Software Documentation template

Release 0.0.

Raphael Dürscheid, based on Template by Dr. Peter

Jul 01, 2020

USAGE AND INSTALLATION

Todo: Introduce your project and describe what its intended goal and use is.

CHAPTER

ONE

RELEVANT BACKGROUND INFORMATION AND PRE-REQUISITS

Todo: Describe what a potential user needs to be familiar with. What should they read and understand beforehand

Describe what a developer needs to be familiar with to further understand the code.

Link to relevant documents or create a new page and add them there.

CHAPTER TWO

REQUIREMENTS OVERVIEW

The **software requirements** define the system from a blackbox/interfaces perspective. They are split into the following sections:

- **User Interfaces** - *User Interfaces*
- **Technical Interfaces** - *Technical Interfaces*
- **Runtime Interfaces and Constraints** - *Technical Constraints / Runtime Interface Requirements*

CHAPTER

THREE

TODO:

Todo: Create a black box view of your software within its intendend environment. Identify all neighboring systems and specify all logical business data that is exchanged with the system under development.

List of all (a-l-l) neighboring systems.

Motivation.

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

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/roboy-sw-documentation-template/checkouts/lite/documentation/ScopeContext/0_system_scope_and_context.rst, line 4.)

Todo: Define what coding guideline they should use, if you differ from the ones below.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/roboy-sw-documentation-template/checkouts/lite/documentation/ScopeContext/1_conventions.rst, line 5.)

Todo: Describe what constraints someone further developing this software should adhere to, and why. Should they not use tool x or operating system y… You can use the table as below, or just put a list.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/roboy-sw-documentation-template/checkouts/lite/documentation/ScopeContext/2_architecture_constraints.rst, line 4.)

Todo: List all technical constraints in this section. This category covers runtime interface requirements and constraints such as:

- Hard- and software infrastructure
- Applied technologies - Operating systems - Middleware - Databases - Programming languages

---

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/roboy-sw-documentation-template/checkouts/lite/documentation/ScopeContext/2_architecture_constraints.rst, line 12.)

Todo: For all your interfaces, define their first 3 levels of interoperability. You can use your doxygen documented source code to i.e. show all members of a class. Find more on the Breathe Documentation

---

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/roboy-sw-documentation-template/checkouts/lite/documentation/ScopeContext/3_technical_interfaces.rst, line 8.)

Todo: If you have a user interacting with the finished system, and especially if you have a UI or GUI, describe how it can be used. A good way of doing this, is by building a

- state transition diagram aka the 'Dynamic UI Behaviuour' and
- a mockup/picture of every screen the user can see - aka 'Static UI'

Don't overdo it. . .

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/roboy-sw-documentation-template/checkouts/lite/documentation/ScopeContext/4_user_interface.rst, line 7.)

Todo: If you want to use it, keep track of decisions that someone further developing this software or using it in the future might want to know about. This might save them time or clarify expectations. You can put them as a list, or whatever.

Things to consider:

- What exactly is the challenge?
- Why is it relevant for the architecture?
- What consequences does the decision have?
- Which constraints do you have to keep in mind?
- What factors influence the decision?
- Which assumption have you made?
- How can you check those assumptions?
- Which risks are you facing?
- Which alternative options did you consider?
- How do you judge each one?
- Which alternatives are you excluding deliberately?
- Who (if not you) has decided?
- How has the decision been justified?
- When did you decide?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/roboy-sw-documentation-template/checkouts/lite/documentation/ScopeContext/5_design_decisions.rst, line 6.)

Todo: Describe the installation process step by step.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/roboy-sw-documentation-template/checkouts/lite/documentation/Usage/0_installation.rst, line 4.)

Todo: How do you start the software after installation

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/roboy-sw-documentation-template/checkouts/lite/documentation/Usage/1_getting_started.rst, line 4.)

Todo: Describe your 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!

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/roboy-sw-documentation-template/checkouts/lite/documentation/development/00_solution_strategy.rst, line 4.)

Todo: If you have any tests describe:

- Where the tests are kept.
- How the tests are invoked.
- What you can/need to test manually.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/roboy-sw-documentation-template/checkouts/lite/documentation/development/05_test_strategy.rst, line 4.)

Todo: Inset a building block view:

- Static decomposition of the system into building blocks and the relationships thereof.
- Description of libraries and software used
-

We specify the system based on the blackbox view from *UML System Context* by now considering it a whitebox and identifying the next layer of blackboxes inside it. We re-iterate this zoom-in until specific granularity is reached - 2 levels should be enough.

Motivation.

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

Tool

- Create diagrams as below.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/roboy-sw-documentation-template/checkouts/lite/documentation/development/06_building_block_view.rst, line 6.)

Todo:

- Define your groups using the **/defgroup** doxygen command
- Add **@addtogroup** tags to doxygen blocks of components in code as described here: http://www.stack.nl/~dimitri/doxygen/manual/grouping.html#modules

- Adapt the doxygencall to match the group name

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/roboy-sw-documentation-template/checkouts/lite/documentation/development/06_building_block_view.rst, line 44.)

