**Architecture Documentation**

lbDMF Distributed Multiplatform Framework

created by

Lothar Behrens

*Template Revision: 6.1 EN  
June 2012*

|  |  |
| --- | --- |
| We acknowledge that this document uses material from the arc 42 architecture  template, <http://www.arc42.de>. Created by Dr. Peter Hruschka & Dr. Gernot Starke. |  |

**Revision History**

| **Version** | **Date** | **Reviser** | **Description** |
| --- | --- | --- | --- |
| 0.1 | 30.08.2013 |  | Initial |
| 0.2 | 31.08.2013 |  | Added coarse architecture diagram |
|  |  |  |  |

**Related documents**

|  |  |
| --- | --- |
| **Document** | **Description** |
|  |  |
|  |  |

**Table of Contents**

1. Introduction and Goals 5

1.1 Requirements Overview 5

1.2 Quality Goals 6

1.3 Stakeholders 6

2. Architecture Constraints 6

2.1 Technical Constraints 6

2.2 Organizational Constraints 7

2.3 Conventions 8

3. System Scope and Context 9

3.1 Business Context 9

3.2 Technical Context 10

3.3 External Interfaces 10

4. Solution Strategy 12

5. Building Block View 13

5.1 Level 1 13

5.2 Level 2 13

5.3 Level 3 14

6. Runtime View 15

6.1 Runtime Scenario 1 15

6.2 Runtime Scenario 2 15

6.3 ... 15

6.4 Runtime Scenario n 15

7. Deployment View 15

7.1 Infrastructure Level 1 15

7.2 Infrastructure Level 2 16

8. Concepts 16

8.1 Domain Models 16

8.2 Recurring or Generic Structures and Patterns 16

8.3 Persistency 16

8.4 User Interface 16

8.5 Ergonomics 16

8.6 Flow of Control 16

8.7 Transaction Procession 16

8.8 Session Handling 17

8.9 Security 17

8.10 Safety 17

8.11 Communications and Integration with other Software Systems 17

8.12 Distribution 17

8.13 Plausibility and Validity Checks 17

8.14 Exception/Error Handling 17

8.15 System Management and Administration 17

8.16 Logging, Tracing 17

8.17 Business Rules 17

8.18 Configurability 17

8.19 Parallelization and Threading 17

8.20 Internationalization 17

8.21 Migration 18

8.22 Testability 18

8.23 Scaling, Clustering 18

8.24 High Availability 18

8.25 Code Generation 18

8.26 Build-Management 18

9. Design Decisions 18

9.1 <Decision Topic 1> 18

9.2 <Decision Topic 2> 18

9.3 <Decision Topic 3> 18

... 18

10. Quality Scenarios 19

10.1 Quality Tree 19

10.2 Evaluation Scenarios 19

11. Technical Risks 19

12. Glossary 19

Remark: The Microsoft-Word™ variant of this template contains hidden remarks and suggestions. You can toggle display of this text by the appropriate Word-command.

# Introduction and Goals

The introduction to the architecture documentation should list the driving forces that software architects must consider in their decisions.

This includes on the one hand the fulfillment of functional requirements of the stakeholders, on the other hand the fulfillment of or compliance with required constraints, always in consideration of the architecture goals.

#### Purpose of the System

Software requires a framework to be written with. My first software projects were written with frameworks that are ready made for this purpose. I have migrated my early projects from MFC, a Microsoft Framework to one provided with Power++ by Sybase. But very early in my carrier I were set up with those frameworks from vendors. The cause was simply the end of life for the last framework. Sybase decided to kill Power++.

My first goal therefore is to provide my own framework that does not end. The next goal is to learn how to write a framework and what complications it brings in.

Over time there are added more goals that are listed here:

* The framework should run not only on one platform, such as Windows that I started with.
* It should be modular like COM that is inspiring me.
* No direct dependency to any GUI framework. A GUI can be encapsulated behind a basic API.
* There are more goals, but these are issues within an application that is using this framework. These are things for fast database application prototyping and required techniques. This project contains the application to aid in rapid database prototyping, but it is a candidate for splitting the project.

## Requirements Overview

Short description of the functional requirements.

Digest (or abstract) of the requirements documents.

Reference to complete requirements documents (incl. version identification and location).

Contents

A compact summary of the functional environment of the system. Answers the following questions (at least approximately):

1. What happens in the system’s environment?
2. Why should the system exist? Why is the system valuable or important? Which problems does the system solve?

Motivation

From the point of view of the end users a system is created or modified to improve execution of a business activity.

This essential architecture driver must not be neglected even though the quality of an architecture is mostly judged by its level of fulfillment of non-functional requirements.

Form

Short textual description, probably in tabular use-case format.

The business context should in any case refer to the corresponding requirements documents.

Examples

Short descriptions of the most important:

1. business processes
2. functional requirements
3. non-functional requirements and other constraints (the most important ones must be covered as architecture goals or are listed in the “Constraints” section), and/or
4. quantity structures
5. background information

Here you can reuse parts of the requirements documents – but keep these excerpts short and balance readability against avoidance of redundancy.

