ECSEL Joint Undertaking

# R5-COP

Reconfigurable ROS-based Resilient Reasoning Robotic Cooperating Systems

# D35.10: Requirements

Jan Wagner, Sönke Michalik (TUBS), Tadeusz Dobrowiecki, István Majzik, András Förhécz, István Engedy, Péter Eredics (BME), Viatcheslav Tretyakov (SYN)

| | | | |
|---|---|---|---|
| **Project** | R5-COP | **Grant agreement no.** | 621447 |
| **Deliverable** | D35.10 | **Date** | 20.02.2015 |
| **Contact Person** | Viatcheslav Tretyakov, Jan Wagner | **Organisation** | SYN, TUBS |
| **E-Mail** | vtretyakov@synapticon.com wagner@c3e.cs.tu-bs.de | **Diss. Level** | PU |

| Document History | | | |
|---|---|---|---|
| **Ver.** | **Date** | **Changes** | **Author** |
| 0.1 | 12.01.2015 | Creation of initial document | Viatcheslav Tretyakov (SYN) |
| 0.2 | 10.02.2015 | First batch of TUBS content added | Jan Wagner (TUBS) |
| 0.3 | 29.01.2015 | Filling sections x.4-x.5 | Tadeusz Dobrowiecki, István Majzik, András Förhécz, István Engedy, Péter Eredics (BME) |
| 0.4 | 16.02.2015 | Take over comments from Istvan | Jan Wagner (TUBS) |
| 0.5 | 17.02.2015 | Added HW/SW profiling requirements | Sönke Michalik (TUBS) |
| 0.6 | 20.2.2015 | Filling in SYN's sections, general editing | Viatcheslav Tretyakov (SYN), Jan Wagner (TUBS) |
| 0.7 | 20.2.2015 | General editing | Viatcheslav Tretyakov (SYN), |
| 0.8 | 09.03.2015 | Modifications requested from review | Jan Wagner (TUBS) |
| 0.9 | 09.03.2015 | Merged in BME's review changes | Jan Wagner (TUBS) |
| 1.0 | 12.03.2015 | Changes version to 1.0 to make it final | Jan Wagner (TUBS) |
| | | | |
| | | | |
| | | | |
| | | | |

***Note: Filename should be***

*"R5-COP_D##_#.doc", e.g. „R5-COP_D91.1_v0.1_TUBS.doc"*

***Fields are defined as follow***

**1. Deliverable number** ***.***

**2. Revision number:**

**draft version** **v**

**approved** **a**

**version sequence (two digits)** ***.***

**3. Company identification (Partner acronym)** *****

**Content**

**List of Figures:**

## List of Acronyms

| ABBREV | Explanation |
|---|---|
| DB | Data Base |
| EDA | Electronic Design Automation |
| GUI | Graphic User Interface |
| JTAG | Joint Test Action Group (IEEE-Standard 1149.1) |
| HCI | Human Computer Interaction |
| KB | Knowledge Base |
| OS | Operating System |
| RTL | Register Transfer Level |
| SaaS | Software as a Service |
| SMKB | Skill Model Knowledge Base |
| SoC | System-on-Chip |
| UART | Universal Asynchronous Receiver Transmitter |

# 1 Introduction

## 1.1 Summary (abstract)

This document is D35.10 delivery report of the R5-COP project. Work package 35 addresses several aspects of supporting engineers in their work. Modern industries rely heavily on embedded systems under the assumption these systems are dependable in terms of completing assigned tasks in a correct and safe way. In order to guarantee this behavior certain properties of the underlying hardware and the hardware/software interfaces need to be well-known.

The objective of this work package is to design tools, which provide information about crucial properties of hardware and hardware-software-interaction as early as possible to the designer and support him to make the right decisions. This feedback enables designers to design tailored components which meet the requirements, where especially the early feedback helps to lower the development costs as well as helps to speed up the design process through avoiding unnecessary design cycles.

This report contains a description of the context for which each tool is considered and a description of the planned tool itself. Furthermore the requirements for all tools are collected within this report.

## 1.2 Purpose of document

This document is a collection of the requirements for the tools to be created in the context of work package 35. It contains a survey of the requirements to be fulfilled by the tools and represents the basis for the upcoming work.

## 1.3 Partners involved

| **Partners and Contribution** | |
|---|---|
| **Short Name** | **Contribution** |
| TUBS | Requirements for T35.1, consultation regarding T35.2, T35.3 |
| BME | Requirements for Configuration and Skill Composer Tools, 2.4-2.5, 3.4-3.5 |
| DTI | No work assigned within the scope of this deliverable |
| FAU | Review |
| SYN | Requirements for T35.2, T35.3 |

# 2 Context of the to-be-developed tools

Each tool created in WP35 is invented to support the designer or user of a system in a certain way. This chapter will give an insight into the context in which these tools will aid the designer to get his work done.

## 2.1 Soft-error-prone components analysis

Electronic components, especially integrated circuits can fail in different ways. There are three classes of failures specified: permanent faults, intermittent faults and transient faults, also known as soft-errors. Soft-errors are caused by radiation. When an alpha particle or neutron hits a transistor it interacts with the silicon chrystals. Soft-errors do not damage the circuit, but cause a bit flip somewhere in the circuit, resulting in a wrong outcome of a calculation or a wrong state of hard- or software. The work done in this task work will only address soft-errors in System-on-Chips (SoCs).

In task T35.1 techniques shall be investigated which allow to make an estimation of the soft-error vulnerability of components within a System-on-Chip (SoC) before the actual chip is produced and even before a RTL model is available, under the assumption that a modern development strategy is applied.

