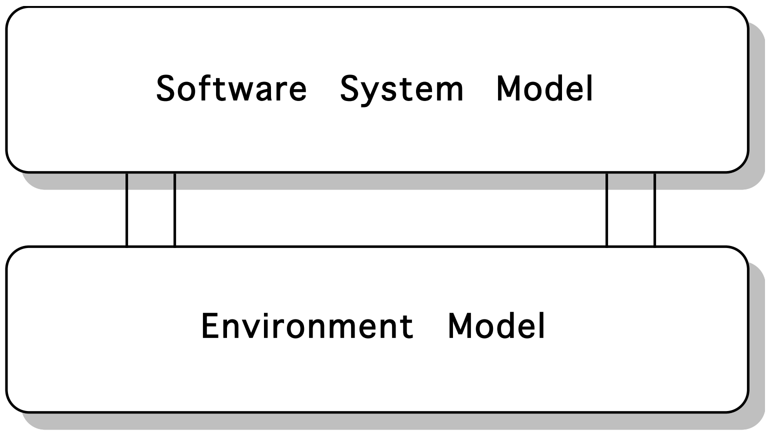
6. Basic Methods

Overview

- 6.1 Models
- 6.2 Taxonomy
- 6.3 Finite State Model
- 6.4 State Transition Model
- 6.5 Dataflow Model
- 6.6 User Manual

6.1 Models

- Existing techniques focus on functional specifications
  - the relation between the system and its environment
- Specifications vary greatly in style and notation
  - semi-formal (specialized notation)
  - formal (mathematics and logics)
- Each method is based on an underlying model of computing

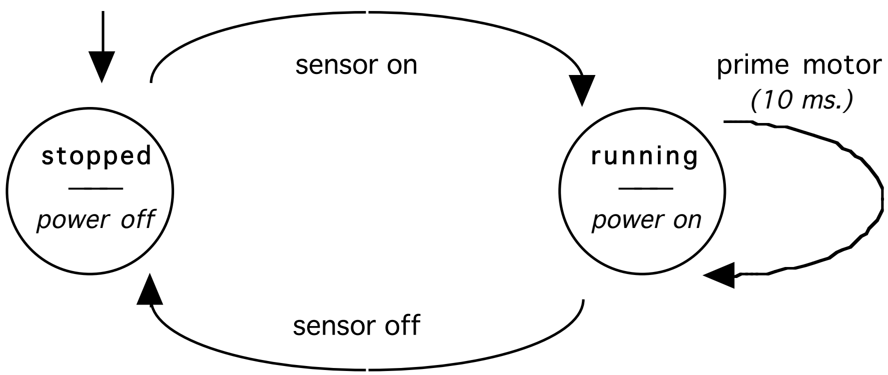


Choosing the Right Model

- The choice of model is determined by the need for precise and concise communication
- Aspects of the specification that do not fit the model are left out or are treated using other models
- When multiple models are used, consistency becomes a critical verification concern
- The document organization is determined by the nature of the model

A Finite State Machine Model

- External actions associated with entry in a state



Documentation implications

3. Specific Requirements

3.1 Initial state

Startup action

3.2 State S

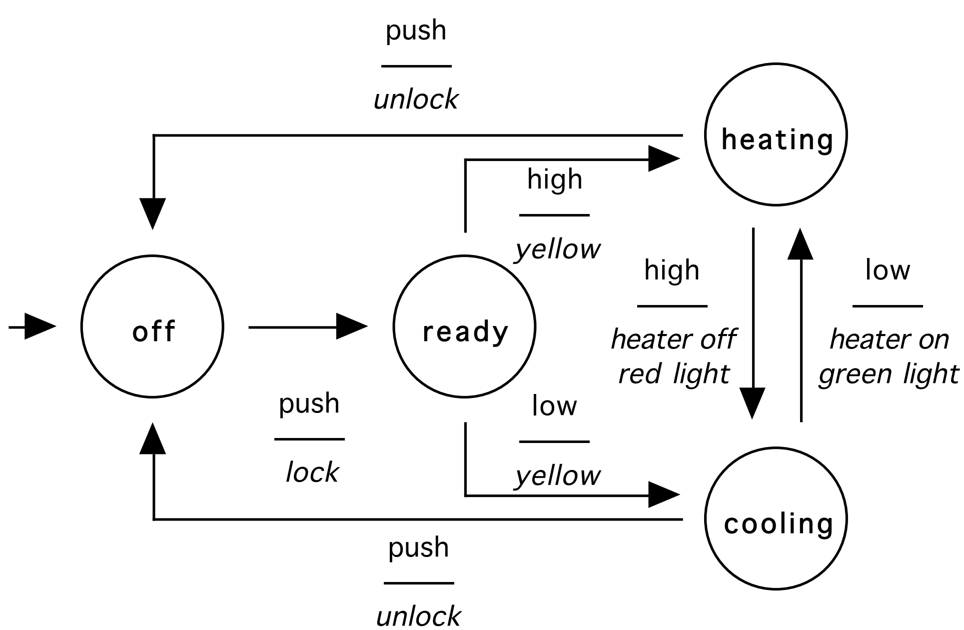
3.2.1Action A

3.2.2Event X

Next State S’

Another Machine Model

- External actions associated with transitions



Documentation implications

3. Specific Requirements

3.1 Initial state

Initialization

3.2 State A

3.2.1 Event X

Next State S'

Action A

6.2 Taxonomy

Type of underlying model

- Operational
  - finite state
  - state transition systems
  - concurrent processes
  - dataflow
  - applicative
- Descriptive
  - data models
  - algebraic
  - logic
  - object-oriented