The most important functional requirements:

* The framework should provide it’s functionality by interfaces.
* The implementation language is C++, thus interfaces are pure abstract classes. (Optionally this may change to structs if that really helps to overcome the multiple compiler issue)
* Each implementation of an interface provides a functor (a constructor as function).
* A functor is a helper function that must be exported by the library (DLL, so) and it returns an instance of the implementation. The returned interface is a base interface, all implementations have to implement.
* The base interface provides querying for other provided interfaces of that implementation. This is inspired by COM.
* A macro provides functionality to be used to implement the basic functionality of the base interface. Each implementation has to use this macro.
* An exported implementation should be registered for the ability to be found and used.
* Not registered implementations are for internal usage of DLL’s or shared objects.
* A main interface plays the role of a virtual application. It is the interface between a real implementation provided by an API. The real implementation is provided without using an API. Instead a generic marshaling is used.
* The main interface must provide a functionality to load a custom application module. This module implements the application logic that cannot be in the framework.
* The implementing GUI or UI instantiates the virtual application and may load an application on behalf of user login.
* The GUI must provide a basic functionality by predefined marshal interfaces.
* Marshal interfaces can be intercepted, conventional API only by custom interception wrapper that implement the same interface and therefore must be interface APIs.
* The virtual application can automatically do this for the last user and application or by an environment variable pointing to the application module (TARGET\_APPLICATION).
* An application module must provide a functor with a name of instanceOfApplication (case sensitive).

## Quality Goals

Contents

The top three (max five) goals for the architecture and/or constraints whose fulfillment is of highest importance to the major stakeholders.

Goals that define the architecture’s quality could be:

1. availability
2. modifiability
3. performance
4. security
5. testability
6. usability

Motivation

If you as an architect do not know how the quality of your work can be judged …

Form

Simple tabular representation, ordered by priorities

Background Information

NEVER start developing an architecture if these goals have not been put into writing and have not been signed by the major stakeholders.

|  |
| --- |
| We have endured projects lacking defined quality goals much too often. We do not like to suffer, therefore we are by now highly convinced that the few hours spent on collecting quality goals are well invested.  PH & GS. |

The quality level for the framework is for daily usage, but not for usage in any backend, server or services based software. This is due to the learning goal to write a framework. If a software is using the framework in such environments, care should be taken to mitigate these quality issues by fault tolerant solutions, restart, recovery and the like mechanisms.

Daily usage means no crashes per day. Occasionally the software can crash, thus the software should not be used within critical environments. Occasionally crashes are tolerated in prototyping applications.

The quality level should increase were needed by the open source community.

Sources

The DIN/ISO 92000 Standard contains an extensive set of possible quality goals.

Or use chapters 10 – 17 of the VOLERE template as a starting point (www.volere.co.uk).  
PH

## Stakeholders

Me, Lothar Behrens, like to have a tool for software development that decreases time to market.

Open Source Community by participation.

Contents

A list of the most important persons or organizations that are affected by can contribute to the architecture.

Motivation

If you do not know the persons participating in or concerned with the project you may get nasty surprises later in the development process. Should your project manager maintain this list, make sure that all the people influencing the architecture are part of it.

Form

Simple table with role names, person names, their knowledge as pertaining to architecture, their availability, etc.

# Architecture Constraints

Contents

Any requirement that constrains software architects in their freedom of design decisions or the development process.

Motivation

Architects should know exactly where they are free in their design decisions and where they must adhere to constraints.

Constraints must always be dealt with; they may be negotiable, though.

***Form***

Informal lists, structured by the sub-sections of this section.

***Examples***

see sub-sections

Background Information

In the optimal case constraints are defined by requirements. In any case, at least the architects must be aware of constraints.

The influence of constraints is described by [Hofmeister et al] (Software Architecture, A Practical Guide, Addison Wesley 1999) using the term “global analysis”.

## Technical Constraints

Contents

List all technical constraints in this section. This category covers hard- and software infrastructure, applied technologies (operating systems, middleware, databases, programming languages, …).

|  |  |
| --- | --- |
| Hardware-Constraints | |
|  | Common PC Hardware |
| Software-Constraints | |
|  | Sqlite Databases must be supported, ODBC may be supported |
|  | wxWidgets must be supported as GUI frontend |
| Operating System Constraints | |
|  | Windows, Linux and Mac OS X must be supported, Solaris may |
| Programming Constraints | |
|  | Implementing language is C++ |

Examples

|  |  |
| --- | --- |
| Constraint | Description |
| Hardware infrastructure | Processors, memory, networks, firewalls and other relevant elements |
| Software infrastructure | Operating systems, database systems, middleware, communications systems, transaction monitors, web servers, directory services |
| System operations | Batch- or online operations of the system or of required external systems? |
| Availability of the runtime environment | Data center with 7x24 uptime?  Will there be service times that cause reduced availability of the system or important parts thereof? |
| Graphical user interface | Are there any restrictions related to GUI (style guide)? |
| Libraries, frameworks, components | Is there any COTS that must be used? |
| Programming languages | Object oriented, structured, declarative, or rule-based languages?  Compiled or interpreted languages? |
| Reference architectures | Are there any comparable or reusable reference projects in the organization? |
| Analysis and design methodologies | Object oriented or structured methodologies? |
| Data structures | Requirements for certain data structures, interfaces to existing databases or files? |
| Software interfaces | Interfaces to existing applications |
| Programming requirements | Programming guidelines, fixed program structure |
| Technical communications | synchronous or asynchronous; protocols |

## Organizational Constraints

Contents

Enter all organizational, structural, and resource-related constraints. This category also covers standards and legal constraints that you must comply with.

|  |  |
| --- | --- |
| Organization and Structure | |
|  | Private Open Source project |
| Resources (Budget, Time, Personnel) | |
|  | One person project, no time constraints, no budget |
| Organizational Standards | |
|  | Most current language features should be omitted due to multiplatform availability |
| Legal Factors | |
|  | No legal liabilities are accepted from users due to Open Source |

