Topic Area Requirements Engineering: Content

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
Live Sequence Charts
Syntax, Semantics
Working Definition: Software
Discussion

Structure of Topic Areas

Example: Requirements Engineering

Vocabulary

Informal
Semi-formal
Formal

In the course:

e.g. "Whenever a crash..."
"Always, if crash occurs at..."
"/c01 t, t/c01/c02 Time..."

Use Cases

Pattern Language

Decision Tables

Live Sequence Charts

User Stories
Use Cases
Use Case Diagrams
Sequence Diagrams

A Brief History

Live Sequence Charts

Syntax:

Towards Semantics:

Cuts
Firedsets
User Stories: Discussion

Natural language requirements can be (tried to be) written as an instance of

User Stories (here: Live Sequence Charts)

In the following, we will discuss two-and-a-half notations (in increasing formality):

- Sequence Diagrams
- Use Cases
- Extreme Programming (part of OOSE)

The idea of scenarios (positive, negative, anti-scenarios or what is also sometimes without scenarios) (iv) Get 50 cent change.
(ii) Press the 'softdrink' button.
(i) Insert one 1 euro coin.

Positive scenario

• Get 50 cent change.
• Get a softdrink.

Negative scenario

• Get 50 cent change.
• Get a softdrink.

Anti-scenario

• Get 50 cent change.
• Get a softdrink.

Example: Vending Machine
Use Case: Definition

A use case is a sequence of interactions between an actor (or actors) and a system triggered by a specific actor, which produces a result for an actor.

More precisely:

- A use case has participants: the system and at least one actor.
- Actor: an actor represents what interacts with the system.
- An actor is a role, which a user or an external system may assume when interacting with the system under design.
- Actors are not part of the system, thus they are not described in detail.
- Actions of actors are non-deterministic (possibly constrained by domain model).
- A use case is triggered by a stimulus as input by the main actor.
- A use case is goal oriented, i.e., the main actor wants to reach a particular goal.
- A use case describes all interactions between the system and the participating actors that are needed to achieve the goal (or fail to achieve the goal for reasons).
- A use case ends when the desired goal is achieved, or when it is clear that the desired goal cannot be achieved.

Use Case Example: ATM Authentication

- **Authentication**
- **Goal:** the client wants access to the ATM
- **Pre-condition:** the ATM is operational, the welcome screen is displayed, card and PIN of client are available
- **Post-condition:** client accepted, services of ATM are offered
- **Post-cond. in exceptional case:** access denied, card returned or withheld, welcome screen displayed
- **Actors:** client (main actor), bank system
- **Open questions:** none
- **Normal case:**
  1. client inserts card
  2. ATM reads card, sends data to bank system
  3. bank system checks validity
  4. ATM shows PIN screen
  5. client enters PIN
  6. ATM reads PIN, sends to bank system
  7. bank system checks PIN
  8. ATM accepts and shows main menu
- **Exceptional cases:**
  - **2a:** card not readable
    1.1 ATM displays "card not readable"
    1.2 ATM returns card
    1.3 ATM shows welcome screen
  - **2b:** card readable, but not ATM card
  - **2c:** no connection to bank system
  - **3a:** card not valid or disabled
  - **5a:** client cancels
  - **5b:** client doesn't react within 5 s
  - **6a:** no connection to bank system
  - **7a:** first or second PIN wrong
  - **7b:** third PIN wrong

Use Case Diagrams

Use Case Diagrams: Basic Building Blocks

```plaintext
⟨actor name⟩ ⟨use case name⟩
```

or:

```plaintext
⟨use case name⟩
```
A Brief History of Sequence Diagrams

Message Sequence Charts, ITU standardized in different versions (ITU Z.120, 1st edition: 1993); often accused of lacking a formal semantics.

Sequence Diagrams of UML 1.x (one of three main authors: I. Jacobson)

SDs of UML 2.x address some issues, yet the standard exhibits unclarities and even contradictions (Harel and Maoz, 2007; Störrle, 2003).

For the lecture, we consider Live Sequence Charts (LSCs) (Damm and Harel, 2001; Klose, 2003; Harel and Marelly, 2003), LSCs have a common fragment with UML 2.x SDs: (Harel and Maoz, 2007).

Live Sequence Charts: Syntax (Body)

LSC Body Building Blocks
Definition.

Let $E$ be a set of events and $C$ a set of atomic propositions, $E \cap C = \emptyset$.

