TOWARDS A NONFUNCTIONAL REQUIREMENT DRIVEN APPROACH TO THE OPTIMIZATION OF SOFTWARE SYSTEMS FOR ROBOTICS

Bernd Langpap[1,2] **and Rolf Drechsler**[1,3]

[1]*DFKI GmbH, Cyber-Physical Systems, 28359 Bremen, Germany*
[2]*Airbus GmbH, Dept. of Autonomous Systems & Robotics Engineering, 28199, Bremen, Germany*
[3]*University of Bremen, Dept. of Computer Science, 28359 Bremen, Germany*

ABSTRACT

Robotic systems are multi-domain mechatronic systems that combine the field of mechanical and electrical engineering, computer science, and control theory. Their design and development, especially the software engineering work, becomes more and more complex due to the rising challenges and tasks they need to accomplish. In order to deal with these requirements and to handle the complexity, model-based software engineering (MBSE) has proven its usefulness, because it allows the analysis of the system in early design stages, e.g. response times, communication throughput, or resource utilizations. This paper presents the road map towards an enhanced approach of how to deploy a network of software components onto an arbitrary net of execution nodes using metrics, semantics, and formalized nonfunctional requirements as optimization criteria on system level.

Key words: Software Engineering, Robotic Software Systems, Nonfunctional Requirements, Optimization, Profiling.

1. INTRODUCTION

Nowadays the development of mechatronic systems is very complex due to the increasing demand for versatile and sophisticated tasks they need to accomplish. This is especially true in the automotive and avionic domain, where this leads to software-intensive systems. Moreover, these software systems run on a hardware environment that is characterized by its heterogeneity. One example is a premium-class car, which contains probably something close to 100 millions lines of code and is only forecast to increase in the future. Nevertheless, as these system are mostly determined to be deployed in a safety- or mission-critical context, they have to fulfill stringent specifications not only related to hard real-time constraints, but also in terms of reliability, performance, robustness, or safety. All these aspects are covered by nonfunctional requirements. Moreover, running these software systems on reliable and failure-tolerant hardware usually imposes additional constraints in terms of limitation in memory space and computational power. While true for a lot of other applications, robotic systems in particular bring to bear the cross-disciplinary facet of mechatronic systems as they combine the domains of mechanical and electrical engineering as well as the field of computer science and control theory in a synergistic manner as illustrated in Figure 1. That makes the robotic domain an intriguing and fascinating field, yet it is the reason why it can easily get very complex from a system's point of view due to the amount of components and their strong interaction among the different aforementioned disciplines. The consequences on software development are significant for a number of mixed reasons, e.g. the heterogeneous nature of software components, the need for a tailored middleware, the capability to handle large number of processors and specific requirements for safety, reliability and many more [1]. Furthermore, due to the rise of multicore processors and different processor architectures, the landscape of interconnected processing entities within a robot has become highly distributed and versatile, e.g. FPGA, x86, ARM, etc. Naturally, this variety of processing entities introduces new algorithms and new opportunities for their implementation, which increases the number of possible system solutions drastically. In order to deal with the variety of so-

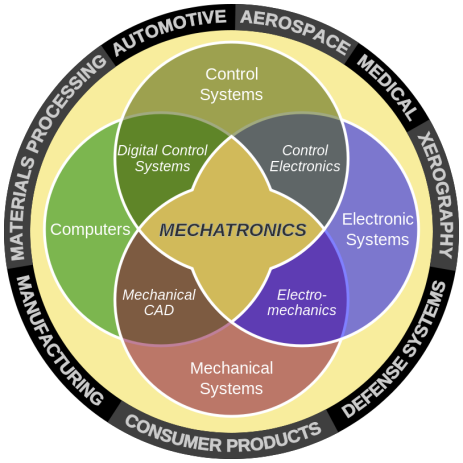

Figure 1. Domain overview for mechatronics

lutions and as a baseline for constraints, various types of requirements are used to describe the desired system and its purpose at the beginning of the development process. Whereas the main objectives of the system are covered by functional requirements (what the system should do), the system's behavior and qualities are specified through nonfunctional requirements (NFRs).

Functional vs. Nonfunctional Requirements Due to their importance here, the distinction between functional and nonfunctional requirements shall be quickly reviewed based upon a simple everyday example, a coffee mug. The main purpose of a coffee mug can be wrapped in a functional requirement stating:

REQ: The mug shall have the ability to contain tea or coffee without leaking.