Examples

| Constraint | Description |
| --- | --- |
| Organization and Structure | |
| Sponsor’s organizational structure | Potential changes of responsibilities?  Changes of contact persons? |
| Project team’s organizational structure | with/without subcontractors  decision-making power of the project manager |
| Decision makers | Experience with similar projects |
| Existing partnerships or co-operations | Are there any co-operations between the organizations and certain software companies?  Such partnerships often influence procurement decisions (independent of system requirements). |
| Internal development or outsourcing | Develop internally or outsource to external service companies? |
| Development of a product or for internal use? | Implies different processes in requirements analysis and decision making.  In the case of product development:  New product for a new market?  Improved product for an existing market?  Productizing of an existing (internal) system?  Development for internal use only? |
| Resources (Budget, Time, Personnel) | |
| Fixed price or time/effort? | Is the project’s budget fixed or is it calculated by time or effort? |
| Schedule | Is the schedule flexible? Is there a fixed delivery date? Which stakeholders control the delivery date? |
| Schedule vs. functionality | What has higher priority: The delivery date or the functionality? |
| Release-schedule | At which dates should which functionality be available in which releases / versions? |
| Project’s budget | Fixed or flexible? What amount is available? |
| Budget for technical resources | Buy or rent development tools? (hardware and software) |
| Team | Number of team members, qualifications, motivation, availability. |
| Organizational Standards | |
| Development process | Requirements concerning development process? This includes internal standards for modeling, documentation and implementation. |
| Quality standards | Is the organization required to adhere to quality standards (such as ISO-9000)? |
| Development tools | Requirements related to development tools (such as CASE-Tool, database, IDE, communications software, middleware, transaction monitor). |
| Configuration and version management | Requirements concerning processes and tools |
| Test tools and processes | Requirements concerning processes and tools |
| Acceptance- and release processes | Data modeling and database design  User interfaces  Business processes (workflow)  Usage of external systems (e.g. write access to external databases) |
| Service Level Agreements | Requirements or standards related to availability or required service levels? |
| Legal Factors | |
| Liability | Are there any legal aspects related to usage or operations of the system?  Could the system cause loss of human life or hazard to human health?  Could the system impact the operations of external systems or businesses? |
| Data privacy and security | Does the system store or process any data worthy of protection |
| Auditing | Are any aspects of the system under legal obligation to present evidence? |
| Aspects of international law | Will the system be used in an international context?  Are there varying constraints on system usage in different countries (example: use of encryption technology)? |

## Conventions

Contents

List all conventions that are relevant for the development of your software architecture.

Form

Either insert the conventions directly in this document or refer to other documents.

Examples

1. Coding guidelines
2. Documentation guidelines
3. Guidelines for version and configuration management
4. Naming conventions

All interfaces are prefixed with lb\_I\_\* to clearly indicate it’s origin. All application modules implement a functor with this name: instanceOfApplication. All implementations have to implement lb\_I\_Unknown. It should be avoided to couple the virtual application against the GUI implementation by using interfaces. Instead, marshal interfaces should be preferred. There are interfaces used that should be deprecated in favor of marshal interfaces. They could be intercepted much easier.

Many of the functionalities in lb\_I\_MetaApplication should be moved to lb\_I\_GUI. But still the implementation of lb\_I\_GUI should use marshal interfaces to decouple from the implementation. That is for sample the function to open a database form for SQL queries or custom forms. This was a very old initial design.

# System Scope and Context

The framework view separates itself from the application logic. The application communicates with interfaces and thus never with implementations directly. This goes down to the very basic interfaces like integer and other primitives. An application developer may decide not to use these basic primitives internally, but must do so for interaction with the framework.

Contents

The context view defines the boundaries of the system under development to distinguish it from neighboring systems. It thereby identifies the system’s relevant external interfaces.

Make sure that the interfaces are specified with all their relevant aspects (what is communicated, in which format is it communicated, what is the transport medium, …), even though some popular diagrams (such as the UML use case diagram) represent only a few aspects of the interface.

Motivation

The interfaces to neighboring systems are among most critical and risky aspects of a project. Ensure early on that you have understood them in their entirety.

Form

1. Various context diagram (see below)
2. Lists of neighboring systems and their interfaces.

## Business Context

Contents

Identify all[[1]](#footnote-1) neighboring systems and specify all logical/business data that is exchanged with the system under development. Add data formats and communication protocols with neighboring systems and the general environment if these are not specified in detail with the relevant components.

Motivation

Understanding the information exchange with neighboring systems (i.e. all input flows and all output flows).

Form

Logical context diagram.

In UML this can be simulated e.g. by class diagrams, use case diagrams, communications diagrams – i.e. all diagrams that represent the system as a black box and explain its interfaces to neighboring systems (in varying degrees of detail).

If there are many neighboring systems you can substitute the context diagram with a table, including all the neighboring systems, their inputs and their outputs.

The main application (lbDynApp) in this case provides the logic to enable database application prototyping (not described in detail here). The lbMetaApplication used by the wxWrapper component is a class that loads the configured application. The wxWrapper provides event handlers that are minimally required by lbMetaApplication to load menu, toolbars and other stuff. The GUI is provided by the wxWrapper in form of an interface and the instance is passed to lbMetaApplication. This instance is in turn passed to the loaded application to allow direct usage of basic functionality provided by the GUI. As noted in the conventions, this should be changed.

The dispatcher forwards calls to be dispatched to their event handlers that are registered at runtime from wxWrapper, lbMetaApplication and lbDynamicApplication. The event manager is used to map event names to ID’s and reverse ID’s to names. The overall event handling mechanism is the key to loose coupling at runtime. This allows loading application modules that do not require to define ID ranges for events. The lbModule provides a repository of interfaces and their implementations and is used to instantiate these.

## Technical Context

TBD

Contents

Specification of the communication channels linking your system to neighboring systems and the environment.

Motivation

Understanding of the media used for information exchange with neighboring systems, and the environment.

