

System Design

©Andrey Kruglyak, 2010

Friday, May 21, 2010

1

Design of Software Systems

- What is system design?

Friday, May 21, 2010

2

Design of Software Systems

- What is system design?
- Why are we developing different approaches to system design, different methodologies, etc.?

Friday, May 21, 2010

3

Design of Software Systems

- What is system design?
- Why are we developing different approaches to system design, different methodologies, etc.?
- Because we want to shorten development times, achieve certain properties of the system by a certain design process, but first of all - to decrease complexity of a task by dividing it into subtasks

Friday, May 21, 2010

4

Design of Software Systems

- What is system design?
- Why are we developing different approaches to system design, different methodologies, etc.?
- Because we want to shorten development times, achieve certain properties of the system by a certain design process, but first of all - to decrease complexity of a task by dividing it into subtasks
- So what are the desirable system properties?

Friday, May 21, 2010

5

Desirable system properties

- ◁ Correctness (and verifiability) - exhibiting desired behavior
- ◁ Reliability - the ability to perform required functions under stated conditions for a specified period of time
- ◁ Robustness - the ability to cope with errors during execution or to continue to operate despite abnormalities in input
- ◁ Maintainability - the ease with which a software product can be modified to correct defects, meet new requirements, etc.
- ◁ Modularity (needed for division of labor)
- ◁ Quality of Service (QoS)

Friday, May 21, 2010

6

What constitutes a 'methodology'?

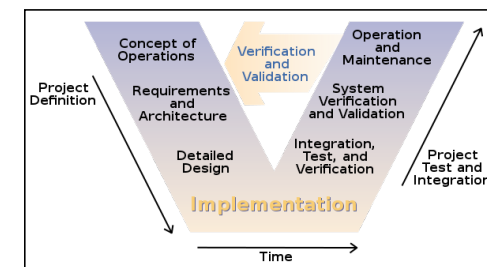
- ◁ A certain staging of work (structuring)
- ◁ A certain organization of the development team
- ◁ A set of engineering best practices
- ◁ Optionally: a set of supporting tools

Friday, May 21, 2010

7

Examples of methodologies

- ◁ V-model



Friday, May 21, 2010

8

Model-based design

- ◀ We will focus on one group of approaches to software design – so-called model-based design, which is based on first creating a model (or multiple models) and then implementing it
- ◀ What is a model – and when is it advantageous to use one?

Friday, May 21, 2010

13

Model

- ◀ A model is a (formal) representation of a system, some aspects of its behavior, or an underlying algorithm
- ◀ A model is an abstraction: some kind of information is preserved and the rest is hidden (e.g. functional model, structural model, model of timing behavior, use cases)
- ◀ In the best of worlds, properties of the model are preserved by the implementation, which should allow us to avoid a separate verification of the latter (hence the desire to have so-called executable models)
- ◀ A fundamental problem: how to combine several models into one implementation?

Friday, May 21, 2010

14

Uses of a model

- ◀ Cognition: to understand parts of the system/algorithm/structure
- ◀ Communication: to discuss requirements and solutions, divide work, etc.
- ◀ Specification: to formalize requirements
- ◀ Simulation: to investigate an artifact as a natural phenomenon, i.e. to observe the consequences of design choices

Friday, May 21, 2010

15

Modeling languages

- ◀ Each model has to be built of well-defined and well-understood blocks
- ◀ Thus an alphabet, syntax rules, and semantics need to be defined...
 - ◀ ...preferably formally, but in general people often settle for an informal understanding or a specification in a natural language
 - ◀ but this is never sufficient if we want to perform a formal verification of the model's properties!
- ◀ In short, for each kind of model we need a modeling language

Friday, May 21, 2010

16

Modeling languages

- ◁ Some modeling languages are entirely graphical, others have a text notation, or both
- ◁ But having a graphical notation helps! Graphics gives us two dimensions (+ possibly colors) and relies on easily understood concepts such as "X is inside Y", "X is connected to Y", "X is above/below/to the left of/to the right of Y"

Friday, May 21, 2010

17

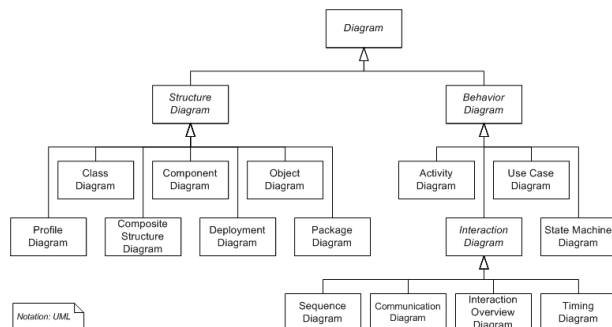
Modeling languages: examples

- ◁ Algebraic modeling languages: high-level programming languages for describing and solving high complexity problems for large scale mathematical computation, e.g. AMPL (any high-level programming language can be viewed as a modeling language)
- ◁ Domain-specific modeling languages: specification languages dedicated to a particular problem domain, e.g. Simulink (tools often come with automatic code generation)
- ◁ Object modeling languages (part of OOD): culminated as UML (industry-standard)