Although this seems to be sufficient, the lack in specification becomes obvious by this missing nonfunctional requirement:

REQ: The mug shall contain hot liquid without heating up to more than 45 degree Celsius.

If we apply this everyday example to the robot technology and system domain, we do not only talk about functionality, e.g. the conversion of a protocol or the capability to grasp an object, but we also need to consider the system's behavior and usability. Transferring the concept of nonfunctional requirements into the domain of robotic software systems, possible examples of nonfunctional requirements can deal with the maximum workload of each processing node or the maximum allowed latency for a response to a given input. The following list provides a brief but not extensive overview over some aspects covered by nonfunctional requirements:

- Availability
- Efficiency
- Maintainability
- Performance
- Portability
- Recovery
- Response time
- Reliability
- Robust
- Safety
- Scalability
- Security

Modeling Approach The complexity during the development process itself remains and demands for new development approaches, where nowadays model-based engineering has emerged as the state of the art. By modeling the components of the system, their links and properties, certain analysis of the system can be made upfront, e.g. verification of interfaces, check for livelocks and deadlocks, and even the overall weight, power consumption or computing performance of the whole system can be derived with the support of specific tools. A lot of the system's functionality can be prototyped and tested - all that supports the overall design process in order to meet the specifications of the system at the end.

One existing approach for modeling component-based software systems and predicting their performance through software simulations is the Palladio-Component-Model (PCM) presented in [2]. It also implements the specification of "extra-functional properties" focusing on performance and reliability.
Besides the analysis of an existing and configured system model, the way a software component network is deployed onto a given hardware network has a huge impact on the overall system's performance and is usually specified through nonfunctional requirements. Their importance has also been recognized by the work of [3], which propose a constraint aggregation language on top of the declarative S-Net language [4], that imposes "extra-functional properties" on a network of asynchronous components. Especially in mission-critical and safety-critical environments, such as the automotive or aviation domain, nonfunctional requirements play a significant role and have a major impact on software deployment. A procedure for an automated software deployment generation and its evaluation is proposed by [5], which already addresses key issues, e.g. criticality level, temporal and spatial deployment constraints, etc. But is only using a limited set of nonfunctional requirements and does not consider internal states of a component for the time analysis.
As we have shown, the field of nonfunctional requirements is a research topic by itself and plays a significant role in system development. Hence, they will form the guideline for this work to an optimized software deployment in robotic software systems. Based on that, this paper will introduce a new integrated concept that includes:

- Analysis and profiling of different software component behavior with respect to metrics and system states
- Semantical meaning of the software components and their validation in a software component network
- Application of robot-specific software constraints imposed by the hardware, e.g. the frame rate of a camera should have an influence on the cycle time of the corresponding camera thread
- Integration of all information into a formal system model
- Optimization of the system with respect to formalized, nonfunctional requirements

The specification of the nonfunctional requirements needs to be incorporated in the requirements elicitation process and the following design phases. Moreover, in order to make nonfunctional requirements usable as an input to the later optimization process, their definition or representation has to be formalized to support the verification and automatic application of this information. The formalization process itself is not an objective of this work.
In summary, an enhanced optimization approach which considers not only functional, but also nonfunctional requirements for the model-based deployment of software component networks to a network of execution platforms shall be developed. In addition, the optimization or **q**uest

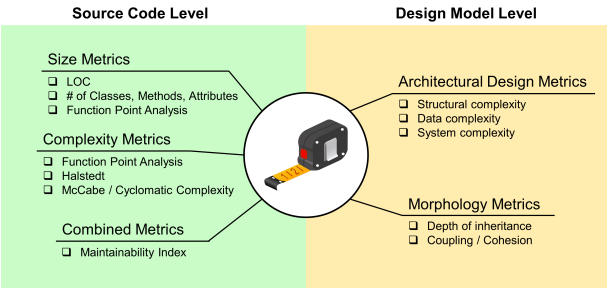

Figure 2. Overview over Metrics used on Source Code and on Model level

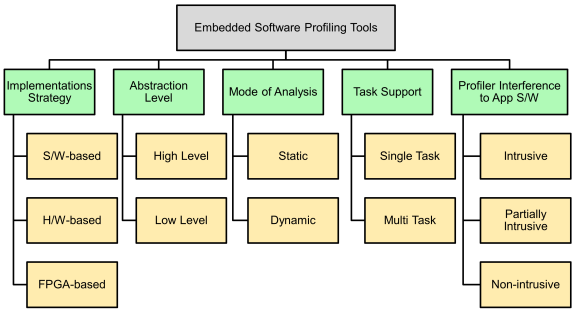

Figure 3. Overview over embedded profiling tools and methods [9]