Form

E.g. UML deployment diagram describing channels to neighboring systems, together with a mapping table of logical input and output flows of the logical context diagram (3.1) to the channels.

## External Interfaces

For many building blocks you can describe its interfaces directly in the black box template of the building block. For complex interfaces – and external interfaces are normally very complex – it is worth while to describe them in separate sections. Use the following interface template to guide you towards many questions that might be relevant for the interface.

#### Interface Id

|  |  |
| --- | --- |
| Name | <name of Interface> |
| Version |  |
| Changes w.r.t previous release |  |
| Who changed it and why? |  |
| Contact person |  |

#### Business Context of the Interface

#### Business Processes

<Diagram or desciption of business processes relevant for this interface>

#### Interface Data

<Description of interface data>

  Technical Context

  Form of interaction

#### Requirements for the Interface

#### Security Aspects

#### Quantities

   Runtime

   Throughput/Volume

  Availability

  Logging

  Archiving

#### Participating Resources

#### Syntax: Data and Formats

   Data Formats

   Validity & Plausibility Rules

   Encoding, Character Sets

   Configuration data

#### Syntax: Methods/Functions

  Check data

#### **Interface Process**

Logical and technical processes

#### Semantics

 Side effects, consequences

#### Technical Infrastructure

 Technical protocols

#### Error and Exception Handling

#### Constraints and Assumptions

 Access Rights

  Temporal constraints

  Parallel Access

  Preconditions for using the interface

#### Operating the Interface

#### Meta Information for the Interface

Person in charge

Costs of using the interface

Organizational Issues

Versioning

#### Examples of Using the Interface

Sample data

Sample flows and interactions

Programming Examples

### External Interface 1

<insert interface template>

### External Interface 2

<insert interface template>

### ...

### External Interface n

<insert interface template>

# Solution Strategy

Contents

A short summary and explanation of the fundamental solution ideas and strategies.

Motivation

An architecture is often based upon some key solution ideas or strategies. These ideas should be familiar to everyone involved into the architecture.

Form

Diagrams and / or text, as appropriate. Keep it short, i.e. 1 or 2 pages at most!

# Building Block View

Contents

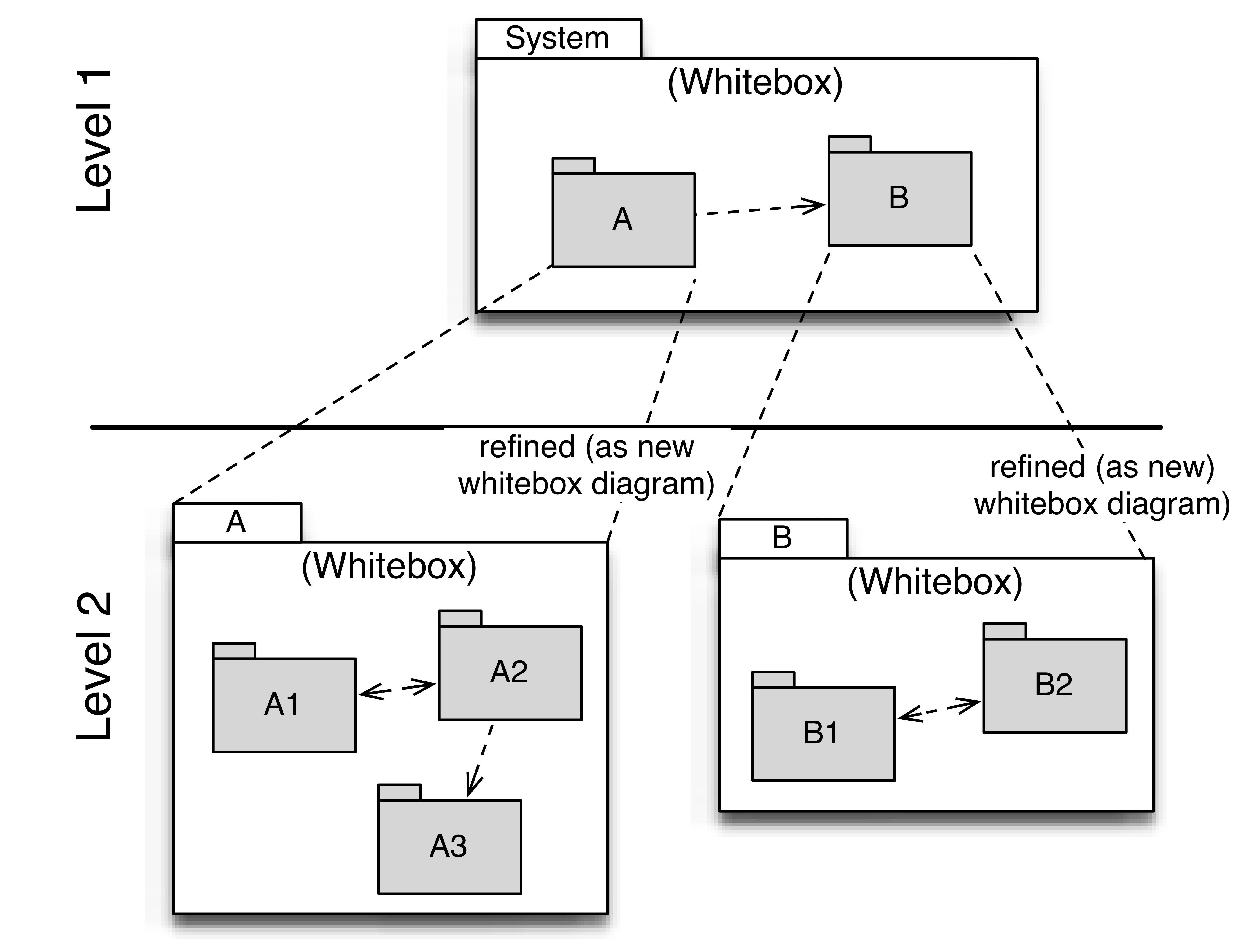
Static decomposition of the system into building blocks (modules, components, subsystems, subsidiary systems, classes, interfaces, packages, libraries, frameworks, layers, partitions, tiers, functions, macros, operations, data structures, …) and the relationships thereof.