Todo:

- Define your groups using the **/defgroup** doxygen command
- Add **addtogroup** tags to doxygen blocks of components in code as described here: http://www.stack.nl/~dimitri/doxygen/manual/grouping.html#modules
- Adapt the doxygencall to match the group name

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/roboy-sw-documentation-template/checkouts/lite/documentation/development/06_building_block_view.rst, line 55.)

Todo: Add a runtime view of i.e. the startup of your code. This should help people understand how your code operates and where to look if something is not working.

Contents. alternative terms: - Dynamic view - Process view - 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:

- How are the most important use cases executed by the architectural building blocks?
- Which instances of architectural building blocks are created at runtime and how are they started, controlled, and stopped.
- How do the system's components co-operate with external and pre-existing components?
- 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.

  Especially 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.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/roboy-sw-documentation-template/checkouts/lite/documentation/development/07_runtime_view.rst, line 6.)

Todo: Add a deployment diagram. **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.

Motivation.

Software is not much use without hardware. These models should enable the operator to properly install the software.

You can separate this into different levels…

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/roboy-sw-documentation-template/checkouts/lite/documentation/development/08_deployment_view.rst, line 6.)

Todo: List libraries you are using

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/roboy-sw-documentation-template/checkouts/lite/documentation/development/9_Libraries.rst, line 8.)

Todo: Introduce your project and describe what its intended goal and use is.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/roboy-sw-documentation-template/checkouts/lite/documentation/index.rst, line 10.)

Todo: Describe what a potential user needs to be familiar with. What should they read and understand beforehand

Describe what a developer needs to be familiar with to further understand the code.

Link to relevant documents or create a new page and add them there.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/roboy-sw-documentation-template/checkouts/lite/documentation/index.rst, line 19.)

Contents:

3.1 Installation

Todo: Describe the installation process step by step.

3.2 Getting started

Todo: How do you start the software after installation

3.3 Context

Todo: Create a black box view of your software within its intendend environment. Identify all neighboring systems and specify all logical business data that is exchanged with the system under development.

List of all (a-l-l) neighboring systems.

Motivation.

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

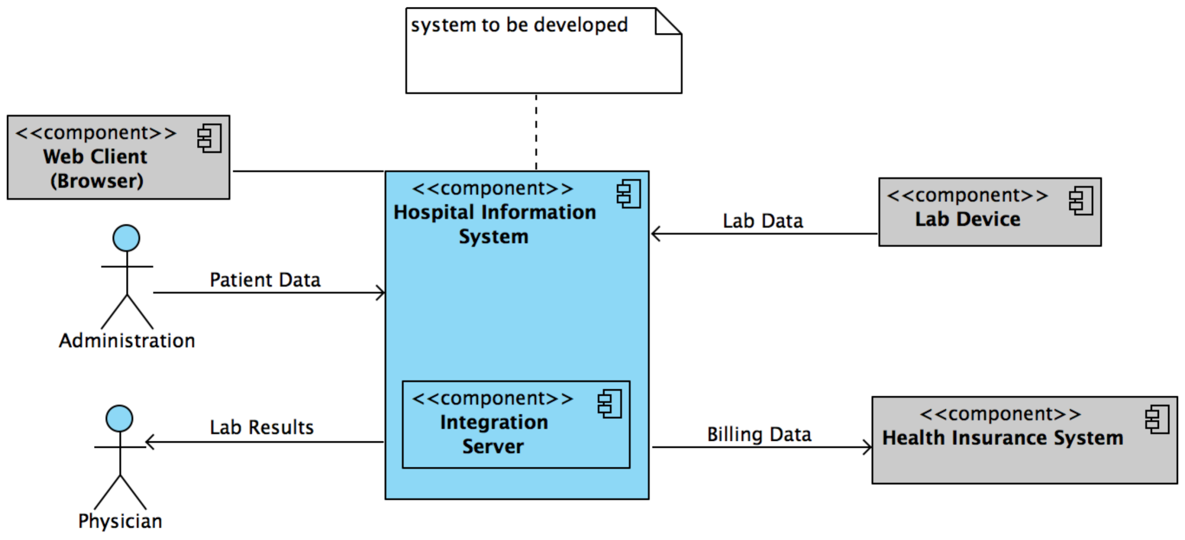


Fig. 1: UML System Context

UML-type context diagram - shows the birds eye view of the system (black box) described by this architecture within the ecosystem it is to be placed in. Shows orbit level interfaces on the user interaction and component scope.

3.4 Conventions

We follow the coding guidelines:

Todo: Define what coding guideline they should use, if you differ from the ones below.

Table 1: Coding Guidelines

Language	Guideline	Tools
Python	https://www.python.org/dev/peps/pep-0008/	
C++	http://wiki.ros.org/CppStyleGuide	clang-format: https://github.com/davetcoleman/roscpp_code_format

3.5 Architecture Constraints

Todo: Describe what constraints someone further developing this software should adhere to, and why. Should they not use tool x or operating system y… You can use the table as below, or just put a list.