phase will be complemented by two preliminary phases dealing with the **q**uantitative and **q**ualitative analysis of the software component network. Because of its structure and naming this approach is coined *Triple-Q*.
Following the presentation of the State of the Art in Section 2 , the idea of this approach and the strategy behind it will be presented in more detail in Section 3. Finally, Section 4 puts this work into context with ongoing development projects at the DFKI and Airbus followed by a brief summary and outlook in Section 5.

2. STATE OF THE ART

This section introduces the state of the art in the different technology domains covered by the Triple-Q approach presented in Section 3.

Metric Analysis In order to assess and compare the behavior of the different components some kind of basis for comparison has to be found. In software engineering it is common practice to use software metrics, e.g. SLOC, cyclomatic complexity, function points, etc., mainly for the purpose of effort and cost estimation[1] or quality-related aspects as shown [6]. A short overview of existing metrics is provided in Figure 2. It is also shown in the studies of [6], [7], and [8] that the quality of a conclusion based upon software metrics strongly depends on the context and that not all software metrics are applicable to every use case.

Profiling The profiling of the software is a very important step as the results of the timing and memory analysis are the foundation for later performance evaluations in the integrated software system model. The determination of the runtime costs can be done through pure analysis of the source code, software simulation, or actual execution. In the open source domain a lot of tools are already available, e.g. gprof, valgrind, visual studio profiler, vtune, just to name a few[2]. These tools not only differ in terms of the supported operating system, hardware or programming languages, but also in the way they interact with the software under test (SUT). A possible classification of profiling methods of embedded software is shown in Figure 3, each of them have its strength and weaknesses, e.g. in terms of intrusion, accuracy, generalization, etc. A trade-off at this stage has to decide which strategy is most beneficial within the Triple-Q approach as it has to be flexible and easy adaptable to various kind of software components and their environment. The WCET execution time is probably the most important value in satisfying the strict timing constraints of hard-real time systems. How much time a certain task needs for its execution is influenced by a wide range of factors, e.g. the parameterization of the input, different behavior of the environment or used middleware [10], [11]. As [12] shows, the process of creating and defining a representative processor model for an accurate simulation is very tedious and needs to be done for every processor type. Because of the versatility of processing nodes in a robotic system and the goal to respond to development changes quickly, the cycle accurate results of the simulations can be traded in for the effort of implementing a daemon process and a benchmark library for each target architecture. Based upon the results of [13] but extending it to profiling of complete components, the daemon is used to receive the SUT executable and parameters for the test run for the whole component, which will then executed by the benchmark library. Moreover, the library also contains tools to setup the runtime environment of the SUT, e.g. other tasks that require processing time or tasks that overwrite the cache of the processor. All of that would increase the execution time and hence lead to a more realistic WCET.

Semantic Languages Today everyone is using the Internet as *the* source of information. The information is presented to the user in a statically or dynamically generated form that cannot be processed by machines in order to apply some kind of reasoning, e.g. the development of advanced search engines [14]. On that account the semantic web was introduced in order to tackle the shortcomings of HTML and Co by structuring the information

[1] strongly used in the COCOMO II model for cost estimations

[2] A good overview of existing tools can be found at `https://en.wikipedia.org/wiki/List_of_performance_analysis_tools`

