Topic Area Architecture & Design: Content

• Introduction and Vocabulary

• Software Modelling
  - model; views / viewpoints; 4+1 view

• Modelling structure
  - (simplified) class & object diagrams
  - (simplified) object constraint logic (OCL)

• Principles of Design
  - modularity, separation of concerns
  - information hiding and data encapsulation
  - abstract data types, object orientation

• Design Patterns

• Modelling behaviour
  - communicating finite automata (CFA)
  - Uppaal query language

• CFA vs. Software

• Unified Modelling Language (UML)
  - basic state-machines
  - an outlook on hierarchical state-machines

• Model-driven/-based Software Engineering

Vocabulary

• System, Architecture, Design

• Software Modelling

• views & viewpoints
  - the 4+1 view

• Class Diagrams
  - concrete syntax,
  - abstract syntax,
  - class diagrams at work,
  - semantics: system states.

• Object Diagrams
  - concrete syntax,
  - dangling references,
  - partial vs. complete,
  - object diagrams at work.

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)

class
— 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)

module — A set of operations and data visible from the outside only in so far as explicitly permitted by the programmers.

(Ludewig and Lichter, 2013)

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)

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)

• 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. design

modulespec. impl. code programmer implementation

views & viewpoints

the 4+1 view

Class Diagrams

concrete syntax, abstract syntax,
class diagrams at work,
semantics: system states.

Object Diagrams

concrete syntax,
dangling references,
partial vs. complete,
object diagrams at work.

Vocabulary Cont'd

Goals and Relevance of Design

Content
Example: Design Models in Construction Engineering

Requirements

(i) Press the 'softdrink' button.
(ii) and press the 'water' button
(iii) and there is water in stock,
(iv) then

we insert 0.50

1

User

I: strict

Requirement

Furniture shall fit into living room.
Shall fit on given piece of land.

To be strictly accepted, the software shall:
- be error-free,
- be maintainable,
- have a clear user interface,
- perform its stated functions reliably.
No sharp boundaries! (would be too easy...)

should (or should not) be computed

"or to set constraints on behavior in preparation for verification."

assertive (or reflective •).

things are computed

how → "in executing the model or in translating it into executable code.

constructs [of description] contain information needed :

constructive descriptions of behaviour:

1997, Harel.

The (possibly partial) function 

is called 

The (possibly partial) function 

0 0 0 0 0 0 0

π

•

is a finite description

of a (possibly in-

finite) set

of (finite or infinite)

computation paths

of the architecture. " ?!

part

purpose of architecture:

later) — so no need for (extensive) particular documentation.

→ employed framework (For, e.g., a simple smartphone app

which function is used when? Event triggered, time triggered, continuous, etc.?

•

•

large number of electronic control units (ECUs) spread all over the car,

how and when are components instantiated and how do they work together at runtime.

•

how is the system under development integrated into (or seen by) its

developer •

etc., e.g., needs to know which component is running on which host,

team leader •

etc.

•

example models

•

•

process view):

communication •

•

systems (including users) does it

interact; with which other components?

dynamic view:

Scenarios

structure vs. behaviour / constructive vs. reflective

viewpoints

viewpoint

process view:

~

(system view)

views and viewpoints

process, performance, scalability

IEEE 1471 (2000)

topology, communication •

•

systems •

1995, Kruchten •

— a representation of a whole system from the perspective of a related set of

concerns.

— A specification of the conventions for constructing and using a view. A pat-

tern or template from which to develop individual views by establishing the purposes

and audience for a view and the techniques for its creation and analysis.

— A representation of the structure or topology of the system, focusing on the

relationships between components and the flow of information or control.

— A model that captures the behavior and functionality of the system, including

events, states, and actions.
Introduction and Vocabulary

- Software Modelling
  - model; views / viewpoints; 4+1 view
  - Modelling structure
    - (simplified) class & object diagrams
    - (simplified) object constraint logic (OCL)
- Principles of Design
  - modularity, separation of concerns
  - information hiding and data encapsulation
  - abstract data types, object orientation
- Design Patterns
- Modelling behaviour
  - communicating finite automata (CFA)
  - Uppaal query language
- Unified Modelling Language (UML)
  - basic state-machines
  - an outlook on hierarchical state-machines
- Model-driven/-based Software Engineering

Concrete Syntax: Example

Definition.
An (Object System) Signature is a 6-tuple $S = (T, C, V, \text{atr}, F, \text{mth})$ where

- $T$ is a set of (basic) types,
- $C$ is a finite set of classes,
- $V$ is a finite set of typed attributes $v : T$, i.e., each $v \in V$ has type $T$,
- $\text{atr} : C \rightarrow 2^V$ maps each class to its set of attributes,
- $F$ is a finite set of typed behavioural features $f : T_1, \ldots, T_n \rightarrow T$,
- $\text{mth} : C \rightarrow 2^F$ maps each class to its set of behavioural features.

A type can be a basic type $\tau \in T$, or $C_{0, 1}$, or $C^*$, where $C \in C$.

Note: Inspired by OCL 2.0 standard OMG (2006), Annex A.