Motivation

This is the most important view, that must be part of each architecture documentation. In building construction this would be the floor plan.

Form

The building block view is a hierarchical collection of black box and white box descriptions as shown in the following diagram:



Level 1 contains the white box description of the overall system (system under development / SUD) made up of black box descriptions of the system’s building blocks.

Level 2 zooms into the building blocks of Level 1 and is thus made up of the white box descriptions of all building blocks of Level 1 together with the black box descriptions of the building blocks of Level 2.

Level 3 zooms into the building blocks of Level 2, etc.

The section is structured as follows:

============================

White Box Template:

Contains multiple building blocks with corresponding black box descriptions.

One or more black box templates:

Each building block appearing in the white box template should be described as follows:

1. Purpose / Responsibility:
2. Interface(s):
3. Implemented requirements:
4. Variability:
5. Performance attributes:
6. Repository / Files:
7. Other administrative information: Author, Version, Date, Revision History
8. Open issues:

## Level 1

Here you describe the white box view of level 1 according to the white box template. The structure is given below.

The overview diagram describes the inner structure of the overall system in terms of building blocks 1 – n, as well as their relationships and interdependencies.

It is also useful to list the most important reasons that led to this structure, esp. as relevant to the interdependencies / relationships among the building blocks at this level.

You should also mention rejected alternatives incl. reasons for their rejection.

The following diagram shows the main building blocks of the system and their interdependencies:

<insert overview diagram here>

Comments regarding structure and interdependencies at Level 1:

### Building Block Name 1 (Black Box Description)

Structure according to black box template:

1. Purpose / Responsibility:
2. Interface(s):
3. Implemented requirements:
4. Variability:
5. Performance attributes:
6. Repository / Files:
7. Other administrative information: Author, Version, Date, Revision History
8. Open issues:

<insert the building block’s black box template here>

### Building Block Name 2 (Black Box Description)

<insert the building block’s black box template here>

### ...

<insert the building block’s black box template here>

### Building Block Name n (Black Box Description)

<insert the building block’s black box template here>

### Open Issues

## Level 2

Describe all building blocks comprising level 1 as a series of white box templates. The structure is given below for three building blocks and should be duplicated as needed.

### Building Block Name 1 (White Box Description)

Shows the inner workings of the building block in form of a diagrams with local building blocks 1 – n, as well as their relationships and interdependencies.

It is also useful to list the most important reasons that led to this structure, esp. as relevant to the interdependencies / relationships among the building blocks at this level.

You should also mention rejected alternatives incl. reasons for their rejection.

<insert diagram of building block 1 here>

#### Building Block Name 1.1 (Black Box Description)

Structure according to black box template:

1. Purpose / Responsibility:
2. Interface(s):
3. Implemented requirements:
4. Variability:
5. Performance attributes:
6. Repository / Files:
7. Other administrative information: Author, Version, Date, Revision History
8. Open issues:

#### Building Block Name 1.2 (Black Box Description)

Structure according to black box template

#### ...

#### Building Block Name 1.n (Black Box Description)

Structure according to black box template

#### Description of Relationships

#### Open Issues

### Building Block Name 2 (White Box Description)

…

<insert diagram of building block 2 here>

#### Building Block Name 2.1 (Black Box Description)

Structure according to black box template

#### Building Block Name 2.2 (Black Box Description)

Structure according to black box template

#### ...

#### Building Block Name 2.n (Black Box Description)

Structure according to black box template

#### Description of Relationships

#### Open Issues

### Building Block Name 3 (White Box Description)

…

<insert diagram of building block 3 here>

#### Building Block Name 3.1 (Black Box Description)

Structure according to black box template

#### Building Block Name 3.2 (Black Box Description)

Structure according to black box template

#### ...

#### Building Block Name 3.n (Black Box Description)

Structure according to black box template

#### Description of Relationships

#### Open Issues

## Level 3

Describe all building blocks comprising level 2 as a series of white box templates. The structure is identical to the structure of level 2. Duplicate the corresponding sub-sections as needed.

Simply use this section structure for any additional levels you would like to describe.

# Runtime View

Contents

alternative terms:

1. Dynamic view
2. Process view
3. Workflow view

This view describes the behavior and interaction of the system’s building blocks as runtime elements (processes, tasks, activities, threads, …).

Select interesting runtime scenarios such as:

1. How are the most important use cases executed by the architectural building blocks?
2. Which instances of architectural building blocks are created at runtime and how are they started, controlled, and stopped.
3. How do the system’s components co-operate with external and pre-existing components?
4. How is the system started (covering e.g. required start scripts, dependencies on external systems, databases, communications systems, etc.)?

Note: The main criterion for the choice of possible scenarios (sequences, workflows) is their *architectural relevancy*. It is not important to describe a large number of scenarios. You should rather document a representative selection.

Candidates are:

1. The top 3 – 5 use cases
2. System startup
3. The system’s behavior on its most important external interfaces
4. The system’s behavior in the most important error situations

Motivation

Esp. for object-oriented architectures it is not sufficient to specify the building blocks with their interfaces, but also how instances of building blocks interact during runtime.

Form

Document the chosen scenarios using UML sequence, activity or communications diagrams. Enumerated lists are sometimes feasible.

