Models of the System

Overview

Standard Formalisms
software engineering notations used to specify the required behaviour of specific interactive systems

Interaction Models
special purpose mathematical models of interactive systems, used to describe usability properties at a generic level

Status/Event Analysis
an example of an engineering level method drawing on both formal modelling and naïve psychology
Relationship with dialogue

Dialogue modelling is linked to semantics.

System semantics affects the dialogue structure.

But the bias is different.

Rather than dictate what actions are legal, these formalisms tell what each action does to the system.
Irony

Computers are inherently mathematical machines.

Humans are not.

Formal techniques are well accepted for cognitive models of the user and the dialogue (what the user should do).

Formal techniques are not yet well accepted for dictating what the system should do for the user!
General computational formalisms

Standard software engineering formalisms can be used to specify an interactive system.

Referred to as formal methods

Model based describe system states and operations
  - Z, VDM

Algebraic describe effects of sequences of actions
  - OBJ, Larch, ACT-ONE

Extended logics describe when things happen and who is responsible
  - temporal and deontic logics
The uses of SE formal notations

For communication

- common language
- remove ambiguity (possibly)
- succinct and precise

For analysis

- internal consistency
- external consistency
  - with eventual program
  - with respect to requirements (safety, security, HCI)
- specific versus generic
## Model based formalisms

Mathematical counterparts to common programming constructs

<table>
<thead>
<tr>
<th>Programming</th>
<th>Mathematics</th>
</tr>
</thead>
<tbody>
<tr>
<td>types</td>
<td>sets</td>
</tr>
<tr>
<td>basic types</td>
<td>basic sets</td>
</tr>
<tr>
<td>constructed types</td>
<td>constructed sets</td>
</tr>
<tr>
<td>records</td>
<td>unordered tuples</td>
</tr>
<tr>
<td>lists</td>
<td>sequences</td>
</tr>
<tr>
<td>functions</td>
<td>functions</td>
</tr>
<tr>
<td>procedures</td>
<td>relations</td>
</tr>
</tbody>
</table>
The model based method

Example: a graphic drawing package

<table>
<thead>
<tr>
<th>Line</th>
<th>Rectangle</th>
<th>Ellipse</th>
<th>Unselect</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Line" /></td>
<td><img src="image2" alt="Rectangle" /></td>
<td><img src="image3" alt="Ellipse" /></td>
<td><img src="image4" alt="Unselect" /></td>
</tr>
</tbody>
</table>

Points are ordered pairs.

\[ \text{Point} == \mathbb{N} \times \mathbb{N} \]

Shapes can be of varying types.

\[ \text{Shape\_type} == \text{line} | \text{ellipse} | \text{rectangle} \]
More type definitions

A graphic object is defined by its shape type, width, height, and centre position.

\[
\text{Shape}
\]
\[
\begin{align*}
&\text{type : Shape\_type} \\
&\text{width, height : } \mathbb{N} \\
&\text{centre : Pt}
\end{align*}
\]

A collection of graphic objects can be identified by a ‘lookup dictionary’

\[
\begin{align*}
&[\text{Id}] \\
&\text{Shape\_dict == Id } \mapsto \text{Shape}
\end{align*}
\]
Defining the state

The system state contains a dictionary of created objects and a set of selected objects.

\[
\begin{array}{l}
\text{State} \\
\quad \text{shapes} : \text{Shape\_dict} \\
\quad \text{selection} : \mathbb{P} \text{Id} \\
\quad \text{selection} \subseteq \text{dom shapes}
\end{array}
\]

Initially, there are no shapes in the dictionary.

\[
\begin{array}{l}
\text{Init\_State} \\
\quad \text{State}' \\
\quad \text{shapes}' = \{\}
\end{array}
\]
Defining operations

State change is represented as two copies of the state

before — State
after — State'

\[ \Delta \text{State} \]
\[ \text{State} \]
\[ \text{State}' \]

The Unselect operation deselects any selected objects.

\[ \text{Unselect} \]
\[ \Delta \text{State} \]
\[ \text{selection}' = \{\} \]
\[ \text{shapes}' = \text{shapes} \]
Interface issues

**Framing problem**  “everything else stays the same” can be complicated with state invariants

**Internal consistency**  do operations define any legal transition?

**External consistency**  must be formulated as theorems to prove

  Clear for refinement, not so for requirements

**Separation**  of system functionality and presentation is not explicit
Algebraic notations

Model based notations emphasise constructing an explicit representations of the system state.