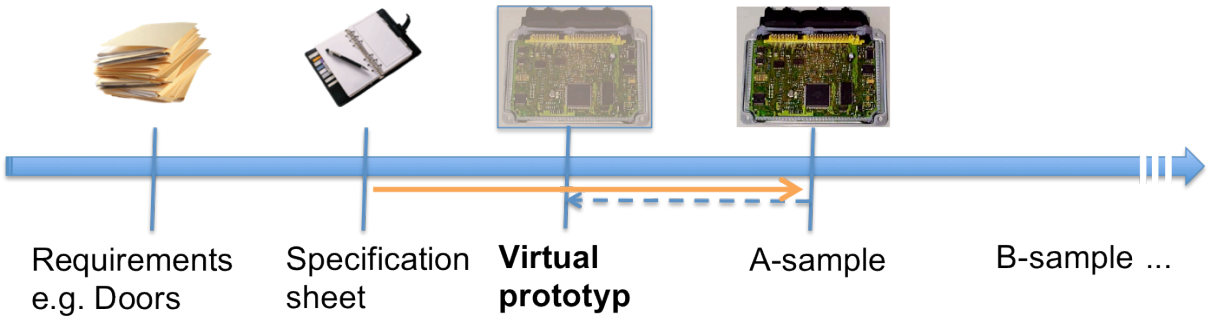


**Figure 1: Development timeline**

As shown in Figure 1 every design starts with a collection of requirements. Based on this collection the specification is build, which is the blueprint for the first sample, including timing specifications for critical features of the planned chip. Since the creation of a first chip is a very time consuming process and often the specification unintentionally leaves space for interpretation the concept of virtual prototypes was introduced. A virtual prototype is described in a high-level language and describes the specified functionality plus estimated timing information for selected actions with a defined accuracy. By disregarding the strong timing restrictions required in a hardware description and the ability of a high-level description for most issues a significant time saving could be achieved in creating the virtual platform compared to a hardware description.

The virtual platform itself can be used as golden-reference for the hardware design since all ambiguities from the specification need to be clarified during the implementation process. Moreover software development for the targeted SoC can start as soon as the virtual platform is available as a simulator of the final system.

The techniques proposed for “Soft-error-prone components analysis” are going to start with the existing virtual platform. At suitable places in the simulator, information about the use of selected components will be collected, while running the actual software in the simulator. Based on an analysis of the gathered information it shall clear which components are essential for the correct function of the SoC.

At the end the virtual prototype will provide information on the functionality of the SoC, the timing (depending on the targeted and implemented timing accuracy) and on which components need special attention regarding soft-error vulnerability (Requirement WP35.1.1). Hav-

ing this information available in an early design stage aids the development of safe and reliable systems, which are an indispensable building block of safe robotic systems. Safety in robotics is an upcoming issue since robots are going to be used real-world environments allowing direct interaction with humans and the environment. This introduces the risk robots will hurt humans or cause severe damage to goods and properties, which must be reduced to a minimum using safe and reliable hardware.

In most cases software is not running bare-metal on a processor, but using an underlying operating system. Operating system boot up can be a very challenging task in a simulator. It can take quite a lot of time to perform it, while during this phase no task is performed for which the system was originally added to the system. Hence the most interesting time-span regarding reliability is after the system was booted and started executing the actual program it was designed for. This results in the requirement the proposed techniques need to consider an adequate runtime of the simulator if possible (requirement WP35.1.4).

There are plenty of possible ways to simulate a system. In order to focus on a feasible range of solutions the choice of the tool needs to be narrowed down.

In recent years SystemC became a very popular simulation framework for simulation in the EDA community. To support the exchangeability of modules from different vendors TLM (transaction level modelling) was introduce as a common interface between modules in SystemC simulations. Both techniques are standardized as IEEE1666 [1] and supported by EDA tool vendors like Cadence, Synopsys or Mentor Graphics. Moreover, there are reference implementations of SystemC and TLM available as open-source (http://www.accellera.org/downloads/standards/systemc). The techniques proposed in T35.1 are considered to be compatible with the latest versions of the reference simulator available when writing this report (SystemC 2.3 and TLM 2.0) (requirement WP35.1.2).

Simulating a whole system makes high demands on the system performing the simulation. A lot of data is expected to accumulate and not all data can be written immediately to disk. Hence it needs to be stored, or at least buffered, in the computers main memory. Due to the fact that 32-bit systems are not able to handle more than 4 GB main memory, a 64-bit operating system is required in conjunction with at least 8 GB RAM. Linux as operating system is selected due to good out-of-the-box support of many software libraries and powerful command-line tools (requirement WP35.1.3)

## 2.2 Hardware and hardware/software profiling

Efficient implementation of embedded software and hardware is the key contributor to embedded system performance and low power consumption. Therefore, an adequate profiling is needed to increase the overall system performance and lowering power consumption of hardware/software co-designs. The profiling will include analysis of execution time, FPGA logic/BRAM occupation, CPU/memory usage and power consumption.

The investigation of profiling of components oriented towards video and signal processing is important, since image processing is one of the most resource demanding tasks in robotic applications.

Hence, the focus of this task lies on profiling of software components and hardware implemented in FPGA on heterogeneous systems combining FPGA and CPU, such as Xilinx Zynq platforms and development of software.
The processing platform and selected image processing algorithms shall be suitable for hardware/software implementations and shall feature modular structure and support partitioning.

The platform shall provide interfaces to the ROS middleware, video image sensors and algorithm debugging.

The algorithm implementation shall be aware of available hardware resources.

## 2.3 Distributed control software editor

Developers of distributed control software for embedded systems are facing numerous challenges. The distributed control software editor will support the designer in creating high quality software for such systems within reasonable time. But since embedded systems are most efficient when tailored to a certain use-case, the software cannot be considered without the hardware. This relationship justifies the demand for a tool helping the designer with creation of customized, synchronized software and hardware.

Software development for embedded devices always needs to deal with very limited hardware resources, resulting in the need of a well-thought-out software architecture to meet all requirements defined by the targeted application. Especially embedded control systems have strong requirements regarding real-time behaviour and correct usage of several external interfaces. Deep knowledge of the underlaying hardware is inevitable to succeed in designing software for such systems.

The complexity of such systems grows significantly, if the control tasks are not handled by a single embedded system, but by a network of several embedded systems. Besides the partitioning, which assigns subtasks to certain nodes in the network, a communication infrastructure needs to be established in hardware as well as in software. Partitioning is considered to be performed by the designer, but once the hardware architecture and physical links are defined, the distributed control software editor is considered to set up the software architecture automatically for the given hardware architecture. The generated software setup shall include the communication infrastructure and stubs for functions where the designer can implement the actual application.

To sum up, a tool is required taking the description of the hardware architecture as input and providing a suitable software framework to the designer. Moreover it shall have capabilities enabling software development, such as a code editor and compiler support.

## 2.4 Configuration tool support

An application introduces high-level problem and goal dependent requirements. To obtain a successfully designed (optimal, optimized, robust, etc.) robotic system such abstract requirements must be matched to the available resources producing then realizable software and hardware requirements. To this end a decision support tool (Configuration tool) will be developed, demonstrating to the user how his application can be realized as a configuration (or reconfiguration) of (possibly hierarchical) robotic skills.