Using object diagrams you can depict snapshots of existing runtime objects as well as instantiated relationships. The UML allows to distinguish between active and passive objects.

## Runtime Scenario 1

1. Runtime diagram (or other adequate description of scenario!)
2. Description of the notable aspects of the interactions between the building block instances depicted in this diagram.

## Runtime Scenario 2

1. Runtime diagram (or other adequate description of scenario!)
2. Description of the notable aspects of the interactions between the building block instances depicted in this diagram.

## ...

## Runtime Scenario n

1. Runtime diagram (or other adequate description of scenario!)
2. Description of the notable aspects of the interactions between the building block instances depicted in this diagram.

# Deployment View

Contents

This view describes the environment within which the system is executed. It describes the geographic distribution of the system or the structure of the hardware components that execute the software. It documents workstations, processors, network topologies and channels, as well as other elements of the physical system environment. The deployment view shows the system from the operator’s point of view.

Please explain how the systems’ building blocks are aggregated or packaged into deployment artifacts or deployment units.

Motivation

Software is not much use without hardware. The minimum that is needed by you as a software architect is sufficient detail of the underlying (hardware) deployment so that you can assign each software building block that is relevant for the system’s operations to some hardware element. (This also holds for any COTS that is a prerequisite for the operations of the overall system.) These models should enable the operator to properly install the software.

Form

The UML provides deployment diagrams for describing this view. Use these – possibly in a nested manner if necessary. (The top level deployment diagram should already be part of your context view, showing your infrastructure as a single black box (cf. chapter 3.2). Here you are zooming into this black box with additional deployment diagrams.)

Diagrams by your hardware-oriented colleagues who describe processors and channels are also usable. You should abstract these to aspects relevant for software deployment.

## Infrastructure Level 1

### Deployment Diagram Level 1

1. Shows the deployment of the overall system to 1 – n processors or sites as well as the physical connections among these elements.
2. Lists the most important reasons that led to this deployment structure, i.e. the specific selection of nodes and channels.
3. Should also mention rejected alternatives incl. reasons for their rejection.

### Processor 1

< insert node template here>

node template:

* Description
* Deployed software building blocks
* Quality attributes (performance, constraints, ...)
* Other administrative information
* Open issues

### Processor 2

< insert node template here>

### ...

### Processor n

< insert node template here>

### Channel 1

Contents

Specification of the channel’s attributes, as relevant for software architecture.

Motivation

Specify at least those attributes of the communications channels that you need for proving fulfillment of non-functional requirements such as maximal throughput, probability for faults, etc.

Form

Often you will refer to a standard (e.g. CAN-Bus, 10Mbit Ethernet, IEEE 1394, ...).

If you need more information use a structure similar to the node template (especially to document quality aspects of channels like throughput, error rates, ...)

### Channel 2

### ...

### Channel m

## Infrastructure Level 2

Contents

Additional deployment diagrams with similar structure as above, refining each node of infrastructure level 1 that needs more details to map the software blocks.

Motivation

To describe additional details of the infrastructure, as needed by software deployment.

# Concepts

Contents

The following chapters cover examples of frequent cross-cutting concerns (a.k.a. aspects in some programming languages)

Fill in these chapters if there is NO building block that covers this aspect. If some of the concepts are not relevant for your project mention this fact instead of removing the section.

Motivation

Some concept cannot be “factored” into a separate building block of the architecture (e.g. the topic “security”). This section of the template is the location where you can describe all decision for such a cross cutting topic in one central place. Nevertheless, you have to make sure that all your building blocks conform to such decisions.

Form

.. can be varied. Some concepts are plain natural language text with a freely chosen structure, some others may include models/scenarios using notations that are also applied in architecture views.

## Domain Models

Contents

Business (or logical) models, domain models - they all describe subject-specific, business aspects, without relation to technology.

Often these structures or concepts are re-used at many places within an architecture, especially within chapter [5. Building Block View](http://confluence.arc42.org/display/templateEN/5.+Building+Block+View)..

Motivation

Often parts of these models reappear in the building block view, but in order to „keep them clean from technnology“ they can be captured as a concept.

Form

Usually either ER-Models (Entity-Relationship Models) or UML class diagrams (mainly containing entity classes) are used to document domain models.

## Recurring or Generic Structures and Patterns

Motivation

Sometimes a hierarchical decomposition of building blocks is insufficient for giving an overview of detailed interdependencies between individual building blocks. The following sections are intended to describe generic or specific dependencies among any set of building blocks – possibly even across different levels.

We call a dependency *generic* if it appears more than once in the architecture, and *specific* if it is unique.

Form

Use building block models (class diagrams, package diagrams, component diagrams, etc.) and related descriptions in the same way as in the hierarchical decomposition.

Often it is practical to support understandability by adding specific runtime views to explain these recurring structures.

### Recurring or Generic Structure 1

<insert diagram and descriptions here>

### Recurring or Generic Structure 2

<insert diagram and descriptions here>

## Persistency

Persistency means moving data from (volatile) memory to a durable storage medium (and back).

Some of the data that a software system is processing must be written to and read from persistent storage media.

1. Volatile storage media (main memory or cache) are not designed for permanent storage. Data is lost if the hardware is switched off.
2. The amount of data processed by commercial software systems normally exceeds the capacity of main memory.
3. Hard disks, optical media and tapes often contain large amounts of existing business data that represent a significant investment.

Persistency is a technical issue that normally does not appear as part of the actual business functionality. An architect must deal with this issue nevertheless because most software systems require efficient access to persistently stored data. This is relevant for essentially all commercial and most technical systems; embedded systems on the other hand often differ in their data management requirements.

## User Interface

Software systems that are used interactively by (human) users require a user interface. These can be graphical, textual, or voice user interfaces.