6.3 Finite State Model

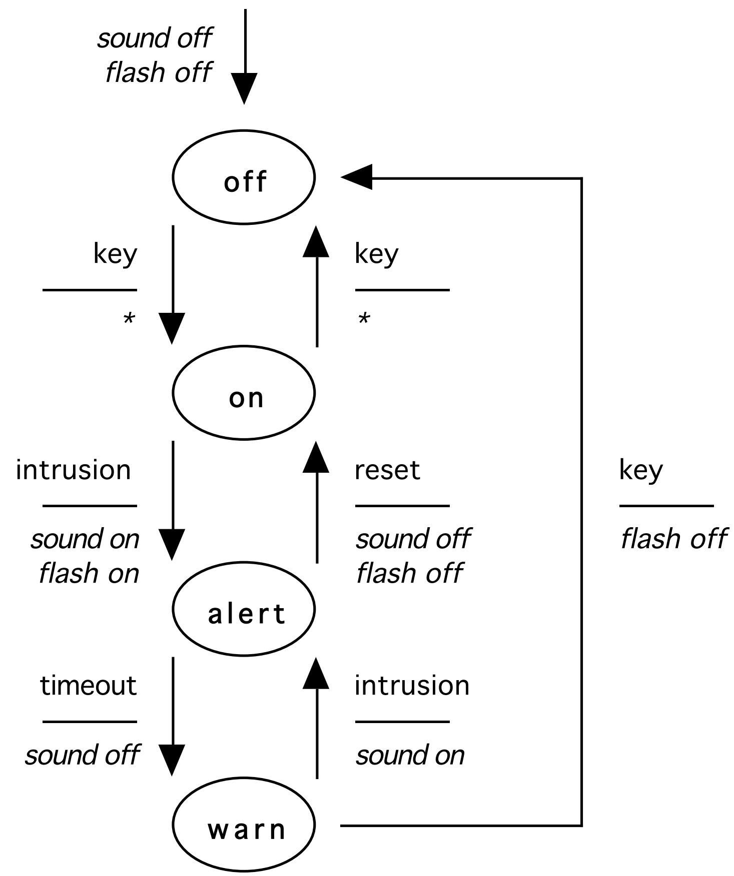
Reactive Control Applications

- Environment (abstract external interfaces)
  - sensor behavior can be reduced to a finite set of input events
  - actuator controls can be reduced to a finite set of output events
- System (software functionality)
  - control logic requires a finite set of states
  - behavior is deterministic

Model

Home safety alarm

- What is a “timeout” event?
- Is the specification correct?



Documentation

1. Introduction
2. General description
3. Specific requirements

   …

4. Performance requirements
5. Design constraints
6. Attributes
7. Other requirements

3.1 External interfaces

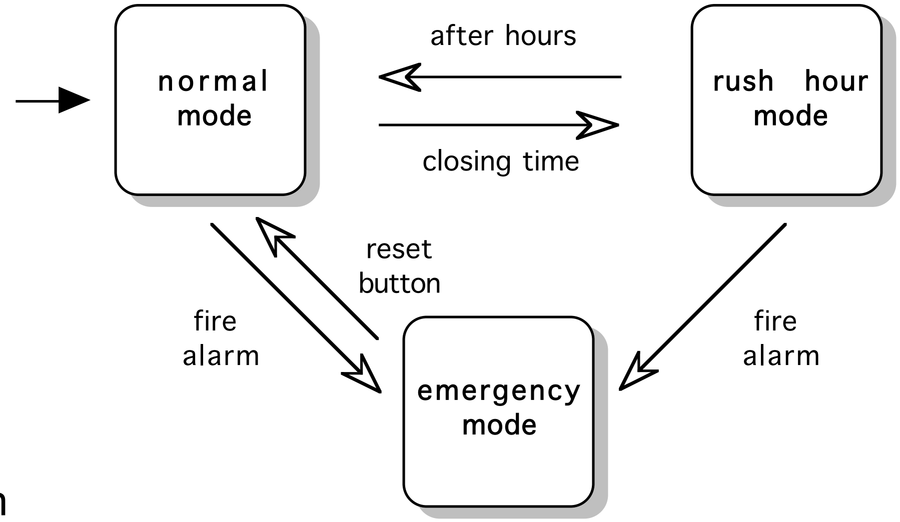
- *input events and output actions*
- *relation to physical devices*

3.2 Control logic

- *diagram, table, or text*

Complexity Control

- State explosion can be avoided often by separating the system behavior into distinct modes
- A finite state diagram can capture the mode transition rules
- Uniform rules must govern the transitions from mode to mode
  - a mode is always entered in the initial state or in specially denoted states
  - a mode can be exited from every state or only from marked states

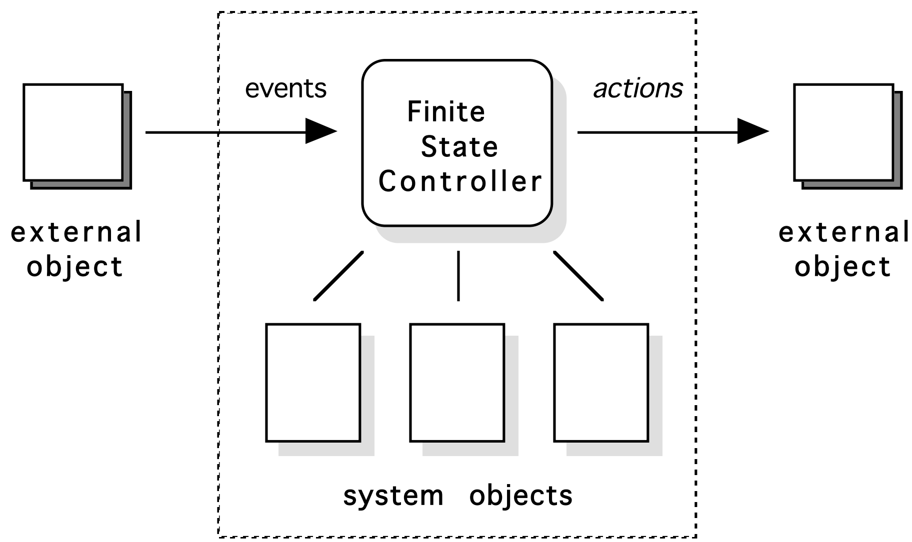


Object-Oriented Variations

- Interfaces are encapsulated as external objects
  - a step towards an object-oriented design
- Non-finite-state aspects of the system are encapsulated in internal objects
  - these objects have little or no implication on the subsequent design structure

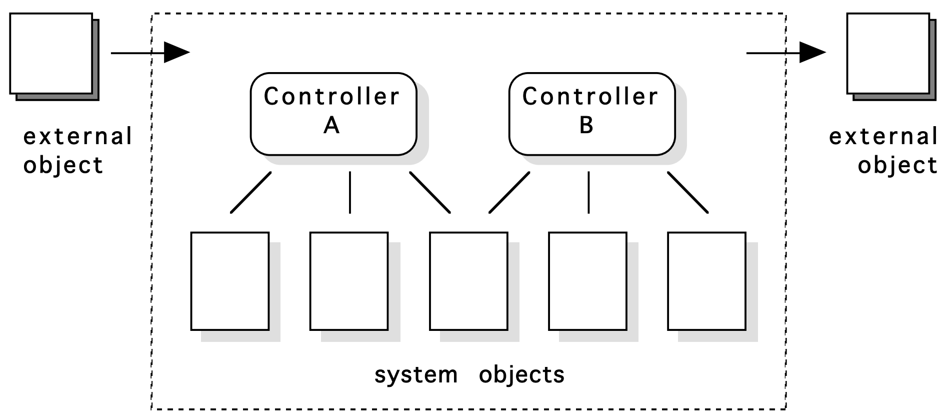
Object-Oriented Variations

- Events may be external or internal (conditions)
- Actions may be external or internal (operations)