3.5.1 Technical Constraints / Runtime Interface Requirements

Todo: List all technical constraints in this section. This category covers runtime interface requirements and constraints such as:

- Hard- and software infrastructure
- Applied technologies - Operating systems - Middleware - Databases - Programming languages

Table 2: Hardware Constraints

Constraint Name	Description
Altera FPGA	All code is highly specific to the Altera FPGA. Intel has bought Altera and aims to integrate their SoC with FPGAs. We are on the right horse!
Intel RealSense	Only the intel real sense can sense it..

Table 3: Software Constraints

Constraint Name	Description
Altera FPGA	All code is highly specific to the Altera FPGA. Intel has bought Altera and aims to integrate their SoC with FPGAs. We are on the right horse!
Intel RealSense	Only the intel real sense can sense it…

Table 4: Operating System Constraints

Constraint Name	Description
Windows 8 or higher	Due to the Intel RealSense SDK only being supported on Windows, we are stuck with Windows

Table 5: Programming Constraints

Constraint Name	Description
CouchDB	We have to use the CouchDB because the type of data we have to store changes at runtime…

3.6 Technical Interfaces

This section describes the data interfaces to other systems around it. It follows 3 of the levels of interoperability (**IO**):

Todo: For all your interfaces, define their first 3 levels of interoperability. You can use your doxygen documented source code to i.e. show all members of a class. Find more on the Breathe Documentation

- Technical interoperability - Datastreams btwn systems. i.e. TCP/IP, RS232, . . .
- Syntactic interoperability - Units within the stream. i.e. XML, CSV, HL7, DICOM
- Semantic interoperability - Common definition of unit meaning.

3.6.1 Powerlink for MotorControl

Technical IO

Powerlink

Syntactic IO

SDO, PDO, NMT messages

Semantic IO

The package names or registers or

`class Nutshell`

Public Types

`enum Tool`

Our tool set.

The various tools we can opt to use to crack this particular nut

Values:

`enumerator kHammer` = 0

What? It does the job.

`enumerator kNutCrackers`

Boring.

`enumerator kNinjaThrowingStars`

Stealthy.

Public Functions

Nutshell()
Nutshell constructor.

~Nutshell()
Nutshell destructor.

void **crack**(*Tool tool*)
Crack that shell with specified tool.

Parameters

- `tool`: - the tool with which to crack the nut

bool **isCracked**()

Return Whether or not the nut is cracked

3.7 User Interfaces

How does a user interact with the system.

Todo: If you have a user interacting with the finished system, and especially if you have a UI or GUI, describe how it can be used. A good way of doing this, is by building a

- state transition diagram aka the 'Dynamic UI Behaviuour' and
- a mockup/picture of every screen the user can see - aka 'Static UI'

Don't overdo it…

3.7.1 Dynamic UI Behaviour

3.7.2 Static UI

3.8 Design Decisions

This can document decisions on the design of the software.

Todo: If you want to use it, keep track of decisions that someone further developing this software or using it in the future might want to know about. This might save them time or clarify expectations. You can put them as a list, or whatever.

Things to consider:

- What exactly is the challenge?
- Why is it relevant for the architecture?
- What consequences does the decision have?
- Which constraints do you have to keep in mind?
- What factors influence the decision?