An LSC body over $E$ and $C$ is a tuple $(L, \preceq, \sim, I, \text{Msg}, \text{Cond}, \text{LocInv}, \Theta)$ where

- $L$ is a finite, non-empty set of locations
- $\preceq \subseteq L \times L$ is a partial order
- $\sim \subseteq L \times L$ is a symmetric simultaneity relation disjoint with $\preceq$, i.e. $\preceq \cap \sim = \emptyset$
- $I = \{I_1, \ldots, I_n\}$ is a partitioning of $L$; elements of $I$ are called instance line
- $\text{Msg} \subseteq L \times E \times L$ is a set of messages with $(l, E, l') \in \text{Msg}$ only if $(l, l') \in \preceq \cup \sim$; message $(l, E, l')$ is called instantaneous iff $l \sim l'$ and asynchronous otherwise
- $\text{Cond} \subseteq (2^L \setminus \emptyset) \times \Phi(C)$ is a set of conditions with $(L, \phi) \in \text{Cond}$ only if $l \sim l'$ for all $l \neq l' \in L$
- $\text{LocInv} \subseteq L \times \{\circlearrowleft, \circlearrowright\} \times \Phi(C) \times L \times \{\circlearrowleft, \circlearrowright\}$ is a set of local invariants with $(l, \iota, \phi, l', \iota') \in \text{LocInv}$ only if $l \prec l'$, $\circlearrowleft$: exclusive, $\circlearrowright$: inclusive
- $\Theta : L \cup \text{Msg} \cup \text{Cond} \cup \text{LocInv} \to \{\text{hot}, \text{cold}\}$ assigns to each location and each element a temperature.
Towards Semantics:

Syntax:

content

LSC Body: Abstract Syntax

From Concrete to Abstract Syntax
Definition.

Let \((L, \preceq, \sim)\), \(I\), \(\text{Msg}\), \(\text{Cond}\), \(\text{LocInv}\), \(\Theta\) be an LSC body. A non-empty set \(\emptyset \neq C \subseteq L\) is called a cut of the LSC body iff \(C\) is downward closed, i.e.
\[
\forall l, l' \in L \quad l' \in C \land l \preceq l' \Rightarrow l \in C,
\]
\(C\) is closed under simultaneity, i.e.
\[
\forall l, l' \in L \quad l' \in C \land l \sim l' \Rightarrow l \in C,
\]
and \(C\) comprises at least one location per instance line, i.e.
\[
\forall I \in I \quad C \cap I \neq \emptyset.
\]

The temperature function is extended to cuts as follows:
\[
\Theta(C) = \begin{cases} 
\text{hot} & \text{if } \exists l \in C \land (\forall l' \in C \quad l \prec l') \land \Theta(l) = \text{hot}, \\
\text{cold} & \text{otherwise}.
\end{cases}
\]
that is, \(C\) is hot if and only if at least one of its maximal elements is hot.
A Successor Relation on Cuts

The partial order "\(\preceq\)" and the simultaneity relation "\(\sim\)" of locations induce a direct successor relation on cuts of an LSC body as follows:

**Definition.** Let \(C \subseteq L\) be a cut of an LSC body \((L, \preceq, \sim)\). A set \(F \subseteq L\) of locations is called a **fired-set** of cut \(C\) if and only if

- \(C \cap F = \emptyset\) and \(C \cup F\) is a cut, i.e. \(F\) is closed under simultaneity,
- all locations in \(F\) are direct \(\preceq\)-successors of the front of \(C\), i.e. \(\forall l \in F \exists l' \in C \cdot l' \preceq l \land (\nexists l'' \in C \cdot l' \preceq l'' \preceq l)\),
- locations in \(F\), that lie on the same instance line, are pairwise unordered, i.e. \(\forall l \neq l' \in F \cdot (\exists I \in I \cdot \{l, l'\} \subseteq I) \Rightarrow l \not\preceq l' \land l' \not\preceq l,
- for each asynchronous message reception in \(F\), the corresponding sending is already in \(C\), i.e. \(\forall (l, E, l') \in \text{Msg} \cdot l' \in F \Rightarrow l \in C\).

The cut \(C' = C \cup F\) is called the **direct successor of** \(C\) via \(F\), denoted by \(C \Rightarrow F\).
Recall: The TBA (L) of LSC L is (C, Q, q ini, →, Q F) with
• Q is the set of cuts of L,
• q ini is the instance heads cut,
• C B = C ˙∪ E
?!,
• → consists of loops, progress transitions (from ⇝ F), and legal exits (cold cond./local inv.),
• F = {C ∈ Q | Θ(C) = cold ∨ C = L} is the set of cold cuts.

So in the following, we “only” need to construct the transitions’ labels:

→ = {(q, ψ loop(q), q) | q ∈ Q} ∪ {(q, ψ prog(q,q′), q′) | q ⇝ F q′} ∪ {(q, ψ exit(q), L) | q ∈ Q}

ψ loop(q): “what allows us to stay at cut q”
ψ prog(q,q′): “characterisation of firedset Fn”
ψ exit(q): “what allows us to legally exit true


