tasks models with a graphical editor



ADEPT Example : Generated Interface



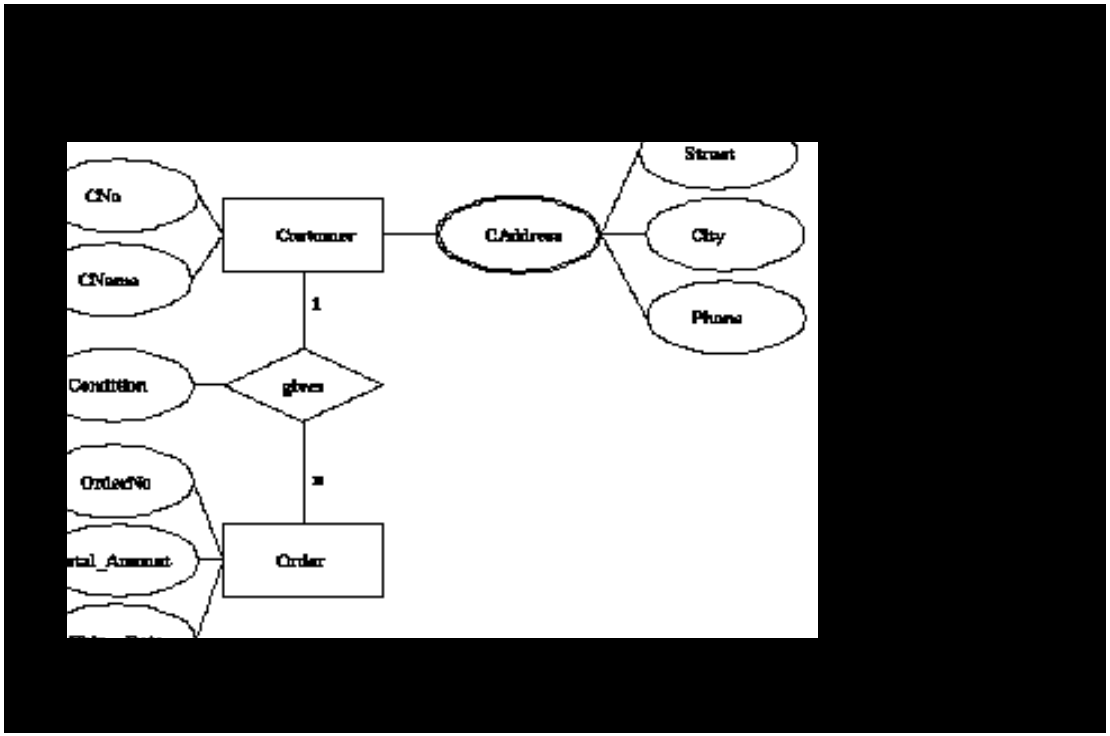
Entry form for radiology workstation application

GENIUS [Weisbecker 93]

- Development environment for database applications
- Integrates
  - » Software engineering techniques
  - » User interface design
- Automatic generation of interfaces
  - » Layout from E-R models
  - » Dialog specifications from Petri nets