Algebraic notations provide only implicit information about the system state.

Model based operations are defined in terms of their effect on system components.

Algebraic operations are defined in terms of their relationship with the other operations.
types
State, Pt

operations
init : \rightarrow State
make_ellipse : Pt \times State \rightarrow State
move : Pt \times State \rightarrow State
unselect : State \rightarrow State
delete : State \rightarrow State

axioms
for all \ st \in State; \ p \in Pt \bullet
1. delete(make_ellipse(st)) = unselect(st)
2. unselect(unselect(st)) = unselect(st)
3. move(p, unselect(st)) = unselect(st)
Issues for algebraic notations

**Ease of use** a different way of thinking than traditional programming

**Internal consistency** are there any axioms which contradict others?

**External consistency** with respect to executable system less clear

**External consistency** with respect to requirements is made explicit and automation possible

**Completeness** is every operation completely defined?
Extended logics

Model based and algebraic notations make extended use of propositional and predicate logic.

**Propositions** expressions made up of atomic terms \( p, q, r, \ldots \) composed with \( (, ), \land, \lor, \neg, \Rightarrow \), etc.

**Predicates** propositions with variables, e.g., \( p(x) \) and quantified expressions \( \forall, \exists \).

These are not convenient for expressing time, responsibility and freedom, notions sometimes needed for HCI requirements.
Temporal logics

Time considered as succession of events

Basic operators:

- always \( \Box \) \( \Box (G \text{ funnier than } A) \)
- eventually \( \Diamond \) \( \Diamond (G \text{ understands } A) \)
- never \( \lnot \Box \) \( \lnot \Box (\text{rains in So. Cal.}) \)

Other bounded operators:

- \( p \text{ until } q \) weaker than \( \Box \)
- \( p \text{ before } q \) stronger than \( \Diamond \)
Explicit time

These temporal logics do not explicitly mention time, so some requirements cannot be expressed.

Active research area, but not so much with HCI

Gradual degradation more important than time-criticality

Myth of the infinitely fast machine
Deontic logics

For expressing responsibility, obligation between separate agents (e.g., the human, the organisation, the computer)

permission \( \text{per} \)
obligation \( \text{obl} \)

For example,

\[
\text{owns}( \text{Jane, file } \text{‘fred’} ) \Rightarrow \\
\text{per}( \text{Jane, request(’print fred’)} )
\]

\[
\text{performs}( \text{Jane, request(’print fred’) ) } \Rightarrow \\
\text{obl}( \text{lp3, print(file ‘fred’)} )
\]
Issues for extended logics

**Safety properties** stipulating that bad things do not happen

**Liveness properties** stipulating that good things do happen

**Executability versus expressiveness** easy to specify impossible situations; difficult to express executable requirements; settle for eventual executable

**Group issues and deontics** obligations for single-user systems have personal impact; for groupware, we must consider implications for other users.
Interaction models

General computational models were not designed with the user in mind.

We need models that sit between the software engineering formalism and our understanding of HCI.

**formal**  the PIE model for expressing general interactive properties to support usability

**informal**  interactive architectures (MVC, PAC, ALV) to motivate separation and modularisation of functionality and presentation

**semi-formal**  status-event analysis for viewing a slice of an interactive system that spans several layers
The PIE model

A black-box model

\[
P \xrightarrow{I} E \xrightarrow{\text{display}} D \xrightarrow{\text{result}} R
\]
More formally

\[ [C, E, D, R] \]
\[ P \mapsto \text{seq} C \]

\[ I : P \rightarrow E \]
\[ \text{display} : E \rightarrow D \]
\[ \text{result} : E \rightarrow R \]

Alternatively, we can derive a state transition function from the PIE.

\[ \text{doit} : E \times P \rightarrow E \]
\[ \text{doit}(I(p), q) = I(p \bowtie q) \]
\[ \text{doit}(\text{doit}(e, p), q) = \text{doit}(e, p \bowtie q) \]
Expressing properties

WYSIWYG

What does this really mean, and how can we test product X to see if it satisfies a claim that it is WYSIWYG?