Technically the required terminological and relational knowledge will be collected and transformed into formal ontological structures via controlled language interfaces. An ontology-based core knowledge base covering configurability related notions, capabilities, and metrics will be developed, coupled to the Skill Model Knowledge Base (SMKB) (to be developed in WP13). To implement system services reasoning algorithms will be developed around the overall ontology knowledge base and a small scale prototype decision support system will be demonstrated, which from the queries to ontologies will find compatible services expressed as skill configurations and will instantiate them according to the specific information provided by the system user.

## 2.5 Skill composer tool

Skill Composer is a decision support tool to create new skills compliant to a skill model used in R5-COP (Skill Model Knowledge Base (SMKB) defined in WP13), based on available software component information from SP2. To this end an ontology-based knowledge base to select both appropriate components for the skill and to add respective semantics to describe the properties and capabilities of the skill must be developed. The tool will also support the hierarchical composition of lower-level skills into higher-level skills. Furthermore, configuration of parameters of the individual skills will be supported.

## 2.6 The use of the Configuration and Skill Composer tools

Configuration and Skill Composer tools complement each other and to obtain a full view of how an application could be implemented as a robotic system, the services of both of them are needed, see Figure 2.[1]

When provided with sufficiently elaborated knowledge bases and algorithms Skill Composer can provide a software/hardware instantiation of the skill configurations proposed by Configuration tool, which serves as a functional blueprint to fulfil the demands of the application. To achieve it the skill configurations (output of the Configuration tool) must be passed over (by the user, or in some other way automatically) to the Skill Composer tool.

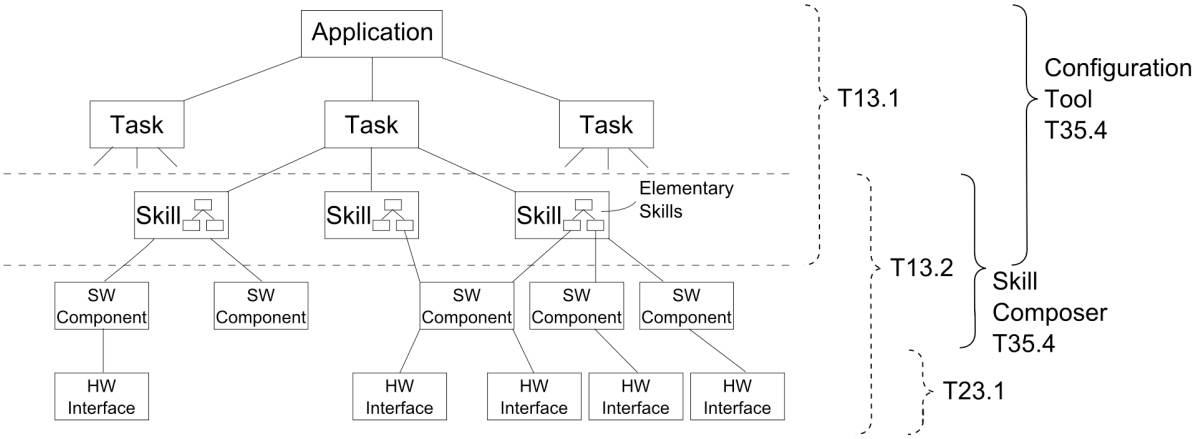


**Figure 2: Hierarchy of task related R5-COP concepts, the project tasks, and the scope of the decision support tools**

The cooperative maintenance and use of both tools can be roughly decomposed into the following activities and interactions, see Figure 3:

(1) Embedding Skill Model Knowledge Base (WP13) into the architecture of the Configuration tools. The embodiment can be direct, with the SMKB being a part of the architecture, or indirect making the SMKB a remotely queried resource.

(2) The same remark applies to the Software component/Hardware interface Knowledge Base to be developed in SP2. Its information must be available to the reasoning algorithms in the Skill Composer. It is however, yet open how it should be solved architecturally.

(3) Application domain knowledge is being brought by the user, who must be qualified enough not only in the application, but also in questions of acceptable alternatives and quality measures.

(4) The Query/Answer interface medium to both, the Skill Composer and the Configuration Tool, must be tailored properly to facilitate the most the influx of the robotic and application related information without the burden of handling special IT concepts (representations) by the user. Possibly, the best will be to use a form structured (limited, controlled) natural language interface.

(5) The working session between the robot designer and the Configuration tool should conclude in a feasible skill configuration. The Configuration tool warrants that the scheme is

[1] In Fig 2. as higher order skills are abstractions realized in system software, SW Components are always placed between them and HW Interfaces to designate the character of the transition between the digital and the analogue system level. In cases when an elementary skill functionality is entirely covered by the HW component, SW component will mean a simple driving software. In more involved cases, when the function of the hardware must be backed up by proper algorithm, a SW component will be a (possibly plug-in) fully blown algorithmic module.

sound and consistent with the general principles of skill based modelling. User accepting it indicates that when implemented, it will satisfy his view of the application requirements (i.e. any later problem can be pin-pointed as user error).

(6) The working session between the robot designer and the Skill Composer tool should conclude in a feasible skill (configuration) implementation. Skill Composer warrants that with the proposed software and hardware components the skill can be realized and provide functions and quality assurances required by the application.

(7) The maintenance of the Configuration tool requires much of knowledge engineering. Skill models used by the tool to build skill configurations implementing applications must be verified, extended, edited, modified, etc. at a proper level of expertise (see D13.11). As this knowledge hardly will be provided by the nominal user - robot designer - another expert user must be involved to keep the tool operational.

(8) Similar remark applies to the Skill Composer also. Technically the knowledge engineering covers the same issues of verifying, editing, modifying, etc., however, the scope of the expert knowledge differs. The knowledge engineer here must draw his knowledge from interactions with different kind of professionals.

(9) Cooperation between knowledge engineers. It can be the same person skilled in transforming various fields' knowledge into IT representation, but it could be different. In that case a strict cooperation is required between them, because the consistency of both decisions must be assured.

(10) Filling input forms is based on the deep knowledge of the application but must be guided by the tool. The input forms must be arranged hierarchically, and filling of the forms will be supported by (controlled) natural language to filter out potentially inconsistent user inputs.

