TU WIEN Informatics

Worst-Case Execution Time Analysis of OPC UA PubSub on a Time-Predictable Processor

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Informatik

eingereicht von

Andreas Kirchberger, BSc.
Matrikelnummer 0626507

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dr. Wolfgang Kastner
Mitwirkung: Dr.techn. Thomas Frühwirth
Dipl.-Ing. Patrick Heinrich Denzler

Wien, 22. Februar 2022

Andreas Kirchberger | Wolfgang Kastner

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

TU WIEN Informatics

Worst-Case Execution Time Analysis of OPC UA PubSub on a Time-Predictable Processor

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Computer Engineering

by

Andreas Kirchberger, BSc.
Registration Number 0626507

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dr. Wolfgang Kastner
Assistance: Dr.techn. Thomas Frühwirth
Dipl.-Ing. Patrick Heinrich Denzler

Vienna, 22nd February, 2022

Andreas Kirchberger	Wolfgang Kastner

Erklärung zur Verfassung der Arbeit

Andreas Kirchberger, BSc.

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 22. Februar 2022

Andreas Kirchberger

Danksagung

Ich möchte mich an dieser Stelle zuerst bei Wolfgang Kastner dafür bedanken, dass ich nach meiner Bachelorarbeit in diesem Themengebiet die Möglichkeit erhalten habe, im Zuge meiner Masterarbeit mein Wissen in diesem Bereich weiter zu vertiefen. Auch möchte ich mich dafür bedanken, dass ich im Zuge dieser Masterarbeit die Möglichkeit erhalten habe, an zwei Publikationen mitwirken zu können und dadurch wertvolle Erfahrungen in diesem spannenden Prozess gesammelt habe.

Ein großer Dank gilt auch Martin Schöberl von der Technischen Universität Dänemark (DTU) für die Unterstützung bei Fragen zur verwendeten Entwicklungsumgebung und Eleftherios Kyriakakis für die Unterstützung bei der Lösung eines speziellen Problems mit der Hardware.

Besonders bedanken möchte ich mich auch bei meinen zwei Betreuern Thomas Frühwirth und Patrick Denzler für die großartige Unterstützung, Motivation und fachliche Expertise.

Weiter möchte ich auch meinen Eltern und Geschwistern danken, die mich immer in meinen Plänen unterstützt haben, mir Halt gegeben haben und immer ein offenes Ohr für mich hatten. Ein besonderer Dank gebührt meiner Freundin, die mich immer unterstützt und motiviert hat, trotz ihrer eigenen Doppelbelastung mit Job und Masterstudium.

Kurzfassung

Die industrielle Automatisierungspyramide erfährt derzeit einen Wandel hin zum industriellen Internet der Dinge, mit einem höheren Integrationsgrad und einem nahtlosen Kommunikationsfluss zwischen allen Ebenen. Die Zahl der Systeme mit Echtzeitanforderungen an die Maschine-zu-Maschine-Kommunikation nimmt ständig zu. Insbesondere im Bereich der Fertigung steigt die Dichte der Systeme mit Echtzeitanforderungen stark an. Die aufkommenden Technologien Time-Sensitive Networking (TSN) und OPC Unified Architecture (OPC UA) PubSub werden wahrscheinlich einen großen Beitrag zur Lösung neuer Herausforderungen leisten, die sich in diesem Anwendungsbereich stellen. TSN wurde entwickelt, um eine Echtzeitkommunikation auch unter hoher Netzwerklast zu gewährleisten, OPC UA hingegen wurde nicht speziell für Echtzeitanforderungen entwickelt, so dass das zeitliche Verhalten des OPC UA-Stacks unbekannt bleibt.

In dieser Arbeit wird der Weg von der Erstellung allgemeiner Worst Case Execution Time (WCET) Code-Transformationsregeln, zur Anwendung dieser Regeln auf einen bestehenden Open-Source OPC UA PubSub-Stack, bis hin zur statischen WCET-Ausführungszeitanalyse vorgestellt. Die WCET-Analyse wird auf der Grundlage des open62541 OPC UA PubSub-Stack jeweils für ein OPC UA-Publisher- und ein OPC UA-Subscriber-Beispiel mit Basisfunktionalität für den Datenaustausch durchgeführt. Für die Evaluierung werden zwei T-CREST-Plattformen, die direkt über Ethernet miteinander verbunden sind, als zeitlich vorhersehbarer Plattform verwendet, um die End-to-End-Latenz zu bewerten. Für die statische Analyse von WCET wird das zur T-CREST-Plattform gehörende Toolset verwendet. Die dynamische WCET-Analyse durch Messung wird mit einem Logikanalysator durchgeführt und ergänzt die statischen WCET Zeitmessergebnisse um die Teile der End-to-End-Latenz, die nicht statisch analysiert werden können.

Abstract

The industrial automation pyramid is currently experiencing a transformation towards the Industrial Internet of Things (IIoT), with a higher level of integration and a seamless communication flow among all levels. The number of systems with real-time requirements for machine-to-machine communication is constantly increasing. Especially towards the shop-floor, the density of systems with real-time requirements is increasing. The emerging Time-Sensitive Networking (TSN) and OPC Unified Architecture (OPC UA) PubSub technologies are likely to contribute greatly to solving new challenges that arise in this application area. TSN was developed to ensure real-time communication, even under high network load, OPC UA on the other hand wasn't especially developed to meet real-time requirements. Thus, timely behaviour of the OPC UA stack remains unknown.

This thesis presents a path from the creation of general Worst Case Execution Time (WCET) code transformation rules, over the application of these rules to an existing open-source OPC UA PubSub stack, to static WCET execution time analysis. The WCET analysis process is applied to the open62541 OPC UA PubSub stack for an OPC UA publisher- and an OPC UA subscriber-example with basic functionality for data exchange. For the evaluation, two T-CREST platforms, that are directly connected over Ethernet with each other, are used as a time-predictable platform to evaluate the end-to-end latency. For static WCET analysis, the toolset that is part of the T-CREST platform is utilized. The dynamic WCET analysis, by means of measurement, is performed with a logic analyzer and complements the static WCET timing results with the parts of the end-to-end latency that cannot be statically analyzed.

Contents

CHAPTER 1

Introduction

In the past, industrial automation systems were formed in a pyramid-like architecture with complex Information Technology (IT) and Operational Technology (OT) components, which consist of software and hardware [1]. Thereby, IT and OT covered fundamentally different tasks and the differing requirements resulted in the development of specialized Information and Communication Technology (ICT) for each domain. Modern communication technologies, however, aim at supporting both IT and OT requirements, a trend known as IT/OT convergence or Industrial Internet of Things (IIoT) transformation. Both, the traditional pyramid-like and the more modern IIoT architecture, are illustrated in Figure 1.1 and briefly discussed in the following.

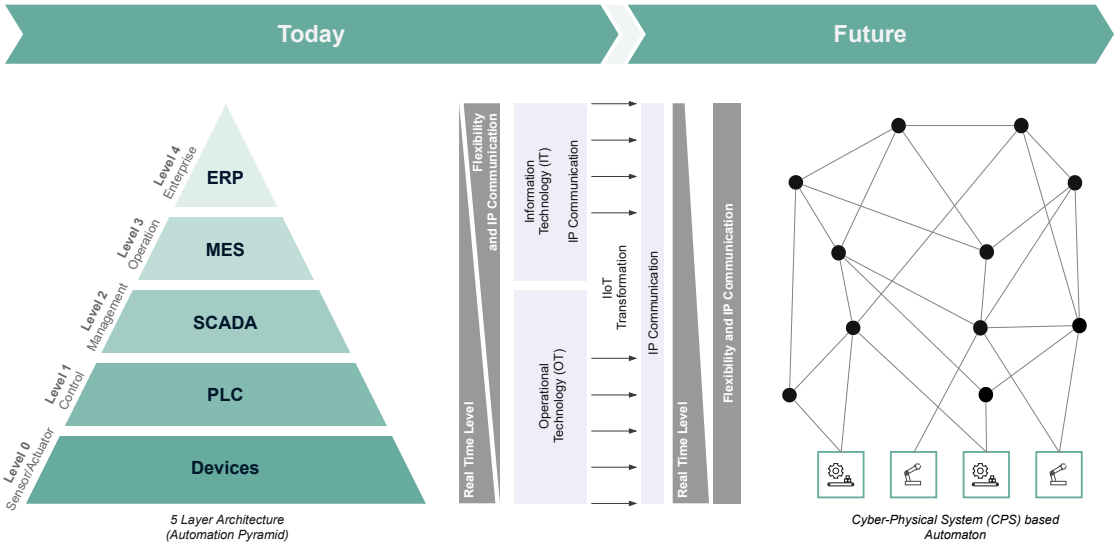

Figure 1.1: Automation pyramid vs. IIoT architecture (adapted from [2], [1], [3])

Devices and Programmable Logic Controllers (PLCs) form Level 0 and Level 1 of the automation pyramid. Sensors and actuators of the lowest level are typically connected with the overlying control level through industrial communication systems like EtherCat or Profibus [4]. Time-critical control loops are mainly found in OT. Therefore, communication and processing need to be timely predictable to meet defined deadlines. Level 2 of the automation pyramid is the Supervisory Control and Data Acquisition (SCADA) level, which manages the two layers below, performs control and monitor tasks, and provides a Human Machine Interface (HMI). This last level of OT represents the connection point to the IT with its Manufacturing Execution System (MES) and Enterprise Resource Planning (ERP), which are part of the enterprise systems. Enterprise systems are typically formed by servers and desktop PCs with off-the-shelf components that communicate over the standard Internet Protocol (IP). The technology differences between IT and OT lead to a gap between these two.

Today's industrial automation systems are facing a transformation from the traditional 5-layer automation pyramid towards Industry 4.0 with its IIoT architecture [5]. IIoT utilizes a more uniform IP-based communication scheme with fewer gateways based on the Internet of Things (IoT) paradigm with a flat architecture of interconnected devices. Emerging technologies like fog computing providing computing power near to the devices and Time-Sensitive Networking (TSN) enabling deterministic networking will help to close the IT/OT gap [6]. The still relatively new OPC Unified Architecture (OPC UA) PubSub extension for publish-subscribe communication in combination with TSN has the possibility to pave the way towards flexible, real-time, Machine-to-Machine (M2M) communication [5].

1.1 Motivation and problem statement

Modern industrial automation systems include a large variety of devices and systems, ranging from small field devices over complex machinery to management systems like MES and ERP. Automating such systems requires not only real-time data transmission but also sophisticated M2M communication protocols. This is where OPC UA comes into play, which combines data transmission and information modeling in a single communication standard. In particular the publish-subscribe communication scheme introduced by the OPC UA PubSub Specification was identified as a viable approach to add real-time capabilities to OPC UA [7].

Furthermore, major requirement of OT is real-time data transmission for a variety of applications like closed-loop process control and safety systems. Therefore, a communication network that supports IT and OT tasks needs to ensure that low priority IT traffic does not compromise the timeliness of real-time traffic in the OT domain under any circumstances. TSN was specially designed for this purpose and supports mixed-criticality traffic via a shared Ethernet network.

TSN is limited to the data link layer 2 of the OSI model and is aimed to replace it. OPC UA extends from the session layer 5 to the application layer 7 in the OSI model. Figure 1.2 presents the layers in the OSI model that are covered by TSN and OPC UA, respectively.

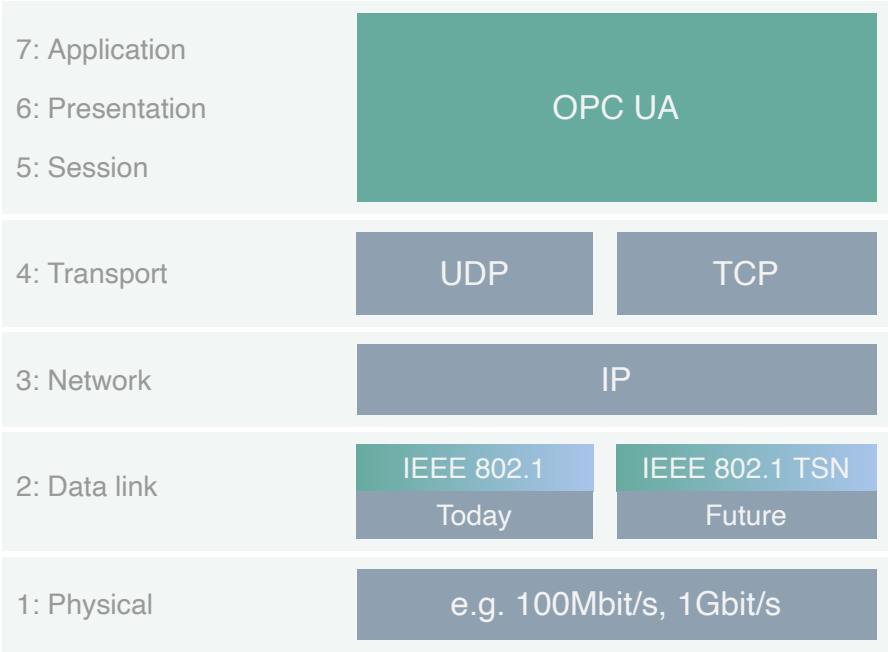

Figure 1.2: TSN and OPC UA in the OSI model (adapted from [8])

In contrast to OPC UA PubSub, TSN was specially designed to meet timing constraints. Therefore mainly the Worst Case Execution Time (WCET) analysis of the OPC UA stack, especially the PubSub part, is a remaining challenge towards end-to-end real-time communication. In other domains like automotive or avionic systems, the WCET is a timing measure that defines a program's execution time characteristic [9]. The most common methods for determining this execution time are static program analysis by means of calculation or dynamic program analysis by means of measurement.

This thesis presents a static WCET analysis of the OPC UA PubSub stack using the open-source open62541 source code for software and the T-CREST platform with its toolset and Patmos processor for a time-predictable hardware platform for the implementation. For verification, the static static WCET analysis results are also dynamically analyzed by measurement to verify the plausibility of the results. This WCET measurement is done on a complete end-to-end system with a publisher and a subscriber by using two time-predictable platforms. Applying static program analysis to existing software, which was not designed for WCET analysis, is a particular challenge and a systematic process to address this task is an additional contribution of this thesis.

Overall, this work presents the first step towards a time-predictable OPC UA PubSub implementation and forms the basis for further research and a possible communication over TSN. Figure 1.3 illustrates how the WCET-analyzed OPC UA publisher and subscriber can be connect via a TSN network to achieve real-time end-to-end data transmission in modern automation systems.

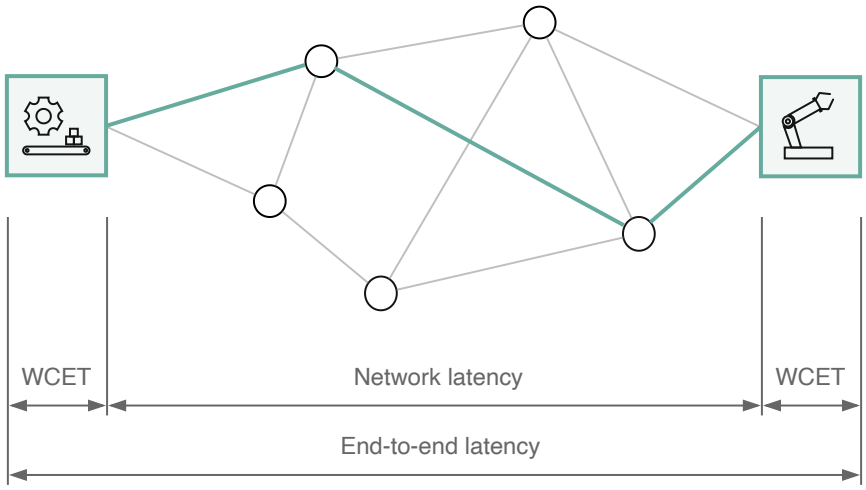

Figure 1.3: End-to-end latency

1.2 Delimitations

The combination of the T-CREST time-predictable platform, the open62541 OPC UA PubSub stack, and TSN as a communication network in combination with WCET analysis is a very complex task, which cannot be solved in its entirety in this thesis. Therefore, some limitations of the work are listed in the following.