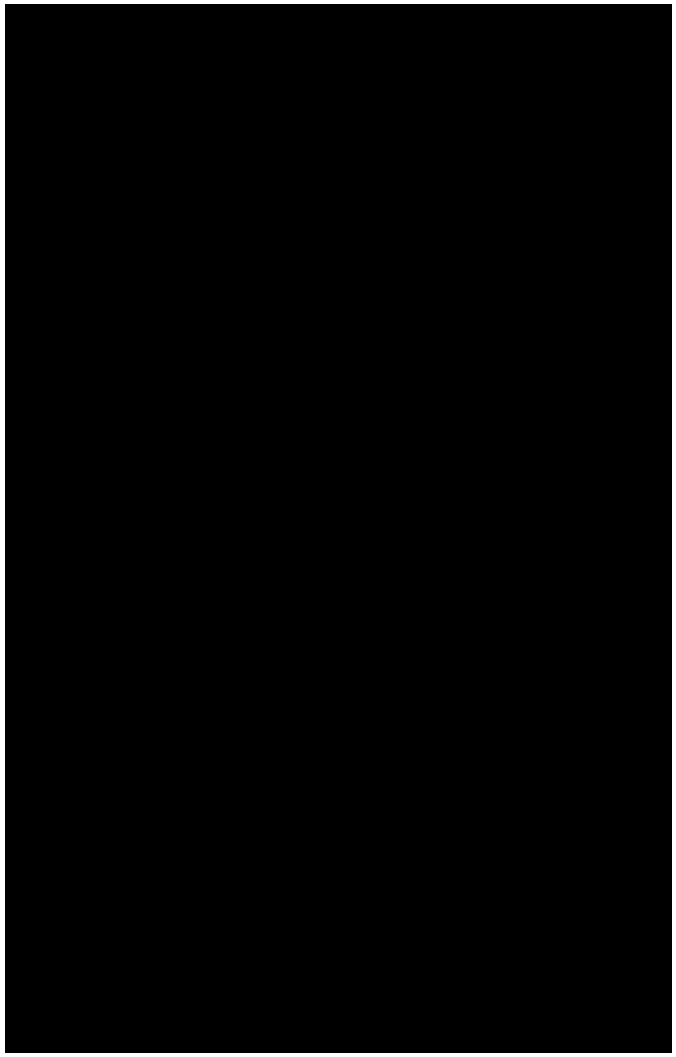
GENIUS Example: The E-R Model

Designers build the application's data models with a graphical editor



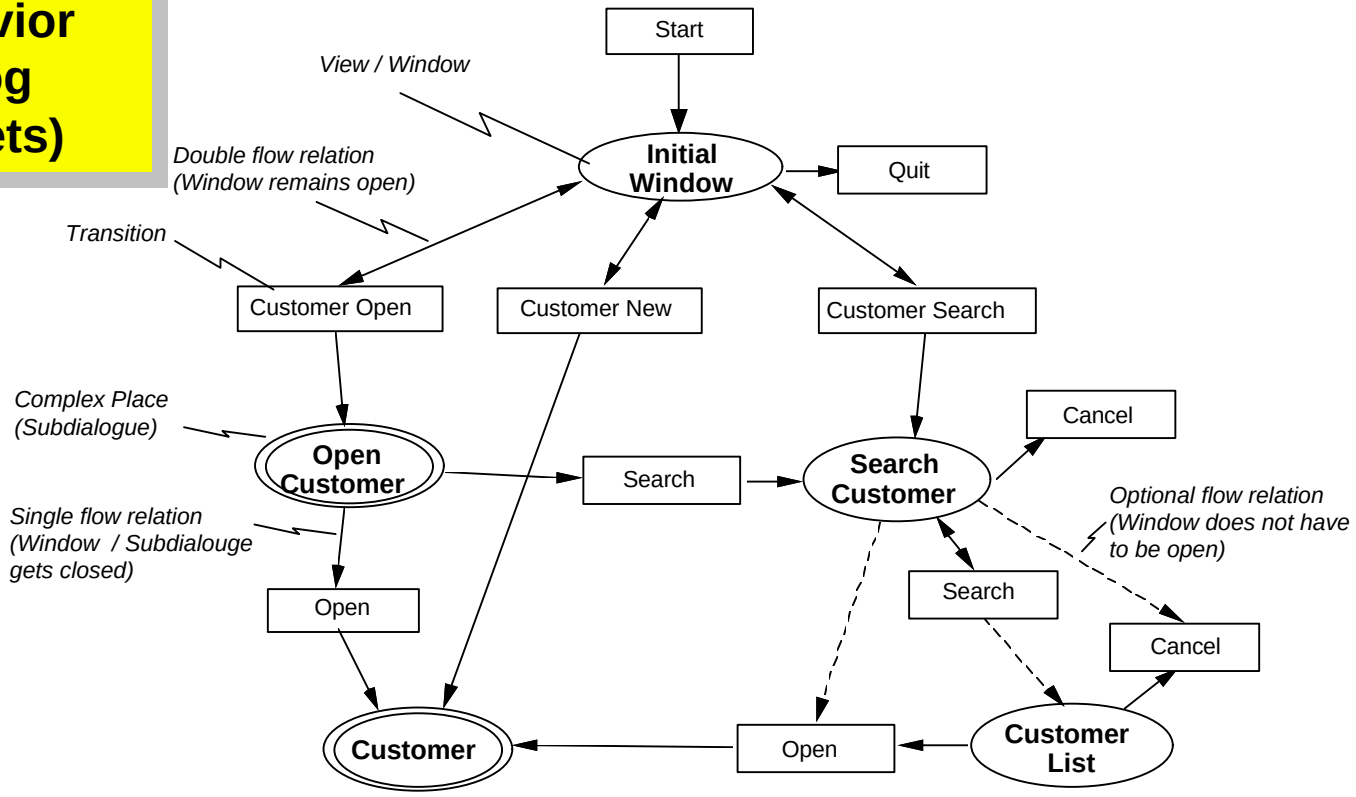
GENIUS Example: The Generated Interface