(11) The question (related to point 9.) is whether it wouldn't be better to fuse (integrate) the two system functions (application → skill configuration, and skill → software/hardware configuration) into one tool, providing solution in a single session, or decompose the problem design into stages. As the mapping from the application to the skill configuration, and also from skill configuration to software/hardware components is one-to-many, a fused system would have to handle too large search, compromising the quality of the proposed solution. Decomposition of the process into two stages cuts down the complexity, permits to design better reasoning algorithms, and also frees the user from thinking across a too large span of abstraction levels. However, then some kind of mechanism (formalism, representation) to link the data (skill configuration as the output of the Configuration tool and as the input to the Skill Composer) must be designed also to help the user to transfer the task between both tools.

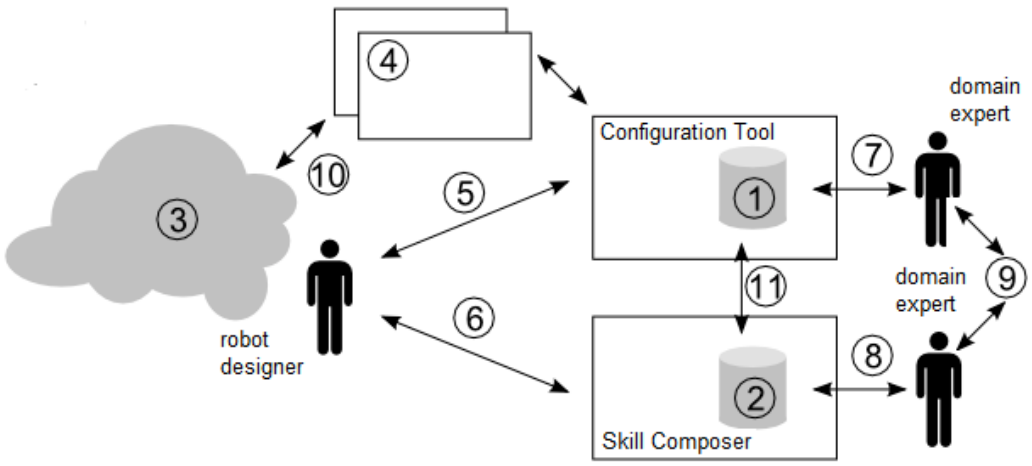


**Figure 3: The functional relation of the Configuration and the Skill Composer Tools**

# 3 Requirements

## 3.1 Soft-error-prone components analysis

| Field Name | Value |
|---|---|
| Requirement ID | WP35.1.1 |
| Title | Statement about soft-error vulnerability of certain components |
| Description | The techniques proposed in T35.1 shall provide the designer of a system with information on soft-error vulnerability during early design stages. |
| Rationale | Based on the information gathered from running a virtual platform, estimation shall be performed on the vulnerability of certain components. This gives hints to the designer if design decisions were correct and if certain components require additional mechanisms in order to comply with the expectations on the system regarding the soft-error ruggedness. |
| Importance | High |
| Assigned Tasks/WPs | T35.1 |
| Partners contributing to this description | TUBS |
| Responsibility and Reference Person | Jan Wagner (TUBS) |

| Field Name | Value |
|---|---|
| Requirement ID | WP35.1.2 |
| Title | Virtual Platform Simulator |
| Description | The techniques reviewed shall be compatible with simulators based on SystemC 2.3 and TLM 2.0 |
| Rationale | Since there are plenty of possible ways to simulate a system a common foundation for the reviewed techniques needs to be defined. In recent years SystemC became a very popular simulation framework for simulation in the EDA community. In order to support the exchangeability of modules from different vendors TLM (transaction level modelling) was introduced as a common interface. Both techniques are supported by the EDA tools vendors like Cadence, Synopsys or Mentor Graphics. Moreover there are reference implementations of SystemC and TLM available as open-source (http://www.accellera.org/downloads/standards/systemc). The techniques proposed in T35.1 shall be compatible with the latest versions of the reference simulator available when writing this report (SystemC 2.3 and TLM 2.0) |
| Importance | High |
| Assigned Tasks/WPs | T35.1 |
| Partners contributing to this | TUBS |

| | |
|---|---|
| description | |
| Responsibility and Reference Person | Jan Wagner (TUBS) |

| **Field Name** | **Value** |
|---|---|
| Requirement ID | WP35.1.3 |
| Title | Host system |
| Description | The simulation host shall run a 64-bit Linux and provide at least 8 GB RAM. |
| Rationale | While simulating a whole system a lot of data accumulate. Not all data can be written immediately to disk, hence it needs to be stored in the computers main memory. Due to the fact that a 32-bit system is not able to handle more than 4 GB main memory, a 64-bit OS is required in conjunction with at least 8 GB RAM. Linux as operating system is selected due to good out-of-the-box support of many libraries and command-line tools. |
| Importance | Low |
| Assigned Tasks/WPs | T35.1 |
| Partners contributing to this description | TUBS |
| Responsibility and Reference Person | Jan Wagner (TUBS) |

| **Field Name** | **Value** |
|---|---|
| Requirement ID | WP35.1.4 |
| Title | Capability to simulate an adequate timespan |
| Description | The observation of a system shall cover an adequate timespan in order to be capable to make meaningful statements regarding the usage of certain components. |
| Rationale | If an operating system is involved in the investigation the simulator needs to run longer than the OS boot up takes plus some time in normal operation mode. Hence the proposed technique needs to consider the vast amount of data, which can be produced while running long simulations, if possible. |
| Importance | Medium |
| Assigned Tasks/WPs | T35.1 |
| Partners contributing to this description | TUBS |

| | |
|---|---|
| Responsibility and Reference Person | TUBS (Jan Wagner) |

### 3.2 Hardware and hardware/software profiling

| **Field Name** | **Value** |
|---|---|
| Requirement ID | WP35.2.1 |
| Title | HW/SW profiling platform |
| Description | The HW/SW profiling platform shall consist of a reconfigurable hardware (FPGA) partition and a general purpose processor (CPU). Both partitions shall be able to exchange information through a bus interface or streaming interface. |
| Rationale | To support various HW/SW profiling strategies the platform has to support algorithm execution in software as well as in dedicated hardware. |
| Importance | High |
| Assigned Tasks/WPs | T35.2 |
| Partners contributing to this description | SYN, TUBS |
| Responsibility and Reference Person | Viatcheslav Tretyakov, Sönke Michalik |

| **Field Name** | **Value** |
|---|---|
| Requirement ID | WP35.2.2 |
| Title | HW/SW allocable algorithms |
| Description | Algorithms shall be suitable for HW and SW implementation |
| Rationale | The selected algorithms shall provide features that can be synthesised in hardware for acceleration in addition to software features that enable superior configuration and modification. |
| Importance | Medium |
| Assigned Tasks/WPs | T35.2 |
| Partners contributing to this description | SYN, TUBS |
| Responsibility and Reference Person | Viatcheslav Tretyakov, Sönke Michalik |