The open-source OPC UA stack open62541 was selected for the implementation of the OPC UA PubSub subscriber and publisher. This stack implements a rich set of data types that are defined in the OPC UA specification. This great selection of data types and the extensive use of recursive functions lead to difficulties in the calculation of the WCET. Therefore some functionality, primarily data types and special functions (e.g. JSON encoding, data encryption, or keyframes and delta frames) that typically are not used in real-time systems, were disabled in the stack. Additionally, the number of data elements that can be transmitted in a single OPC UA PubSub message was limited to get a tight WCET bound.

The time-predictable platform T-CREST was chosen for the implementation, mainly because of its included toolset for WCET analysis and because most of the T-CREST technology is available as open-source software [10]. It needs to be mentioned that the proposed concept was specifically developed for this platform. However, compatibility

with standard computing platforms has been preserved and it is still possible to run the same code also on standard Linux systems, however without real-time functionality.

TSN enables deterministic real-time communication with a bounded maximum latency over Ethernet. To guarantee a bounded end-to-end latency in a network setup with many devices, a TSN switch is necessary. As TSN is not yet implemented in the T-CREST project, the development platforms are instead connected through a point-to-point Ethernet connection. This limitation ensures a uniform latency for the communication as no other devices and services are communicating on the network in the evaluation setup.

1.3 Structure of the work

The remaining chapters of this thesis are briefly summarized in the following.

- **Chapter 2 – Technical background and related work:** This chapter gives a brief introduction into the technical background of the utilized technologies, platforms, standards, and definitions. It covers the definition of real-time, WCET analysis, the T-CREST project, the history of OPC UA, the relatively new OPC UA PubSub specification, and scientific work related to this thesis.
- **Chapter 3 – Methodological approach:** The applied methodological approach entitled „design and creation“ and the interrelationship are presented in this chapter. Also, the timeline of this thesis is presented at this point.
- **Chapter 4 – Code transformation for WCET analysis:** This chapter presents the code transformation process from porting applications to new platforms, over call graphs, and control flow graphs, towards code transformation and annotation rules. Finally, a small example application is presented, on which the previously defined annotation and transformation rules are applied.
- **Chapter 5 – WCET analysis of OPC UA PubSub:** In this chapter, the previously defined code transformation process is applied to the OPC UA PubSub stack and results of the WCET analysis are presented.
- **Chapter 6 – Evaluation:** In this chapter, the evaluation procedure, the utilized toolset, and the interrelationship within the toolchain are briefly described at the beginning. In this part, the various intermediate file formats and the flash process are also stated. Also, the utilized hardware, the measurement tools, and the measurement approach are presented here. The evaluation results and the end-to-end latency analysis are presented at the end of this chapter.
- **Chapter 7 – Discussion:** The difficulties that arose during the implementation and evaluation are discussed at the beginning of this chapter. Furthermore, the measured and calculated WCET results are discussed at this point.

- **Chapter 8 – Conclusion:** This chapter covers the findings during the WCET analysis of a OPC UA PubSub stack in this thesis. Future research questions that arose during the implementation and evaluation process are covered in the last part of this chapter and lay the ground for further research efforts.

CHAPTER 2

Technical background and related work

The first section of this chapter presents the definition of real-time and its classification into soft, firm, and hard real-time. The next section introduces the fundamentals of the T-CREST Specific Targeted Research Project (STREP) project of the Technical University of Denmark, the Patmos processor, and the WCET analysis toolset. The following section gives a review of the history of OPC UA and an introduction into the relatively new OPC UA PubSub specifications. In the last section of this chapter, scientific work related to this thesis is presented.

2.1 Definition of realtime

In a *real-time* system not only the computational correctness of the results is of interest, but the time when the result is produced is also important [11]. A constraint that is important for real-time systems is the deadline. It describes the maximum time until a result must be produced. The consequence of a missed deadline classifies real-time systems into three categories [11], [12], [13]:

- **Soft real-time:** In a soft real-time system, there is no hard deadline, after which the result gets useless. It can still have utility for the system, but it decreases over time. This is illustrated in Figure 2.1 (a), which shows that the value of the result decreases towards zero over time, after passing the deadline.
- **Firm real-time:** In a firm real-time system, occasionally missing a deadline causes no economical damage or human harm, but the result has no utility for the system

thereafter. This sudden loss of value can be seen in Figure 2.1 (b). The value of the data immediately drops to zero once the deadline has passed.

- **Hard real-time:** In a hard real-time system, meeting the deadline is of utmost importance. Missing a deadline results in a failure. Results that arrive after the deadline, even when they are computationally correct, have zero value. Missing a deadline not only means a complete loss of the value, but it can also have a negative effect, which can result in economical damage or human harm (cf. Figure 2.1 (c)).

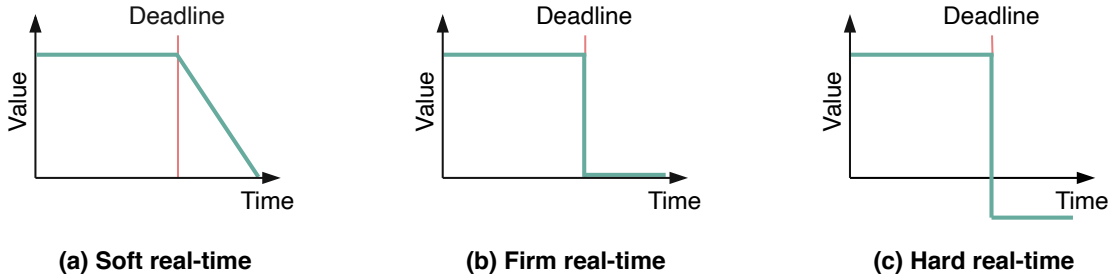

Figure 2.1: Comparison of soft, firm and hard real-time (adapted from [14])

For real-time systems, it is important that the execution time of tasks is known a priori. The upper bound of this execution time is called Worst Case Execution Time (WCET), and is typically defined for a program or program part on a specific platform. This characteristic determines if a real-time system can meet its deadline under any circumstances for which the system has been designed. Figure 2.2 shows the execution times of an example program with different input data. The WCET defines the upper bound for these execution times. One can also see that the calculated WCET is smaller than the set deadline. Because of the different input data, the execution time is subject to a certain amount of jitter. However, the execution time for any given in put data is bound by the calculated WCET.

2.2 WCET analysis

Timing analysis is a crucial part of real-time systems design because strict timing constraints need to be satisfied to meet the requirements of the controlled system [9]. The main target is to determine the timing behavior of systems or programs and to find upper timing bounds. The execution time of a program typically varies with the input data and other effects determined by the environment like previously executed programs or the contents of caches of the processor. Figure 2.3 shows the distribution of the execution times for different program runs. Two values are important in this context, the Best Case Execution Time (BCET) forms the shortest execution time and the Worst Case Execution Time (WCET) the longest execution time. In general, these values can be determined either via dynamic or via static timing analysis techniques.

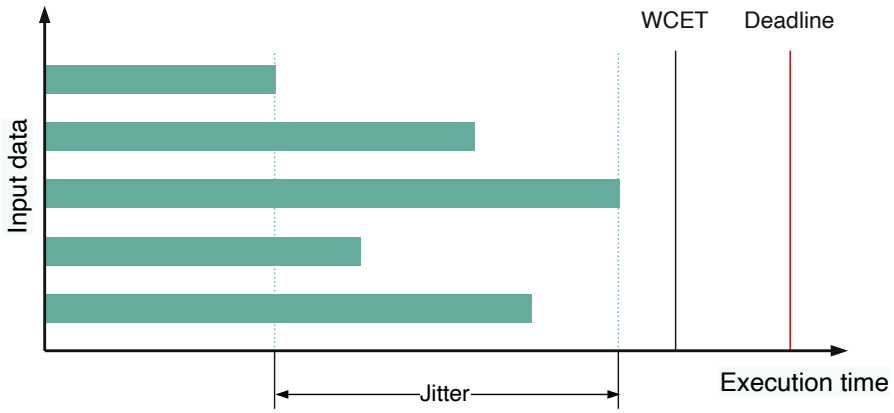

Figure 2.2: Execution times for different input data

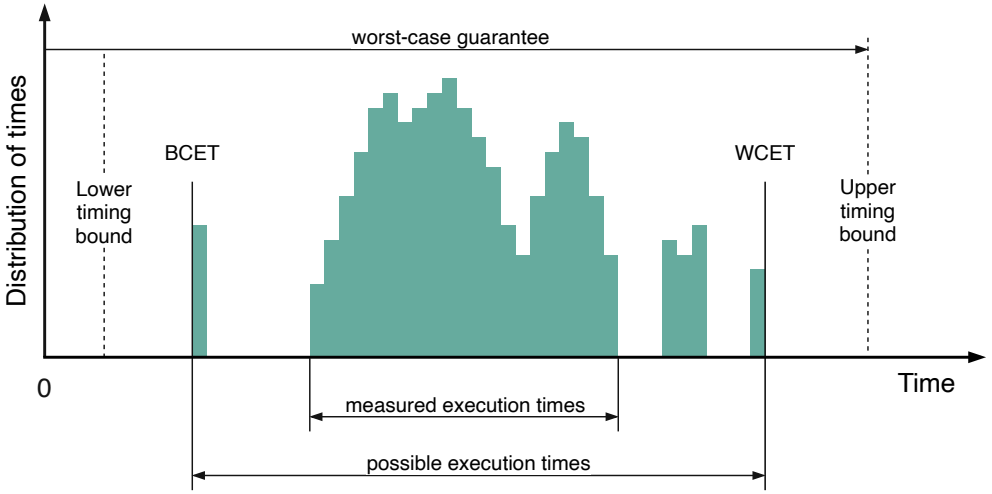

Figure 2.3: Basic notions of timing analysis (adapted from [9])

A commonly used timing analysis approach in the industry is end-to-end time measurement with varying input values and multiple test runs, also called dynamic analysis [15]. Timing measurements are typically performed by logic analyzers, oscilloscopes, in-circuit emulators, or platform timers. This method can give an average timing range with a rough determination of the WCET. However, complex programs with many execution paths can lead to an underestimated WCET, because every run follows a single program path without any guarantee to cover the longest path. Therefore, a safety margin needs to be added to the highest measured execution time to ensure that the WCET is not set too low. A drawback of the dynamic WCET analysis method is that adding a safety margin to the WCET can lead to scheduling problems caused by over- or under-dimensioned computing resources.

To meet the requirements of systems or programs with strict timing constraints (in particular hard real-time systems) and complex execution paths, static timing analysis is more suitable. Static timing analysis doesn't execute the program, but rather statically analysis the timing behavior of the program [15]. Tools for static WCET analysis can generally be split into three parts: first, the program flow is analyzed to determine the possible execution paths by generating a Control-Flow Graph (CFG) from the source code or binary code. Next, the timings of the basic blocks of the CFG are analyzed and calculated. Finally, the WCET is formed by a combination of the two previous steps and evaluation of the longest execution path.

A key aspect of program flow analysis is determining loop bounds and finding the maximum loop iteration count [16], because loop bounds have a major effect on the WCET. Some loop bounds can be determined by advanced tools automatically, but manual annotation of the loop bounds is often still required. In addition, automatic program flow analysis may identify a longest path that is actually infeasible when studying the semantics of the program and taking the input data values into account. Therefore, static WCET analysis often requires manual effort to determine a correct but sufficiently tight WCET.

2.3 T-CREST

Modern processors and many programming languages are optimized for the average use-case. They use various techniques such as caches, dynamic frequency scaling, paging, and code optimization to deliver a good user experience and fast response times for most applications. However, at the same time, these optimization techniques led to a point where it is infeasible to precisely determine the WCET of programs, as they may cause pauses and stalls, e.g., if a cache miss occurs. To solve these problems, the T-CREST project was launched. T-CREST is a STREP of the Technical University of Denmark funded by the Seventh Framework Programme for research and technological development (FP7) which was created by the European Union Commission. The project was supported by several partners, including GMVIS Skysoft, AbsInt Angewandte Informatik, TU Wien, Eindhoven University of Technology, Intecs, The Open Group, and the University of York. The T-CREST project provides a set of tools including a compiler and WCET analysis tools for static timing analysis. Another target of the T-CREST project is the reduction of complexity and costs for systems with safety-critical requirements, whilst gaining faster time-predictable execution times. As part of the T-CREST project, a time-predictable processor named PATMOS was developed [17].

2.3.1 Platform tools

The T-CREST project comes with a rich toolset that handles the toolchain from compilation to the calculation of the WCET bounds [10]. This toolchain is illustrated in Figure 2.4. The starting point is the application code (C-Code) and, if applicable,

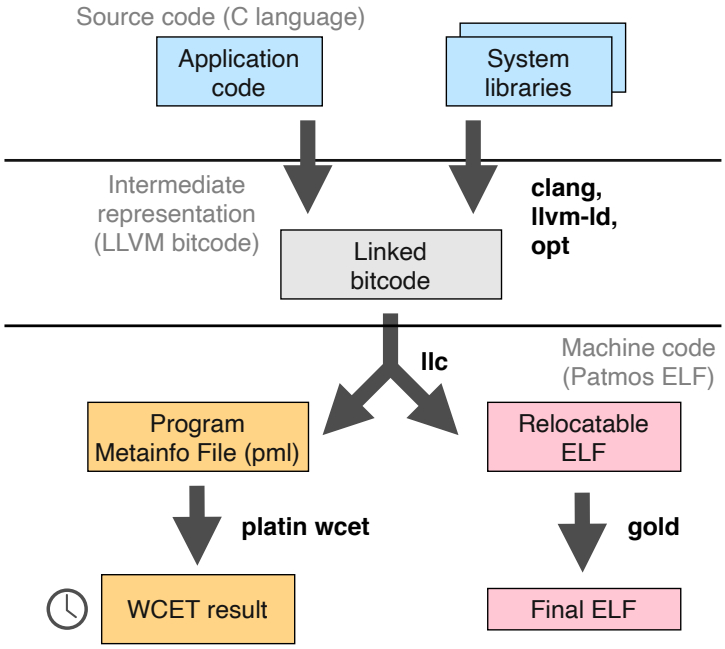

Figure 2.4: T-CREST toolchain (adapted from [10])

additional necessary system libraries. For compilation, the LLVM compiler infrastructure project is used to extend the Patmos compiler [18].

The first step in the compilation process is the translation of every C source code file with the C frontend *clang* to the LLVM intermediate *bitcode* format. This language- and target-independent intermediate representation format represents the program using a CFG in a static single assignment form. The application code and the system libraries are linked by the *llvm-link* tool at the bitcode level.