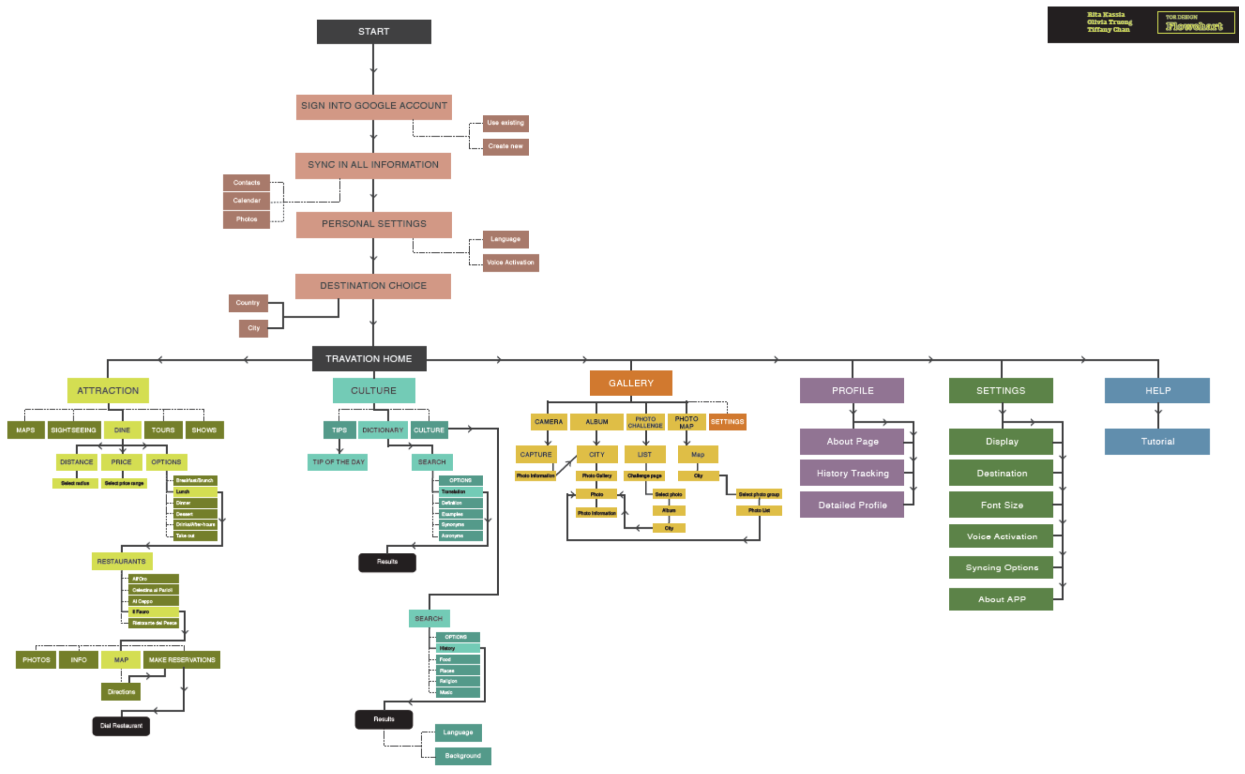


Fig. 2: Diagram showing the dynamic behaviour of the user interface i.e. State diagram

- Which assumption have you made?
- How can you check those assumptions?
- Which risks are you facing?
- Which alternative options did you consider?
- How do you judge each one?
- Which alternatives are you excluding deliberately?
- Who (if not you) has decided?
- How has the decision been justified?
- When did you decide?

3.9 Public Interfaces

class Nutshell

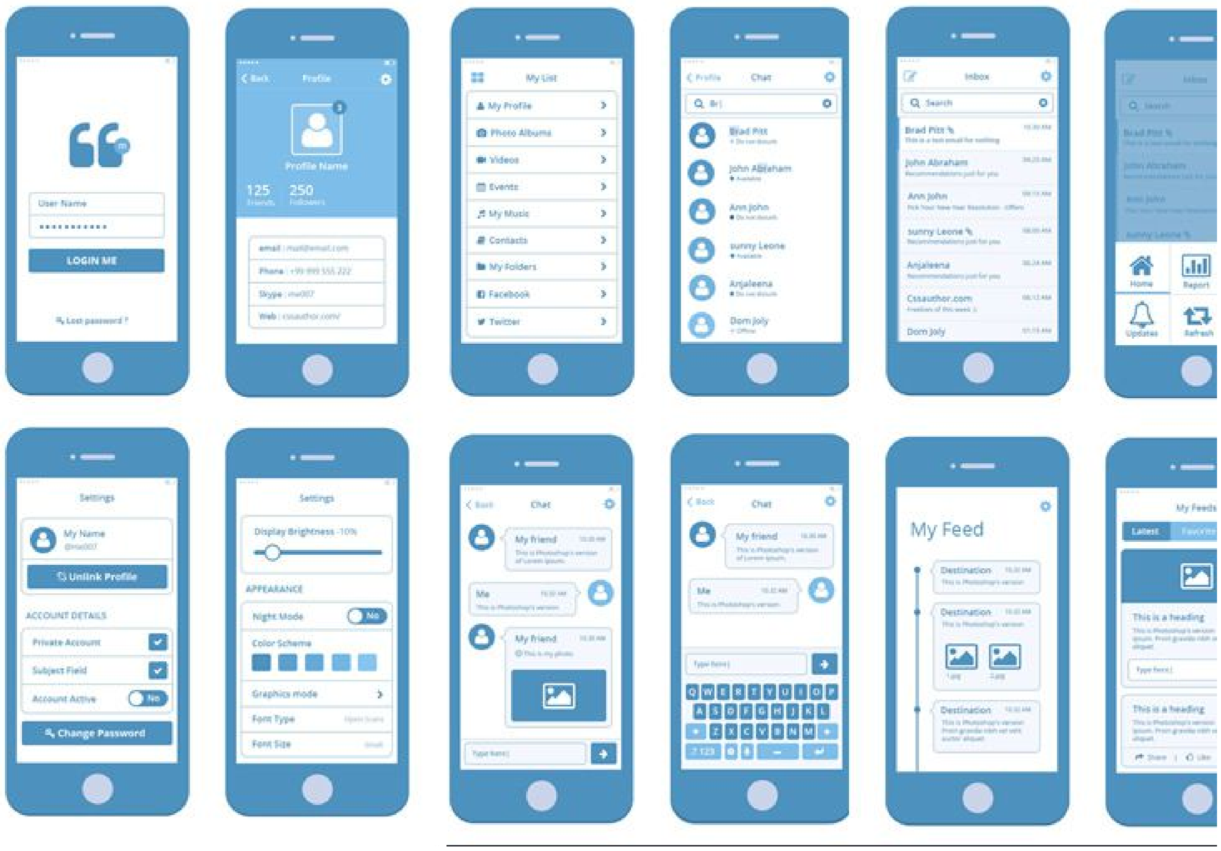


Fig. 3: Mock-up screens of the individual views/screens of the GUI i.e. Wireframes, whiteboard sketches

Public Types

`enum Tool`
Our tool set.

