• LSCs: Automaton Construction
• Excursion: Symbolic Büchi Automata
• LSCs vs. Software
• Methodology
• Requirements Engineering with scenarios
• Strengthening scenarios into requirements
• Requirements Engineering Wrap-Up

Topic Area: Architecture & Design
• Vocabulary
  • (software) system, component, module, interface
  • design, architecture
• Software Modelling
  • model
  • views & viewpoints, the 4+1 view
  • model-driven software engineering

Excursion: Symbolic Büchi Automata

From Finite Automata to Symbolic Büchi Automata

Definition. A Symbolic Büchi Automaton (TBA) is a tuple $B = (C_B, Q, q_{ini}, \rightarrow, Q_F)$ where

- $C_B$ is a set of atomic propositions,
- $Q$ is a finite set of states,
- $q_{ini} \in Q$ is the initial state,
- $\rightarrow \subseteq Q \times \Phi(C_B) \times Q$ is the finite transition relation. Each transition $(q, \psi, q') \in \rightarrow$ from state $q$ to state $q'$ is labelled with a formula $\psi \in \Phi(C_B)$.
- $Q_F \subseteq Q$ is the set of fair (or accepting) states.

Run of TBA

Definition. Let $B = (C_B, Q, q_{ini}, \rightarrow, Q_F)$ be a TBA and $w = \sigma_1, \sigma_2, \sigma_3, \ldots \in (\Phi(C_B) \rightarrow B)^\omega$ an infinite word, each letter is a valuation of $\Phi(C_B)$. An infinite sequence $\rho = q_0, q_1, q_2, \ldots \in Q^\omega$ of states is called a run of $B$ over $w$ if and only if

- $q_0 = q_{ini}$,
- for each $i \in \mathbb{N}_0$ there is a transition $(q_i, \psi_i, q_{i+1}) \in \rightarrow$ s.t. $\sigma_i | \psi_i = \psi_i$.

Example: $q_1 q_2$ even $(x)$ odd $(x)$
How to Prove that a Software Satisfies an LSC?

Example 2

LSCs vs. Software

The Language of a TBA
How to Prove that a Software Satisfies an LSC?

Software $S$ satisfies existential LSC $L$ if there exists $\pi \in [L]$ such that $L$ accepts $w(\pi)$. Prove $S \not|= L$ by demonstrating $\pi$.

Note: Existential LSCs may hint at test-cases for the acceptance test! (As well as (positive) scenarios in general, like use-cases)

LSC: buy softdrink
$AC: true$
$AM: invariant$
$I: permissive$

User
End. Ma.

LSC: get change
$AC: true$
$AM: invariant$
$I: permissive$

User
End. Ma.

$C_50$
$E_1$
$pSOFT$
$SOFT$
$SOFT$
$chg_{-}$
$C_50$

• Universal LSCs (and negative/anti-scenarios!) in general need an exhaustive analysis! (Because they require that the software never ever exhibits the unwanted behaviour.)

Prove $S \not|= L$ by demonstrating one $\pi$ such that $w(\pi)$ is not accepted by $L$.

Pushing Things Even Further

(Harel and Marelly, 2003)

Requirements Engineering with Scenarios
Wrap-Up

• (as safety precaution, e.g., in lawsuits).

Inconsistencies in the requirements.

Hints at diagrams with text (formal) complement.

If in doubt, •

SW Specification & Software Definition:

• (Formal) inconsistency of, e.g., a decision table to communicate with your developers.

• Syntax, Semantics against a formal design description. Verified •

VL 10...

• Formal representations •

• For, e.g., completeness, consistency analysed can be very

• Use formal notations •

• Natural Language Patterns, objective, testable, very useful. Thus, •

• Scenarios •

• Use Cases •

• Natural Language — use case •

• Desirable Properties •

• Requirements on Requirements Speciﬁcations •

• Analysis Techniques •

• Desired Properties •

• Care for •

• Requirements / constraints. That's in general not their "fault"!