| Field Name | Value |
|---|---|
| Requirement ID | WP35.2.3 |
| Title | Modular algorithms |
| Description | Modular algorithms shall be based on pipelined architectures that can be subdivided in modular components for partitioning and easy adaption. |
| Rationale | The selected algorithms shall provide modular subcomponents that can be implemented in hardware and/or software. |
| Importance | Medium |
| Assigned Tasks/WPs | T35.2 |
| Partners contributing to this description | SYN, TUBS |
| Responsibility and Reference Person | Viatcheslav Tretyakov, Sönke Michalik |

| Field Name | Value |
|---|---|
| Requirement ID | WP35.2.4 |
| Title | Image Sensor Interface |
| Description | Image Sensor interface shall provide interfaces for High Resolution Video/Image capturing as input for image processing algorithms. |
| Rationale | The Sensor interface shall provide a direct and low latency connection to the processing building blocks. The Image sensors shall be accessible from hardware and software. |
| Importance | High |
| Assigned Tasks/WPs | T35.2 |
| Partners contributing to this description | SYN, TUBS |
| Responsibility and Reference Person | Viatcheslav Tretyakov, Sönke Michalik |

| Field Name | Value |
|---|---|
| Requirement ID | WP35.2.5 |
| Title | ROS Interface |
| Description | The interface to the robot middleware shall be compatible to ROS middleware framework:<br>- ROS message parsing<br>- ROS publisher/subscriber |

| | |
|---|---|
| | - Ethernet interface |
| Rationale | The processed data shall be forwarded to the robotic middleware ROS for easy integration into robotic systems. |
| Importance | High |
| Assigned Tasks/WPs | T35.2 |
| Partners contributing to this description | SYN, TUBS |
| Responsibility and Reference Person | Viatcheslav Tretyakov, Sönke Michalik |

| **Field Name** | **Value** |
|---|---|
| Requirement ID | WP35.2.6 |
| Title | Debug Interface |
| Description | The Platform shall provide interfaces for debugging of HW/SW development:<br>- JTAG<br>- UART<br>- HDMI (Video/Data-output) |
| Rationale | The results and execution status of the processing algorithms shall be displayable over debug interfaces. |
| Importance | Low |
| Assigned Tasks/WPs | T35.2 |
| Partners contributing to this description | SYN, TUBS |
| Responsibility and Reference Person | Viatcheslav Tretyakov, Sönke Michalik |

| **Field Name** | **Value** |
|---|---|
| Requirement ID | WP35.2.7 |
| Title | Resource aware implementation |
| Description | Algorithm implementation shall be based on performance metrics:<br>- Execution time<br>- FPGA logic/BRAM occupation<br>- CPU/memory usage |
| Rationale | The hardware and software implementation shall be aware of available resources as memory/FPGA logic resources and processing system ca- |

| | |
|---|---|
| | pabilities. |
| Importance | Medium |
| Assigned Tasks/WPs | T35.2 |
| Partners contributing to this description | SYN, TUBS |
| Responsibility and Reference Person | Viatcheslav Tretyakov, Sönke Michalik |

### 3.3 Distributed control software editor

| **Field Name** | **Value** |
|---|---|
| Requirement ID | WP35.3.1 |
| Title | Embedded System Editor |
| Description | The distributed control software editor shall enable inexperienced designers to compose an embedded system from predefined building blocks. |
| Rationale | Software for embedded systems always needs to deal with the limited resources available on such systems. Moreover control software needs to meet real-time requirements. These requirements can only be fulfilled by using dedicated hardware components. But also other features of embedded systems, such as many interfaces to external devices, rely on the close interaction of hard- and software. Hence a deep knowledge of the underlaying hardware is necessary during the software design process.<br>The hardware architecture description of the embedded system needs to be provided to the distributed control software editor. As interface between designer and distributed software control editor the embedded system editor is required. |
| Importance | High |
| Assigned Tasks/WPs | T35.3 |
| Partners contributing to this description | SYN, TUBS |
| Responsibility and Reference Person | Viatcheslav Tretyakov, Jan Wagner |

| **Field Name** | **Value** |
|---|---|
| Requirement ID | WP35.3.2 |
| Title | OS-agnostic |
| Description | The distributed control software editor shall run on multiple operating sys- |

| | |
|---|---|
| | tems to be compatible with the preferred OS of the customer. |
| Rationale | Customers usally have their own well tested workflow. The distributed control software editor is considered to be integrated in this workflow without hassles like requiring a certain operating system or extraordinary libraries. Hence it shall follow an OS-agnostic approach, such as using Java or Python or even without any extra software on the customer's computer as a Software-as-a-Service (SaaS) application running in a web browser. |
| Importance | Medium |
| Assigned Tasks/WPs | T35.3 |
| Partners contributing to this description | SYN, TUBS |
| Responsibility and Reference Person | Viatcheslav Tretyakov, Jan Wagner |

| **Field Name** | **Value** |
|---|---|
| Requirement ID | WP35.3.3 |
| Title | Software framework generator |
| Description | The distributed control software editor shall generate a fully operational software framework based on the created embedded system architecture. |
| Rationale | The distributed control software editor is intended to support the designer in creating applications for a specific embedded system. This requires code making the hardware layer usable to the software. This includes defining which software part is executed on which processor and initialisation of all hardware drivers. To free the designer from the burden of typing in a lot of boilerplate code at the beginning of such a project an initial code framework shall be generated based on the hardware description generated with the Embedded System Editor (requirement WP35.3.1) |
| Importance | High |
| Assigned Tasks/WPs | T35.3 |
| Partners contributing to this description | SYN, TUBS |
| Responsibility and Reference Person | Viatcheslav Tretyakov, Jan Wagner |

| **Field Name** | **Value** |
|---|---|
| Requirement ID | WP35.3.4 |
| Title | Source code editor |

| | |
|---|---|
| Description | The Distributed control software editor shall enable the customer to develop the whole application code. Hence a source code editor is mandatory. |
| Rationale | Editing source code is an inevitable feature of the distributed control software editor. Hence it shall support the designer with features for effective code editing, such as syntax highlighting. |
| Importance | High |
| Assigned Tasks/WPs | T35.3 |
| Partners contributing to this description | SYN, TUBS |
| Responsibility and Reference Person | Viatcheslav Tretyakov, Jan Wagner |