The various tools we can opt to use to crack this particular nut

Values:

`enumerator kHammer` = 0
What? It does the job.

`enumerator kNutCrackers`
Boring.

`enumerator kNinjaThrowingStars`
Stealthy.

Public Functions

`Nutshell``()`
Nutshell constructor.

`~Nutshell``()`
Nutshell destructor.

void **`crack`**`(`*Tool tool*`)`
Crack that shell with specified tool.

Parameters

- `tool`: - the tool with which to crack the nut

bool **`isCracked`**`()`

Return Whether or not the nut is cracked

Private Members

bool **`m_isCracked`**
Our cracked state.

file **`nutshell.h`**
An overly extended example of how to use breathe.

Enums

`enum` **[anonymous]**
Values:

`enumerator YellowRoller`

`enumerator RedRoller`

`enumerator GreenRoller`

`enum` **[anonymous]**
Values:

enumerator Chocolate

enumerator Mossy

enumerator CremeFraiche

group **nuttygroup**

The group for all nutjobs.

No nut can withstand us!

Enums

enum [anonymous]

Values:

enumerator YellowRoller

enumerator RedRoller

enumerator GreenRoller

group **nuttygroup2**

The group for all nutjobs.

More documentation for the second group.

Enums

enum [anonymous]

Values:

enumerator Chocolate

enumerator Mossy

enumerator CremeFraiche

dir **/home/docs/checkouts/readthedocs.org/user_builds/roboy-sw-documentation-template/checkou**

3.10 Solution Strategy

Todo: Describe your 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!

3.11 Test Strategy

Todo: If you have any tests describe:

- Where the tests are kept.
- How the tests are invoked.
- What you can/need to test manually.

3.11.1 System Tests

3.11.2 Integration Tests

3.11.3 Unit Tests

3.12 Building Block View

3.12.1 Overview

Todo: Inset a building block view:

- Static decomposition of the system into building blocks and the relationships thereof.
- Description of libraries and software used
-

We specify the system based on the blackbox view from *UML System Context* by now considering it a whitebox and identifying the next layer of blackboxes inside it. We re-iterate this zoom-in until specific granularity is reached - 2 levels should be enough.

Motivation.

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

Tool

- Create diagrams as below.

The white box view of the first level of your code. This is a white box view of your system as shown within the in Context in figure: *UML System Context*. External libraries and software are clearly marked.