For •

• Customers •

• Requirements / constraints.

That's in general not their "fault"!

• Customers •

• Requirements / constraints.

That's in general not their "fault"!

• Customers •

• Requirements / constraints. That's in general not their "fault"!

• Customers •

• Requirements / constraints. That's in general not their "fault"!

• Customers •

• Requirements / constraints. That's in general not their "fault"!

• Customers •

• Requirements / constraints. That's in general not their "fault"!

• Customers •

• Requirements / constraints. That's in general not their "fault"!

• Customers •

• Requirements / constraints. That's in general not their "fault"!

• Customers •

• Requirements / constraints. That's in general not their "fault"!

• Customers •

• Requirements / constraints. That's in general not their "fault"!

• Customers •

• Requirements / constraints. That's in general not their "fault"!

• Customers •

• Requirements / constraints. That's in general not their "fault"!

• Customers •

• Requirements / constraints. That's in general not their "fault"!

• Customers •

• Requirements / constraints. That's in general not their "fault"!

• Customers •

• Requirements / constraints. That's in general not their "fault"!

• Customers •

• Requirements / constraints. That's in general not their "fault"!

• Customers •

• Requirements / constraints. That's in general not their "fault"!

• Customers •

• Requirements / constraints. That's in general not their "fault"!

• Customers •

• Requirements / constraints. That's in general not their "fault"!

• Customers •

• Requirements / constraints. That's in general not their "fault"!

• Customers •

• Requirements / constraints. That's in general not their "fault"!

• Customers •

• Requirements / constraints. That's in general not their "fault"!

• Customers •

• Requirements / constraints. That's in general not their "fault"!

• Customers •

• Requirements / constraints. That's in general not their "fault"!

• Customers •

• Requirements / constraints. That's in general not their "fault"!
Alphabet:

• M – dispense cash only,
• C – return card only,
• MC – dispense cash and return card.

Customer 1:

S₁ = (M.C⏐⏐⏐C.M⏐⏐⏐MC)ω

Customer 2:

S₂ = (M.C)ω or (C.M)ω

Customer 3:

S₃ = (C.M)ω

Literature Recommendation

(Rupp and die SOPHIST en, 2014)

Topi Area Architecture & Design: Content

• Introduction and Vocabulary
• Software Modelling
  (i) views and viewpoints, the 4+1 view
  (ii) model-driven/-based software engineering
  (iii) Unified Modelling Language (UML)
  (iv) Modelling structure
    a) (simplified) class diagrams
    b) (simplified) object diagrams
    c) (simplified) object constraint logic (OCL)
  (v) Principles of Design
    a) modularity
    b) separation of concerns
    c) information hiding and data encapsulation
    d) abstract data types, object orientation
  (vi) Modelling behaviour
    a) communicating finite automata
    b) Uppaal query language
    c) basic state-machines
    d) an outlook on hierarchical state-machines
• Design Patterns

Survey: Previous Experience

Project Management

Requirements Engineering

Programming

Design Modelling

Software Quality Assurance
Vocabulary

**system** — A collection of components organized to accomplish a specific function or set of functions. (IEEE 1471, 2000)

**software system** — A set of software units and their relations, if they together serve a common purpose. This purpose is in general complex, it usually includes, next to providing one (or more) executable program(s), also the organisation, usage, maintenance, and further development. (Ludewig and Lichter, 2013)

**component** — One of the parts that make up a system. A component may be hardware or software and may be subdivided into other components. (IEEE 610.12, 1990)

**software component** — An architectural entity that (1) encapsulates a subset of the system’s functionality and/or data, (2) restricts access to that subset via an explicitly defined interface, and (3) has explicitly defined dependencies on its required execution context. (Taylor et al., 2010)

**module** — (1) A program unit that is discrete and identifiable with respect to compiling, combining with other units, and loading; for example, the input to, or output from an assembler, compiler, linkage editor, or executive routine. (2) A logically separable part of a program. (IEEE 610.12, 1990)

**interface** — A boundary across which two independent entities meet and interact or communicate with each other. (Bachmann et al., 2002)

**interface (of component)** — The boundary between two communicating components. The interface of a component provides the services of the component to the component’s environment and/or requires services needed by the component from the requirement. (Ludewig and Lichter, 2013)

Even More Vocabulary
Even More Vocabulary

**design**

(1) The process of defining the architecture, components, interfaces, and other characteristics of a system or component.