Optimizations like sub-expression elimination and constant propagation are performed by the *opt* optimizer in a generic, target-independent way. The linked LLVM bitcode is then translated into machine code by the *llc* tool. In addition, target-specific information for WCET analysis is also handled by this tool. The backend, which is constituted by the *llc* tool, generates a relocatable binary ELF file. This file contains symbolic address information. The final ELF file is then generated by the *gold* tool and contains the final data, resolved symbol relocations, and memory layout.

The backend also generates a *Program Metainfo File*, which contains information for the optimization and WCET analysis via subsequent tools. The WCET analysis is performed by the platin's internal analysis tool WCA, which outputs its results to the command line.

2.3.2 Patmos

The Patmos processor is part of the T-CREST platform. In contrast to general-purpose processors, this processor was not optimized for high performance in the average case but was specially designed to deliver narrow WCET bounds and to simplify the determination of these. In order to deliver good single-thread performance, specially designed caches and a Very Long Instruction Word (VLIW) pipeline is used. Patmos follows a RISC-style microprocessor architecture with a 32-bit instruction set and a dual-issue pipeline [19].

2.4 OPC Unified Architecture

In the 1990s, the number of computer-based control and monitor systems increased continuously. A lot of different industrial communication systems were introduced to gain access to the automation data. Many vendor-specific solutions emerged that actually addressed very similar requirements. Consequently, it was and still is difficult and time-consuming to build an automation system with components from different vendors.

Eventually, vendors of HMI and SCADA software joined their forces as they had to solve similar challenges. A task force was formed by the companies Intuitive Technology, Intellution, Opto 22, Rockwell Software, and Fisher-Rosemount in 1995. To unify the data access in Windows-based automation systems, standards were defined under the name OLE for Process Control and later renamed to Open Platform Communications (OPC). The first specification for OPC Data Access was released in August 1996. The reuse of Windows Component Object Model (COM)/Distributed COM (DCOM) base technology and the focus on key features enabled rapid adoption of the standard in the corresponding areas. A variety of OPC specifications followed to meet the requirements of modern industrial applications.

As stated, the focus on the key features and the reuse of existing Windows standards lead to a fast spread of Classic OPC in many products in the automation pyramid. However, as there were not only Windows-based systems, many applications were excluded from using Classic OPC. Other disadvantages of Classic OPC are the lack of a data model and deficient security features. As a result, OPC UA was developed to overcome all these issues regarding performance and features. Comprehensive modeling features and interfaces for various platforms were enabled by OPC UA to fulfill the requirements of state-of-the-art systems. OPC UA has been shown to be applicable on all levels of the automation pyramid up to ERP systems.

2.4.1 Introduction to OPC UA

The OPC UA architecture is typically represented as two pillars. Figure 2.5 shows the Transport pillar left and OPC UA Meta Model pillar right. Data encoding and the message exchange between devices are handled in the Transport pillar. The initial version of OPC UA provided a binary TCP protocol, which was specifically designed for high

performance applications, but also mappings for standard Internet standards like Web Services, XML and HTTP are available for firewall-friendly communication over the Internet. Since the abstract communication model is not dependent on a specific protocol mapping, new protocols can be added in the future, such as the new UDP UA PubSub extension. An overview of current available OPC UA communication technologies is present in [20].

The OPC UA Meta Model pillar on the other hand covers the information modeling. It describes the basic building blocks that are necessary to expose the OPC UA information model. Base types for building the type hierarchy and entry-points into the address space are also defined within this pillar. Also enhanced modeling concepts are available for special applications.

OPC UA Services form the interface between the server and the client. The information model is supplied by the server and accessed by the client. The data is exchanged between the server and the client via the transport mechanisms.

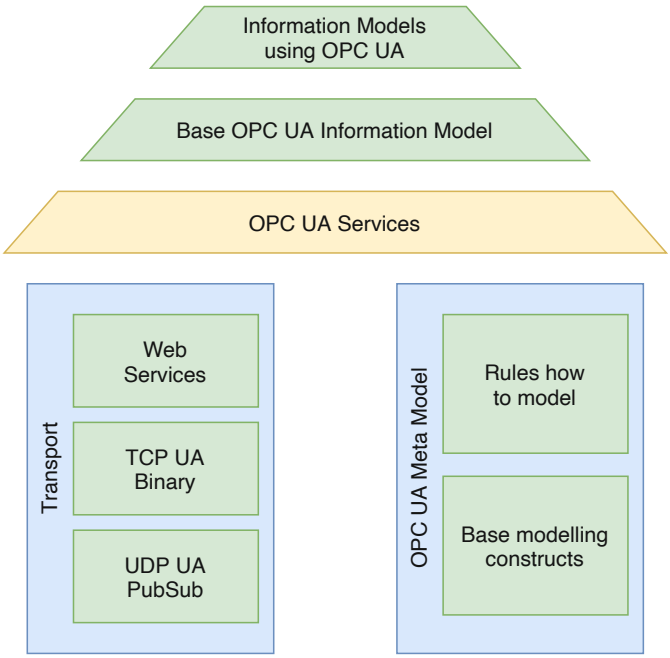

Figure 2.5: The foundation of OPC UA (adapted from [21])

OPC UA mainly uses a Client-Server communication paradigm, but also a Publish-Subscribe (OPC UA PubSub) communication model is available. More complex services like information model browsing and method calls are provided by the Client-Server model. OPC UA PubSub with its minimal communication overhead is mainly used for process data exchange. Figure 2.6 shows the two communication paradigms and their coexistence in an OPC UA application.

In the OPC UA Client-Server communication model the address space is exposed in an object-oriented way, which for example contains periodically updated temperature values of an industrial system. Each client is accessing individual values in the address space or registers for periodically updated values from the server. The server sends the periodically sampled values from the address space, according to the client configuration, over the client TCP connection. For each client a separate session and TCP connection is created, which increases the memory and CPU usage for a higher number of clients [22].

The OPC UA PubSub communication model that is defined in the OPC UA Specification Part 14 [23] introduces a completely decoupled connection between the communication partners. This also includes additional nomenclature, namely *publisher* for the information producer and *subscriber* for the information consumer. The OPC UA publisher can group values from the address space into DataSets, which are then sent to UDP multicast groups in an OPC UA PuSub broker-less setup. Thereby, no session data needs to be stored for each subscriber. This leads to a lower CPU and memory usage compared to the Client-Server communication scheme. However, it must be mentioned that UDP transmission is not reliable and packet loss cannot be detected. In addition to the broker-less system, the OPC UA also supports a broker-based setup, which utilizes a message broker like a Message Queuing Telemetry Transport (MQTT) server to exchange DataSets. More detail on the OPC UA PubSub communication mechanisms is provided in the following section.

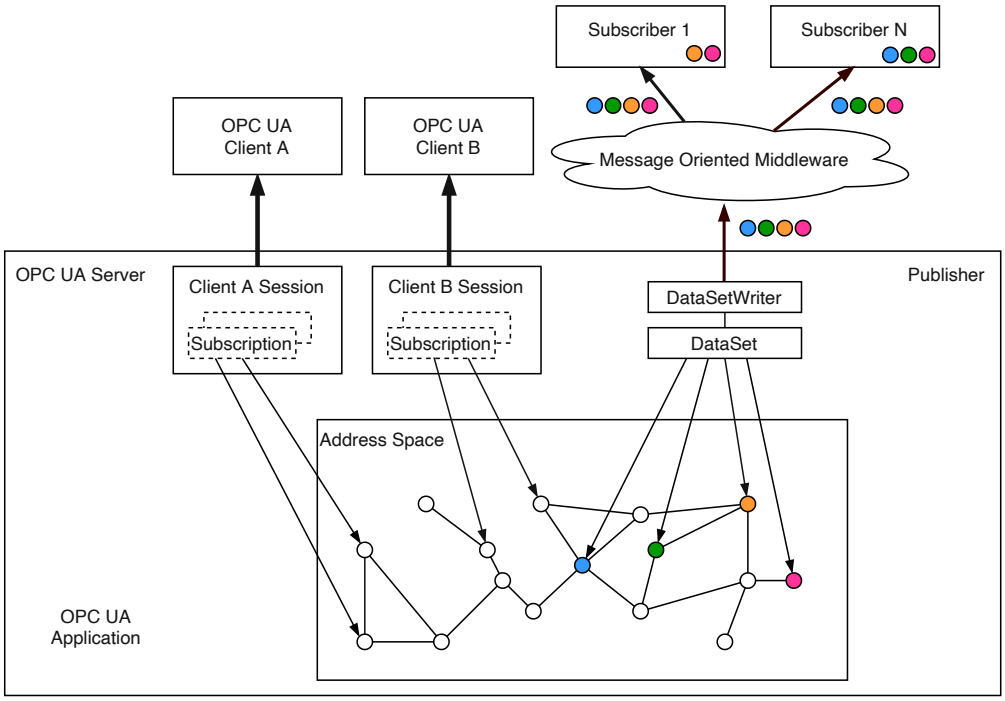

Figure 2.6: Integrated Client-Server and Publish-Subscribe models (adapted from [24])

2.4.2 OPC UA PubSub

To propagate the use of OPC UA in particular in the lower levels of the automation pyramid, i.e., the control level and the field level (cf. Figure 1.1), OPC UA PubSub [23] was released. It fulfills the requirements for low power, optimized, and local low-latency communication on embedded devices, controllers, and sensors.

Figure 2.7 shows the process of packing and publishing data from the information space via a network message. The main components are described below. Further details can be found in OPC UA Specification Part 14 [23]. A *DataSet* is a list of name and value pairs. Each item is called *DataSet field* and can represent arbitrary information. The configuration of which data elements from the information space shall be published is called the *PublishedDataSet* and used by the data collector. The publisher encodes each *DataSet* into a *DataSetMessage*. One or multiple DataSetMessages are then combined in a *NetworkMessage* and transmitted via a communication medium to all subscribers. Semantic information like the data type and the full name of the variable of a *DataSet* can optionally be stored and transmitted via the *DataSetMetaData*. Figure 2.7 shows this process in the context of the publisher.

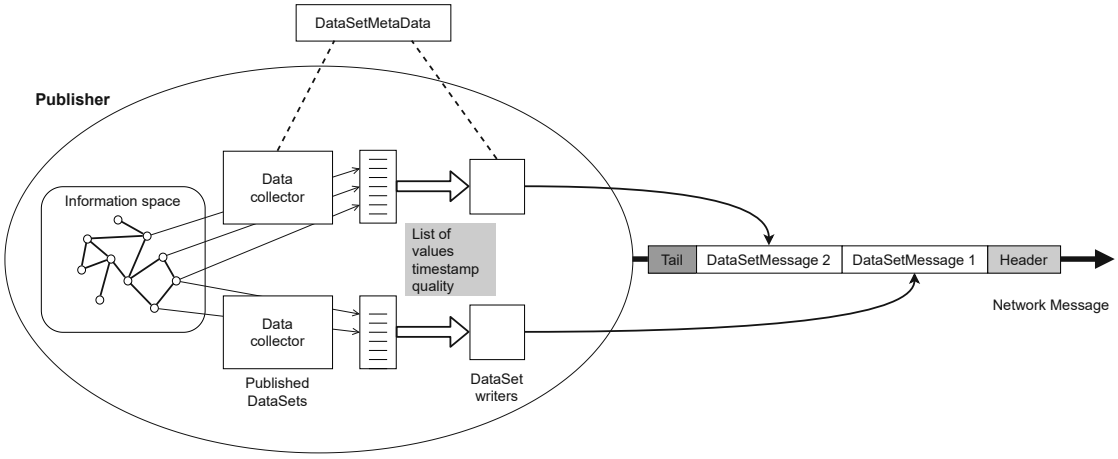

Figure 2.7: DataSet in the process of publishing (adapted from [23])

Figure 2.8 illustrates in a simplified form how an OPC UA PubSub message is built from DataSet fields. Multiple DataSet fields are combined to a DataSetMessage, which in turn is encoded in a NetworkMessage with other DataSetMessages. Finally, the NetworkMessage is transmitted in the payload field via the *TransportProtocol* of a specific Message Oriented Middleware (MOM).

OPC UA PubSub supports two fundamentally different types of MOMs. The broker-less middleware leverages existing network infrastructure that is able to route datagrams. UDP is typically used for this. Figure 2.9 shows how datagrams are transmitted from the publisher through the Network Infrastructure to the Subscriber. In a broker-based

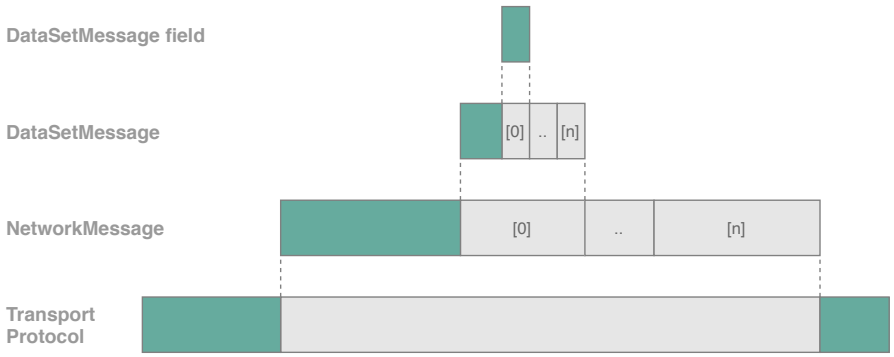

Figure 2.8: OPC UA PubSub message layers (adapted from [23])

implementation, the central point of datagram exchange is formed by a Broker (cf. Figure 2.10). The communication to and from the Broker is based on protocols like MQTT and Advanced Message Queuing Protocol (AMQP). Messages are published to defined queues, which Subscribers can listen to. Both, the broker-less and the broker-based system have certain benefits, which are listed in the following.

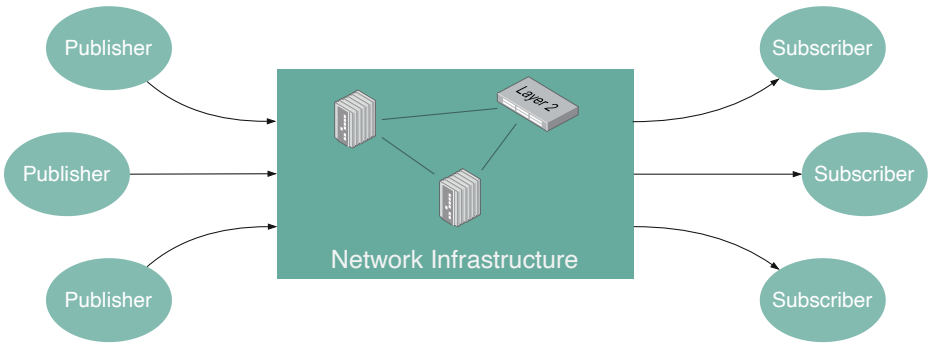

Figure 2.9: OPC UA PubSub using network infrastructure (adapted from [23])

Advantages of a broker-less middleware model:

- This model builds on standard network hardware and no special software like a broker is necessary.
- Messages are directly delivered without software interaction over a switched Ethernet network, which reduces the latency significantly.
- The support for multicast addressing in UDP and in switched Ethernet networks enables the transmission of messages to multiple subscribers.
- Periodic message transmission via UDP is very well supported by TSN.