Controlling State Explosion

- Model complexity can be reduced significantly by employing multiple parallel controllers
- Decoupling and limited object sharing simplify analysis
- Direct implementation of the model is sometimes feasible

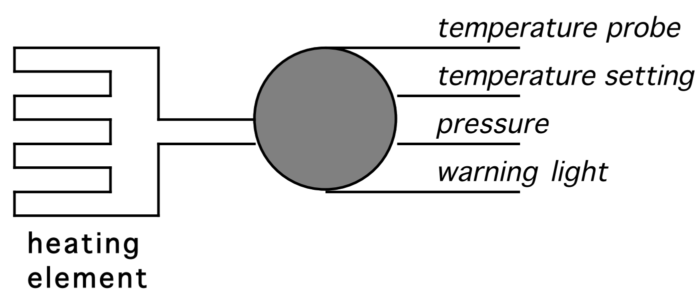


Case Study: Hot Water Heater

- During the preheating period (first 2 minutes) the heating element is on continuously.
- Continuous hot water output requires the heating element to stay on for 50% of the time while water is needed (low pressure).
- If the heater is no longer able to provide a continuous hot water supply (temperature drop), a warning light must turn on and the heating element must remain on for as long as water is needed.

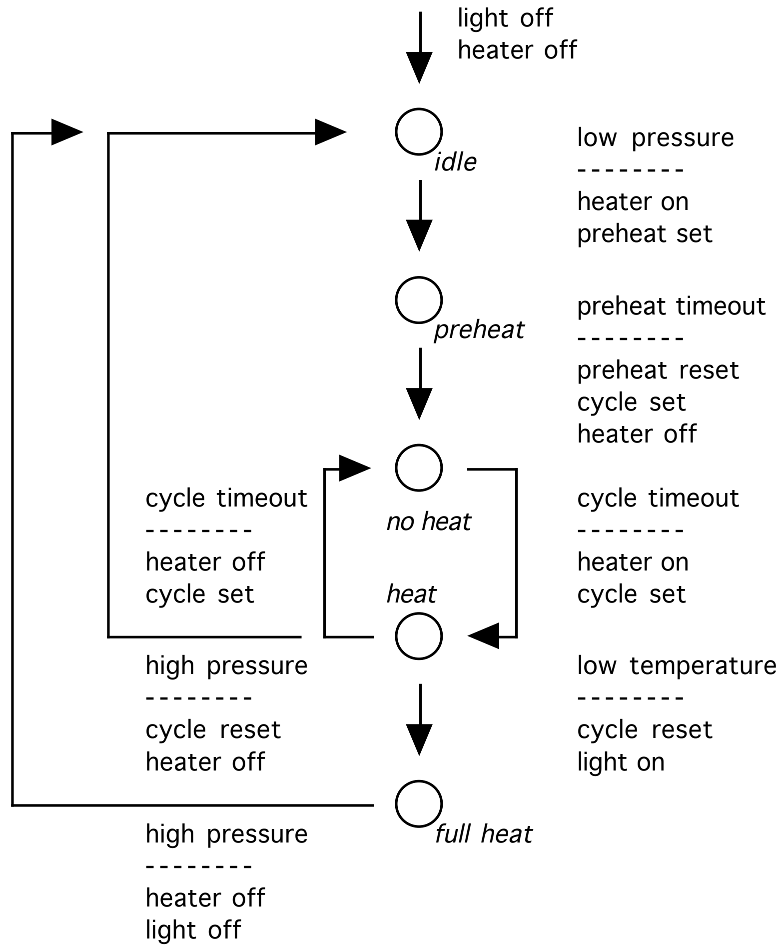
Case Study: Hot Water Heater

- Environment (objects)
  - heating element
    - on/off
  - temperature probe & temperature setting
    - low temperature
  - pressure sensor
    - low/high
  - warning light
    - on/off
- Virtual environment objects
  - preheat timer (2 minutes)
    - set, reset, timeout
  - cycle timer (1 second)



Case Study: Hot Water Heater

- Analysis:
  - Can the temperature drop without heating coming on?
  - *[idle]*



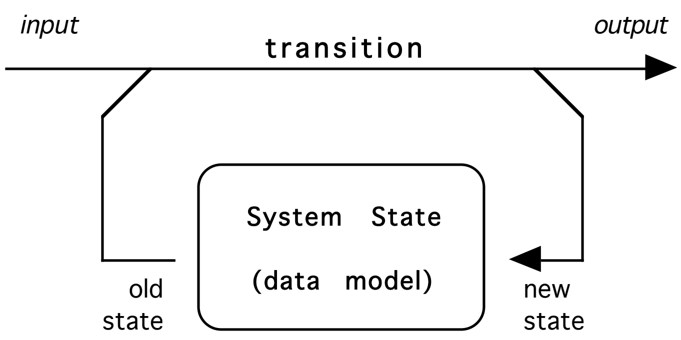
6.4 State Transition Model

Data Processing Applications

- Environment (abstract external interfaces)
  - inputs are data records supplied in a nondeterministic manner
  - outputs are immediately consumed data records
- System (software functionality)
  - the system maintains significant amounts of data
  - system operation involves data updating and output computation in response to input arrival

Model

- Data models are used to characterize the system state
- Operations
  - are associated with the arrival of input
  - are atomic
  - result in a state change and output generation
- Virtual inputs may be added to model periodic activities, time-dependent processing, etc.



Documentation

1. Introduction
2. General description
3. Specific requirements

   …

4. Performance requirements
5. Design constraints
6. Attributes
7. Other requirements

3.1 External interfaces

- *inputs and outputs*
- *relation to physical devices*

3.2 Data model

- *text, tables, etc.*

3.3 Operations

- *grouped by some organizing principle*

Operational Specifications

- Operations can be specified as abstract procedures
- They may be described using natural language or pseudocode

cancel (passenger P, flight F, date D)

if	there is some reservation for R seats for passenger P on flight F on date D
then	delete this reservation and decrease by R the number of seats reserved for flight F on date D
else	ignore the request and generate an error message
endif	

Axiomatic Specifications

- Operations can be specified as pairs of assertions capturing the relation between the data state before and after the operation

cancel (passenger P, flight F, date D)

precondition:

(1) R seats are reserved for passenger P on flight F on date D

(2) the total number of reserved seats for flight F on date D is K

postcondition:

(1) there are no reservations for passenger P on flight F on date D

(2) the number of reserved seats for flight F on date D is (K-R)

exception:

(1) an error message is displayed

Data Modeling

- Data modeling must emphasize information contents and simplicity, not design
- Any data modeling technique can be used
  - abstract variables
  - relations
  - entity-relation diagrams
  - objects
- Concept modeling (e.g., classes and inheritance) is helpful for complex systems

Abstract Variables

- The system state may be characterized by a set of components that represent information maintained by the system
- The domain of values assumed by such components should be application specific
- The representation should be abstract, mathematical forms are preferred

Abstract Variables

An airline reservation system

- Job queue—sequence of reservation and cancellation requests from travel agents

  (agent, action, passenger, flight number, date, number of sets)

- Schedule—set of flights

  (flight number, date, time, destination, capacity, bookings)

- Volume—counter of processed requests

Relations

- Relational models are particularly useful because of their intuitive appeal—they may be viewed as tables

Schedule

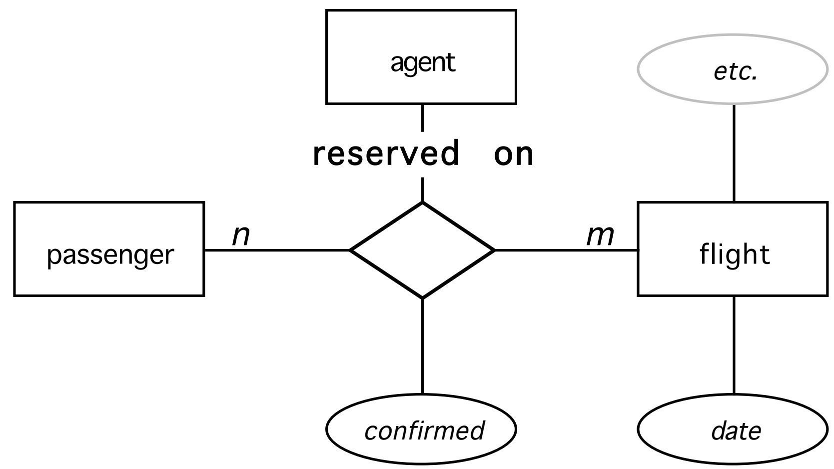
Flight	Date	Time	Dest.	Capacity	Booked
TW23	23-1-96	10:23	...	100	24
NW183	23-1-96	...	...	50	41
…	…	...	...	…	…

Reservations

Flight	Date	Name	Booked
TW23	23-1-96	Mary	1
TW23	23-1-96	Bob	3
TW23	23-1-96	Dana	1
…	…	…	…

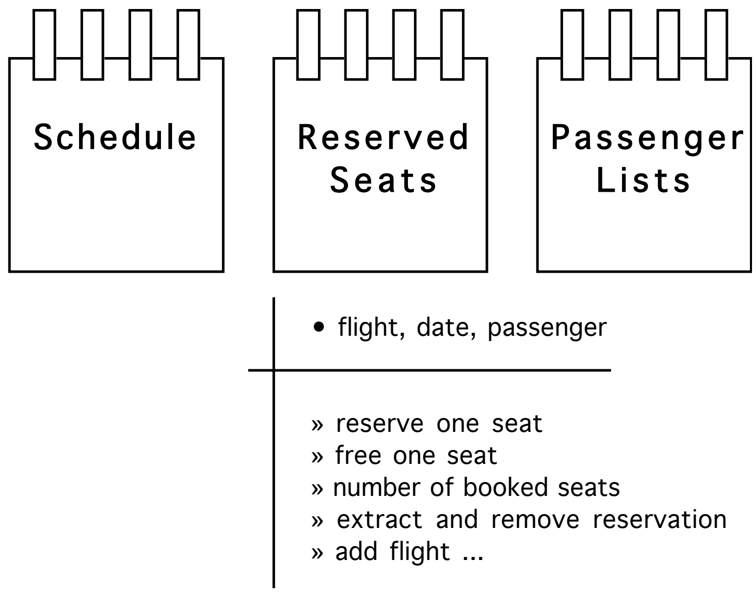
Entity-Relation Diagrams

- The approach has its roots in the database schema definition
- The key concepts are
  - entity set (contains entities)
  - relations over entity sets (may be recursive)
  - attributes (of entities and relations)



Objects

- Objects can be used to encapsulate data and common operations on it
- Objects are independent of each other
- Objects simplify the specification of the processing requirements



Integrity

- Semantic constraints capture important aspects of the application

    "reservations and cancellations involve only future flights"

- Local constraints can be tested by examining some data component in isolation

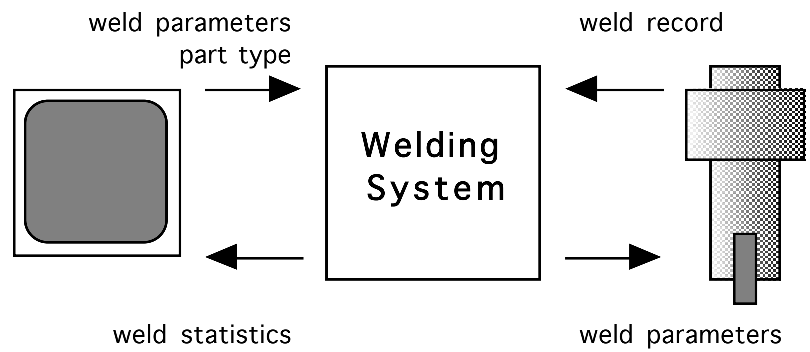
    "a flight has a single departure time for each individual date, if any"

- Global constraints involve relations among multiple components

    "the number of reserved seats cannot exceed the flight capacity plus the general overbooking factor"

Case Study: Welding

- Consider an automated welding tool used in the assembly of plastics in a medical application.
- Complete records must be maintained for each performed weld (e.g., duration, pressure, frequency).
- The goal is to be able to extract data about a specific part and to analyze failure patterns.



Case Study: Welding

- Data Model
  - Current part
  - Specifications
    - part number, duration, pressure, frequency
  - Statistics
    - part number, number of welds, number of failures
- Questions
  - Calibration, tolerances, etc.
- Operations
  - Work on part N
  - New welding record
  - Add part specification
  - Remove part specification
  - Display statistics
- Questions
  - Can modifications take place during welding?
  - How long are statistics kept?