## Ergonomics

Ergonomics of software systems deals with the improvement (optimization) of their usability with respect to objective and subjective factors. Key ergonomic factors are user interface, reactivity (subjective performance) as well as availability and robustness of the system.

## Flow of Control

Flow of control for software systems is related to visible flows (on the - graphical - user interface) as well as the flow of background activities. Therefore this section should cover control of the user interface as well as control of workflows.

## Transaction Procession

A transaction is a sets of operations or activities that must be processed either in its entirety or not at all. The term is especially relevant in the database area with the important notion of ACID-transactions (atomic, consistent, isolated, durable).

## Session Handling

A session identifies an active connection between a client and a server. The session state must be preserved, which is esp. important if stateless protocols such as HTTP are used for communications. Session handling is a critical challenge esp. for intra- and internet-systems and can strongly influence the performance of a system.

## Security

The security of software systems deals with mechanisms that ensure data confidentiality, integrity, and availability.

Typical issues are:

1. How can data be protected during transport (e.g. via open networks such as the internet)?
2. How can communicating entities ensure mutual trust?
3. How can communicating entities identify each other and be protected against faked identities?
4. How can communicating entities prove data provenience or certify validity of data?

The topic of IT-security often touches upon legal aspects, sometimes even international law.

## Safety

The safety of software systems deals with mechanisms that ensure that human life or our environment is not endangered.

Describe your concept here: identify those parts of the system that might endanger life and describe mechanism to ensure proper safety.

## Communications and Integration

Communication: Exchange of data between system components. Covers communications within one process or address space, between different processes (inter-process communication – IPC), and between different systems.

Integration: Combination of existing systems in a new context. Also known as: (Legacy) Wrapper, Gateway, Enterprise Application Integration (EAI).

## Distribution

Distribution: Design of software systems whose parts are executed on different – physically separated – hardware systems.

Distribution covers issues such as calling methods on remote systems (remote procedure call – RPC or remote method invocation – RMI), the transfer of data or documents among distributed parties, the choice of optimal modes of interaction or communications patterns (such as synchronous / asynchronous, publish-subscribe, peer-to-peer).

## Plausibility and Validity Checks

How and where do you check plausibility and validity of (input) data, esp. user inputs?

## Exception/Error Handling

How are exceptions and errors handled systematically and consistently?

How can the system reach a consistent state after an error? Is this done automatically or is manual interaction required?

This aspect is also related to logging and tracing,

Which kind of exceptions and errors are handled by the system? Which kinds of errors are forwarded to which external interface and which are handled fully internally?

How do you use the exception handling mechanisms of your programming language? Do you use checked or unchecked exceptions?

## System Management and Administration

Larger software systems are often executed in controlled environments (data centers) under oversight of operators or administrators. These stakeholders require specific information on the applications’ states during runtime as well as special means of control and configuration.

## Logging, Tracing

There are two ways of documenting an application’s status during runtime: *Logging* and *Tracing*. In both cases the application is extended with function or method calls that write state information, but there is a difference in their usage:

1. Logging can cover business or technical aspects or any combination of both.
2. Business logs are normally prepared for end users, administrators or operators. They contain information on the business processes that are executed by the application. This kind of logging may also be related to auditing.
3. Technical logs contain information for operators or developers. These are used for error detection and system optimization.
4. Tracing is intended to provide debugging information for developers or support personnel. It is primarily used for error detection and analysis.

## Business Rules

How do you deal with business logic and business rules? Is business logic implemented in the corresponding business classes or is it handled in a central component? Do you use a rule engine for the interpretation of business rules (production system, forward-/backward-chaining)?

## Configurability

The flexibility of a software system is influenced by its configurability, i.e. the possibility to make certain decisions about usage of the system at a late point in time.

Configurability can occur at the following events:

1. During development: For example server, file, or directory names could be stored directly in the code (“hard-coded”).
2. During deployment or installation: Configuration information for a specific installation (such as the installation path) can be given.
3. At system startup: Information can be read dynamically before or during system startup.
4. During application execution: Configuration information is queried or read during runtime.

## Parallelization and Threading

Applications can be executed in parallel processes or threads. This creates a need for synchronization points. The theory of parallel processing serves as a foundation for this aspect. The architecture and implementation of parallel systems needs to consider many technical details such as address spaces, applied mechanisms for synchronization – guards, semaphores, etc. – processes and threads, parallelism in the operating system, parallelism in virtual machines. etc.

## Internationalization

This section covers support for usage of the system in different countries, i.e. adjusting the system to country specific attributes. Internationalization (often abbreviated as “i18n” where “18” refers to the eighteen characters between the I and the n) covers translation of text, usage of character encodings, display of fonts, writing of numbers and dates, and other (external) aspects.

## Migration

In many cases a new software system is intended to replace an existing legacy system. As an architect you should not only consider your shiny new architecture but also all organizational and technical aspects that must be considered for the introduction or migration of the architecture.

Examples:

1. Concept, process, or tools for data transfer and initial data creation.
2. Concept for system introduction or temporary parallel operations of legacy system and new system.

Is it necessary to migrate existing data? How do you execute any needed syntactic or semantic transformations?

## Testability

Support for simple (and if possible automated) tests. This aspect is the basis for the important implementation pattern of “continuous integration”. Projects should support at least daily build-and-test cycles. Important keywords for this aspect are unit tests and mock objects.

## Scaling, Clustering

How can your system grow in a way that can cope with more load or a larger number of users and still keep up performance and throughput.

## High Availability

How can you achieve high availability of your system? Do you use redundancy of major parts? Or do you distribute your system to different processors or locations. Are you running standby- systems?

## Code Generation

How and where do you use code generators to create parts of the system from models or domain specific languages (DSL’s)?