Advantages of a broker-based middleware model:

- The number of subscribers can be easily extended to a large number, even beyond local networks with scalable or chained brokers.
- The publisher and its subscribers do not need to be online at the same time. The broker can store messages and forward them as soon as the subscriber is online. This is especially important for low power devices, where low power states are important.
- The broker enables the use of different communication protocols for the publisher and the subscriber.
- Using standard protocols such as MQTT and AMQP makes the firewall traversal more easy, as there is no need for the subscribers to directly establish a connection with the publisher. Instead, only the broker needs to be accessible by all communication partners.

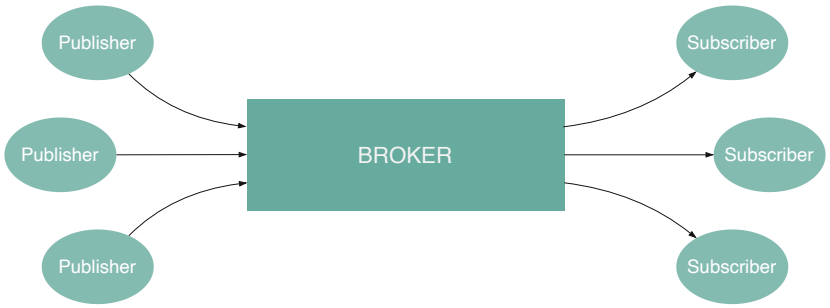

Figure 2.10: OPC UA PubSub using broker (adapted from [23])

2.5 Related work

The number of publications in the field of timing analysis, in contrast to studies especially focused on industrial software analysis, is significantly more extensive. In addition, WCET studies on the quite new OPC UA PubSub standard are not available at the moment. Therefore, this section first focuses on the related work in the area of WCET analysis in industrial automation, and then covers several approaches regarding of real-time capable OPC UA data transmission.

A comparison of static and dynamic WCET analysis on five industrial case-studies was presented by Gustafson et al. [25]. Static and measurement-based tools were used on typical industrial systems for evaluation. The results were very accurate but the authors reported the high complexity of the process caused by the necessary annotation work. In

this context it was mentioned how important a well-developed graphical user interface is for the analysis tools to get a clear and holistic view of the whole system. Possible execution paths can be visualized and the influence of various input values on these paths can be evaluated.

The focus of other authors lies on general challenges in the WCET field. Sehlberg et al. [26], for example, found that industry in this area combats complexity with strict guidelines that prohibit programming constructs that would lead to problems in WCET analysis. This concerns especially language constructs like pointers, recursive functions, recursive data structures, variable-length loops, dynamic memory allocation and assignments with side effects [27]. Some constructs can be handled by modern WCET tools in combination with code annotations, which are used to provide additional information about possible execution paths (e.g., maximum recursion depth). However, partly reprogramming cannot be avoided sometimes. In general, the desired accuracy of the WCET bounds affects the amount of work that is necessary [28].

Automatic flow analysis was used by Barkah et al. [29] on industrial real-time applications to minimize manual work. They showed that the WCET times are the same for automatically generated and manually provided flow information. This result indicates that automatic flow analysis techniques can reduce the need for manual annotation, making WCET analysis tools easier to use.

WCET analysis is often performed at the binary level, which makes it difficult to apply the annotation rules. To circumvent these annotation problems, Lisper et al. [30] proposed to face this problem by program flow analysis at the source level and to resolve specific program flow problem statements on the binary level. They also suggested introducing standards for source-level analysis tools and information for build processes and tools, such as linkers. This should relieve tool vendors from having to offer multiple versions of their tools.

Combining OPC UA programs (a special form of OPC UA state machines) with real-time communication over TSN was proposed in [31]. This should enable real-time communication between distributed processes running on multiple components. A handshake protocol was introduced to synchronize the state machines of various distributed OPC UA programs. For this purpose, the open-source OPC UA open62541 stack was extended to enable the communication between the distributed state machines. The main contribution presented in this paper is the concept of long-running and real-time capable tasks in OPC UA. Since only direct network connections without additional load were used for the evaluation, future experiments with more practical network topologies and stress testing with additional network loads are planned.

The authors in [7] also evaluated the potential of sending OPC UA PubSub messages over a real-time TSN network. To enable real-time properties, a modified open62541

publisher was proposed that is triggered by a hardware interrupt to access a shared information model and publish a DataSet. The setup was based on a standard PC with a special network interface that supports TSN. A Linux operation system patched with the well-known RT-Preempt patch was used as a real-time capable evaluation platform. The measurements showed that a Linux operation system is capable to deliver sub-millisecond intervals with minimal jitter. Porting to other TSN implementations, porting to other embedded platforms, and long-term tests are planned.

Other works in this area focus on field devices based on commercial off-the-shelf (COTS) hardware and software components with OPC UA and TSN. [32] showed that various traffic shaping methods can yield to a reduced latency in the OPC UA communication. This improvement of the latency was even possible on devices with no dedicated TSN hardware support. Prinz [33] investigated ways to use TSN and OPC UA to form an application layer protocol.

The future of industrial network configuration was explored by the authors of [34]. A Software Defined Networking (SDN)-based TSN network architecture in combination with OPC UA for dynamic interconnected devices and applications was proposed. This concept is also especially useful for dynamically networked devices and applications with time-critical requirements. Also, future plug- and produce applications, with their dynamic applications and always-changing interconnections between applications and devices, may benefit from this solution. The implementation was evaluated through several experiments that verified the correct routing, the correct timing of the traffic between the interconnected applications, and the correct auto-configuration capability. The authors also highlighted the complexity caused by the combination of the three technologies OPC UA, TSN, and SDN, and stated that standardization is required in this topic.

OPC UA Field Level Communication simulation was explored in [35] by using OPC UA PubSub. A shared information model with a best-effort IP-based OPC UA Server communication and real-time TSN OPC UA PubSub was used in this approach. The open-source project open62541 was used as the OPC UA library for the simulation environment. They showed that with the developed publisher and subscriber modules, a simulation of the end-to-end jitter and latency is possible.

A large part of the current research related to OPC UA PubSub utilized the open-source OPC UA project open62541. This project is licensed under the Mozilla Public License v2.0. It is royalty-free available on github [36] and supported by a broad and active community.

Industrial software that requires timing analysis can be found in a variety of applications, from space applications [37, 38] to the avionics industry [39, 40]. A case study to find the upper bounds with static WCET analysis for time-critical code, was presented in [41],

which is especially relevant in the field of automotive communication. The conclusion of this paper was how labor-intensive the analysis is and what the practical difficulties are, especially the need for better loop bound analysis was mentioned at this point. It was also stated that a fully automated static WCET analysis is not fully mature, yet. Therefore, detailed knowledge of the target system and source code are required for manual annotations.

CHAPTER 3

Methodological approach

This chapter briefly covers the research methodology that was followed during this thesis. Section 3.1 presents the design and creation research methodology in general. Section 3.2 describes how this methodology was applied to enable WCET analysis of OPC UA PubSub. Finally, Section 3.3 provides a summary of the time and effort that went into the creation of this thesis.

3.1 Design and creation methodology

The thesis aims at contributing knowledge to the field of computer science. A strategy often used in computer sciences is design and creation research [42]. The main focus of this research strategy lies in developing new IT products (artifacts) [43]. An artifact is everything from simple constructs (symbols and vocabulary), models (abstractions and representations), methods (algorithms and practices), and finished instantiations (programs and prototype systems) [44]. In most research activities, however, several artifacts in combination create new knowledge and insights. To distinguish design and creation research strategy from product development, rigorous analysis, explanation, justification, and critical evaluation of the results is necessary.

3.2 Design and creation for WCET analysis

As indicated beforehand, the design and creation methodology uses an artifact as a vessel to create knowledge. Examples for new knowledge are that an IT application incorporates a new theory or produces new insights while in use or the process to create the artifact is new [45]. In this thesis, the artifact is the OPC UA PubSub stack, and the unknown is its timing behavior while executed. Additional knowledge can be gained in the process to obtain the desired information. Figure 3.1 depicts the two knowledge outputs.

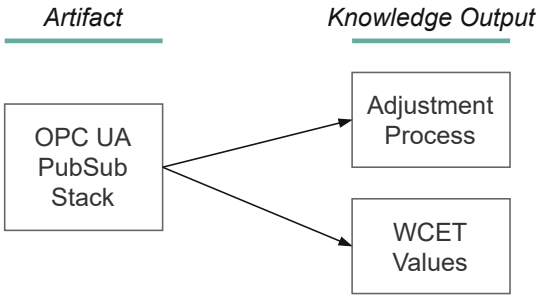

Figure 3.1: Knowledge output produced by this thesis

The design and creation strategy builds upon multiple system development principles, which typically involve awareness, suggestion, development, evaluation, and conclusion [46]. These elements of the problem-solving process are used in an iterative manner, where steps are executed several times when new insights appear. Specifically for computer-based research, the development part requires the most effort and is often further divided into analysis, design, implementation and testing. As there is no specific development methodology covering WCET analysis, typical industrial methods provided the necessary framework. The used T-CREST environment described in Section 2.3 is used in various other projects that act as best practice guidelines. In summary, the combination of industry best practices and related work provided the framework applied in the implementation and testing phase. Figure 3.2 visualizes the key elements of the applied methodology.

The following Chapter 4 *Code transformation for WCET Analysis* and Chapter 5 *WCET analysis of OPC UA PubSub* introduce the developed process and findings made while adjusting the publisher and subscriber of the OPC UA PubSub stack. The *Evaluation* is presented in Section 6. Section *Technical background and related work* 2, which also contains industry best practices, serves as a source of information for the evaluation and development process.

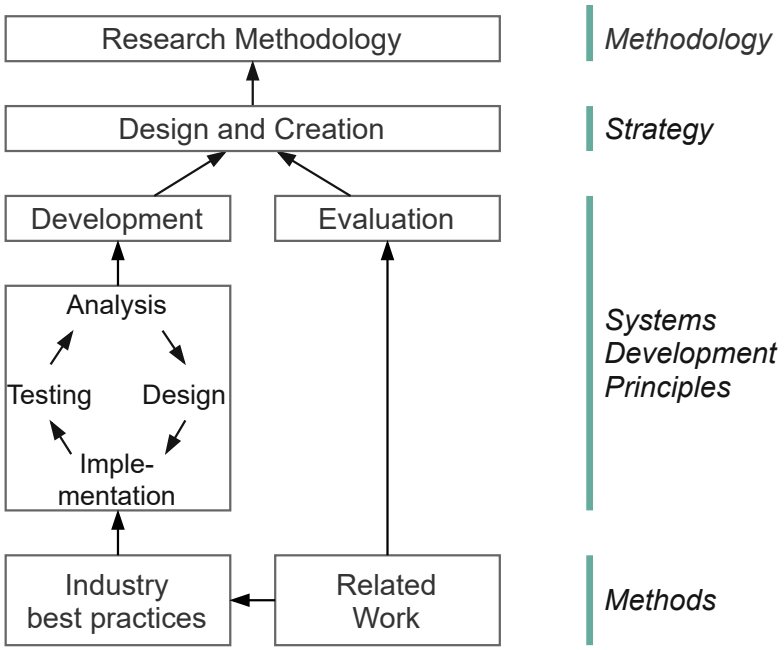

Figure 3.2: Research method overview and elements (based on [42])

3.3 Timeline

The timeline in Table 3.1 separates the activities conducted while following the above methodology into twelve phases and provides estimations of the time that was required for conducting each phase. In total, about 97 full 8 h working days were necessary.

Table 3.1: Estimated workload for each activity conduced during this thesis

Phase	Activity description	Duration
I	Problem statement, literature review, proposal	18 days
II	Familiarization with the evaluation system (T-CREST)	5 days
III	Platform porting	5 days
IV	OPC UA publisher example implementation	3 days
V	Publisher adaptations on OPC UA stack	5 days
VI	Publisher code annotations and transformation for WCET analysis	7 days
VII	OPC UA subscriber example implementation	4 days
VIII	Subscriber adaptations on OPC UA stack	5 days
X	Subscriber code annotations and transformation for WCET analysis	10 days
XI	Measurements and creation of the charts	5 days
XII	Thesis write-up	30 days
	Total	97 days

CHAPTER 4

Code transformation for WCET analysis

This chapter starts with an introduction to the WCET code transformation process. It covers the main steps required to prepare existing software for WCET analysis, including application porting, call graph and control-flow graph generation, code annotations, and code transformations. In the subsequent section, the previously defined annotation and transformation rules are exemplified using an existing C application.

4.1 Code transformation process

One major goal of the WCET community is the advancement towards further automation in WCET analysis. This resulted in well known commercial tools like Rapita Systems *RapiTime* and Absint *aiT* but there are also open-source alternatives available like *Bound-T*, *Chronos*, and the *T-CREST* platform. All these tools offer considerable assistance in the process of calculating the WCET, but a fully automated determination is not possible. Often, existing code that was not developed specifically for real-time applications needs manual adjustment to obtain a result with static WCET analysis. Furthermore, additional effort may be required to get a reasonable tight WCET bound and make the existing code suitable for real-time applications.

The route towards the calculation of the WCET is shown in Figure 4.1. It shows the steps, that need to be performed to systematically get the WCET of an existing application. This process is partly derived from WCET tools and is meant to be an abstract helping guide and a starting point towards WCET analysis. The process consists of different activities that can be separated into porting the application to the real-time platform (Section 4.1.1), generating the the call graph and the CFG (Section 4.1.2), applying

code annotations (Section 4.1.3), and applying code transformations (Section 4.1.4). The output of this process is an application that can be WCET analyzed.

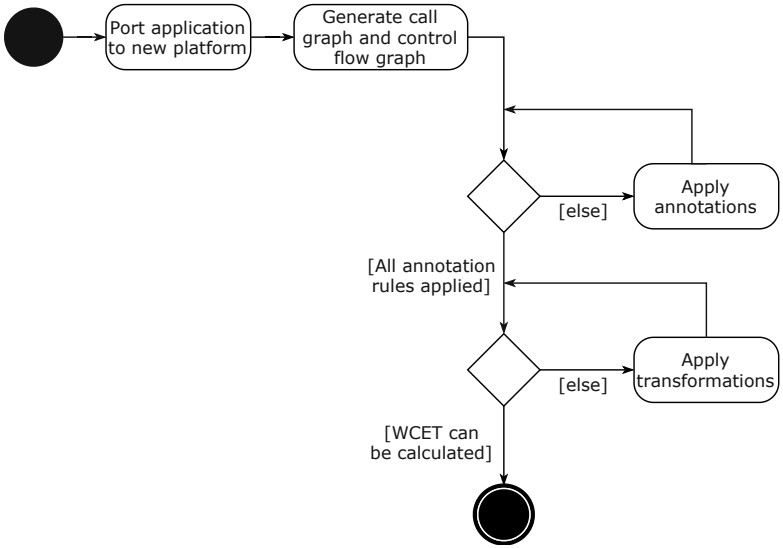

Figure 4.1: Process for adjusting existing software for WCET analysis

4.1.1 Port application to new platform

Nowadays, industrial applications are mostly running on specific computing hardware with platform-specific functionality, especially in real-time applications. Existing software is, on the other hand, mostly developed for general-purpose computers with standard peripheral devices and interfaces. Because of this, one of the first steps is often porting the existing application code from a general-purpose to a time-predictable platform. Platform-specific functions that need may require porting include:

- I/O operations for user interaction
- Filesystem access
- Interrupts
- Network interface access
- Memory management

The outcome of this process step should be code that can be compiled and executed on the new target platform.