6.5 Dataflow Model

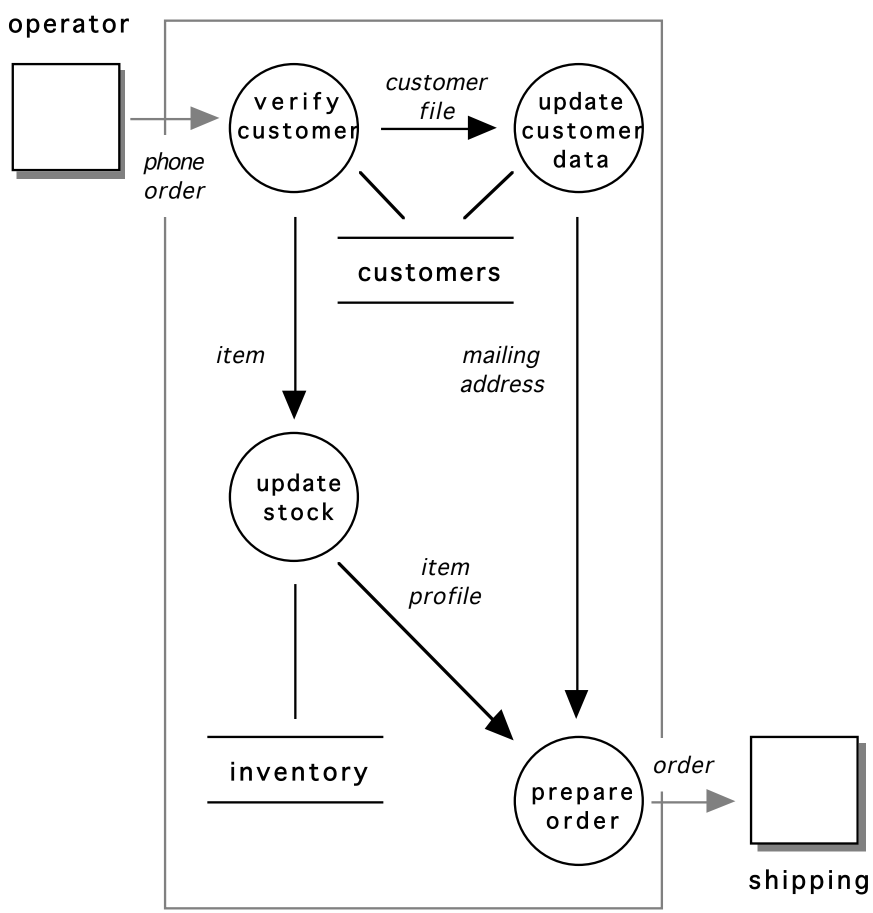
Process Control Applications

- Environment (abstract external interfaces)
  - inputs (data and events) are stimuli
  - outputs are externally visible responses
- System (software functionality)
  - the system maintains a certain amount of data
  - the system's response to inputs is a complex set of internal actions and external outputs

Model

- Dataflow diagrams notation
  - functions—deterministic input/output transformations triggered by the presence of all needed inputs
  - flows—unbounded queues
  - stores—global system data
  - terminators—interface models
  - minispecs—semantics of the lowest level functions
- Top-down functional decomposition provides a complexity control mechanism
- Data dictionary tracks the (global) names and their interpretation
- Execution model
  - fully concurrent interpretation
  - explicitly marked critical sections
  - stimulus/response interpretation

Notation


operator
phone
order
verify
customer
customer
file
update
customer
data
customers
item
mailing
address
update
stock
item
profile
inventory
prepare
order
order
shipping

Documentation

1. Introduction
2. General description
3. Specific requirements

   …

4. Performance requirements
5. Design constraints
6. Attributes
7. Other requirements

3.1 Context diagram

- *system and its environment*
- *terminator definitions*

3.2 Dataflow diagrams

- *hierarchical decomposition of functions*
- *minispecs for lowest-level functions*

3.3 Data dictionary

- *flows and stores*

Critique

- Semantics of dataflow is often left undefined
  - atomicity (granularity of actions)
  - fairness (scheduling)
- Concurrent semantics are complex and may allow race conditions to occur
- The lack of minispecs for the high-level functions makes analysis and precise understanding difficult
- The diagrams are easy to understand only when the complexity is low, the development cost is high, and the communication bandwidth may be low
- Improper abstraction for the terminators leads to complex specifications

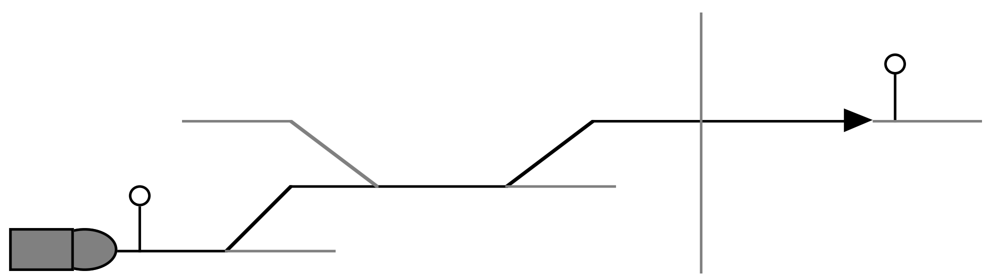
Complexity Control

- Hierarchical decomposition
- Horizontal partitioning

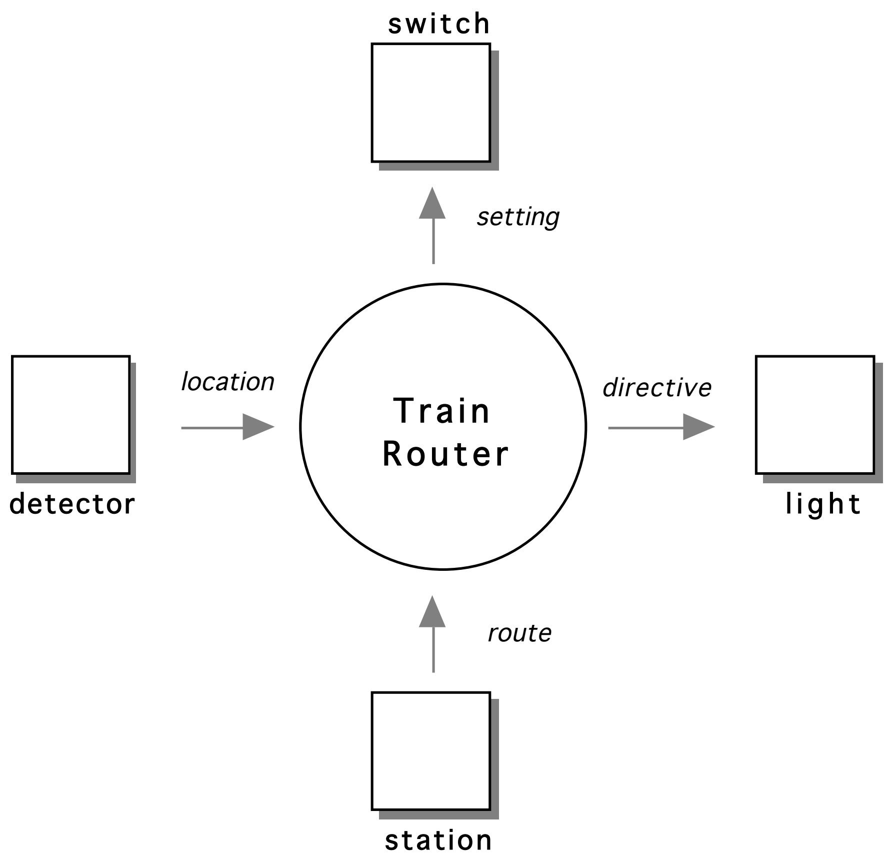
- Terminators can become abstract objects
- Stores can become abstract data objects
- Messages can become objects
- All objects may be instances of classes
- Classes may be defined in terms of each other by employing inheritance
- Inheritance and instance diagrams can by used to capture the relations among classes and objects