| **Field Name** | **Value** |
|---|---|
| Requirement ID | WP35.3.5 |
| Title | Compiler tools |
| Description | All required compilers shall be accessible from the distributed control software editor. |
| Rationale | The distributed control software editor is planned as the central tool to create applications for the targeted embedded systems. This includes giving feedback to the designer if the source code is valid and providing binary executable files generated from the source code. |
| Importance | Medium |
| Assigned Tasks/WPs | T35.3 |
| Partners contributing to this description | SYN, TUBS |
| Responsibility and Reference Person | Viatcheslav Tretyakov, Jan Wagner |

| **Field Name** | **Value** |
|---|---|
| Requirement ID | WP35.3.6 |
| Title | Sanity checks |
| Description | The distributed control software editor shall perform sanity checks during the design process and guide the customer to a realizable and sensible design. |
| Rationale | The designer of an embedded system shall get feedback regarding the correctness of his design as quick as possible. The sanity check shall warn the designer for example if more devices are connected to a CPU |

| | |
|---|---|
| | than interfaces are available. This enables the designer to adapt the system in an early design stage, avoiding an expensive redesign at later design stages. |
| Importance | Medium |
| Assigned Tasks/WPs | T35.3 |
| Partners contributing to this description | SYN, TUBS |
| Responsibility and Reference Person | Viatcheslav Tretyakov, Jan Wagner |

| **Field Name** | **Value** |
|---|---|
| Requirement ID | WP35.3.7 |
| Title | Common robotic interface support |
| Description | The distributed control software editor shall support robotic control applications. |
| Rationale | Since robotic application are the main target for the distributed control software editor, the most frequently used interfaces in the robotic domain shall be supported. A survey on these interfaces can be found in deliverable D23.11 *Component interfaces*. |
| Importance | Medium |
| Assigned Tasks/WPs | T35.3 |
| Partners contributing to this description | SYN, TUBS |
| Responsibility and Reference Person | Viatcheslav Tretyakov, Jan Wagner |

| **Field Name** | **Value** |
|---|---|
| Requirement ID | WP35.3.8 |
| Title | Backend to the vendor |
| Description | The distributed control software editor shall enable the user to send his design to the vendor. |
| Rationale | In order to establish easy collaboration between the customer and the vendor of embedded systems, the distributed control software editor shall provide support for easy data exchange. |
| Importance | Low |
| Assigned Tasks/WPs | T35.3 |

| | |
|---|---|
| Partners contributing to this description | SYN, TUBS |
| Responsibility and Reference Person | Viatcheslav Tretyakov, Jan Wagner |

| **Field Name** | **Value** |
|---|---|
| Requirement ID | WP35.3.9 |
| Title | Interoperability |
| Description | The distributed control software editor shall be able to cooperate with other software development tools. |
| Rationale | As aforementioned every customer has already a well proven workflow established, including reusable source code. In order to foster the migration from existing software projects to the distributed control software editor an import of projects from tools like eclipse shall be considered. |
| Importance | Low |
| Assigned Tasks/WPs | T35.3 |
| Partners contributing to this description | SYN, TUBS |
| Responsibility and Reference Person | Viatcheslav Tretyakov, Jan Wagner |

### 3.4 Configuration tool support

| **Field Name** | **Value** | |
|---|---|---|
| Requirement ID | WP35.4.1 | |
| Title (goal) | Skill configuration design for an application (without user constraints) | |
| Description | Actors | User (C - Client), Configuration tool (S) |
| | Description | C enters an application description, S proposes a configuration of skills functionally implementing the application. |
| | Preconditions | C has an access permission or an account in the system.<br>C can handle (understand) the HCI specification.<br>C has suitably deep knowledge of the application. |
| | Postconditions | C receives skill configuration for his application.<br>S prepares an internal case representation of application-skill configuration (optional: case representation is stored |

| | | |
|---|---|---|
| | | in case DB). |
| | Normal Flow | 1. C fills in initial query forms about application (high level data).<br>2. S helps C to fill the forms by guiding queries with controlled natural language prompts.<br>3. S evaluates the input information and computes what next forms (and guiding information) are to be presented to C. Back to 1.<br>4. S compiles the provided information into an internal application representation and performs reasoning to find out compatible skill configurations.<br>5. S presents pre-selected configurations to C.<br>6. C selects a configuration for usage.<br>7. S outputs the case configuration (application + configuration) to C.<br>8. S stores case information in a case database. |
| | Alternative Flow | In 6. C selects more configurations.<br>In 7. C asks S to output more case configurations and provides tags for them.<br>In 7. C asks for a summary about the proposed configuration listing e.g. required resources (sensors, actuators), conditions, etc. |
| | Exceptions | C entry refused.<br>C cannot answer essential questions posed by the tool.<br>S cannot provide skill configuration due to missing/inconsistent items in its knowledge base.<br>S cannot provide skill configuration due to faulty reasoning. |
| Rationale | Basic activity between the application user and the tool. Any other application related interaction is a variant of it. | |
| Importance | High (When performed without problems, the functional skeleton of the robotic system is provided.) | |
| Assigned Tasks/WPs | T35.4/WP35 | |
| Partners contributing to this description | Tadeusz Dobrowiecki, István Majzik, András Förhécz (BME) | |
| Responsibility and Reference Person | Tadeusz Dobrowiecki (BME) | |

| Field Name | Value | |
|---|---|---|
| Requirement ID | WP35.4.2 | |
| Title (goal) | **Skill configuration design for an application with user constraints** | |
| Description | Actors | User (C - Client), Configuration Tool (S) |
| | Description | C delivers an application description with some implementation constraints (type of robot, sensors/actuators YES/NO, quality measures, etc.) and S proposes a skill configuration with qualifications. |
| | Preconditions | C has access permission or an account in the tool.<br>C can handle (understand) the HCI specification.<br>C has suitably deep knowledge of the application. |
| | Postconditions | C receives skill configuration for his application or explanation why it is not possible.<br>S prepares an internal case representation of application-skill configuration (optional: case representation is stored in case DB). |
| | Normal Flow | 1. C fills-in Query forms about application (high level data) involving constraints.<br>2. S helps C to fill the forms by guiding queries with controlled natural language prompts.<br>3. S evaluates the input information and computes what next forms (and guiding information) to present to C. Back to 1.<br>4. S compiles the provided information into an internal application representation and performs reasoning to find out compatible skill configurations.<br>5a. S presents pre-selected configurations to C, or<br>5b. S presents explanation why no skill configuration is feasible.[2]<br>6. C selects a configuration for usage.<br>7. S outputs the case configuration (application + configuration) to C.<br>8. S stores case information in a case data base. |
| | Alternative Flow | In 6. C selects more configurations.<br>In 6. (after 5b.) C asks which constraints, or application properties are critical to the skill configuring.<br>In 7. C asks S to output more case configurations and provides tags for them.<br>In 7. C asks for a summary about the proposed configura- |