4.1.2 Generate call graph and control flow graph

To gain a structured overview of a software program, the call graph and the CFG are very helpful tools. The call graph gives an overview of the interdependencies between the different functions of a program. Thereby, functions are represented as nodes and function calls are indicated by directed edges in the call graph. For example, a function call of g in function f shows up in the call graph as a directed edge from the nodes f to g, g is thus a sub-routine of f. An example of a call graph can be found in Figure 4.2, the function *simple_function* has a directed edge from the *main* function an is thus a sub-routine of *main*. Figure 4.2 also shows how a direct recursive function (cfg. *recursive_infinite*) and a indirect recursive function (cfg. *recursive_function_loop*) will show up in the CFG.

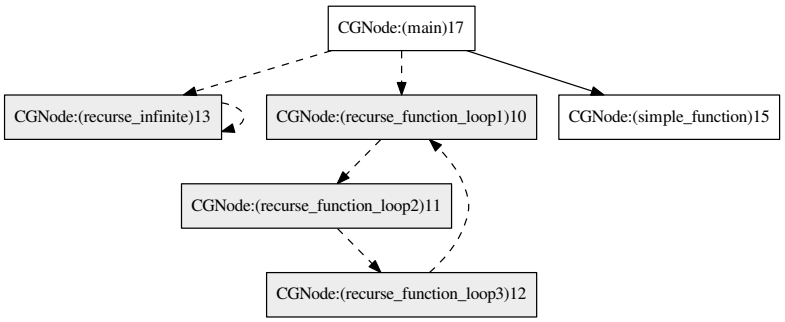

Figure 4.2: Example call graph with indirect and direct recursions

Figure 4.3 shows the CFG for the code example provided in Listing 4.1. The CFG consists of program statements (one or multiple lines) in one or several nodes. The control flow is visualized with directed edges between nodes. One can distinguish between intraprocedural (only a single function is covered) and interprocedural (multiple functions are covered).

```
int a = z;
int b = z*2;

if(c > 0)
{
    a = a*2;
    b = b+1;
}
else
{
    a = a-1;
    b = b*3;
}

z = a+b;

while(z > 0)
{
    c = c+z;
```

```
        z = z-1;
}
```

Listing 4.1: C application serving as an example for a simple control flow graph

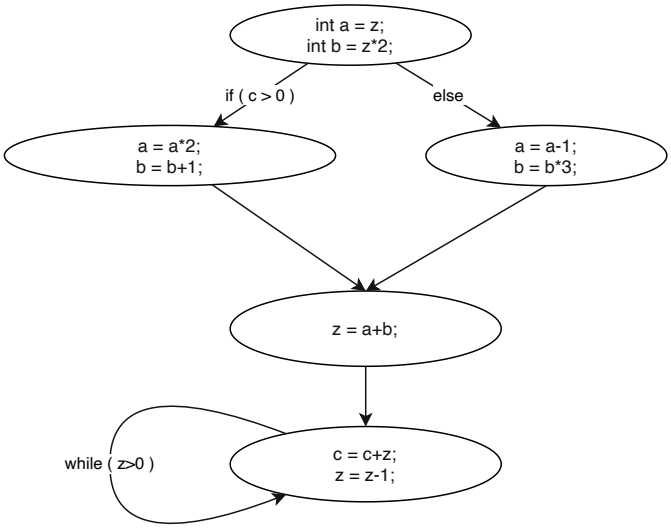

Figure 4.3: Example of a control flow graph with condition and loop

4.1.3 Apply code annotations

To perform the WCET analysis of a source code with WCET analysis tools, code annotations have to be added to the source code or in a separate file. This section presents some pseudo-code examples before and after applying the code annotations. Thereby, the syntax recommended by TACLeBench [47] is applied.

While loop

The code annotations that are shown in Algorithm 1 (top) need to be applied to enable WCET analysis of a while loop. The number of loop iterations is defined via the code annotation (*_Pragma*) before the while loop with a minimum and maximum loop iteration count. To perform a runtime check of the iteration count, an assert statement may be added within the while loop. This may be necessary if the exact loop bounds are impossible to determine by static value analysis or other techniques before runtime. Appropriate error handling routines need to be implemented by the programmer to handle violations against this assert statement because violating the assertion also presents a WCET violation.

Algorithm 1 Annotations: loops

While loop

▷ Original code:
while loop condition **do**
    execute loop content

▷ Code annotated for WCET analysis:
_Pragma("loopbound min X max Y")
while loop condition **do**
    assert loopbound defined by _Pragma is not violated
    execute loop content

Do-while loop

▷ Original code:
do
    execute loop content
while loop condition

▷ Code annotated for WCET analysis:
_Pragma("loopbound min X max Y")
do
    assert loopbound defined by _Pragma is not violated
    execute loop content
while loop condition

For loop

▷ Original code:
for initialization; loop condition; iteration statement **do**
    execute loop content
end for

▷ Code annotated for WCET analysis:
_Pragma("loopbound min X max Y")
for initialization; loop condition; iteration statement **do**
    assert loopbound defined by _Pragma is not violated
    execute loop content
end for

Do-while loop

Do-while loops are treated very similar to while loops. However, the minimum loop bound in Algorithm 1 (middle) is at least one ($X \geq 1$). This is because in do-while loops the loop condition is first checked after the loop content has already been executed once.

For loop

The annotations that need to be applied to a for loop are presented in Algorithm 1 (bottom). The applied annotations are semantically analogous to while loops.

Recursion

Recursion can be split into two types: direct and indirect recursions. Direct recursions are functions that call themselves inside their code. In the call graph, direct recursions show up as edges where the source and the target are the same nodes, also called self-referencing node. Algorithm 2 illustrates the annotations required for a direct recursion in order to enable WCET analysis.

Algorithm 2 Annotation: direct recursion

```
▷ Original code:
function Rec(F)
    if termination condition then
        return F // base case
    else
        F := calculations before recursive call
        R := Rec(F) // solve subproblem
        R := calculations after recursive call
        return R

▷ Code annotated for WCET analysis:
function Rec(F)
    if termination condition then
        return F // base case
    else
        assert recursion depth defined by _Pragma is not violated
        F := calculations before recursive call
        _Pragma("marker recursivecall")
        R := Rec(F) // solve subproblem
        _Pragma( "flowrestriction 1*REC≤X*recursivecall" )
        R := calculations after recursive call
        return R
```

Indirect recursions are more difficult to locate in a call graph as they show up as cycles that span across multiple nodes. The simplest ordinary indirect recursion spans over two functions, where function a is calling function b, which again calls a. Indirect recursions

are more difficult to handle and only a few WCET analysis tools provide annotations to treat them. Therefore, often code transformations are needed.

4.1.4 Apply code transformations

Even if the WCET analysis tool does not provide the possibility of code annotation for a specific programming construct, it is often still possible to change the code in way such that the WCET can be determined. Such changes are called *code transformation*. Common programming constructs that require code transformation, specifically to be WCET analyzable on the T-CREST platform, are presented in the following.

Recursion

Some WCET analysis tools do not support recursion annotations. In this case, iterations need to be used to replace the recursion. The original code in Algorithm 3 shows a directly recursive function that performs calculations before and after the recursive call. Calculations before the recursive call may be performed on the state/variable F. In the next step, the recursive call itself is executed with the state/variable F as a parameter. Finally, calculations on the return state/variable R might be performed.

One way to transform a directly recursive function into a function that only uses iterations is to simulate the stack of the recursive function as shown in Algorithm 3. The iterative function consists of two loops. The first loop pushes data to the stack and the second pulls data back from the stack in a Last-In, First-Out (LIFO) manner. By means of determining the maximum stack size, the loop bound for the two loops can be defined and, thus, the calculation of the WCET is possible. Again, the input data may need to be checked with an assert statement before the loops to ensure that the loop bound will not be exceeded.

Algorithm 3 Transformation rule: direct recursion

```
▷ Original code: cf. Algorithm 2

▷ Code transformed for WCET analysis:
function RECURSION(F)
    Stack S
    _Pragma("loopbound min X max Y")
    while base case not reached do
        assert loopbound defined by _Pragma is not violated
        F := calculations before recursive call
        S.push(F)
    _Pragma("loopbound min X max Y")
    while S not empty do
        assert loopbound defined by _Pragma is not violated
        F := S.pop()
        R := calculations after recursive call
```

To deal with indirect recursions, a process called inlining can be used. Thereby the function call is replaced by the body of the called function, resulting in a direct recursion function. One simple example would be a function a that calls function b, whereby function b calls again function a. Therefore, the call of function b in a needs to be replaced by the contents of function b. Algorithm 3 can then be applied to the resulting direct recursion.

Function pointer / callback function

In C programming, callback functions and function pointers are often used to simplify or to avoid redundant code. In contrast to pointers that store the reference of a variable, function pointers store references to a function. This reference is typically the start of executable code. Function pointers that are passed as an argument to a function are called callback functions. Function pointers and callback functions are difficult to be WCET analyzed because it is often unknown at compile-time which callback functions will be called at runtime. To allow the WCET analysis of callback functions, the possible functions need to be explicitly stated as shown in Algorithm 4.

Algorithm 4 Transformation rule: callback function

```
▷ Original code:
FunctionPointer cb
function RegisterCallback(FunctionPtr)
    cb = FunctionPtr
function CallCallback
    cb()

▷ Code transformed for WCET analysis:
FunctionPointer cb
function RegisterCallback(FunctionPtr)
    cb = FunctionPtr
function CallCallback
    assert cb is one of Function0 .. FunctionX
    switch cb
        case Function0 do
            Function0()
        case Function1 do
            Function1()
        case FunctionX do
            FunctionX()
    end switch
```

Jump table

A jump table or also called branch table is used to alter/branch the program flow with an array of function pointers. The called function is selected at runtime based on the array index. Therefore, the called function is not known at compile-time. Jump Tables

can be replaced by an if-else-if chain or by a switch statement for WCET analysis. The corresponding transformation rule is shown in Algorithm 5.

Algorithm 5 Transformation rule: jumptable

```
▷ Original code:
Array JumpTable[] = {Function0, Function1, FunctionX}
function FUNCTION(I)
    JumpTable[I]()

▷ Code transformed for WCET analysis:
function FUNCTION(I)
    assert I is a valid index for the JumpTable
    switch I
        case 0 do
            Function0()
        case 1 do
            Function1()
        case X do
            FunctionX()
    end switch
```

Non-WCET-analyzable code

Besides the previously presented annotations and transformations rules, often non-analyzable code fragments exist in software projects. Such problems are often very specific and must be treated individually since general annotations and transformation rules are difficult to determine.

One example of this category are existing programs that use pre-compiled libraries where the source code is sometimes not fully available, which leads to problems in the WCET analysis process. The affected parts in the source code that use pre-compiled libraries must be narrowed down and replaced either by a re-implemented version with the same functionality or be replaced by a WCET-analyzable library alternative.

Other problems that are often encountered when WCET analyzing existing source code are functions that are very complex resulting in a calculated WCET bound that is impractical for the target application. Finally, it is sometimes impossible to calculate the WCET of many existing implementations of output functions like printf, blocking input function like getch or scanf, and functions for dynamic memory management like malloc for allocation and free for de-allocation of memory. Such functions need either be removed or replaced by WCET analyzable alternative.

4.2 Code transformation example

In this section, the previously defined annotation and transformation rules are applied to an existing C application. The result should be C code for which the WCET can be computed using the T-CREST toolset. The existing example C application can be seen in Listing 4.2. In this example, the power of a number is calculated using either an iterative or a recursive approach. Only unsigned integer values are possible for simplicity.

The function to calculate the power b^e is called cal_pow. Four arguments are passed to this function and one number is returned as a result of the power calculation. The first two arguments are the base b and the exponent e. The argument t determines the applied calculation method (iterative or recursive), and the last argument is used to return the measured execution time as a pointer to the calling function.

The function cal_pow uses a jumptable to select the function that is called based on the argument t. The execution time is measured with a clock_gettime call before and after the calculation method. The difference is calculated with the timespec_diff function which gives the difference in nanoseconds. This value is then returned as a pointer to the calling function of cal_pow. In its current form, the program cannot be WCET analyzed but has to ported to the T-CREST platform first and then code annotation and transformation rules have to be applied.

In Listing 4.3 the transformed version of the application can be found. It is executable on the Patmos processor and the T-CREST toolset can be applied for WCET analysis. The function clock_gettime for measuring the start and end timestamp was replaced by the platform-specific function get_cpu_usecs, which returns a timestamp in microseconds. It needs to be mentioned that in this simple case, the jumptable that is used to select the function that is called based on the argument t was recognized and solved by T-CREST toolset. It is also visible in the call graph of the function cal_pow that two paths (rec , iter) are possible due to the jumptable (c.f. Figure 4.4).

According to the „for loop code annotation“ (cf. Section 4.1.3), first the preceding _Pragma specifying the min and max loop bound needs to be added to the iterative version of the implementation. In this case, the minimum loop bound is 0 and the maximum is defined as a value MAX that is in the range of a 32-bit unsigned integer value. As the _Pragma is only used for WCET calculation but not checked at runtime, calling the function with $e >$ MAX is possible and would result in a WCET violation. For this reason, an assert statement is added to ensure that the loopbound defined by _Pragama is not violated. As neither the loop index i, nor the loop bound e are changed within the loop, the assertion can, in this very simple case, also be placed before the loop.

Transforming the recursive version of the power calculation function to an iterative equivalent requires two loops. The first loop pushes the variables that are needed after

```
uint32_t iter(uint32_t b, uint32_t e)
{
    uint32_t pow=1;
    for(uint32_t i=0;i<e;i++)
        pow=pow*b;
    return pow;
}

uint32_t rec(uint32_t b, uint32_t e)
{
    if(e==0)
        return 1;
    else
        return b*rec(b,e-1);
}

uint32_t (*pw[])(uint32_t b, uint32_t e) = {iter, rec};

uint32_t calc_pow(uint32_t b, uint32_t e, uint32_t t, struct timespec *diff)
{
    struct timespec start, end;
    clock_gettime(CLOCK_MONOTONIC, &start);
    uint32_t ret = pw[t](b, e);
    clock_gettime(CLOCK_MONOTONIC, &end);
    timespec_diff(&start,&end,diff);

    return ret;
}
```

Listing 4.2: Example C application to be WCET analyzed

the recursive call on the stack. In this example, this is only the base of the exponent. In the next step, the values are popped from the stack and the calculations after the recursive call are executed. In the example, the *ret* variable is initialized with the base case 1 and updated in every iteration with the *ret* variable multiplied with the value popped from the stack. Furthermore, both loops require additional code for the _Pragma and the assertion. With this transformed application code, the T-CREST toolset can be used for the WCET analysis.

```
uint32_t iter(uint32_t b, uint32_t e)
{
    int pow=1;

    assert(e<MAX);

    _Pragma("loopbound min 0 max MAX")
    for(uint32_t i=0;i<e;i++) {
        pow=pow*b;
    }
    return pow;
}

uint32_t rec(uint32_t b, uint32_t e)
{
    uint32_t ret=1;

    _Pragma("loopbound min 0 max MAX")
    for(uint32_t i=e;i>0;i--) {
        assert(e<MAX);
        push(b);
    }

    _Pragma("loopbound min 0 max MAX")
    for(uint32_t i=e;i>0;i--) {
        assert(e<MAX);
        ret=ret*pop();
    }

    return ret;
}

uint32_t (*pw[])(uint32_t b, uint32_t e) = {iter, rec};

uint32_t calc_pow(uint32_t b, uint32_t e, uint32_t t, uint64_t *diff)
{
    uint64_t start, end;
    start=get_cpu_usecs();
    uint32_t ret = pw[t](b, e);
    end=get_cpu_usecs();
    *diff=end-start;

    return ret;
}
```

Listing 4.3: Example C application after applying the annotation and transformation rules

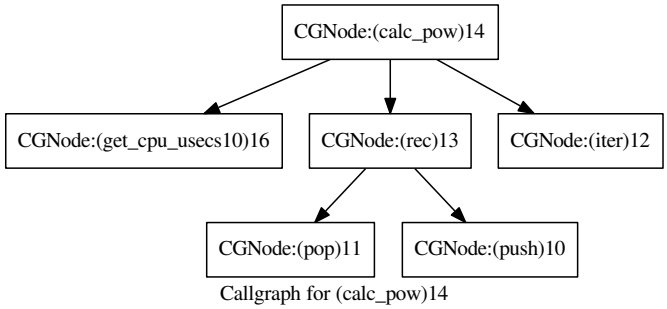

Figure 4.4: Call graph of the transformed C example application

CHAPTER 5

WCET analysis of OPC UA PubSub

To enable real-time communication with OPC UA PubSub, the timings that are involved in the end-to-end communication need to be determined. The whole timing path between an input signal change on one end system and the corresponding action on another end system is discussed in Section 5.1. In Section 5.2, the process of preparing existing code is applied to the publisher implementation using the open-source open62541 stack and the WCET analysis toolset provided by the T-CREST project. The last Section 5.3 applies the same process to the subscriber.