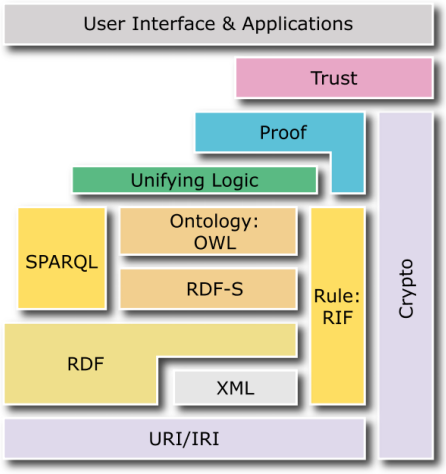

Figure 4. Building Blocks of the Semantic Web Stack [16]

semantically and make that information available for machine reading.
To allow intelligent applications and agents to interact with the semantic web, a multitude of formal and descriptive languages can be used, e.g. data and metadata query languages, ontology languages or transformation languages [15]. An overview over the different building blocks of the semantic web is presented in Figure 4. In addition to the hierarchical structure of the building blocks, some languages also have different levels of expressiveness and therefore different levels of computational properties [17]. For example, the Web Ontology Language (OWL), a well-known language for writing ontologies, comes in three different layers: OWL-Lite, OWL DL, or OWL Full. Some famous examples of ontologies would include "Friend of a Friend" (FOAF), an ontology that describes people and the links between them and other people, or "Simple Knowledge Organization System" (SKOS) a standard built upon RDF and RDFS and recommended by the W3C when representing controlled structured vocabularies, e.g. thesauri or classification schemes [18].

Nonfunctional Requirement Specification In software and systems engineering, requirements are used to describe and specify the expectations towards the final system. NFRs are particularly important for the behavior of the system and its usability. This has been recognized by [19], who presents a theoretical approach of modeling NFRs and using them to deploy software, predominantly focusing on the automotive sector. However, this approach does not incorporate the description and measurement of performance parameters. Once the NFRs have been modeled, they can be used to assess the system and evaluate Quality of Service (QoS) predictions. This topic has been investigated by [20], where they extend an already existing development process by integrating an early, model-based QoS-analysis. After presenting the development process and developer roles, the authors illustrate the interaction of these roles within the workflow and what kind of information needs to be provided to facilitate an early QoS-analysis. Although an example is presented by using the UML/SPT profile, this is a more generalized approach on how NFRs are integrated into the development process, which does not consider the technical details nor implementations about the information needed for QoS evaluations.
A more component-oriented idea residing in the robotic domain is presented by [21] with a meta-model separating individual concerns on component level from configuration questions on system level on which the non-functional properties cast into a domain specific language (DSL) come into focus. The proposed meta-model is well-designed and based upon other existing component meta-models, but it is concentrating on the software component characteristics and neglecting the additional parameters imposed by the robotic hardware.

Software Modeling Due to its significant impact on the evaluation capabilities provided through the system model and its derived tools, the way a the system is modeled and the NFRs are specified and applied to this model is a key point during development. This can range from textual descriptions over graphical models with properties and annotations up to a formal representation. As [22] have shown in the related field of man-machine-interaction, here specifically embedded Brain Reading (eBR) for the control of a robot, the usage of a formal model enabled early error detection and verification as well as the application of optimization techniques. This can be helpful in reaching the final goal of this work, which is an integrated software system model that includes all specifications and properties for each component, e.g WCET, periodicity, as well as necessary descriptions of the robotic system, e.g. hardware constraints, response times, latencies, dissimilarity of components, etc. Furthermore, this model shall then allow for the analysis of the system and the verification of the NFRs.
There already exist some frameworks and tools that support the representation of such a model. UML and SysML are the most common languages used to describe and model systems and software. Although their semi-formalism can be upgraded to a formal notation through the usage of extensions and profiles, e.g. MARTE or SPT, there exist other languages and modeling techniques that are more suitable to a comprehensive and precise description of a whole robotic system, capturing its requirements and facilitating further analysis of the modeled system. The most promising are presented in the following paragraphs.

Palladio The Palladio Component Model (PCM) presented in [23] and [2] is a key concept of the Palladio approach. It was initially developed by the University of Oldenburg and is currently maintained and developed by a group of institutes, e.g. Karlsruhe Institute of Technology, FZI Research Center for Information Technology, in collaboration with some industrial partners, e.g. ABB,

SAP, etc[3]. The main concept of the PCM is the parametric specification of component-based software architectures aligned to the different developer roles in a software development process. Furthermore, it allows to run simulations of the modeled software architecture in order to predict extra-functional properties or service attributes. It is a generic and valid approach that covers a wide area of aspects of the final system, but the evaluation of the system model lacks the consideration of systems states as well as the runtime environment.