Automatic designer produces static layout from E-R model



GENIUS Example: Dialog Specification

Designers specify dynamic behavior through “Dialog Nets” (Petri Nets)



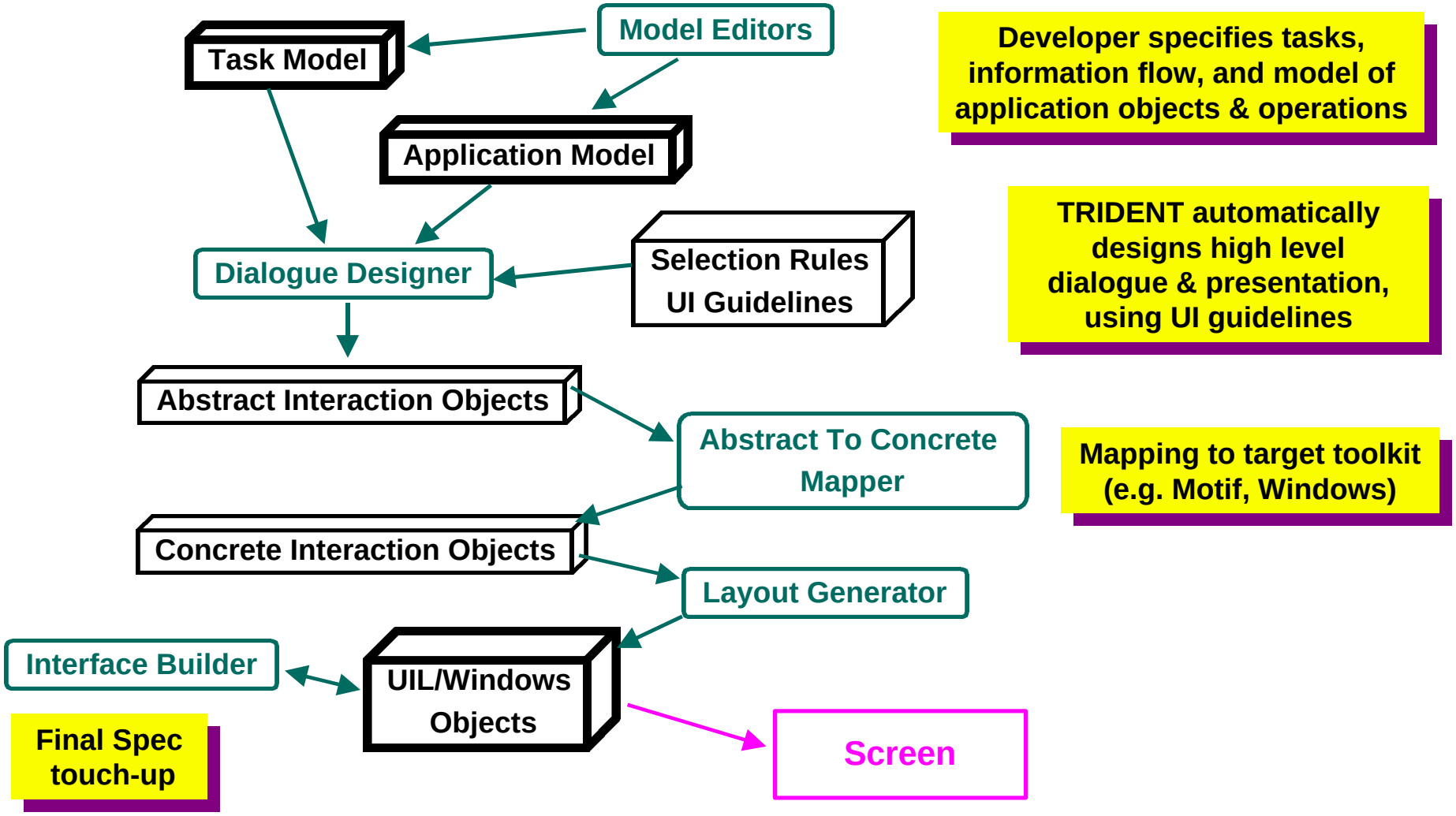
TRIDENT: Rapid Development of Business-Oriented Applications

