Introduction

Requirements Specification

Desired Properties

Kinds of Requirements

Analysis Techniques

Documents

Dictionary, Specification

Specification Languages

Natural Language

Decision Tables

Syntax, Semantics

Completeness, Consistency, ...

Scenarios

User Stories, Use Cases

Working Definition: Software

Live Sequence Charts

Syntax, Semantics

Discussion

Tell Them What You’ve Told Them ...

Requirements Documents are important—e.g., for negotiation, design & implementation, documentation, testing, delivery, re-use, re-implementation.

A Requirements Specification should be correct, complete, relevant, consistent, neutral, traceable, objective.

Note: vague vs. abstract.

Requirements Representations should be easily understandable, precise, easily maintainable, easily usable.


It is the task of the analyst to elicit requirements.

Natural language is inherently imprecise, counter-measures:

natural language patterns.

Do not underestimate the value of a good dictionary.

The effect is called $(\text{Fim}_i)$ where $k, i, 1 \leq i \leq n$ rule names, $\ldots$, $w, w \text{\star} = v$, $\text{true}$, $\text{false}$.

Then $\text{eff}_i (\text{Fim}, \ldots, \text{Fim}, \text{Fim}, \ldots, \text{Fim}) := (\text{Fim}) := a, \ldots, a, \text{Fim}$ over $A \cup C$.

$s.t.$ Actions $A$ and Conditions $C$ are a set of

$\{\text{rule} \text{ conditions}\} \rightarrow \{\text{actions}\}$
Corresponding valuation: button not pressed, ventilation on, we (only) plan to stop the ventilation.

Assume (iii): \( r = |\sigma T \rightarrow \text{stop}, \text{false} \rightarrow \text{true} \rightarrow \text{on} \). 

Is our intention (to stop the ventilation now) allowed by \( \exists \vdash \text{true} \rightarrow \text{stop} \) ?

• Assume (ii): button pressed, ventilation on, we (only) plan to stop the ventilation.

Assume \( b = \{ \text{true} = \text{stop}, \text{false} = \text{on}, \text{false} = \text{go} \} \).

Is our intention (to start the ventilation now) allowed by \( \exists \vdash \text{true} = \text{start} \) ?

• Assume (i): button pressed, ventilation off, we (only) plan to start the ventilation.

Example:

Example:

Decision Tables as Requirements Specification

Decision Table Semantics: Example
Definition. A decision table \( T \) is called complete if and only if the disjunction of all rules' premises is a tautology, i.e. if \( \bigvee_{r \in T} \text{pre}(r) \).
Useless rules do not hurt as such.

Useless rules "do not hurt" as such.

If decision table has an 'else' rule, then...

For Convenience: The 'else' Rule

\[
\begin{array}{c|c|c|c|c}
\text{button pressed?} & \text{ventilation on?} & \text{ventilation off?} & \text{start ventilation} & \text{stop ventilation} \\
\hline
\text{true} & \text{true} & \text{false} & \text{on} & \text{off} \\
\text{true} & \text{false} & \text{true} & \text{off} & \text{on} \\
\text{false} & \text{true} & \text{false} & \text{off} & \text{on} \\
\text{false} & \text{false} & \text{true} & \text{on} & \text{off} \\
\end{array}
\]
**Definition.**

A decision table \( T \) is called **deterministic** if and only if the premises of all rules are pairwise disjoint, i.e. if

\[
\forall r_1 \neq r_2 \in T \implies \neg (F_{\text{pre}}(r_1) \land F_{\text{pre}}(r_2)).
\]

Otherwise, \( T \) is called **non-deterministic**.

And again: determinism is decidable; reduces to SAT.

**Determinism: Example**

\[
T: \begin{array}{ccc}
\text{r} & \text{b} & \text{on} \\
1 & \times & \times \\
2 & \times & * \\
3 & - & * \\
\end{array}
\]

Is \( T \) deterministic? Yes.

**Determinism: Another Example**

\[
T_{\text{abstr}}: \begin{array}{ccc}
\text{r} & \text{b} & \text{on} \\
1 & \times & \times \\
2 & \times & - \\
3 & - & - \\
\end{array}
\]

Is \( T_{\text{abstr}} \) deterministic? No.

By the way. . .

• Is non-determinism a bad thing in general? Just the opposite: non-determinism is a very, very powerful modelling tool.

Read table \( T_{\text{abstr}} \) as:

• the button may switch the ventilation on under certain conditions (which I will specify later), and
• the button may switch the ventilation off under certain conditions (which I will specify later).

We in particular state that we do not (under any condition) want to see on and off executed together, and that we do not (under any condition) see go or stop without button pressed.

• On the other hand: non-determinism may not be intended by the customer.

**Domain Modelling**

**Example:**

\[
T: \begin{array}{ccc}
\text{r} & \text{b} & \text{on} \\
1 & \times & \times \\
2 & \times & * \\
3 & - & * \\
\end{array}
\]

If on and off model opposite output values of one and the same sensor for "room ventilation on/off" , then \( \sigma | = \text{on} \land \text{off} \) and \( \sigma | = \neg \text{on} \land \neg \text{off} \) never happen in reality for any observation \( \sigma \).

• Decision table \( T \) is incomplete for exactly these cases. (\( T \) "does not know" that on and off can be opposites in the real-world).

• We should be able to "tell" \( T \) that on and off are opposites (if they are). Then \( T \) would be relative complete (relative to the domain knowledge that on/off are opposites).

**Bottom-line:**

• Conditions and actions are abstract entities without inherent connection to the real world.