Limited scope general properties which support WYSIWYG.

Observability what you can tell about the current state of the system from the display

Predictability what you can tell about the future behaviour
Observability and predictability

Two possible interpretations of WYSIWYG:

What you see is what you:

- *will get* at the printer
- *have got* in the system

Predictability is a special case of observability
Formally

\[ P \xrightarrow{l} E \]

\[ \exists f : D \rightarrow R \bullet \]
\[ \forall e : E \bullet f(\text{display}(e)) = \text{result}(e) \]

Determining effect from display:

\[ \exists f : D \rightarrow E \bullet \]
\[ \forall e \in E \bullet f(\text{display}(e)) = e \]
Relaxing the property

\[ \exists f : O \to R \quad \forall e \in E \quad f(\text{observe}(e)) = \text{result}(e) \]

\[ \exists f : O \to E \quad \forall e \in E \quad f(\text{observe}(e)) = e \]
Reachability and undo

Reachability — getting from one state to another.

\[ \forall e, e' \in E \bullet \exists p \in P \bullet doit(e, p) = e' \]

Too weak

Undo — reachability applied between current state and last state.

\[ \forall c \in C \bullet doit(e, c \triangleleft undo) = e \]

Impossible except for very simple system with at most two states!

Better models of undo treat it as a special command to avoid this problem
Issues for PIE properties

**Insufficient** define necessary but not sufficient properties for usability.

**Generic** can be applied to any system

**Proof obligations** for system defined in SE formalism

**Scale** how to prove many properties of a large system

**Scope** limiting applicability of certain properties

**Insight** gained from abstraction is reusable
Status/event analysis

semi-formal technique
“engineering” level analysis
based on formal models
uses naïve psychology

clocks and calendars as example
status – analogue watch face

event – an alarm
Properties of events

status change event
- the passing of a time

actual and perceived events
- usually some gap

polling
- glance at watch face
- status change becomes perceived event

granularity
- birthday – days
- appointment – minutes
Design implications

actual/perceived lag…
matches application timescale?

too slow
  • response to event too late
  • e.g., power plant emergency

too fast
  • interrupt more immediate task
  • e.g., stock level low
Naïve psychology

Predict where the user is looking

mouse – when positioning
insertion point – intermittently when typing
screen – if you’re lucky

Immediate events

audible bell – when in room (and hearing)
peripheral vision – movement or large change

Closure

lose attention (inc. mouse)
concurrent activity
Example – email interface

mail has arrived!
timeline at each level

Perceived event in minutes – not guaranteed

<table>
<thead>
<tr>
<th>alternative</th>
<th>timescale</th>
</tr>
</thead>
<tbody>
<tr>
<td>explicit examination</td>
<td>– hours/days</td>
</tr>
<tr>
<td>audible bell</td>
<td>– seconds</td>
</tr>
</tbody>
</table>

want minutes – guaranteed
Example – screen button widget (i)

screen button often missed, ... but, error not noticed

a common widget, a common error: Why?

Closure

mistake likely – concurrent action

not noticed – semantic feedback missed

Solution

widget feedback for application event

*perceived event* for the user

N.B., an expert slip – testing doesn’t help
Screen button widget (ii)

A HIT

<table>
<thead>
<tr>
<th>application</th>
<th>dialogue</th>
<th>screen</th>
<th>user</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>do delete</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>changes in text</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>depress mouse button over 'delete'</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>highlight 'delete'</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>release mouse button</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>remove highlight</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>closure so NO perceived feedback</td>
<td></td>
</tr>
</tbody>
</table>

or a MISS

<table>
<thead>
<tr>
<th>application</th>
<th>dialogue</th>
<th>screen</th>
<th>user</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>depress mouse button over 'delete'</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>highlight 'delete'</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>move off 'delete'</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>remove highlight</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>release mouse button</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>no feedback</td>
<td></td>
</tr>
</tbody>
</table>