- **Separation of concerns**
  - » **Task analysis**
  - » **Application functionality**
  - » **Dialogue**
  - » **Presentation**
  - » **Platform dependencies**
- **Automatic interface generation based on**
  - » **Task analysis**
    - – **Information flow between tasks**
  - » **User interface guidelines**

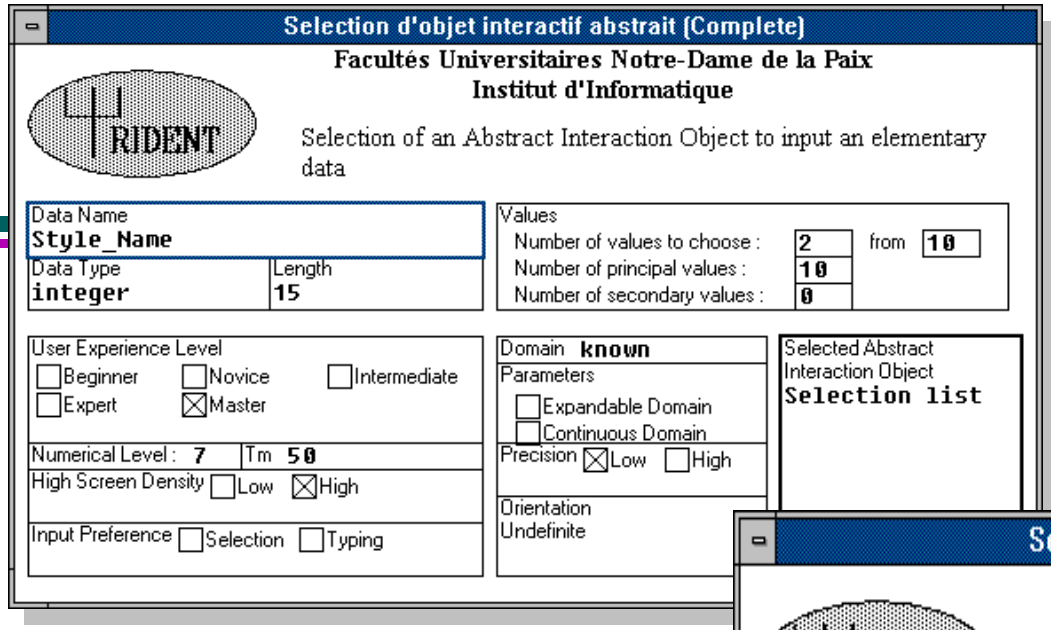
F. Bodart, A.M. Hennebert, J.M. Leheureux, B. Sacre, I. Provot, J. Vanderdonckt
University of Namur, Belgium