The SmartMDSD Toolchain The SmartMDSD toolchain also resides in the domain of component-based modeling of software specific for robotic systems. It therefore utilizes a set of different DSLs and tools, which are embedded in the standard Eclipse tooling world, in order to model the complete system. The main focus is service robots, the guidance through the complex development process towards an integrated system and the ability of the robot to fulfill a diversity of services [24]. On this account, it also includes a behavior model and its variability expressed through a Variability Modeling Language (VML), which relates to an actual behavior of a robot, i.e. tasks that the robot can accomplish, rather than to the behavior of a system in terms of nonfunctional requirements. This approach to model variability on an abstract level and its resolution at runtime can also be applied to the flexibility needed in mapping software components to hardware components constraint by nonfunctional requirements.

Architecture Analysis & Design Language (AADL) Developed by an international committee of experts, AADL has been approved and published in 2004 as a SAE standard (Society of Automotive Engineers) and has been revised in 2009 for its current version AADL v2. This formal language deals with the design, specification, analysis, automated integration and code generation of real-time and safety-/performance-critical distributed systems. It therefore models both the software and hardware component architecture (here: execution platform) and their relation to each other, i.e. communication paths, allocations, allowed bindings, etc. The different components can be grouped into three categories [25]:

- Software: Process, Thread, Thread Group, Subprogram, Data
- Hardware: Processor, Memory, Device, Bus, and
- Composite: Systems, Systems of Systems

Using the default or self-defined property sets allows for the detailed specification of the various components. These properties are the baseline for different kinds of analysis, e.g. timing/latencies, schedulability, fault tolerance, etc. The different behavior of the system in the different operational states can be modeled by making the properties of the components and their connections mode-dependent. These modes are based on a state machine modeled on system level, which also supports the idea of a common state machine among the software components.

[3]More information can be found at `http://www.palladio-simulator.com`

3. TRIPLE-Q APPROACH

This section provides an overview over a model-based approach enhancing the mapping of software to hardware entities by the usage of nonfunctional requirements. For the simple reason that consists of three phases and each one of them starts with a "Q", it is henceforward referred to as the *Triple-Q* approach. By using not only functional, but also nonfunctional requirements for the optimization of a system, this approach offers several advantages, e.g. the development of more reliable and performant systems, which is especially useful for safety-critical and mission-critical applications. It is also an essential requirement for systems that are supposed to operate remote and autonomously.
The Triple-Q concept is hierarchically structured in three main phases as shown in Figure 5. Although these phases can be investigated independently and the main workflow is considered to be iterative, in this paper the phases are put in an aligned context, in which they provide their results as an input to the subsequent phase. This supports a clearer understanding while presenting the idea of Triple-Q. The first phase Quantification deals with

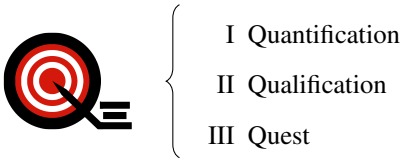

Figure 5. The Triple-Q Phases

the determination of communication and runtime costs by means of analysis tools and software simulation and their correlation to internal behavior. The second step Qualification uses the benefits of semantic annotations from the software components to not only check the syntactic correctness of the linked software components, but also the correct semantic connection. Furthermore, this information can also be used to propose alternative network configurations to achieve the same goal as the original network. The final step named Quest deals with the application of formalized nonfunctional requirements (NFRs) as constraints to the optimization process. Each phase of the Triple-Q approach is divided into three sub-steps and will be explained in the following sections.

3.1. Phase I: Quantification

The main purpose of the Quantification phase is the determination of costs in terms of resources, i.e. time needed for execution and communication, memory usage, etc.,

and the linkage of those costs to internal structure of each software component, e.g. metrics or internal structure. Therefore, the Quantification phase is divided into three sub-phases:

I.a Automatic Metric Analysis

I.b Profiling

I.c Derive Implementation Strategies

The first step is the analysis of available metrics of software components for both their source code and model representation, in order to determine usable measures for the classification and characterization of software entities. This process also involves the combination of existing metrics or the creation of new metrics. The goal of these metrics is to provide a baseline for the comparison of software components not only in terms of their runtime characteristics, but also in terms of their internal structure and properties, e.g. number of memory operations, algorithmic operations, or behavior defined through states. This facilitates a more detailed assessment of the different software components as well as a more refined distinction of the runtime behavior of each component. In a second step the provided software components are profiled, i.e. their runtime costs are determined. Although memory consumption can be of concern in small embedded systems, a major focus is put on the "runtime" itself, more precisely the worst case execution time (WCET), which is an important factor in scheduling components on a network of execution nodes. This data of WCETs can already be normalized and be used as an input into a basic mapping method. Once the profiling data of a component is available, the retrieved results can then be used to populate a database, in which the metrics of each software component is correlated with their performance measurements related to the various hardware execution platforms and its properties.
In a final step and depending on the size and diversity of the database's entries, this data can then also be used to derive implementation strategies, i.e. doing performance trade-offs for different components or making predictions regarding an implementation on a different hardware architecture.

3.2. Phase II: Qualification

Whereas the first phase is dealing with the inspection and profiling of single software components, the second phase of the Triple-Q approach, the Qualification Phase, focuses on networks of software components and ensures the syntactical as well as the semantic correctness of the software component model. This phase again is split into three sub-phases:

II.a Vocabulary/Grammar Definition

II.b Network Construction

II.c Semantic Analysis

In a first step, a set of entities, their relations, and their properties are defined, i.e. a vocabulary and grammar is constructed. In order to do so, a description language is needed, which can be found in the domain of the semantic web. In this case a web ontology language is applied, which provides the ability to formally describe the conceptual schema in this functionality-based domain model ("functionality-based" emphasizes the focus on to the purpose of a software component). As pointed out in the state of the art in Section 2, there is a huge variability in the level of expressiveness among the different languages of the semantic web. Here, the use of the ontology language will be restricted to a simple and lightweight layer as this dictionary shall only support the categorization of software components in terms of their meaning and sets the baseline for the following step.
Of course a semantic description cannot only be applied to a single software component. The second step allows for the lateral and transversal combination of multiple software components forming a network of software components to achieve a common goal, e.g. pose estimation. This network of software components can then again act as a new "virtual" component with its own semantic description.
Once the software network is configured and ready for deployment, it can be checked in a third step not only for syntax, but also for semantic correctness. Moreover, having the library of semantically tagged components in the background, alternative compositions of software modules can be recommended to reach the same goal with a better performance, i.e. the solution space of possible network configurations is widened.

3.3. Phase III: Quest

The Quest Phase represents the final phase of the Triple-Q approach and uses the results of the other first two phases as an input to the optimization process targeting the optimal satisfaction of the nonfunctional requirements. By achieving this goal, the Quest phase also follows the three-steps pattern:

III.a Constraint Modeling

III.b Initial Deployment

III.c Optimization

First, the hardware constraints of the system, i.e. robot, will be analyzed. This is not related to the characteristics of the execution hardware. On the contrary, here it will focus on the limitations of the remaining hardware components typical for a robotic system, e.g. maximum speed and acceleration of the joints, highest possible acquisition rates of sensors, etc. In addition to these hardware-imposed constraints, the specified NFRs are also formalized and transferred into a constraint model.
The next step involves the creation of a system model that consists of two parts. A hardware part, which provides a sufficient representation of the underlying execution hardware, and a software part, which reflects all soft-

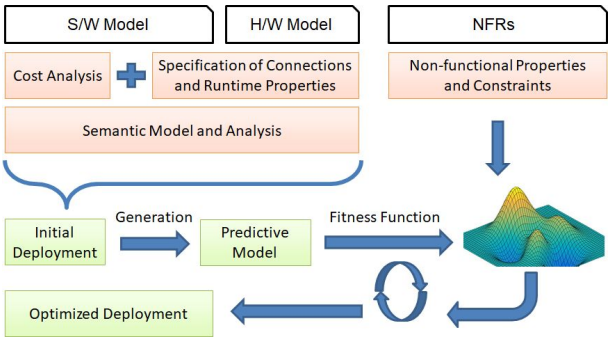

Figure 6. Basic Workflow of the Triple-Q Approach

ware components, their connections among each other, as well as hardware connections, e.g. for driver related software. Furthermore, the system model also provides the ability to specify latencies to account for communication overhead between the components. Once all this information are available, the software entities can be initially deployed, that is every software component will be assigned to an execution unit. Having the system model and a first deployment allows for the evaluation of the complete software system, i.e. the software components will be scheduled and performance parameter can be retrieved and analyzed. This provides the baseline for the final step, the optimization.
Similar to regular optimization tasks, a fitness or target function is used to optimize the configuration of a semantic and syntactic correct system in terms of costs and nonfunctional requirements. In order to facilitate the understanding, this workflow is presented in Figure 6.