Friday, May 21, 2010

18

Unified Modeling Language



Friday, May 21, 2010

19

Unified Modeling Language

- ◁ Advantages: industry standard, relatively easy to learn
- ◁ Drawbacks: individual diagrams are not well formalized, relation between diagrams is undefined, relation between a model and its implementation is not defined for all diagrams
- ◁ Advantage and drawback at the same time: fragmentation (UML profiles)

Friday, May 21, 2010

20

Interface description languages

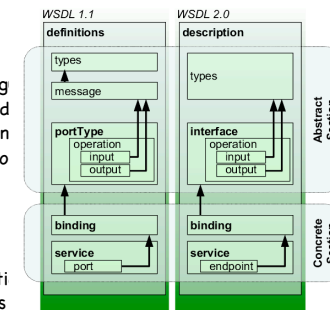
- An IDL is a specification language used to describe a software component's interface. IDLs describe an interface in a language- and machine-independent way, enabling communication between software components that do not share a language (e.g. written in C++ and in Java)
- Examples:
 - Android IDL
 - Microsoft Interface Definition Language
 - Universal Network Objects
 - Web Services Description Language
 - Protocol Buffers

Friday, May 21, 2010

21

Interface description languages

- An IDL is a specification language used to describe a software component's interface. IDLs describe an interface in a language- and machine-independent way, enabling communication between software components that do not share a language (e.g. written in C++ and in Java)
- Examples:
 - Android IDL
 - Microsoft Interface Definition Language
 - Universal Network Objects
 - Web Services Description Language
 - Protocol Buffers



Friday, May 21, 2010

22

Interface description languages

- An IDL is a specification language used to describe a software component's interface. IDLs describe an interface in a language- and machine-independent way, enabling communication between software components that do not share a language (e.g. written in C++ and in Java)
- Examples:
 - Android IDL
 - Microsoft Interface Definition Language
 - Universal Network Objects
 - Web Services Description Language
 - Protocol Buffers

```

message Person {
    required string name = 1;
    required int32 id = 2;
    optional string email = 3;
}

enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
}

message PhoneNumber {
    required string number = 1;
    optional PhoneType type = 2 [default = HOME];
}

repeated PhoneNumber phone = 4;
    
```

Friday, May 21, 2010

23

Contracts

- An attempt to define what a certain method actually does - provide formal, precise and verifiable interface specifications for software components (Design by Contract approach, Bernard Meyer, 1986)
- Consist of formal statements describing acceptable input values or types, return values or types, pre- and post-conditions, invariants, side-effects, error and exception conditions of a method invocation
- This should allow to separate unit testing from integration testing

Friday, May 21, 2010

24

Timber as a modeling language

- As any high-level programming language, Timber can be used as a modeling language for software (and even hardware)
- This is amplified by Timber being a functional language, when we can parametrize objects or components by functions
- Timber allows to abstract from interaction and describe computations - by focusing on functions
- Timber allows to abstract from computations and describe interaction - by focusing on objects and methods

Friday, May 21, 2010

25

Timber as a modeling language

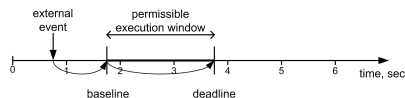
- Timber allows to describe static or dynamic structure of a system (using classes and objects)
- Timber allows to describe concurrent behavior (using objects)
- Perhaps most importantly, Timber allows to specify timing requirements for execution of each asynchronous message, and thus combine timing requirements with a functional model of the system
 - though aggregation and derivation of timing specification is very much work in progress (my research)

Friday, May 21, 2010

26

Timing requirements in Timber

- For each action (including those installed as interrupt handlers) we can specify a baseline offset and/or a deadline (relative to baseline) at the point of invocation, at the point of definition, or both:
Ex1. after (sec 5) before (sec 1) myaction
Ex2. act = after (sec 1) before (sec 2) action {...}



- Note that verification of system's behavior (which requires knowing characteristics of input, execution times, and includes termination analysis) is a separate issue that is not in any way addressed by the language itself

Friday, May 21, 2010

27

Next time:

- Design of distributed systems
- Design of resource-constrained systems
- Design of embedded systems
- Component-based design of general-purpose software systems and of real-time systems

Friday, May 21, 2010

28