Case Study: Train Routing

- Consider a system designed to automatically route trains.
- Upon arrival at some light, the train is assigned a new route which takes it to the next light.
- The system selects the proper position for each switch along the route.



Case Study: Train Routing



Case Study: Train Routing

- Data stores
  - network layout
  - traffic
- Stimuli
  - arrival at a light
    - unlock side protection lights
    - identify blocked trains
    - reprocess their routes
    - get new route
    - lock side protection lights on red
    - turn light green (if successful)

6.6 User Manual

Applications Involving Human Interfaces

- Effective specifications often require the integration of multiple related models
- Human/computer interactions are too complex for commonly used requirements techniques
- The User Manual can be used as a substitute for large sections of the SRS

Documentation

1. Introduction
2. General description
3. Specific requirements

   …

4. Performance requirements
5. Design constraints
6. Attributes
7. Other requirements

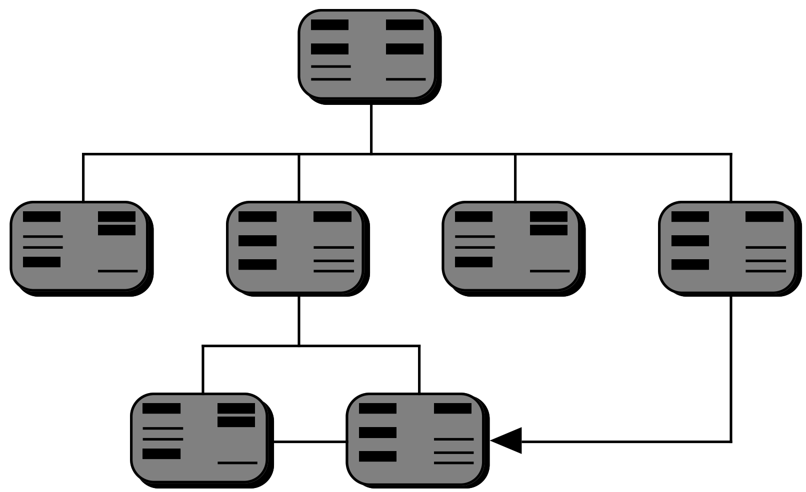
3. Specific requirements
   - *conceptual model*

   3.1 Navigation
   - *screen types and flow among them*
   - *common interactions*

   3.2 Screens
   - *layout and information contents*
   - *command semantics*

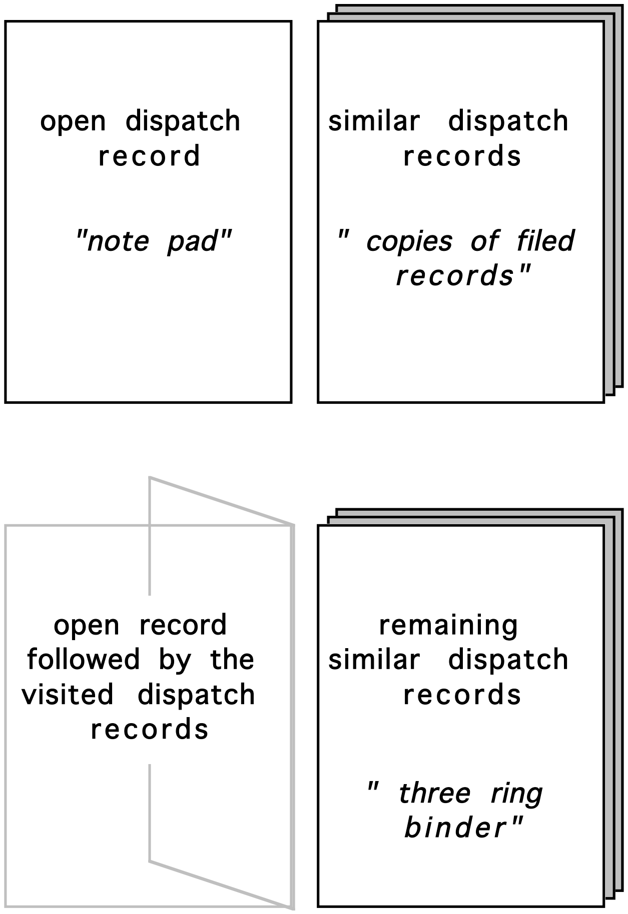
Navigation

- Identify the screens and the permissible transitions among them as a graph
- Identify the events that cause moves from one screen to another
- Identify interactions common among screens and specific to the user interface paradigm in use