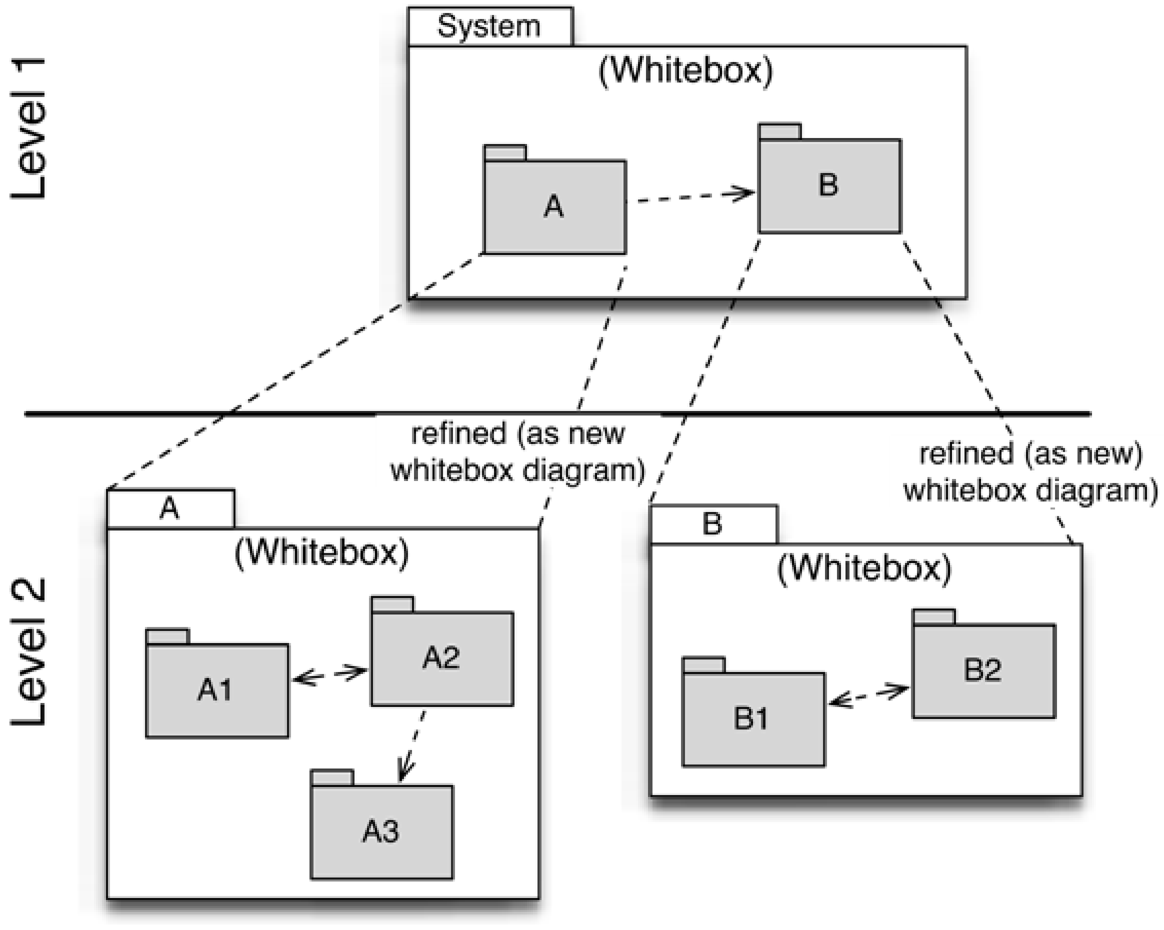

Level 1
System
(Whitebox)
A
B
refined (as new whitebox diagram)
refined (as new) whitebox diagram)
Level 2
A
(Whitebox)
A1
A2
A3
B
(Whitebox)
B1
B2

Fig. 4: Building blocks overview

3.12.2 Level 1 - Components

The highest level components

Todo:

- Define your groups using the **/defgroup** doxygen command
- Add **@addtogroup** tags to doxygen blocks of components in code as described here: http://www.stack.nl/~dimitri/doxygen/manual/grouping.html#modules
- Adapt the doxygencall to match the group name

---

group **`nuttygroup`**

The group for all nutjobs.

No nut can withstand us!

Enums

`enum` **[anonymous]**

Values:

`enumerator YellowRoller`

`enumerator RedRoller`

`enumerator GreenRoller`

`class Nutshell`

3.12.3 Level 2 - Components within each component of level 1

Todo:

- Define your groups using the **/defgroup** doxygen command
- Add **addtogroup** tags to doxygen blocks of components in code as described here: http://www.stack.nl/~dimitri/doxygen/manual/grouping.html#modules
- Adapt the doxygencall to match the group name

---

group **`nuttygroup2`**

The group for all nutjobs.

More documentation for the second group.

Enums

`enum [anonymous]`

Values:

`enumerator Chocolate`

`enumerator Mossy`

`enumerator CremeFraiche`

3.13 Runtime View

Todo: Add a runtime view of i.e. the startup of your code. This should help people understand how your code operates and where to look if something is not working.

Contents. alternative terms: - Dynamic view - Process view - 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:

- How are the most important use cases executed by the architectural building blocks?
- Which instances of architectural building blocks are created at runtime and how are they started, controlled, and stopped.
- How do the system's components co-operate with external and pre-existing components?
- 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.

  Especially 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.