[Vanderdonckt 93]

TRIDENT: Architecture
Automatic Interface Generation

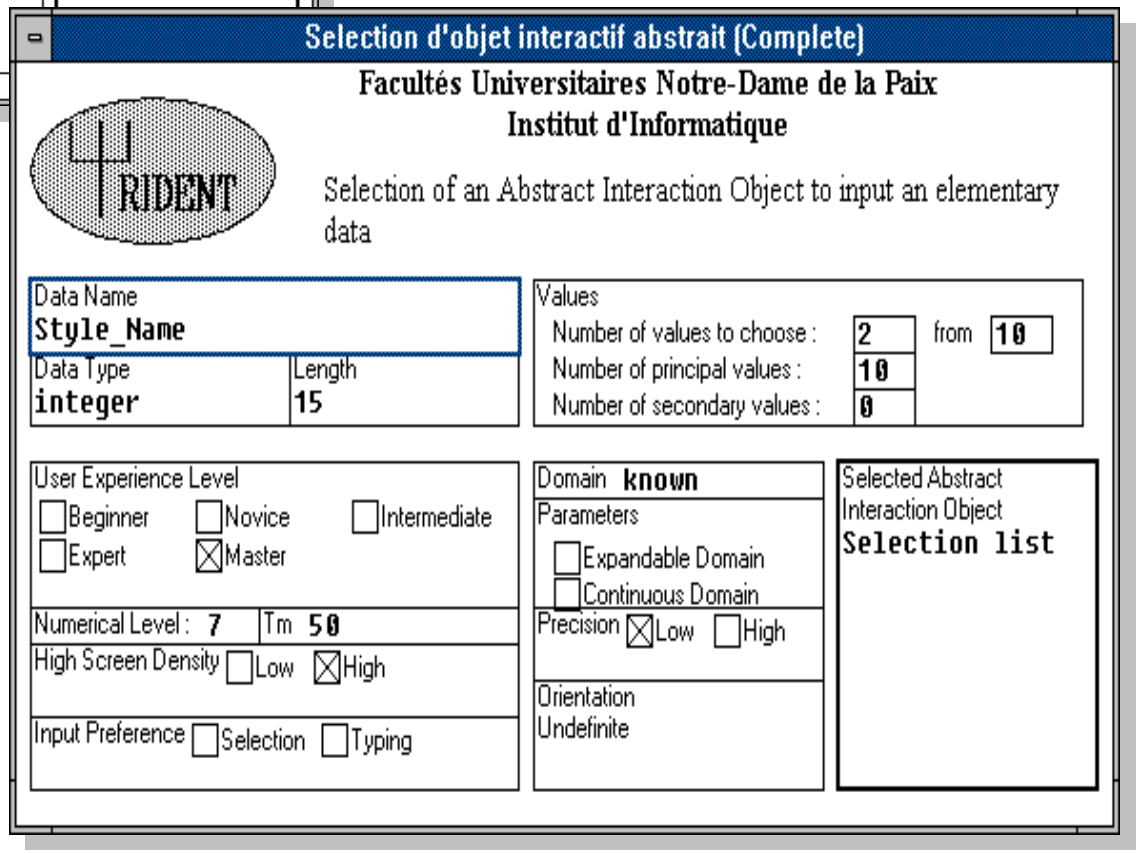


Interaction Object Selection



TRIDENT selects interaction techniques using a decision tree

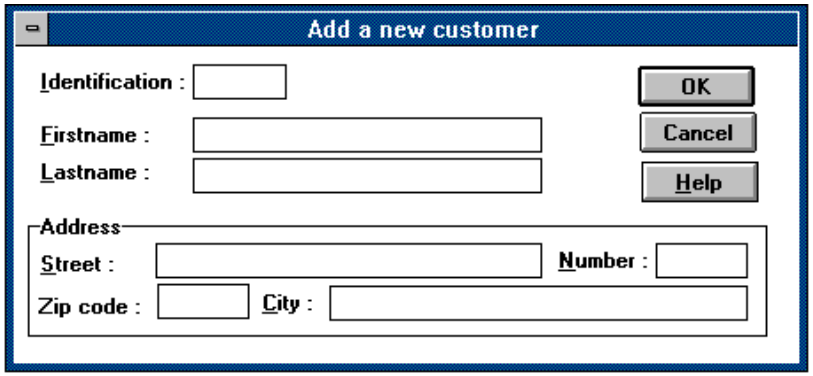
Developer specifies properties of data and user preferences