• When modelling real-world aspects by conditions and actions, we may also want to represent relations between actions/conditions in the real-world (Bjørner, 2006).
Collecting Semantics

\( \text{ϕ} \) wrt. Vacuity

\[ \text{Conflict Relation} \]

\[ \text{Vacuous Rules,} \quad \text{off} \land \neg \text{on} \left( \lor \text{off} \land \neg \right) \]

\[ \text{Conflict Axiom,} \quad \text{ventilation on?} \quad \text{ventilation off?} \]

\[ \text{Stop ventilation} \quad \text{button pressed?} \]

\[ \text{r}_1 : \text{room ventilation} \quad \text{T} \]

\[ \text{is complete wrt. its conflict axiom.} \]

\[ \text{is possible in reality} \quad \text{off} \quad \text{on} \]

\[ \text{are outputs of} \quad \text{off} \quad \text{on} : \text{if} \quad \text{on} \]

\[ \text{is called} \quad \text{in our understanding} \quad \text{conflicts} \]

\[ \text{sets} \quad \text{pre} \quad \text{ϕ} \quad \text{satisfiable} \]

\[ \text{false} = \text{ϕ} : \text{if conditions} \]

\[ \text{fits intuition} \quad \text{anti-crosswind manoeuvre puts} \]

\[ \text{enables} \quad \text{thrust-reverse} \quad \text{at least 6.3 tons weight on each landing gear strut} \]

\[ \text{wheels turning faster than 133 km/h} \quad \text{enable spoilers} \]

\[ \text{enables} \quad \text{thrust-reverse requested} \]

\[ \text{spoilers requested} \]

\[ \text{thrq} \quad \text{splq} \]

\[ \text{too little weight} \]

\[ \text{fatal consequences} \]

\[ \text{Enabling one of those while in the air,} \quad \text{can have} \]

\[ \text{procedures:} \]

\[ \text{Stop a plane} \quad \text{after touchdown,} \quad \text{there are} \]

\[ \text{pitfalls in domain modelling} \]

\[ \text{Example} \quad \text{for requirements analysis} \]

\[ \text{Conflicts in domain modelling} \]
Definition.\[\text{Conflict Relation}\]
A conflict relation \(\leq\) on actions \(A\) is a transitive and symmetric relation \(\leq \subseteq (A \times A)\).

Definition.\[\text{Consistency}\]
Let \(r\) be a rule of decision table \(T\) over \(C\) and \(A\).
(i) Rule \(r\) is called consistent with conflict relation \(\leq\) if and only if there are no conflicting actions in its effect, i.e. if \(\emptyset = \text{eff}(r) \implies \bigwedge (a_1, a_2) \in \leq \neg(a_1 \land a_2)\).
(ii) \(T\) is called consistent with \(\leq\) iff all rules \(r \in T\) are consistent with \(\leq\).

Again: consistency is decidable; reduces to SAT.

Example: Conflicting Actions
\[
\begin{array}{ccc}
\text{r} & \text{b} & \text{c} \\
1 & \text{on} & \text{ventilation} \quad \text{go} \\
2 & \text{on} & \text{ventilation} \quad \text{stop} \\
3 & \text{off} & \text{ventilation} \quad \text{stop} \\
\end{array}
\]

Let \(\leq\) be the transitive, symmetric closure of \(\{(\text{stop}, \text{go})\}\).
"actions stop and go are not supposed to be executed at the same time".

Then rule \(r_1\) is inconsistent with \(\leq\).

A decision table with inconsistent rules may do harm in operation!
Detecting an inconsistency only late during a project can incur significant cost!

Inconsistencies — in particular in (multiple) decision tables, created and edited by multiple people, as well as in requirements in general — are not always as obvious as in the toy examples given here! (would be too easy...)
And is even less obvious with the collecting semantics (in a minute).
Let $T$ be a decision table over $C$ and $A$ and $\sigma$ be a model of an observation of $C$ and $A$.

Then $F_{coll}(T) := \bigwedge a \in A \ (a) \leftrightarrow \bigvee r \in T, r(a) = \times F_{pre}(r)$ is called the collecting semantics of $T$.

We say, $\sigma$ is allowed by $T$ in the collecting semantics if and only if $\sigma|_T = F_{coll}(T)$. That is, if exactly all actions of all enabled rules are planned/executed.

Example:

$T$: room ventilation

\begin{tabular}{c|c|c|c}
  & button pressed? & ventilation off? & ventilation on? \\
  $r_1$ & $\times$ & $\times$ & $\times$ \\
  $r_2$ & $\times$ & $\times$ & $\times$ \\
  $r_3$ & $\times$ & $\times$ & $\times$ \\
  $r_4$ & $\times$ & $\times$ & $\times$
\end{tabular}

- Whenever the button is pressed, let it blink (in addition to go/stop action.

Consistency in the Collecting Semantics

Definition. [$Consistency in the Collecting Semantics$]

Decision table $T$ is called consistent with conflict relation $\triangleright$ in the collecting semantics (under conflict axiom $\phi_{confl}$) if and only if there are no conflicting actions in the effect of jointly enabled transitions, i.e. if $|_T = F_{coll}(T) \land \phi_{confl} \rightarrow \bigwedge (a_1, a_2) \in \triangleright \neg (a_1 \land a_2)$.

Discussion

- Decision Tables: one example for a formal requirements specification language with
  - formal syntax,
  - formal semantics.
- Requirements analysts can use DTs to
  - formally (objectively, precisely) describe their understanding of requirements.
  - Customers may need translations/explanation!
- DT properties like
  - (relative) completeness, determinism,
  - uselessness,
  - can be used to analyse requirements.
  The discussed DT properties are decidable, there can be automatic analysis tools.
- Domain modelling formalises assumptions on the context of software; for DTs:
  - conflict axioms, conflict relation,
  Note: wrong assumptions can have serious consequences.