3.13.1 Runtime Scenario 1

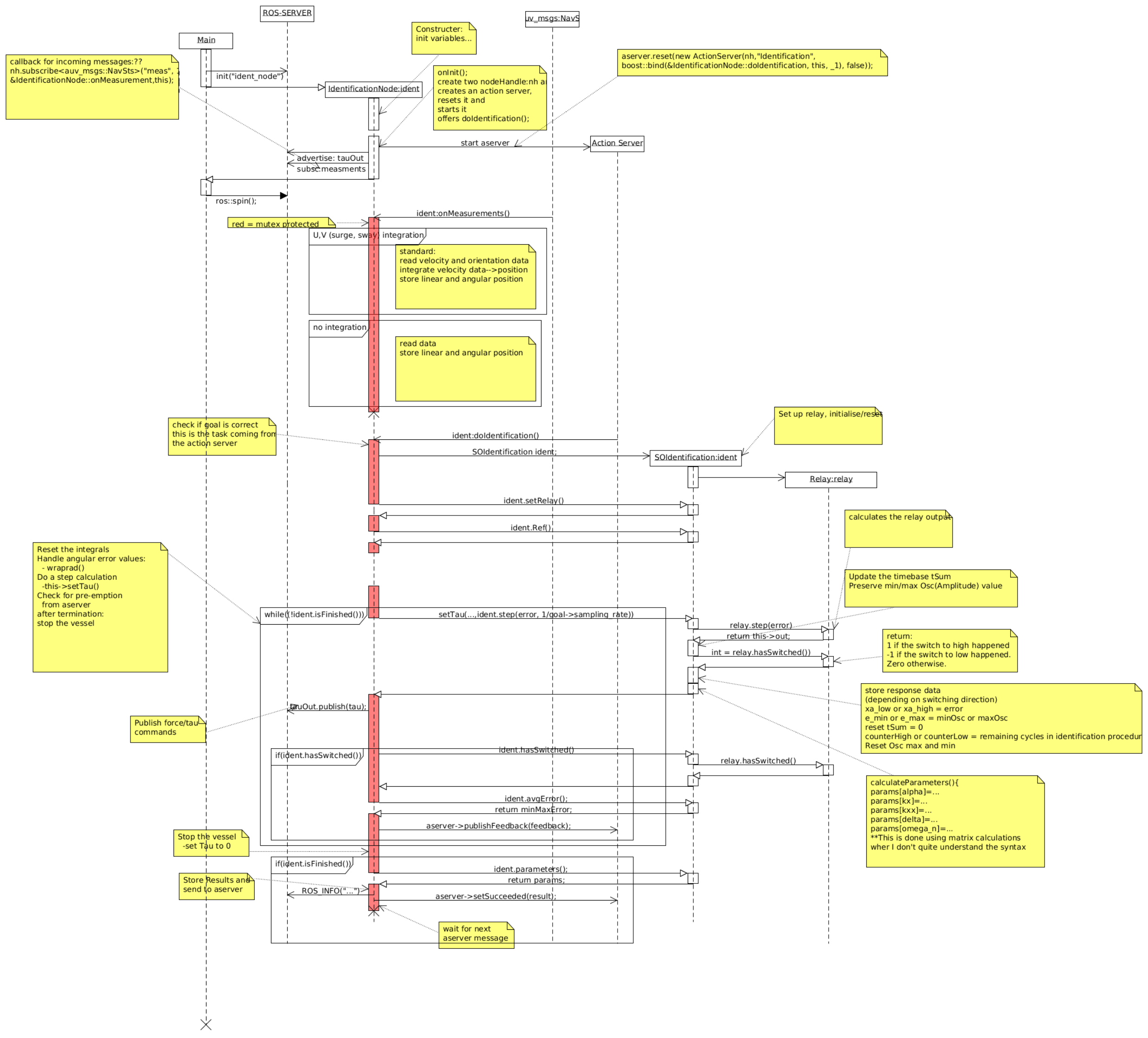

ROS-SERVER
Main
callback for incoming messages:??
nh.subscribe<auv_msgs::NavSts>("meas",
&IdentificationNode::onMeasurement,this);
init("ident_node")
Constructer:
init variables...
uv_msgs:NavS
IdentificationNode:ident
onInit();
create two nodeHandle:nh a
creates an action server,
resets it and
starts it
offers doIdentification();
aserver.reset(new ActionServer(nh,"Identification",
boost::bind(&IdentificationNode::doIdentification, this, _1), false));
start aserver
Action Server
advertise: tauOut
subsc:measments
ros::spin();
ident:onMeasurements()
red = mutex protected
U,V (surge, sway) integration
standard:
read velocity and orientation data
integrate velocity data-->position
store linear and angular position
no integration
read data
store linear and angular position
Set up relay, initialise/reset
check if goal is correct
this is the task coming from
the action server
ident:doIdentification()
SOIdentification ident;
SOIdentification:ident
Relay:relay
ident.setRelay()
ident.Ref()
calculates the relay output
Reset the integrals
Handle angular error values:
- wraprad()
Do a step calculation
-this->setTau()
Check for pre-emption
from aserver
after termination:
stop the vessel
Update the timebase tSum
Preserve min/max Osc(Amplitude) value
while((!ident.isFinished()))
setTau(...,ident.step(error, 1/goal->sampling_rate))
relay.step(error)
return this->out;
int = relay.hasSwitched())
return:
1 if the switch to high happened
-1 if the switch to low happened.
Zero otherwise.
store response data
(depending on switching direction)
xa_low or xa_high = error
e_min or e_max = minOsc or maxOsc
reset tSum = 0
counterHigh or counterLow = remaining cycles in identification procedur
Reset Osc max and min
Publish force/tau
commands
tauOut.publish(tau);
if(ident.hasSwitched())
ident.hasSwitched()
relay.hasSwitched()
calculateParameters(){
params[alpha]=...
params[kx]=...
params[kxx]=...
params[delta]=...
params[omega_n]=...
**This is done using matrix calculations
wher I don't quite understand the syntax
ident.avgError();
return minMaxError;
aserver->publishFeedback(feedback);
Stop the vessel
-set Tau to 0
if(ident.isFinished())
ident.parameters();
return params;
Store Results and
send to aserver
ROS_INFO("...")
aserver->setSucceeded(result);
wait for next
aserver message

Fig. 5: **UML-type sequence diagram** - Shows how components interact with each other during runtime.