[2] E.g. if application calls for covering distances and doing task underwater AND in the air, and the user stipulates to have a single robotic system, pairing underwater swimming AND flying skills, although theoretically possible, is practically not feasible.

| | | |
|---|---|---|
| | | tion listing e.g. required resources (sensors, actuators), conditions, etc. |
| | Exceptions | C entry refused.<br>C cannot answer essential questions posed by the tool.<br>S cannot provide skill configuration due to missing/inconsistent items in its knowledge base.<br>S cannot qualify selectively the explanation about configuration problems (lack of deeper knowledge).<br>S cannot provide skill configuration due to faulty reasoning. |
| Rationale | User always has in mind some particulars about the designed system. Taking them early into account further limits the search space and makes it easier to find good implementation platform. | |
| Importance | High (When performed without problems, a better-balanced functional skeleton of the robotic system is provided.) | |
| Assigned Tasks/WPs | T35.4/WP35 | |
| Partners contributing to this description | Tadeusz Dobrowiecki, István Majzik, András Förhécz (BME) | |
| Responsibility and Reference Person | Tadeusz Dobrowiecki (BME) | |

| **Field Name** | **Value** | |
|---|---|---|
| Requirement ID | WP35.4.3 | |
| Title (goal) | **Reconfiguring skill configuration for a modified application (with user constraints)** | |
| Description | Actors | User (C - Client), Configuration tool (S) |
| | Description | C used the Configuration and Skill Composer Tools earlier, re-evaluated his requirements and problem definition and now brings a modified description of the application (and/or modified constraints). S proposes a new (re)configuration of skills. |
| | Preconditions | C has an access permission or an account in the tool.<br>C can handle (understand) the HCI specification.<br>C has suitably deep knowledge of the application and the modifications to the applications.<br>C recalls the ID of the application configuration (earlier). |
| | Postconditions | C obtains skill configuration for the modified application or |

| | | |
|---|---|---|
| | | explanation why it is not possible.<br>S prepares an internal case representation of application-skill configuration (optional: case representation is stored in case DB). |
| | Normal Flow | 1. C enters the ID of the earlier configuration.<br>2. S presents the Query entry forms filled in with the case data.<br>2. S helps C to update the forms acc. to the application modifications, by guiding queries with controlled natural language prompts.<br>3. (optional) S evaluates the input information and computes what next forms (and guiding information) to present to C. Back to 2.<br>4. S compiles the input information and performs case based (incremental) reasoning to find out how the original skill configuration (or configurations) should be modified to accommodate changes in the application or constraints.<br>5a. S presents pre-selected configurations to C, or<br>5b. S presents explanation why no skill configuration is feasible.<br>6. C selects a configuration for usage.<br>7. S outputs the case configuration (application + configuration) to C.<br>8. S stores case information in a case data base. |
| | Alternative Flow | In 6. C selects more configurations.<br>In 6. (after 5b.) C asks which constraints, or application properties are critical to the skill configuring.<br>In 7. C asks S to output more case configurations and provides tags for them.<br>In 7. C asks for a summary about the proposed configuration listing e.g. required resources (sensors, actuators), conditions, etc. |
| | Exceptions | C entry refused.<br>C cannot answer essential questions posed by the tool.<br>S cannot provide skill configuration due to missing/inconsistent items in its knowledge base.<br>S cannot qualify selectively the explanation about configuration problems (lack of deeper knowledge).<br>S cannot provide skill configuration due to faulty reasoning. |
| Rationale | Specification of an application can change after configuration has been computed (e.g. as a reaction to the poor configurability). If the new (modified) application differs little from the original one, the skill configuration could be modified without complex computations. | |
| Importance | High (Modifying applications belongs to the basic system designing strategies, and as such should be supported without fears of excessive computations.) | |

| | |
|---|---|
| Assigned Tasks/WPs | T35.4/WP35 |
| Partners contributing to this description | Tadeusz Dobrowiecki, István Majzik, András Förhécz (BME) |
| Responsibility and Reference Person | Tadeusz Dobrowiecki (BME) |

| **Field Name** | **Value** | |
|---|---|---|
| Requirement ID | WP35.4.4 | |
| Title (goal) | **Knowledge engineering - introducing new / modifying old skills in the Configuration tool** | |
| Description | Actors | User (C - Knowledge Engineer), Configuration tool (S) |
| | Description | C is approached by application domain experts asking to expand/modify the scope of configured knowledge represented by the tool. C analyses the new knowledge in terms of concepts and relations used in the tool and via a special maintenance GUI introduces the new knowledge in the correct places in the knowledge base data structures. C runs suitable verification procedures to assure the consistency of the extended/modified knowledge base. |
| | Preconditions | C has an advanced access permission to the tool.<br>C can handle (understand) the application domain problem.<br>C has suitable knowledge of the tool's inner functions.<br>C has suitable knowledge of the tool maintenance GUI. |
| | Postconditions | S is equipped with extended/modified and consistent configuration knowledge (knowledge representation structures to run reasoning and additional data to maintain the dialogue with the user). |
| | Normal Flow | 1. C is ready with the analysis of the application demands (his decision: to expand, to modify).<br>2. C uses the maintenance GUI to identify the place of modification to the knowledge structures.<br>3. C uses the maintenance GUI to enter the modification and related descriptions.<br>4. C makes a choice of knowledge base maintenance algorithms to run.<br>5. C acknowledges the changes to the knowledge base and exits. |

| | | |
|---|---|---|
| | Alternative Flow | In 4. S indicates that some descriptors related to the new piece of knowledge are not filled-in with information (making future configuration reasoning impossible). Go to 3.<br>In 4. In case when modification means structural change, or deletion, S indicates that some other concepts/relations/entities are to be deleted, lacking ontological support. |
| | Exceptions | C entry refused.<br>C is professionally inapt to maintain the knowledge base (poor understanding of the domain and the S).<br>C cannot answer essential questions posed by the tool.<br>C did not answer essential questions posed by the tool.<br>C did not run verification reasoning on the modified knowledge base.<br>S cannot verify knowledge base due to faulty reasoning. |
| Rationale | Quality of service of decision support systems always depends on the proper maintenance of the knowledge bases: introducing new knowledge items expressing new technology and theory, and modifying knowledge items found faulty in the earlier system sessions. | |
| Importance | Medium (Proper knowledge-level maintenance increases the useful service lifespan and versatility of an information system.) | |
| Assigned Tasks/WPs | T35.4/WP35 | |
| Partners contributing to this description | Tadeusz Dobrowiecki, István Majzik, András Förhécz (BME) | |
| Responsibility and Reference Person | Tadeusz Dobrowiecki (BME) | |