5.1 End-to-end latency

The time between signal change and action perform in an OPC UA PubSub system is called end-to-end latency. It depends on numerous delays between the sending system (i.e., the publisher), the network, and the receiving end system (i.e., the subscriber). Thus, to get a guaranteed upper limit of the end-to-end transmission latency, the upper bounds of all involved delays need to be determined and accumulated. Figure 5.1 shows the different delays that affect the end-to-end latency.

The sending end system can be split into four types of delays and starts with the change of a signal that initiates a data transfer to another system. Most real-time systems perform signal sensing and software tasks in fixed periods of time. Triggering on external events is not commonly used because it can lead to unpredictable behaviors caused by rapidly changing signals. The *TaskSchedulingDelay* in Figure 5.1 represents the delay between the signal change and the sensing and processing in the next publisher task. The upper bound of this delay is the task period, but it can be reduced towards zero if

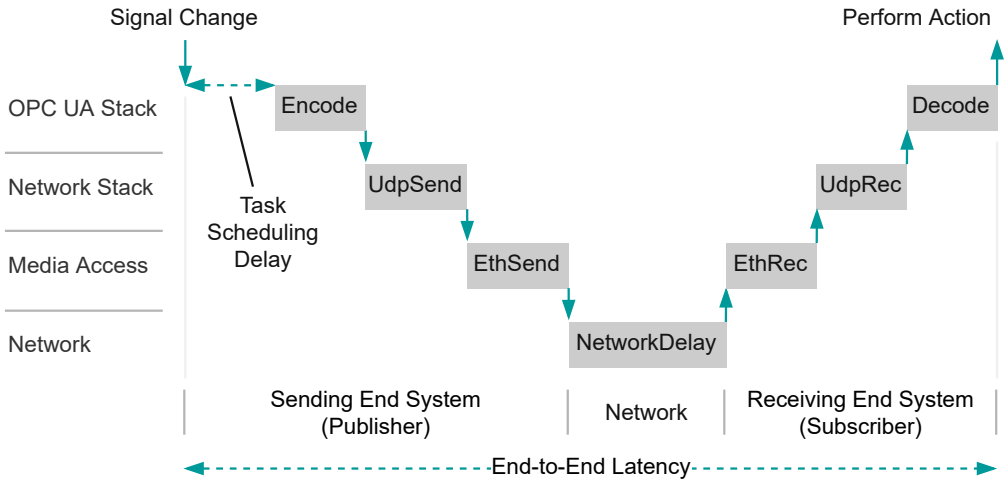

Figure 5.1: OPC UA PubSub end-to-end latency diagram

the signal change can be synchronized with the task interval or if the data value to be transferred only depends on internal data values.

Next, the data is encoded by the OPC UA publisher as defined in Part 6 [48] and Part 14 [23] of the specification. After data encoding, the data from the OPC UA stack is passed to the network stack. *UdpSend* adds additional headers like UDP, IP, and Ethernet headers to the message. The operations up to this point are all done in software. Therefore, the software can be WCET analyzed and the upper bounds of the execution times can be computed.

Finally, the Ethernet frame is sent by the Ethernet controller to the network. This controller is typically implemented in hardware. The delay that is caused in this part (*EthSend*) is typically predictable, but only if the Ethernet controller message queue is empty and no message is currently processed. This can be ensured either by implementing TSN on the publishing end system or by carefully timing the sending of messages in a way such that the network interface is idle whenever a time-critical message shall be transmitted. This topic has already been discussed in the study [49].

The next delay that connects the publisher and subscriber part is the *NetworkDelay*. To ensure that the message is sent within bounded delays, it is mandatory that a real-time network like TSN is used.

The subscriber introduces analogous delays as the publisher, whereby *EthRec* is caused by hardware. *UdpRec* and *Decode* are caused by software and can be determined by performing WCET analysis of the OPC UA subscriber.

5.2 Adjusting the OPC UA Publisher

The task of the OPC UA publisher is encoding and transmitting the message containing information from the application over the network interface. The message encoding is defined in the OPC UA Specification Part 6 [48] and Part 14 [23]. The number of data fields (i.e., the data field length) in a PubSub message is limited by the specification to Int32, which gives a maximum count of 2,147,483,647. This huge number gives the software stack flexibility, but for static WCET analysis this is not practically usable. Therefore, a compromise between narrow WCET bounds and flexibility of the stack has to be found.

This leads to several changes that need to be made on the open62541 stack to restrict the maximum number of data fields that are supported per message. For testing, the number of data fields was limited to two and only a limited amount of data types, mainly the primary ones to build a message, are supported. The primary data types with fixed lengths like Integer, Float, and Boolean are supported but data types with variable lengths like Strings or Arrays are removed from the stack. For real-time systems, these limitations seem acceptable for a first step, as real-time data like sensor values are mostly of fixed size. The stack is further compiled with the open62541 UA_PUBSUB_RT_FIXED_SIZE flag to tell the stack that the structure of a PubSub message is frozen and does not change. This helps to reduce the processing effort for encoding messages because the memory positions of the encoded values are known in advance and, thus, the data value is simply copied into the predefined message structure.

The process presented in Section 4.1 was then applied to the publisher part of the open62541 [50] OPC UA stack and then WCET analyzed. In Table 5.1 the number of annotations and transformations that were necessary to perform the WCET analysis are shown. The toolset provided by the T-CREST project was used to gather the WCET for the publisher with 18,632 processor cycles. With the given clock cycle of 80 MHz, this corresponds to a WCET of about 232.9 μs.

Table 5.1: Programming constructs and number of occurrences for the OPC UA publisher and subscriber

Programming construct	**Number of occurrences**	
	Publisher	**Subscriber**
While loop	1	0
Do-While loop	0	0
For loop	1	11
Indirect recursion	1	3
Jumptable	1	1
Other, non-WCET-analyzable code	6	7

5.3 Adjusting the OPC UA Subscriber

On the receiving side, the OPC UA subscriber decodes the received PubSub network messages and passes the data to the application. The data values can now be processed by the application, which, for example, performs calculations or sets outputs. As for the publisher, the data types and number of data fields need to be restricted for the subscriber to enable a WCET analysis.

The subscriber in the open62541 stack additionally adds the problem of dynamic memory allocation, which is performed for every received message and data field. Dynamic memory allocation is part of the standard C library. However, the standard implementations of *malloc*, *free*, and related functions are non-WCET-analyzable code and had to be re-implemented. A statically allocated memory partition with 32 memory blocks of 512 bytes and an array that holds the memory block status (used/unused) is allocated at startup. The new implementation of *malloc* returns the next free block of memory and marks the block as used in the status array. If no free memory block is available, the function raises an out-of-memory error (ENOMEM), which needs to be handled by the calling function. *Free* releases the memory block by marking the corresponding status array entry as unused. As the newly implemented functions are replacing the standard C library functions one by one, no additional changes have to be performed in the open62541 stack.

The number of annotations and transformations that had to be applied for the subscriber is shown in Table 5.1. The WCET analysis with the T-CREST toolset calculated a WCET of 443,543 processor cycles. On a processor with a 80 MHz clock, this response to about 5,544.29 μs.

CHAPTER 6

Evaluation

This chapter starts with a discussion of the evaluation procedure, the utilized tools, and their interaction within the toolchain in Section 6.1. Also, the different intermediate file formats and the flash process are briefly described. This section is followed by a description of the evaluation setup in Section 6.2, which presents the utilized hardware for the implementation, the selected measurement tool, and the measurement approach. The evaluation results are shown in Section 6.3 and the measured end-to-end latency is compared to the theoretical WCET analysis results in Section 6.4.

6.1 Evaluation procedure

In the process of developing the application code for the publisher and subscriber, tools to compile, analyze, visualize, and flash the program are used. Figure 6.1 gives an overview of the utilized tools, their interaction in the toolchain, and the file formats or interfaces that are used for information exchange.

The starting point for both applications is the application code in C language after the WCET code transformation process has been applied. The *patmos-clang* compiler,m which is especially adapted for the Patmos processor, compiles the source code for the applications. The compiler creates an ELF binary and automatically a PML file that contains meta-information as well as information about the program structure required for WCET analysis.

This PML file can be used to generate control-flow graphs, call-graphs, and relation graphs for both applications with the tool *patmos visualize*. These graphs are especially of interest in the development process to visualize and understand the interrelationships and find direct and indirect recursions. To find recursions, the existing tool was adapted

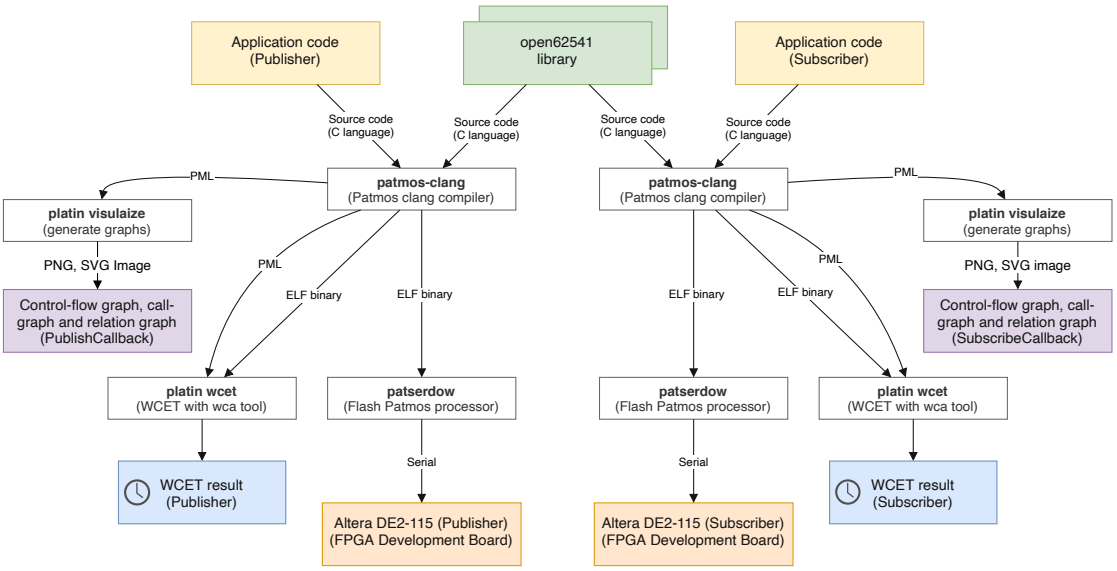

Figure 6.1: Overview of the WCET analysis and deployment process

to detect nodes and edges that form a recursion in the CFG. Thereby, edges and nodes that form a direct or indirect recursion are marked with dashed edges and gray nodes in the CFG (cf. Figure 4.2).

To perform the WCET analysis for the publisher and the subscriber, the tool *platin wcet* is used. By default, it uses the internal tool *wca* to perform the timing analysis, which prints the result directly to the command line.

To enable the measurements on the real hardware, the Patmos processor is flashed with the tool called *patserdow*, which is also part of the T-CREST toolset. The program code is downloaded over a serial connection to the FPGA board and can then be executed to to perform the timing measurements.

6.2 Evaluation setup

Figure 6.2 shows the evaluation setup, which includes two Altera DE2-115 boards with one being the OPC UA publisher and the other the OPC UA subscriber. The two FPGA boards are directly connected with a 2 m point-to-point Ethernet cable at a speed of 100 Mbit/s. Both boards include a Cyclon IV FPGA running the Patmos processor at 80 MHz. On each board, three General Purpose Input/Output (GPIO) pins were used to measure different program positions in the publisher and subscriber.

To measure these timings, a Salea Logic Pro 8 logic analyzer is connected to the two FPGA boards with six probe pins. The main parts of the PublishCallback (publisher)

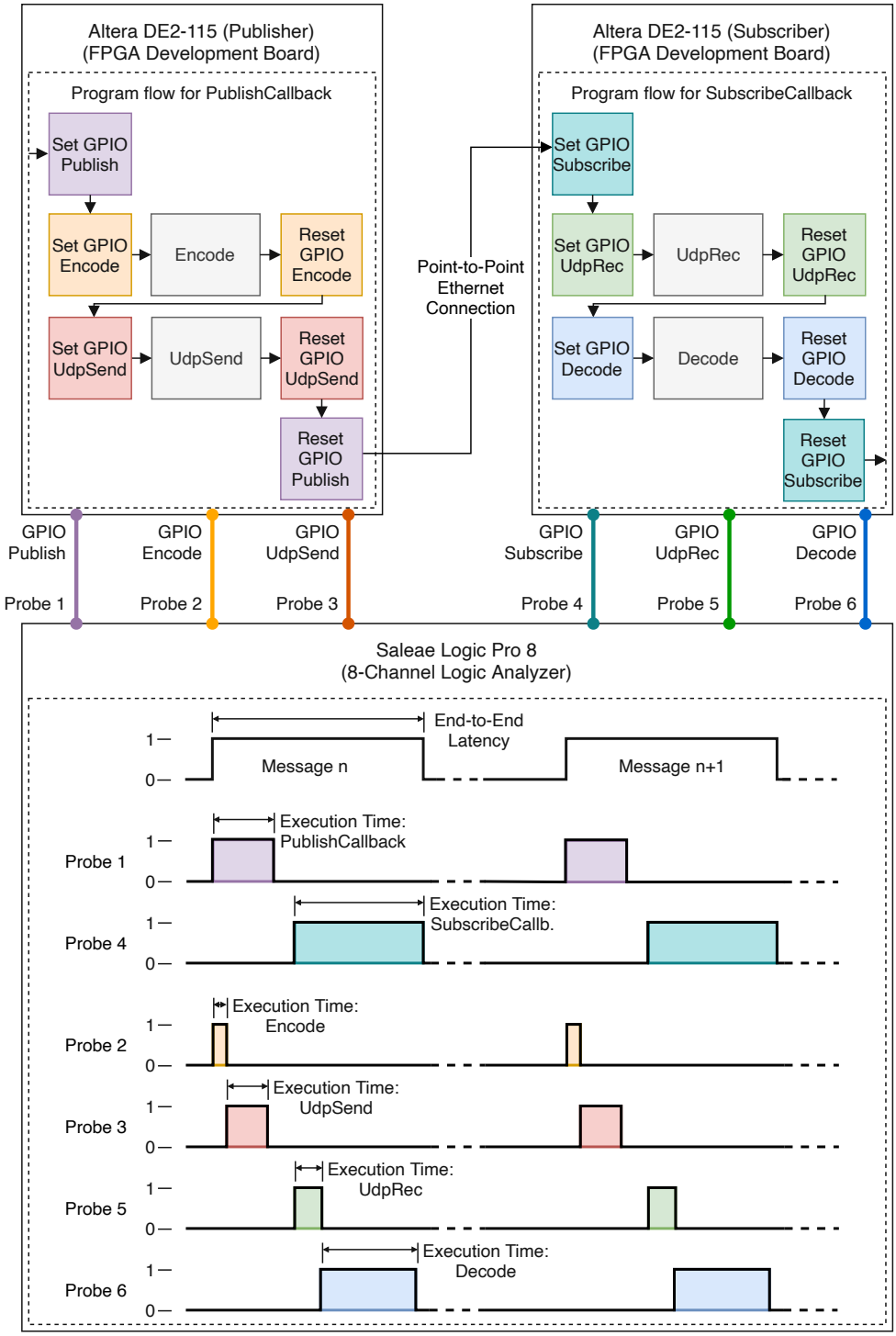

Figure 6.2: Evaluation setup

and the SubscribeCallback (subscriber), which handle the OPC UA PubSub messages, are also shown in Figure 6.2. In addition to the measured times of Encode/Decode and UdpSend/UdpRec, times for pre- and post-processing are covered by the whole execution time of PublishCallback/SubscribeCallback. This time is measured by the GPIO probe pins GPIO Publish and GPIO Subscribe. The example application for evaluation exchanges a single Int32 value, which is incremented with every message.