Conceptual Model

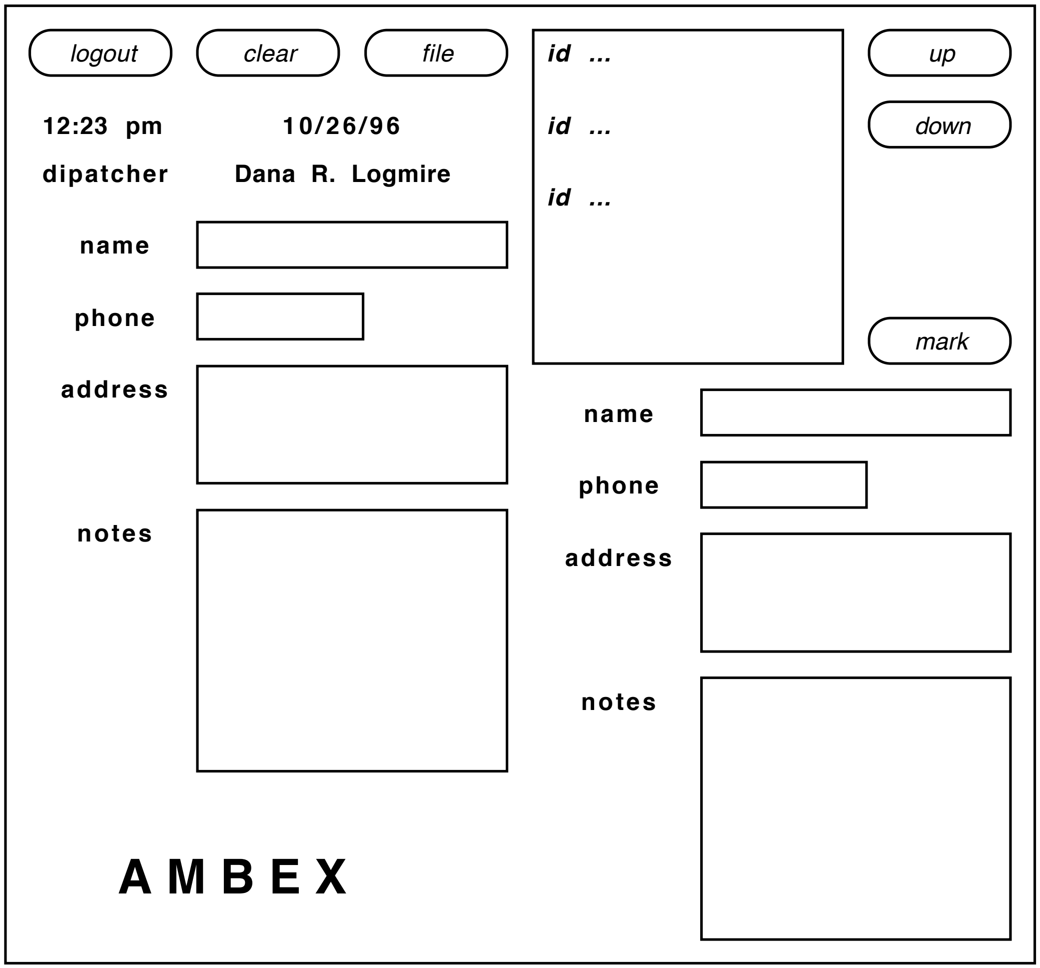
- The conceptual model is a mental map which helps the user anticipate correctly the expected system behavior (navigation, information, commands)
- Metaphors are effective tools for building conceptual models because the user can rely on previous experience



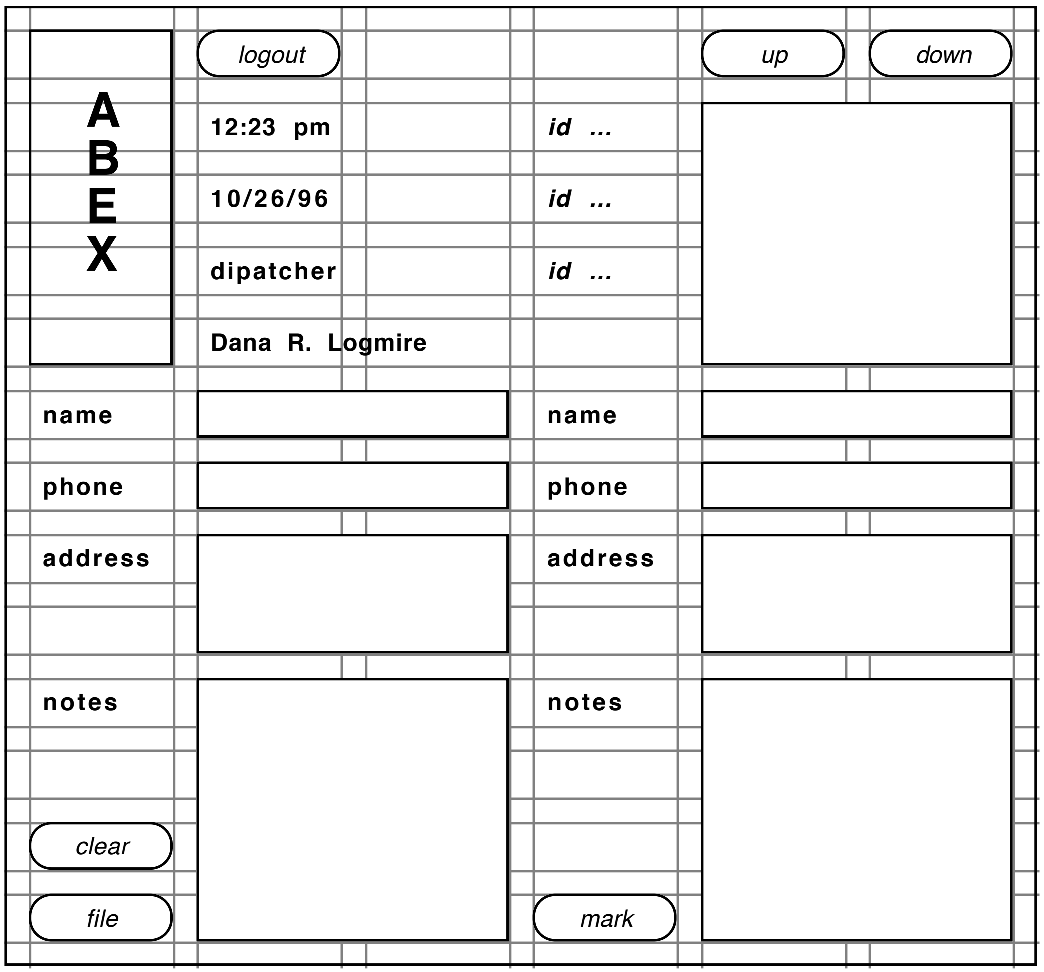
Screens

- Maximize readability (layout and structure)
  - simplicity
  - regularity of structure
  - minimality
- Minimize the potential for operator errors (interactions and general feel)
  - predictability
  - uniformity
  - forgiveness
- Optimize performance for the typical workflow
- Engineer each screen and the entire ensemble
  - grid-based design