4. TRIPLE-Q IN CONTEXT

Although the Triple-Q framework is self-standing and aimed to be versatile, meaning it can be applied to a wide range of different scenarios, the development of the tools and the testing is done as part of the development work of the D-Rock[4] project at the DFKI and partly at the Space Robotics and Automation Laboratory of Airbus, both at its Bremen locations. The D-Rock project at the DFKI deals with the development of complex robotic systems in an efficient, cost-effective and robust way. It answers to the challenge of increasing complexity and required flexibility of robotic systems, especially in the software domain, that are the consequence of even more sophisticated and demanding operations in the work and living environments. For that reason, the project is dedicated to the development of tools and methods that support the user in the creation of software for robotic systems.
Beyond the need for a more flexible and guided way of software development, the usage of requirements for software specification and their verification at the final product is good practice in software engineering. Furthermore, the need for a predictable and fail-safe (or failure-tolerant) software is especially high in safety- or mission-critical environments, in which an error can result in human or huge financial loss. A typical example for this kind of software is flight software in the aeronautical or space domain of Airbus, where the software development process is regulated by many standards. This is a very beneficial and structured environment that emphasizes the importance of functional and nonfunctional requirements, software simulation, testing and verification in order to prove that the requirements have been met.

[4] http://robotik.dfki-bremen.de/en/research/projects/d-rock.html

5. NEXT DEVELOPMENT STEPS

Once the Triple-Q concept is implemented and tested, it should provide a harmonized approach for analyzing and profiling existing software modules, the assignment of semantical meaning to software components and their validation in a software component network, and the optimization of the software deployment to a robotic system with respect to hardware constraints and formalized, nonfunctional requirements. In order to verify the matching of the system predictions to real world characteristics, the usability and correctness of the Triple-Q approach is verified in two use cases, one end-to-end demonstration in a testbed developed in the DRock-project as well as for a robotic control system on space hardware at Airbus.
The next steps entail the definition of a preliminary list of nonfunctional requirements. Although they are needed at the end of Triple-Q, this list acts as a golden thread to guide through the development and to tie the results together. Having this list ready enables the entry into the first phase. Here, a set of software metrics need to be identified that are suitable for performance predictions on different hardware architectures. This is the baseline to compare software components with each other. Given the source code or model of an existing software component, these software metrics should then automatically be retrieved by a tool.
In a following step the different profiling strategies (see section 2 “Profiling”) need to be reviewed. As they differ in their characteristics, e.g. intrusion or accuracy, a trade-off defines what strategy is used for an automatic cost determination for a given software component. This represents a major milestone of the first phase of the Triple-Q concept.

6. ACKNOWLEDGEMENT

The authors would like to thank the D-Rock team and all supporting staff at DFKI Robotics Innovation Center and Cyber-Physical Systems, Bremen and Airbus DS GmbH, Bremen. The work presented is part of the D-Rock project, which is commissioned by the German Space Agency with funds of the Federal Ministry of Education and Research in accordance with a resolution of the German Parliament (No. 01IW15001).

REFERENCES

[1] Alexander Pretschner, Manfred Broy, Ingolf H Krüger, and Thomas Stauner. Software Engineering for Automotive Systems : A Roadmap. *Future of Software Engineering*, 2007.

[2] Steffen Becker, Heiko Koziolek, and Ralf Reussner. Model-Based Performance Prediction with the Palladio Component Model. *Proceeding WOSP '07 Proceedings of the 6th international workshop on Software and performance*, pages 54–65, 2007.

[3] Raimund Kirner, Frank Penczek, and Alex Shafarenko. Compilers must speak properties, not just code: CAL: constraint aggregation language for declarative component-coordination. In *DAMP '12 Proceedings of the 7th workshop on Declarative aspects and applications of multicore programming*, pages 51–54, 2012.

[4] Clemens Grelck, Frank Penczek, Jukka Julku, Haoxuan Cai, Clemens Grelck, Sven-bodo Scholz, Alex Shafarenko, and Bert Gijsbers. S-Net Language Report. (August), 2013.