6.3 WCET measurement results

To generate a histogram of the execution time distribution, 1000 PubSub messages were sent with a delay of 2 ms in between. The distribution of the execution times of the PublishCallback, encoding (Encode), sending (UdpSend), SubscribeCallback, receiving (UdpRec) and decoding (Decode) are shown in the histograms in Figure 6.3.

The calculated WCET for the PublishCallback, which was presented in Section 5, must be higher than the added execution time of Encode and UdpSend. The logic analyzer recorded the maximum execution time for Encode + UdpSend with 136.852 μs, which is approximately 70 % higher than the calculated WCET.

The same applies to the SubscribeCallback. The WCET which was presented in Section 5 must be higher than the sum of UdpRec and Decode. With a measured maximum execution time of 513.08 μs, the WCET bound is about 980 % higher than the maximum recorded execution time of UdpRec + Decode. The higher complexity and the difficult problem with dynamic memory allocation lead to this far higher over-estimation of the WCET compared to the publisher.

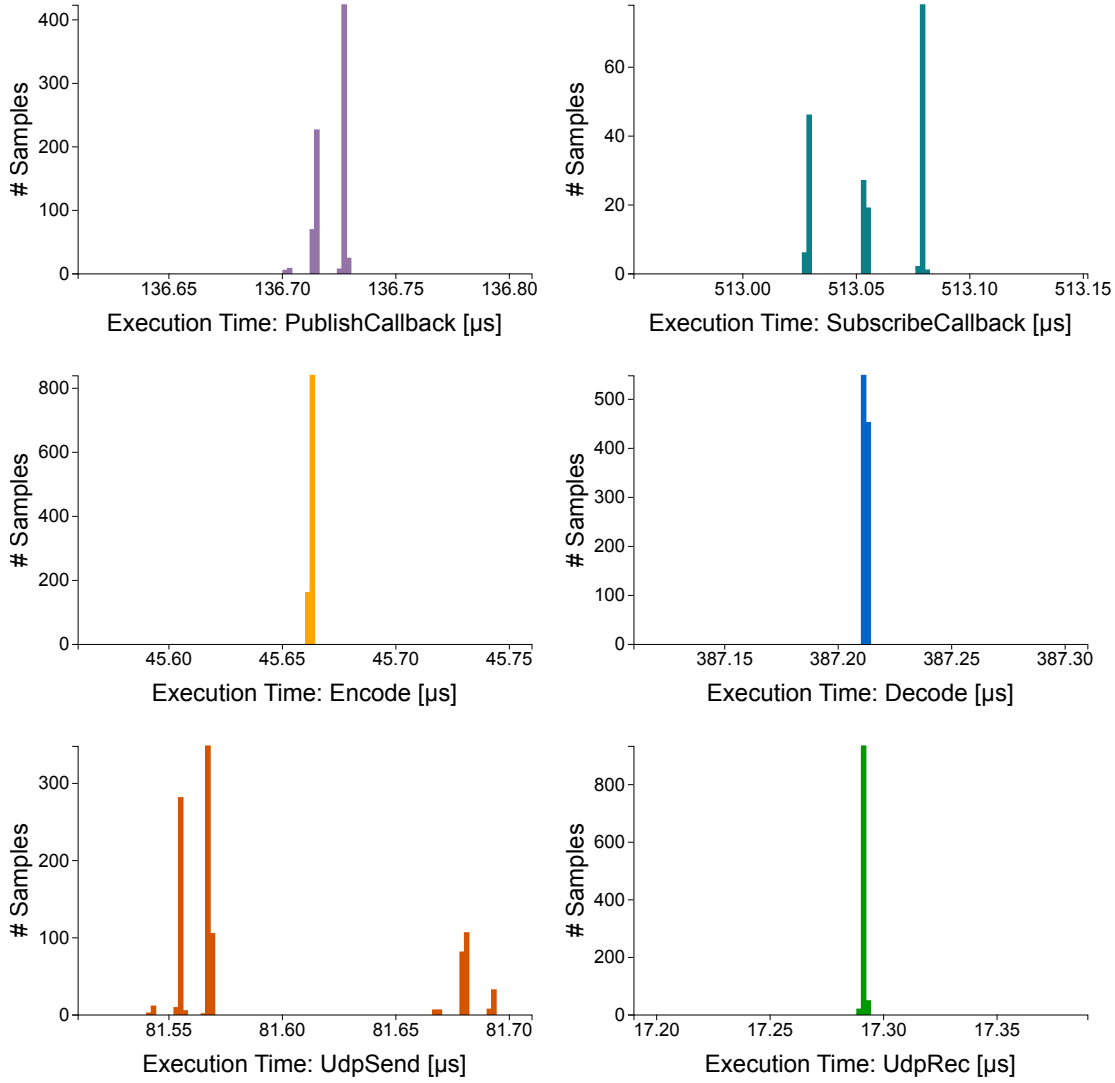

Figure 6.3: Measurement results for PublishCallback, Encode, UdpSend, SubscribeCallback, UdpReceive, and Decode

6.4 End-to-end latency analysis

As discussed in Section 5.1, the execution times of the OPC UA publisher and subscriber are only a part of the end-to-end latency. Table 6.1 presents the calculated WCET values and the measured execution times of all hardware and software delays that affect the end-to-end latency, whereby the sum of the upper bounds defines the upper bound of the end-to-end latency. The sum of EthSend, NetworkDelay, and EthRec depends on the evaluation hardware platform and is briefly discussed in the following.

Table 6.1: Comparison of highest timing measurement result and theoretical upper bounds

SW/HW component	**Upper bound**	**Measurement result**
TaskSchedulingDelay	not relevant	not relevant
PublishCallback ▷ Encode ▷ UdpSend ▷ Pre- and postprocessing	232.9 μs ▷ - ▷ - ▷ -	136.85 μs ▷ 45.67 μs ▷ 81.69 μs ▷ 9.49 μs
EthSend + NetworkDelay + EthRec	HW-dependent	15.4 μs
SubscribeCallback ▷ UdpRec ▷ Decode ▷ Pre- and postprocessing	5,544.29 μs ▷ - ▷ - ▷ -	513.08 μs ▷ 17.29 μs ▷ 387.22 μs ▷ 108.57 μs
End-To-End Latency	5,777.19 μs + NetworkDelay	642.32 μs

Sending Ethernet frames with the Altera DE2-115 evaluation kit requires the Ethernet MAC and Ethernet PHY, which are both implemented in hardware [51]. The Ethernet PHY is implemented on an external IC (Marvell 88E1111). The Ethernet MAC, on the other hand, is implemented in the FPGA itself. This part of the delay is subsumed on the publisher as EthSend and on the subscriber as EthRec. Furthermore, the two Altera DE2-115 evaluation boards are connected with a point-to-point network cable. Therefore, the NetworkDelay is in the range of nanoseconds and only of minor importance. The histogram of the NetworkDelay distribution during the measurement is shown in Figure 6.4. The longest delay of EthSend + NetworkDelay + EthReceive was measured with 15.4 μs.

With the assumption that the network delay is $\leq$20 μs, a guaranteed end-to-end upper bound can be calculated with 5,797.19 μs. The distribution histogram of the measured end-to-end delay can be seen in Figure 6.5. The maximum measured end-to-end delay

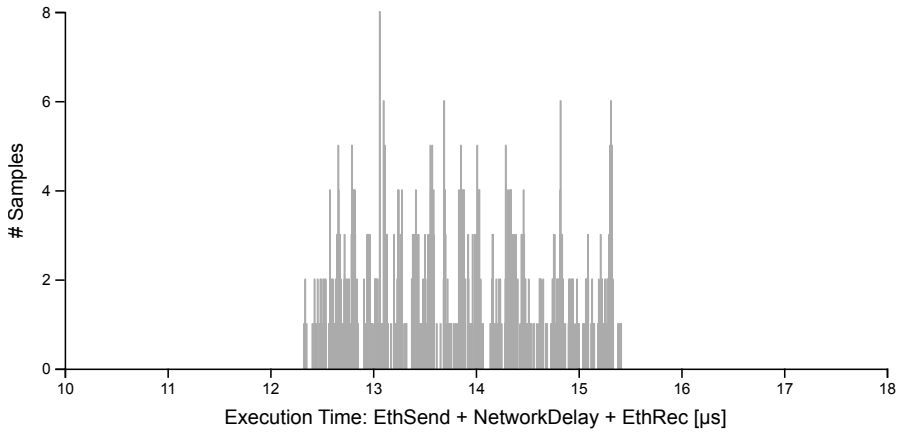

Figure 6.4: Measured NetworkDelay including hardware delays

is 642.32 μs. Therefore, the measurement setup confirms that the practical end-to-end latency for publishing, transmitting, and receiving a message over OPC UA PubSub is well within the theoretical upper limit determined with WCET analysis.

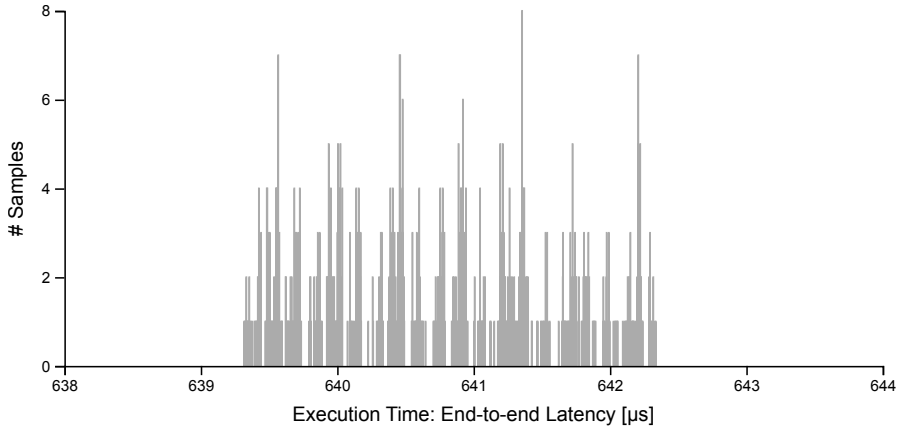

Figure 6.5: Measured end-to-end latency

CHAPTER 7

Discussion

The aim of this thesis is the modification and adjustment of the open62541 OPC UA PubSub stack [52] to enable static WCET analysis and to determine difficulties that arise during this process. The analysis of the publisher and subscriber showed that the subscriber turned out to be more complex. This is due to the extensive use of dynamic memory allocation caused by the unknown message size and structure at compile time. Reprogramming the affected code parts is often necessary [41, 28], because standard dynamic memory allocation is not WCET anaylzable [25], although there are approaches to make dynamic allocation possible in WCET analysis [53]. To enable the WCET analysis of the subscriber, some delimitations like the possible data types and the allowed number of data fields were defined in Section 1.2. These limitations are valid for specific applications, but it needs to be ensured that these assumptions are not violated. Therefore, it is also possible to get tighter WCET bounds by limiting the supported data types and maximum message size for a specific application.

The the evaluation setup confirmed that the calculated WCET values are higher than the measurement results for the publisher and the subscriber individually, as well as for the overall end-to-end latency. In particular, the obtained WCET values of the subscriber are much higher than the measured values. This is due to the higher complexity and the dynamic memory allocation as stated in Section 6.3 and could certainly be improved. It needs to be mentioned that the calculated WCET values are guaranteed upper-bounds. This is in contrast to the closely related work done by Pfrommer et al. [7], which is based on dynamic WCET analysis for one specific message. By changing the identified code parts, even closer guaranteed WCET bounds are possible.

The results of this thesis show that the WCET analysis of the OPC UA PubSub stack is possible and applicable for end-to-end real-time applications over standard Ethernet. The lack of a TSN implementation makes it hard to precisely compare the end-to-end

latency with other research results, but they showed to be in a similar range. Eymüller et al. [31] measured a round trip time for floating-point values with 8 to 160 bytes in the range of 268 μs – 369 μs. This is equivalent to an end-to-end latency of 134 μs – 184.5 μs, which is approximately 25% of the highest value of 642 μs measured by the evaluation setup used in this thesis. This higher latency is likely caused by the significantly lower clock frequency of the Patmos processor (80 MHz) compared to the standard desktop processor selected by Eymüller et al. [31]. It also needs to be stated that a higher network speed of 1 Gbit/s was used in their work.

The claim that the delays in the hardware part (EthSend, NetworkDelay, EthRec) of the evaluation setup are constant may need some additional investigation if certification is required or for specific application scenarios. It can lead to problems if other software also accesses the network interface. Additional hardware support (like a TSN interface) in the end system may be used to overcome this issue.

CHAPTER 8

Conclusion

This chapter concludes the thesis by summarizing the main findings in Section 8.1. Furthermore, it provides a summary of ongoing and future work in this research topic in Section 8.2.

8.1 Findings

In this thesis, the WCET analysis of a OPC UA PubSub stack and the necessary adaptations for a time-predictable platform are presented. In particular, the pitfalls, challenges, and approaches while adjusting the open-source open62541 OPC UA PubSub stack for WCET analysis are elaborated in this work.

A code transformation process was introduced, which allows to adapt existing software in a way such that it can be analyzed with standard WCET analysis tools. After specifying the process, its main steps, and the required annotation and transformation rules, the applicability of the process was demonstrated using a small example application.

Next, the process was applied to the much more complex open62541 OPC UA publisher and subscriber. The open-source time-predictable T-CREST platform with its rich toolset for WCET analysis was leveraged to obtain theoretical upper limits for the time required to send and receive OPC UA PubSub messages.