(2) The result of the process in (1).

IEEE 610.12 (1990)

**architecture**

The fundamental organization of a system embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution.

IEEE 1471 (2000)

**software architecture**

The software architecture of a program or computing system is the structure or structures of the system which comprise software elements, the externally visible properties of those elements, and the relationships among them.

(Bass et al., 2003)

**architectural description**

A model – document, product or other artifact – to communicate and record a system’s architecture. An architectural description conveys a set of views each of which depicts the system by describing domain concerns.

(Ellis et al., 1996)

System

Software System

Component

Software Component

Module

Interface

Component Interface

consists of 1 or more "is a"

is a

may be a

has"

is an

is described by

is the result of

Goals and Relevance of Design

• The structure of something is the set of relations between its parts.

• Something not built from (recognisable) parts is called unstructured.

Design.

(i) structures a system into manageable units (yields software architecture),

(ii) determines the approach for realising the required software,

(iii) provides hierarchical structuring into a manageable number of units at each hierarchy level.

Oversimplified process model "Design":

req. design
design arch.
designer
design
modespec.
impl.
code
programmer
impl.

LSCs:

• Automaton Construction

• Excursion:

• Symbolic Büchi Automata

• LSCs vs. Software

• Methodology

• Requirements Engineering with scenarios

• Strengthening scenarions into requirements

• Requirements Engineering Wrap-Up

Top Topic Area Architecture & Design

• Vocabulary

• (software) system, component, module, interface

• design, architecture

Software Modelling

• model

• views & viewpoints, the 4+1 view

• model-driven software engineering

Three properties are constituent:

(i) the image attribute (Abbildungsmerkmal), i.e. there is an entity (called original) whose image or archetype the model is,

(ii) the reduction attribute (Verkürzungsmerkmal), i.e. only those attributes of the original that are relevant in the modelling context are represented,

(iii) the pragmatic attribute, i.e. the model is built in a specific context for a specific purpose.

Definition. (Folk) A model is an abstract, formal, mathematical representation or description of structure or behaviour of a (software) system.

Definition. (Glinz, 2008, 425) A model is a concrete or mental image (Abbild) of something or a concrete or mental archetype (Vorbild) for something. Three properties are constituent:

(i) the image attribute (Abbildungsmerkmal), i.e. there is an entity (called original) whose image or archetype the model is,

(ii) the reduction attribute (Verkürzungsmerkmal), i.e. only those attributes of the original that are relevant in the modelling context are represented,

(iii) the pragmatic attribute, i.e. the model is built in a specific context for a specific purpose.
1. Requirements
   • Shall fit on given piece of land.
   • Each room shall have a door.
   • Furniture shall fit into living room.
   • Bathroom shall have a window.
   • Cost shall be in budget.

2. Design model
   [Diagram of building layout]

3. System
   [Diagram of building system]

Observation (1): Floorplan abstracts from certain system properties, e.g.,
• kind, number, and placement of bricks,
• subsystem details (e.g., window style),
• water pipes/wiring, and
• wall decoration → architects can efficiently work on appropriate level of abstraction

Observation (2): Floorplan preserves/determines certain system properties, e.g.,
• house and room extensions (to scale),
• presence/absence of windows and doors,
• placement of subsystems (such as windows). → find design errors before building the system (e.g. bathroom windows)