[5] Robert Hilbrich. *Platzierung von Softwarekomponenten auf Mehrkernprozessoren*. Springer Vieweg, 1 edition, 2015.

[6] Y Jiang, B Cuki, T Menzies, and N Bartlow. Comparing design and code metrics for software quality prediction. *PROMISE '08: Proceedings of the 4th international workshop on Predictor models in software engineering*, pages 11–18, 2008.

[7] Kaushal Bhatt, Vinit Tarey, and Pushpraj Patel. Analysis Of Source Lines Of Code (SLOC) Metric. *International Journal of Emerging Technology and Advanced Engineering*, 2(5):3–7, 2012.

[8] Rachel Harrison, Steve J. Counsell, and Reuben V. Nithi. An evaluation of the MOOD set of object-oriented software metrics. *IEEE Transactions on Software Engineering*, 24(6):491–496, 1998.

[9] Rajendra Patel. A Survey of Embedded Software Profiling Methodologies. *International Journal of Embedded Systems and Applications*, 1(2):19–40, 2011.

[10] Reinhard Wilhelm, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, Per Stenström, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, and Reinhold Heckmann. The worst-case execution-time problem—overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems*, 7(3):1–53, 2008.

[11] Sergey Zhuravlev, Sergey Blagodurov, and Alexandra Fedorova. Addressing Shared Resource Contention in Multicore Processors via Scheduling. *Asplos'10*, page 129, 2010.

[12] R.H. Massoud, M.B. Abdelhalim, and M. Allam. Towards mature temporal accuracy assessment of processors models and simulators for real-time systems development. *Proceedings of the 2011 International Symposium on Performance Evaluation of Computer and Telecommunication Systems, SPECTS 2011*, pages 235–240, 2011.

[13] Julian Godesa. Vorhersage der Ausführungszeit von Anwendungen auf Multicore-Architekturen basierend auf einer empirischen Analyse von Einflussfaktoren. 2013.

[14] N. Yadagiri and P. Ramesh. Semantic Web and the Libraries: An Overview. *International Journal of Library Science^{TM}*, 7(1):80–94, 2013.

[15] Jakob Henriksson and Uwe Aßmann. Chapter 5 Component Models for Semantic Web Languages. *Techniques*, pages 233–275, 2009.

[16] Steve Bratt. Semantic web and other W3C technologies to watch. *Talks at W3C, January*, 2007(January), 2007.

[17] Grigoris Antoniou, Enrico Franconi, and Frank Van Harmelen. Introduction to Semantic Web Ontology Languages. *Reasoning Web*, pages 1–21, 2005.

[18] D. Grant Campbell. The Birth of the New Web: A Foucauldian Reading of the Semantic Web. *Cataloging & Classification Quarterly*, 43(3/4):9–20, 2007.

[19] Michael Dinkel and Uwe Baumgarten. Modeling Nonfunctional Requirements: A Basis for dynamic Systems Management. *Seas 2005 (@Icse 2005)*, pages 1–8, 2005.

[20] Heiko Koziolek and Jens Happe. A QoS Driven Development Process Model for Component-Based Software Systems. *Proceedings of the 9th International Symposium on Component Based Software Engineering (CBSE 2006)*, pages 336–343, 2006.

[21] Alex Lotz, Arne Hamann, Ingo Lütkebohle, Dennis Stampfer, Matthias Lutz, and Christian Schlegel. Modeling Non-Functional Application Domain Constraints for Component-Based Robotics Software Systems. 2016.

[22] Elsa Andrea Kirchner and Rolf Drechsler. A formal model for embedded brain reading. *Industrial Robot: An International Journal*, 40(6):530–540, 2013.

[23] Ralf Reussner, Steffen Becker, Jens Happe, Heiko Koziolek, Klaus Krogmann, and Michael Kuperberg. The Palladio component model. (March), 2007.

[24] Dennis Stampfer. The SmartMDSD Toolchain : An Integrated MDSD Workflow and Integrated Development Environment (IDE) for Robotics Software. *Journal of Software Engineering for Robotics*, 1(July):3–19, 2016.

[25] P. H. Feiler and D. P. Gluch. The architecture analysis & design language (AADL): An introduction. (February), 2012.