## Build-Management

How is the overall system created from is (source code) building blocks? Which repositories contain source code, where are configuration files, test cases, test data and build scripts (make, ant, maven) stored?

# Design Decisions

Contents

Document all important design decisions and their reasons!

Motivation

It is advantageous if all important design decisions can be found in one place. It is up to you to decide if a decision should be documented here or rather locally (e.g. in the white box descriptions of building blocks). In any case avoid redundancies.

Form

Informal list, if possible ordered by the decisions’ importance for the reader. Use the following template to structure your decisions.

**9.x.1 Problem**

What exactly is the challenge?

Why is it relevant for the architecture?

What consequences does the decision have?

**9.x.2 Constraints**

Which constraints do you have to keep in mind?

What factors influence the decision?

**9.x.3 Assumptions**

Which assumption have you made?

How can you check those assumptions?

Which risks are you facing?

**9.x.4 Considered Alternatives**

Which alternative options did you consider?

How do you judge each one?

Which alternatives are you excluding deliberately?

**9.x.5 Decision**

Who (if not you) has decided?

How has the decision been justified?

When did you decide?

## <Decision Topic 1>

## <Decision Topic 2>

## <Decision Topic 3>

## ...

# Quality Scenarios

This chapter summarizes all you (or other stakeholders) might need to systematically evaluate the architecture against the quality requirements.

It contains:

* Quality Tree (sometimes called utility tree), an overview of the quality requirements
* Evaluation or quality scenarios - detailed descriptions of the quality requirements or goals.

## Quality Tree

Content

The quality tree ( as defined in ATAM – Architecture Tradeoff Analysis Method) with quality / evaluation scenarios as leafs.

Motivation

When you want to evaluate the quality (especially risks to certain quality attributes) with methods like ATAM, you need to systematically refine your quality goals (from chapter 1.2). The quality tree shows the top-down refinement of the stakeholder-specific notion of quality.

Form

We personally prefer mind maps to a pure tree-like structure, as mind maps allow arbitrary cross-references between scenarios, attributes and intermediate nodes.

Often it is difficult to assign scenarios to single quality attributes, as the scenario refers to several qualities at once. Simply draw references from such scenarios to all affected nodes!

## Evaluation Scenarios

Contents

Scenarios describe a system’s reaction to a stimulus in a certain situation. They thus characterize the interaction between stakeholders and the system. Scenarios operationalize quality criteria and turn them into measurable quantities.

Two scenarios are relevant for most software architects:

1. Usage scenarios (also called application scenarios or use case scenarios) the system’s runtime reaction to a certain stimulus. This also includes scenarios that describe the system’s efficiency or performance. Example: The system reacts to a user’s request within one second.
2. Change scenarios describe a modification of the system or of its immediate environment. Example: Additional functionality is implemented or requirements for a quality attribute change.

If you design safety critical systems a third type of scenarios is important for you:

1. Boundary or stress scenarios describe how the system reacts to exceptional conditions. Examples: How does the system react to a complete power outage, a serious hardware failure, etc.

**Source of**

**the Stimulus**

**System**

**artifact**

**Response measure**

**Stimulus**

**Response**

**Figure: Schematic depiction of scenarios (cf. [Bass+03])**

Scenarios comprise the following major parts (according to [Starke05], original structure from [Bass+03]):

1. Stimulus: Describes a specific interaction between the (stimulating) stakeholder and the system. Example: A user calls a functions, a developer implements an extension, an administrator installs or configures the system.
2. Source of the stimulus: Describes where the stimulus comes from. Examples: internal or external, user, operator, attacker, manager.
3. Environment: Describes the system’s state at the time of arrival of the stimulus. This should list all preconditions that are necessary for comprehension of the scenario. Examples: Is the system under normal or maximal load? Is the data base available or down? Are any users online?
4. System artifact: Describes the part of the system is affected by the stimulus. Examples: The whole system, the data base, the web server.
5. System response: Describes the system’s reaction to the stimulus as determined by the architecture. Examples: Is the function called by the user executed. How long does the developer need for implementation? Which parts of the system are affected by the installation / configuration?
6. Response measure: Describes how the response can be measured or evaluated. Examples: Downtime in hours, correctness yes/no, time for code change in person days, reaction time in seconds.

Motivation

You need scenarios for the evaluation and review of architectures. They take the role of a “benchmark” and aid in measuring the architecture’s achievement of its objectives regarding the non-functional requirements and quality attributes.

Form

Tabular or free text. Explicitly highlight the scenario’s elements (source, environment, artifact, response, measure).

Alternatively, use similar notations as those suggested in [6. Runtime View](http://arc42.org:8090/display/templateEN/6.+Runtime+View).

Background Information

There are relations between scenarios and the runtime view. Often you can use scenarios of the runtime view fully or as a basis for evaluation. Evaluation scenarios additionally contain response measures that are often not considered in the pure execution focus of runtime scenarios.

# Technical Risks

Contents

A list of identified technical risks, ordered by priority

Motivation

“Risk management is project management for grown-ups” (Tim Lister, Atlantic Systems Guild.) This should be your motto for systematic detection and evaluation of technical risks in the architecture, which will be needed by project management as part of the overall risk analysis.

Form

List of risks with probability of occurrence, amount of damage, options for risk avoidance or risk mitigation, …

# Glossary

Contents

The most important terms of the software architecture in alphabetic order.

Motivation

It should not be necessary to explain the usefulness of a glossary …

Form

A simple table with columns <Term> and <Definition>

1. We often tend to a pragmatic approach – but here we insist on a list of all (a-l-l) neighboring systems. Too many projects have failed because they were not aware of their neighbors. ☹ [↑](#footnote-ref-1)