For evaluation of the WCET timing results, a test setup with two development boards, one for the publisher and one for the subscriber, which are directly connected through a point-to-point network cable, was selected. The setup was used to determine the network delay for the end-to-end latency estimation, and the different states of the publisher and subscriber, which were measured with GPIO signals and a logic analyzer. All obtained

measurement results were well within the expected limits. However, the evaluation also revealed that the WCET for the subscriber is much higher than for the publisher, which is because of the higher complexity and the dynamic memory allocation.

8.2 Future work

The presented work showed the implementation of a point-to-point connected OPC UA publisher and subscriber that can be WCET analyzed on a time-predictable platform within the given network setup and no other network traffic. This is laying the ground for further research efforts towards a distributed end-to-end data transfer environment with multiple participants.

The most promising network standard in the area of time-sensitive networks for automation systems is TSN. Therefore an implementation based on TSN would be a good reference for future industrial implementations. More complex network constellations and the inevitable presence of other network traffic require more comprehensive measurements within different setup scenarios.

The presented implementation focuses on the Patmos processor with its toolset for WCET analysis. Further evaluations of other time-predictable platforms, especially with open-source licenses, would be also of interest for comparison of the evaluation results and to provide an alternative development platform.

In the proof-of-concept implementation, the number of supported data types was limited because of the complexity and due to the excessive-high WCET times caused by the encoding or decoding function for some data types. Most of the data types that were not implemented are hardly ever used in real-time applications (e.g. strings), but some could be of interest for specific applications and, in general, it would be wishful to fully support all data types supported by the OPC UA PubSub stack. Therefore, further development and research efforts could be reasonable in this area.

List of Figures

List of Tables

List of Algorithms

Acronyms

AMQP Advanced Message Queuing Protocol. 16, 17

BCET Best Case Execution Time. 8

CFG Control-Flow Graph. 10, 11, 25, 27, 44

COM Component Object Model. 12

COTS commercial off-the-shelf. 19

DCOM Distributed COM. 12

ERP Enterprise Resource Planning. 2, 12

GPIO General Purpose Input/Output. 44, 46, 53

HMI Human Machine Interface. 2, 12

ICT Information and Communication Technology. 1

IIoT Industrial Internet of Things. xi, 1, 2

IoT Internet of Things. 2

IP Internet Protocol. 2

IT Information Technology. 1, 2, 22

LIFO Last-In, First-Out. 31

M2M Machine-to-Machine. 2

MES Manufacturing Execution System. 2

MOM Message Oriented Middleware. 15

MQTT Message Queuing Telemetry Transport. 14, 16, 17

OPC Open Platform Communications. 12

OPC UA OPC Unified Architecture. ix, xi, 2, 3, 4, 5, 6, 7, 12, 13, 14, 15, 16, 17, 18, 19, 21, 22, 39, 40, 41, 42, 44, 46, 48, 49, 51, 53, 54, 55

OT Operational Technology. 1, 2

PLC Programmable Logic Controller. 2

SCADA Supervisory Control and Data Acquisition. 2, 12

SDN Software Defined Networking. 19

STREP Specific Targeted Research Project. 7, 10

TSN Time-Sensitive Networking. ix, xi, 2, 3, 4, 5, 16, 18, 19, 40, 52, 54, 55

VLIW Very Long Instruction Word. 12

WCET Worst Case Execution Time. ix, xi, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 17, 18, 19, 20, 21, 22, 25, 26, 28, 30, 31, 32, 33, 34, 35, 39, 40, 41, 42, 43, 44, 46, 48, 49, 51, 53, 54, 55

Bibliography

[1] Theodore J. Williams. The Purdue enterprise reference architecture. *Computers in Industry*, 24(2-3):141–158, 1994.

[2] S. Schriegel, T. Kobzan, and J. Jasperneite. Investigation on a distributed SDN control plane architecture for heterogeneous time sensitive networks. In *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)*, pages 1–10, 6 2018.

[3] Robert Harrison, Daniel Vera, and Bilal Ahmad. Engineering methods and tools for cyber–physical automation systems. *Proceedings of the IEEE*, 104(5):973–985, 2016.

[4] M. Wollschlaeger, T. Sauter, and J. Jasperneite. The Future of Industrial Communication: Automation Networks in the Era of the Internet of Things and Industry 4.0. *IEEE Industrial Electronics Magazine*, 11(1):17–27, 3 2017.

[5] D. Bruckner, M. Stănică, R. Blair, S. Schriegel, S. Kehrer, M. Seewald, and T. Sauter. An Introduction to OPC UA TSN for Industrial Communication Systems. *Proceedings of the IEEE*, 107(6):1121–1131, 2019.

[6] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the Internet of Things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16, New York, NY, USA, 2012. Association for Computing Machinery.

[7] J. Pfrommer, A. Ebner, S. Ravikumar, and B. Karunakaran. Open source OPC UA PubSub over TSN for realtime industrial communication. In *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 1087–1090, 2018.

[8] Sven Gottwald. TSN – Time-Sensitive Networking. `https://new.siemens.com/de/de/produkte/automatisierung/industrielle-kommunikation/industrial-ethernet/tsn.html`. [Online; accessed 21-April-2021].

[9] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann,

Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The Worst-Case Execution-Time Problem—Overview of Methods and Survey of Tools. *ACM Trans. Embed. Comput. Syst.*, 7(3), 5 2008.

[10] Martin Schoeberl, Sahar Abbaspour, Benny Akesson, Neil Audsley, Raffaele Capasso, Jamie Garside, Kees Goossens, Sven Goossens, Scott Hansen, Reinhold Heckmann, Stefan Hepp, Benedikt Huber, Alexander Jordan, Evangelia Kasapaki, Jens Knoop, Yonghui Li, Daniel Prokesch, Wolfgang Puffitsch, Peter Puschner, André Rocha, Cláudio Silva, Jens Sparsø, and Alessandro Tocchi. T-crest: Time-predictable multi-core architecture for embedded systems. *Journal of Systems Architecture*, 61(9):449–471, 2015.

[11] Giorgio C. Buttazzo. *Hard Real-Time Computing Systems; Predictable Scheduling Algorithms and Applications*. Real-Time Systems Series. Springer US, Boston, MA, Boston, MA, 2011.

[12] Douglas Wilhelm Harder, Jeff Zarnett, and Vajih Montaghami. *A Practical Introduction to Real-Time Systems for Undergraduate Engineering*. 2014. Online; version 0.14.12.22.

[13] Hermann Kopetz. *Real-Time Systems - Design Principles for Distributed Embedded Applications*. Real-Time Systems Series. Springer, 2011.

[14] Christine. Rochange, Sascha. Uhrig, and Pascal. Sainrat. *Time-predictable architectures /*. Focus Computer Engineering Series. ISTE Ltd : John Wiley & Sons„ London, England ; Hoboken, New Jersey :.

[15] M. Lv, N. Guan, Y. Zhang, Q. Deng, G. Yu, and J. Zhang. A Survey of WCET Analysis of Real-Time Operating Systems. In *2009 International Conference on Embedded Software and Systems*, pages 65–72, 2009.

[16] J. Gustafsson, A. Ermedahl, C. Sandberg, and B. Lisper. Automatic Derivation of Loop Bounds and Infeasible Paths for WCET Analysis Using Abstract Execution. In *2006 27th IEEE International Real-Time Systems Symposium (RTSS'06)*, pages 57–66, 2006.

[17] C. Ferdinand, K. Goossens, J. Baptista, S. Mazzini, Martin Schoeberl, and Scott Hansen. D 8.2 T-CREST white paper technical university of denmark with contributions from all partners. 2013.

[18] C. Lattner and V. Adve. Llvm: a compilation framework for lifelong program analysis transformation. In *International Symposium on Code Generation and Optimization, 2004. CGO 2004.*, pages 75–86, 2004.

[19] Martin Schoeberl, Pascal Schleuniger, Wolfgang Puffitsch, Florian Brandner, and Christian Probst. Towards a time-predictable dual-issue microprocessor: The Patmos approach. pages 11–21, 03 2011.

[20] Andreas Eckhardt, Sebastian Muller, and Ludwig Leurs. An evaluation of the applicability of OPC UA Publish Subscribe on factory automation use cases. pages 1071–1074, 09 2018.

[21] Wolfgang Mahnke, Stefan-Helmut Leitner, and Matthias Damm. *OPC Unified Architecture*. Springer Science & Business Media, 2009.

[22] Andreas Burger, Heiko Koziolek, Julius Rückert, Marie Platenius-Mohr, and Gösta Stomberg. Bottleneck identification and performance modeling of OPC UA communication models. In *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, ICPE '19, page 231–242, New York, NY, USA, 2019. Association for Computing Machinery.

[23] OPC Foundation. OPC Unified Architecture Specification Part 14: PubSub, Release 1.04, 2018.

[24] OPC Foundation. OPC Unified Architecture Specification Part 1: Overview and Concepts 1.04, 2017.

[25] J. Gustafsson and A. Ermedahl. Experiences from Applying WCET Analysis in Industrial Settings. In *10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)*, pages 382–392, 2007.

[26] D. Sehlberg, A. Ermedahl, J. Gustafsson, B. Lisper, and S. Wiegratz. Static WCET Analysis of Real-Time Task-Oriented Code in Vehicle Control Systems. In *Second International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (isola 2006)*, pages 212–219, 2006.

[27] Philip Axer, Rolf Ernst, Heiko Falk, Alain Girault, Daniel Grund, Nan Guan, Bengt Jonsson, Peter Marwedel, Jan Reineke, Christine Rochange, Maurice Sebastian, Reinhard Von Hanxleden, Reinhard Wilhelm, and Wang Yi. Building Timing Predictable Embedded Systems. *ACM Trans. Embed. Comput. Syst.*, 13(4), 03 2014.

[28] M. Platzer and P. Puschner. A Real-Time Application with Fully Predictable Task Timing. In *Proceedings - 2020 IEEE 23rd Int. Symposium on Real-Time Distributed Computing, ISORC*, pages 43–46, 2020.

[29] D. Barkah, A. Ermedahl, J. Gustafsson, B. Lisper, and C. Sandberg. Evaluation of Automatic Flow Analysis for WCET Calculation on Industrial Real-Time System Code. In *2008 Euromicro Conference on Real-Time Systems*, pages 331–340, 2008.

[30] Björn Lisper, Andreas Ermedahl, Dietmar Schreiner, Jens Knoop, and Peter Gliwa. Practical experiences of applying source-level WCET flow analysis to industrial code. *International Journal on Software Tools for Technology Transfer*, 15(1):53–63, 2013.

[31] C. Eymüller, J. Hanke, A. Hoffmann, M. Kugelmann, and W. Reif. Real-time capable OPC-UA programs over TSN for distributed industrial control. In *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 278–285, 2020.

[32] A. Gogolev, R. Braun, and P. Bauer. Tsn traffic shaping for OPC UA field devices. In *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*, volume 1, pages 951–956, 2019.

[33] F. Prinz, M. Schoeffler, A. Eckhardt, A. Lechler, and A. Verl. Configuration of application layer protocols within real-time I4.0 components. In *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*, volume 1, pages 971–976, 2019.

[34] T. Kobzan, I. Blöcher, M. Hendel, S. Althoff, A. Gerhard, S. Schriegel, and J. Jasperneite. Configuration solution for TSN-based industrial networks utilizing SDN and OPC UA. In *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 1629–1636, 2020.

[35] S. K. Panda, M. Majumder, L. Wisniewski, and J. Jasperneite. Real-time industrial communication by using OPC UA field level communication. In *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 1143–1146, 2020.

[36] Open source implementation of OPC UA (OPC Unified Architecture) aka IEC 62541 licensed under Mozilla Public License v2.0. `https://github.com/open62541/open62541`.

[37] Niklas Holsti, Thomas Langbacka, and Sami Saarinen. Using a worst-case execution time tool for real-time verification of the DEBIE software. *Proceedings of DASIA 2000 Conference (Data Systems in Aero- space 2000, ESA SP-457)*, 457:307–312, 2000.

[38] Manuel Rodríguez, Nuno Silva, João Esteves, Luis Henriques, Diamantino Costa, Niklas Holsti, and Kjeld Hjortnaes. Challenges in Calculating the WCET of a Complex On-board Satellite Application. In *Proceedings of 3rd International Workshop on Worst-Case Execution Time Analysis (WCET'2003)*, 2003.

[39] P. Montag, S. Görzig, and P. Levi. Challenges of Timing Verification Tools in the Automotive Domain. In *Second International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (isola 2006)*, pages 227–232, 2006.

[40] Stephan Thesing, Jean Souyris, Reinhold Heckmann, Famantanantsoa Randimbivololona, Marc Langenbach, Reinhard Wilhelm, and Christian Ferdinand. An abstract interpretation-based timing validation of hard real-time avionics software. In *Proceedings of the IEEE International Conference on Dependable Systems and Networks (DSN)*, pages 625–632, 2003.

[41] S. Byhlin, A. Ermedahl, J. Gustafsson, and B. Lisper. Applying static WCET analysis to automotive communication software. In *17th Euromicro Conference on Real-Time Systems (ECRTS'05)*, pages 249–258, 2005.

[42] B. J. Oates. Researching information systems and computing. 2005.

[43] S. March and G. F. Smith. Design and natural science research on information technology. *Decis. Support Syst.*, 15:251–266, 1995.

[44] P. Checkland. Soft systems methodology: A thirty year retrospective a. *Systems Research and Behavioral Science*, 17, 2000.

[45] A. Hevner, S. March, Jinsoo Park, and S. Ram. Design science in information systems research. *MIS Q.*, 28:75–105, 2004.

[46] J. Hughes and T. Wood-Harper. Systems development as a research act. *Journal of Information Technology*, 14:83–94, 1999.

[47] Heiko Falk, Sebastian Altmeyer, Peter Hellinckx, Björn Lisper, Wolfgang Puffitsch, Christine Rochange, Martin Schoeberl, Rasmus Bo Sørensen, Peter Wägemann, and Simon Wegener. TACLeBench: A benchmark collection to support worst-case execution time research. In Martin Schoeberl, editor, *16th International Workshop on Worst-Case Execution Time Analysis (WCET 2016)*, volume 55 of *OpenAccess Series in Informatics (OASIcs)*, pages 2:1–2:10, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[48] OPC Foundation. OPC Unified Architecture Specification Part 6: Mappings, Release 1.04, 2017.

[49] T. Frühwirth, W. Steiner, and B. Stangl. TTEthernet SW-based End System for AUTOSAR. In *Proceedings of the 10th IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 1–8, Siegen, Germany, June 2015.

[50] Florian Palm, Sten Grüner, Julius Pfrommer, Markus Graube, and Leon Urbas. open62541-der offene OPC UA-Stack. *5. Jahreskolloquium "Kommunikation in der Automation"(KommA 2014)*, 2014.

[51] Luca Pezzarossa, Martin Schoeberl, and Jens Sparsø. Towards utilizing reconfigurable shared resources in multi-core hard real-time systems. In *9th Junior Researcher Workshop on Real-Time Computing JRWRTC 2015*, pages 21–24, 2015.

[52] Real-Time OPC UA Pub/Sub implementaion using the open62541 library on Patmos. `https://github.com/t-crest/rt-ua`.

[53] Jörg Herter and Jan Reineke. Making dynamic memory allocation static to support wcet analysis. 01 2009.