### 3.5 Skill composer tool

| **Field Name** | **Value** | |
|---|---|---|
| Requirement ID | WP35.5.1 | |
| Title (goal) | **Knowledge engineering - introducing new / modifying old skill compositions (software/hardware modules) in the Skill Composer tool** | |
| Description | Actors | User (C - Knowledge Engineer), Skill Composer tool (S) |
| | Description | C is approached by technology experts with new/modified options for the software/hardware modules. C analyses the newly available functionalities from the point of view of |

| | | |
|---|---|---|
| | | implementing skills and via a special maintenance GUI introduces the new knowledge in the correct places in the knowledge base data structures. C runs suitable verification procedures to assure the consistency of the modified knowledge base. |
| | Preconditions | C has an advanced access permission to the tool.<br>C can handle (understand) the software/hardware component descriptions.<br>C has suitable knowledge of the tool's inner functions.<br>C has suitable knowledge of the tool maintenance GUI. |
| | Postconditions | S is equipped with extended/modified and consistent component knowledge. |
| | Normal Flow | 1. C is ready with the analysis of the newly presented software and hardware options.<br>2. C uses the maintenance GUI to identify the place of introducing the options to the tool knowledge base (KB).<br>3. C uses the maintenance GUI to enter the options and the related skill configuring descriptions.<br>4. C makes a choice of KB maintenance algorithms to run.<br>5. C acknowledges the changes to the KB and exits. |
| | Alternative Flow | In 3. If modifications include deleting items, and there is pre-prepared skill configuration data base, then the data base is recursively screened for consistency. Go to 4. |
| | Exceptions | C entry refused.<br>C is professionally inapt to maintain the knowledge base (poor understanding of the domain and the S).<br>C cannot answer essential questions posed by the tool.<br>C did not answer essential questions posed by the tool.<br>C did not run verification reasoning on the modified knowledge base.<br>S cannot verify knowledge base due to faulty reasoning. |
| Rationale | Quality of service of decision support systems always depends on the proper maintenance of the KB: introducing new knowledge items expressing new technology and theory, and modifying knowledge items found faulty in the earlier system sessions. | |
| Importance | Medium (Proper knowledge-level maintenance increases the useful service lifespan and versatility of an information system.) | |
| Assigned Tasks/WPs | T35.4/WP35 | |
| Partners contributing to this description | Tadeusz Dobrowiecki, István Majzik, András Förhécz (BME) | |

| | |
|---|---|
| Responsibility and Reference Person | István Majzik (BME) |

| **Field Name** | **Value** | |
|---|---|---|
| Requirement ID | WP35.5.2 | |
| Title (goal) | **Designing skill implementations (with user constraints)** | |
| Description | Actors | User (C - Client), Skill Composer tool(S) |
| | Description | C used the Configuration Tool earlier, evaluated his application requirements and now has a candidate skill configuration. Using Skill Composer C asks for skill implementations at different levels of detail (providing optional implementation constraints). S proposes software/hardware implementation of skills. |
| | Preconditions | C has an access permission or an account in the tool.<br>C can handle (understand) the HCI specification.<br>C has suitably deep knowledge of skill implementation with software and hardware components to maintain a dialogue with S. |
| | Postconditions | C obtains skill implementation proposal in terms of software and hardware components, or an explanation why it is not possible.<br>S prepares an internal representation of the skill implementation. |
| | Normal Flow | 1. C enters the skill implementation session.<br>2. S presents the query entry forms to be filled in with the skill data.<br>3. S helps C to update the forms, by guiding queries with controlled natural language prompts.<br>4. S compiles the input information and performs case based (incremental) reasoning to find out how the queried skill can be implemented with software or hardware components, taken into account user constraints.[3]<br>5a. S presents skill composition to C, or<br>5b. S presents explanation why no skill composition is feasible.<br>6. S stores case information in a case data base.<br>7. C asks for next skill implementation, go to 2.<br>8. C asks for a summary about the proposed implementation. |

[3] E.g. that particular function must be implemented in software, or that a particular function should be implemented in hardware, if available.

| | | |
|---|---|---|
| | Alternative Flow | In 2. C passes to S the full skill configuration description obtained from the Configuration tool. C selects one of the skills in the configuration to investigate.<br>In 3. C adds constraints on skill implementations applicable to the full skill configuration.[4]<br>In 6. (after 5b.) C asks which constraints, or application properties are critical to the skill implementation.<br>In 7. C asks S to release some of the constraints.<br>In 8. C asks for a summary about the implementation of the whole skill configuration listing required components and demanded and released constraints. |
| | Exceptions | C entry refused.<br>C cannot answer essential questions posed by the tool.<br>S cannot provide skill implementation due to missing/inconsistent items in its knowledge base.<br>S cannot qualify selectively the explanation about implementation problems (lack of deeper knowledge).<br>S cannot provide skill implementation due to faulty reasoning. |
| Rationale | A number of robotic skills (skill configurations) can be implemented with various available software and hardware components leading to a situation where a number of redundant solutions can be qualified from secondary (however important) points of view. It is important to have a decision support tool which could present alternatives and could vary alternatives adaptively according to user demands. | |
| Importance | High (With Configuration and Skill Composer tools equipped with appropriately developed KB and reasoning the designer of the robotic system can do a fast and cheap test whether his conception of the solution is sound, or how to make it sound, respectively.) | |
| Assigned Tasks/WPs | T35.4/WP35 | |
| Partners contributing to this description | Tadeusz Dobrowiecki, István Majzik, András Förhécz (BME) | |
| Responsibility and Reference Person | István Majzik (BME) | |

[4] E.g. that the number of hardware components should be minimized, or that asynchronous skills can share implementations.

# 4 References

**[1] IEEE Standard for Standard SystemC Language Reference Manual (IEEE Std 1666-2011), IEEE, 9th January 2012**