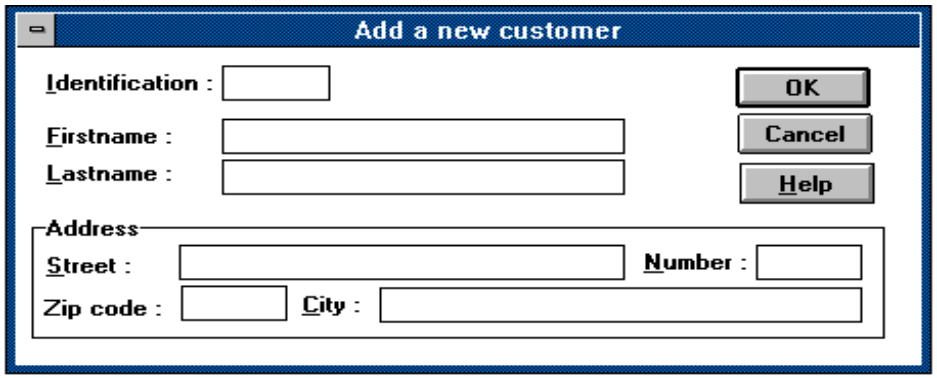


TRIDENT: Examples

Customer Support: 3 tasks

Window design based on info needed to perform tasks, and task sequencing. E.g. after finding customer task, modify address task is possible





EDICM [Cockton 93]

- Extended Designer's Intended Conceptual Model
- A notation for application models
  - » Object-oriented, framed based, multiple inheritance
  - » Multiple levels of abstraction
  - » Extensive command structure
- Model validation
  - » By reverse-engineering existing interfaces
  - » By conducting large-scale usability studies

EDICM Example

Object Class	Attributes	Commands
Documents	name format path to	rename&save set format move
Folders	name path to contents	rename move add to …
Drives	type contents name physical address	link to contents
Desktops	contents	add to …
Contexts	current folder current volume	set current folder set current volume change to desktop current volume

CLASS FOR a DrawingDocument
ISA — Document
INSTANCES — unlimited
ALIASES
CurrentDrive is CurrentDrive of AccessibleMacObjectS
CurrentFolder is CurrentFolder of AccessibleMacObjec
OpenFiles is OpenFiles of PIHR
Status is SaveState of SaveAsController of PIHR
ATTRIBUTES
Contents: DisplayTrees/EmptyTree
Format: (Drawing, Stationary, Pict, Pict2)/Drawing
FUNCTIONS
Make Contents for Drawing into <DisplayFile>
IMPLEMENTATION: DisplayFile <- FileRep(Conte
Make Contents for Stationary into <StatDisplayFile>
IMPLEMENTATION: StatDisplayFile <- StatFileRep(Cc

Multiple levels of abstraction allow multiple levels of model editing

E-R Models [Benyon 93]

- **E-R models as a notation to support design**
  - » **Primarily used to develop task models**
- **E-R models provide**
  - » **Common language for representing and talking about interfaces**
  - » **A mechanism to gain a *structured* insight into the design of an interface**
- **E-R paradigm**
  - » **Is limited in expressiveness**
  - » **May necessitate extensions to represent complex designs**

Dimensions for Comparing Tools (1)

- **Model expressiveness:**
  - What aspects of the design can be controlled explicitly
- **Roles**
  - Generation
  - Automated analysis
- **Level of automation**
  - Described based on level of the interface model
  - Levels
    - Task
    - Application
    - presentation/behavior (what)
    - presentation/behavior (how)
- **Environment tools**
  - What kinds of tools are available in that environment