3.13.2 Runtime Scenario 2

…

3.14 Deployment View

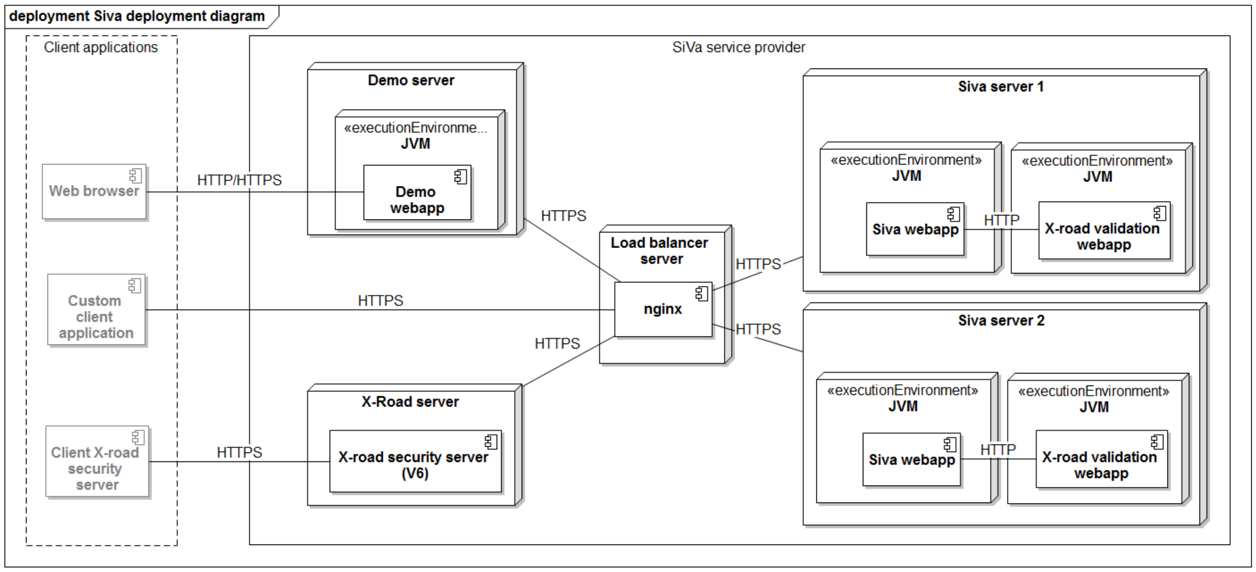
Todo: Add a deployment diagram. **Contents.**

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.

Motivation.

Software is not much use without hardware. These models should enable the operator to properly install the software.

You can separate this into different levels…



3.15 Libraries and external Software

Contains a list of the libraries and external software used by this system.

Todo: List libraries you are using

Table 6: Libraries and external Software

Name	URL/Author	License	Description
arc42	http://www.arc42.de/template/	Creative Commens Attribution license.	Template for documenting and developing software

3.16 About arc42

This information should stay in every repository as per their license: http://www.arc42.de/template/licence.html

arc42, the Template for documentation of software and system architecture.

By Dr. Gernot Starke, Dr. Peter Hruschka and contributors.

Template Revision: 6.5 EN (based on asciidoc), Juni 2014

Note

This version of the template contains some help and explanations. It is used for familiarization with arc42 and the understanding of the concepts. For documentation of your own system you use better the *plain* version.

3.16.1 Literature and references

Starke-2014 Gernot Starke: Effektive Softwarearchitekturen - Ein praktischer Leitfaden. Carl Hanser Verlag, 6, Auflage 2014.

Starke-Hruschka-2011 Gernot Starke und Peter Hruschka: Softwarearchitektur kompakt. Springer Akademischer Verlag, 2. Auflage 2011.

Zörner-2013 Softwarearchitekturen dokumentieren und kommunizieren, Carl Hanser Verlag, 2012

3.16.2 Examples

- HTML Sanity Checker
- DocChess (german)
- Gradle (german)
- MaMa CRM (german)
- Financial Data Migration (german)

3.16.3 Acknowledgements and collaborations

arc42 originally envisioned by Dr. Peter Hruschka and Dr. Gernot Starke.

Sources We maintain arc42 in *asciidoc* format at the moment, hosted in GitHub under the aim42-Organisation.

Issues We maintain a list of open topics and bugs.

We are looking forward to your corrections and clarifications! Please fork the repository mentioned over this lines and send us a *pull request*!

3.16.4 Collaborators

We are very thankful and acknowledge the support and help provided by all active and former collaborators, uncountable (anonymous) advisors, bug finders and users of this method.

Currently active

- Gernot Starke
- Stefan Zörner
- Markus Schärtel
- Ralf D. Müller
- Peter Hruschka
- Jürgen Krey

Former collaborators

(in alphabetical order)

- Anne Aloysius
- Matthias Bohlen
- Karl Eilebrecht
- Manfred Ferken
- Phillip Ghadir
- Carsten Klein
- Prof. Arne Koschel
- Axel Scheithauer

INDEX