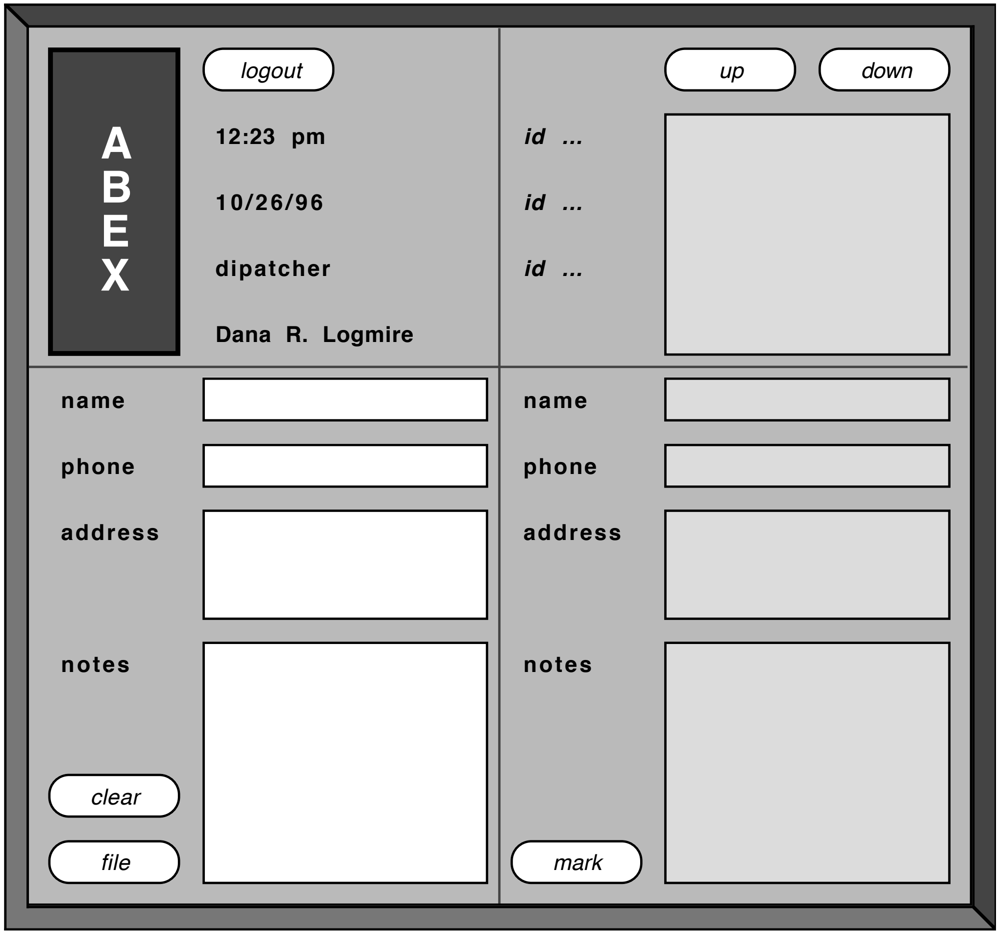
Ad-Hoc Solution



A Grid-based Design



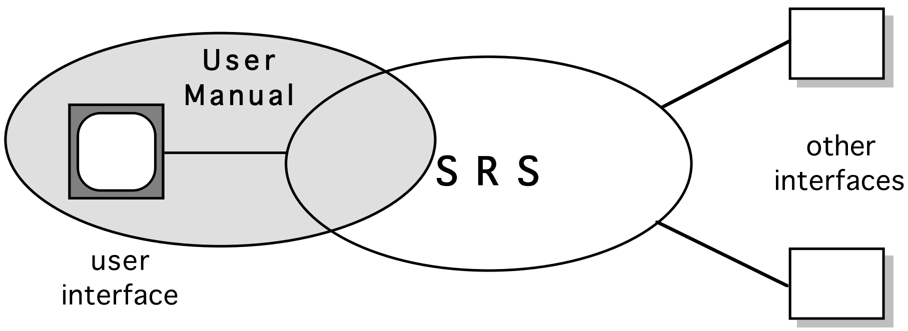
Final Design

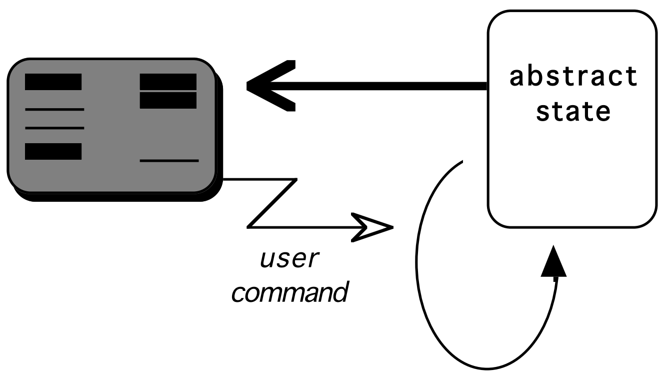


Multiple Models

- If the user manual cannot capture all functional aspects of the system two models are needed
- Consistency must be maintained
- The state representation should be that used in the SRS (common abstract state)
- User commands are specified only once as state transitions over the abstract state of the system
- State information needed for screen presentation is assumed to be directly available

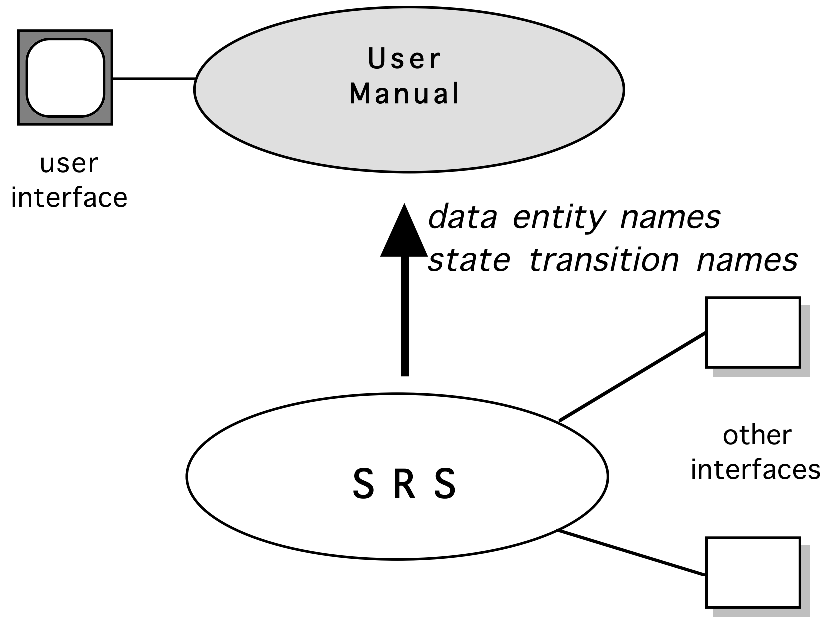
Multiple Models





Building on the SRS

- If most of the complexity in the user interface is due to the navigation and interaction pattern its development can be postponed until the SRS is completed
- Data and transition names can be used directly as in the SRS when needed



Case Study: Paint Drying

- Consider a controller for a paint drying application.
- Painted parts enter the oven one at a time.
- The control software ensures that the part is dried at the specified temperature for the specified duration before leaving the oven.
- The operator can override the duration or the dry cycle for custom parts.

- The system monitors duration and temperature settings and allows for changes in these settings.
- Only one setting is used at any one time.
- Production costs limit the interface to a simple display and two buttons.