Dimensions for Comparing Tools (2)

- **Maturity of tools**
  - » Scale of applications built
  - » Performance
- **Domain dependencies**
  - » Class of interfaces they can create
  - » Class of applications
- **Platform dependencies**
  - » Machine
  - » programming language
  - » window system/toolkit
- **Availability**
  - » Cost
  - » License required
  - » Not available

Comparison Table

	Model Expressiveness	Model Levels	Run-time Tools	Environment Tools	Level of Automation	Maturity	Domain Dependencies	Platform Dependencies	Availability
ITS	Extensive control, but limited set of applications	Application, Presentation, Behavior, Dialogue	Generation	Text editor	Low	Very high, applied commercially	Mostly business information systems, information kiosks	IBM-OS/2	None
Mecano	Restricted control	Application, Presentation, Behavior, Dialogue	Generation	Automated designer Model editor Interface builder	High	Medium, applied research	Form and graph-based interfaces	NeXT Step	Available to researchers No documentation
UIDE	High for input Low for output	Application, Presentation, Behavior, Dialogue	Generation Animated help	Design critics Automatic dialogue box generator Model editor	Low, high for dialogue boxes	Medium, but only small scale applications	Unknown	UNIX, X, C++ UNIX, Lisp for design critics	Available No documentation
Humanoid	Extensive control	Application Presentation Behavior, Dialogue	Generation Generation from partial models Balloon help	Model editors Design assistant	Low, high for dialogue boxes	Medium, several large applications	WIMP interfaces	UNIX, X, Lisp	Available to researchers No documentation
ADEPT	Moderate control	Task, Application, Presentation, Behavior, Dialogue, User, Workplace	Generation	Model editors	High	Medium	Form-based interfaces	Smalltalk	Unknown
Trident	Extensive control, but limited set of applications	Task, Application, Presentation, User, Workplace	Generation	Model editors, automatic designer, interface builder	High	Medium, several large applications	Mostly business information systems	UNIX, Windows	Unknown
GENIUS	Restricted to expressiveness of E-R notation	Application, Presentation, Dialogue	Generation	Layout generator, Dialog net editor	High for layout, moderate for behavior	Research	Database applications	SUN/UNIX	Unknown

Agenda

- Model-based paradigm
- Case studies: UIDE, Mecano
- Architectures
- Break
- Case studies
- Survey of Model-Based Tools
- Conclusions
- Questions

Conclusions

- Advantages
- Disadvantages
- Where will this technology be in 5 years?
- How can I use this technology today?
- Take-home messages

Advantages

- **Model allows development of tools for**
  - » **faster, cheaper interface development**
  - » **more reusable designs**
  - » **more portable interfaces**
  - » **more principled and consistent interfaces**
  - » **interfaces with more services (e.g., usability support)**

Disadvantages of Current Model-Based Systems

- **Building models requires effort**
  - » more than just drawing,
  - » but much less than programming
- **Limited control over interface designs**
- **Efficiency of generic run-time systems**
  - » lower than current application-specific interface code

Where will this technology be in 5 years?

- Better coverage of interface design space
  - » Richer models, better modeling languages
- Lower model building costs
  - » Modeling tools, demonstrational approaches
- Reusable generic models available
  - » Researchers currently merging individual models
- Incorporation of human factors research
  - » GOMS-based analysis of designs
  - » Run-time support for usability studies
- Widely available research model-based tools

How can I use this technology today?

- **Model-based systems can generate platform-independent interface specifications**
  - **An appropriate run-time system must be built for your platform**
- **Existing run-time libraries and toolkits can be incorporated into new model-based development environments**

Take-home messages

- **Interface building = Model building**

- **Interface models enable comprehensive development environments**

Agenda

- Model-based paradigm
- Case studies: UIDE, Mecano
- Architectures
- Break
- Case studies: Humanoid, ITS
- Survey of Model-Based Tools
- Conclusions
- Questions

Questions