

Industrial Validator White Paper

Siemens AG

Nature:	Report
Dissemination Level:	Public
Distribution to Participants ALL INTERESTED companies represented	Additional Distribution interested_all@interested-ip.eu www.interested-ip.eu
DocID: INTERESTED_whitepaper_sie	Creation Date: 17/5/2011

Project	INTERESTED	Contract Number	214889
Author	Stefan Gerken Ralf Pinger Uwe Steinke	Organization	Siemens AG

Project	INTERESTED	Contract Number	214889
Internal Reviewers	Eric Bantegnie	Organization	Esterel

1 Table of Contents

1	Table of Contents	2
2	Motivation for the SIEMENS Validator	3
2.1	Overview.....	3
2.2	Market Requirements.....	3
2.3	Process and Tools	4
2.4	High-level Modeling Advantages.....	5
2.5	Timing Analysis Advantages.....	6
2.6	Cost-benefit Estimation.....	6
3	Technical Implementation of the SIEMENS ZLB Validator	7
3.1	Current Approach	8
3.1.1	Weaknesses of the Current Approach	9
3.1.2	Strengths of the Current Approach.....	9
3.2	INTERESTED Approach.....	9
3.2.1	Evaluating the Use of SysML/SCADE	12
3.2.2	Strengths of the INTERESTED Approach.....	13
3.2.3	Weaknesses of the INTERESTED Approach.....	13
3.3	Metrics.....	13
3.3.1	Engineering Costs	14
3.3.2	Qualitative Evaluation	15
4	Technical Implementation of the SIEMENS Timing Validator ...	16
4.1	Timing Analysis – Current Approach.....	16
4.1.1	Strengths and Weaknesses of the Current Approach	19
4.2	INTERESTED Approach.....	20
4.3	Worst-case Timing Analysis (Code Level) with aiT	20
4.3.1	AbsInt's WCET Analyzer aiT.....	21
4.3.2	aiT Usage	21
4.3.3	AIS Annotations.....	23
4.3.4	Strengths of the aiT Approach	28
4.3.5	Weaknesses of the aiT Approach	28
4.4	Worst-case Timing Analysis (System Level) with SymTA/S	29
4.4.1	Identification of the System Architecture	29
4.4.2	Identification of Internal Scheduling Elements.....	30
4.4.3	Specification of Timing Requirements and Parameters.....	31
4.4.4	Set-up of the Entire System Timing Model.....	31
4.4.5	Refined SymTA/S Model based on TraceAnalyzer Results.....	33
4.4.6	Performance Analyses for Resource Internals	35
4.4.7	Analyzing for Maximum End-to-end Delays	37
4.4.8	Comparison of Worst-case Analysis Results with Constraints.....	40
4.4.9	Identification of Optimization Potential and Strategies	40
4.4.10	Strengths and Weaknesses of the SymTA/S Approach	41
4.5	Timing Analysis Summary	41
5	Summary of Validation Results.....	43
6	References.....	44

2 Motivation for the SIEMENS Validator

2.1 Overview

The railway market is changing significantly. In the past, it mainly focused on high-speed, long-distance mainline and metropolitan mass transit networks. The new arising challenges are to increase safety for regional railways offering low-demand services. Due to the low level of demand, these regional railways have operated completely manually without technical support systems.

These low-demand regional railways differ significantly in their trackside equipment, rolling stock, legal requirements and operating rules. Even the safety levels they have to comply with may vary from one country to another. Hence, technical aids for operating such railways with improved safety and reliability have to be adapted to the very individual requirements and constraints of each operator.

Creating individual solutions is usually very cost-intensive and therefore constitutes a high barrier for introducing technical support systems, only generic and flexible system architectures can provide a sustainable basis for meeting the differing requirements of regional railways for automatic train operation and protection systems, especially also meeting the economic goals of commercial operators. Consequently, architectural goals such as scalability, modifiability and the reuse of subsystems not only relate to safety and dependability, but also play an important part in the design of such systems.

UML and SysML are becoming the standard modeling languages for system engineering in the transportation sector. Most new projects are supported by UML or SysML. The majority of railway operators use these modeling approaches to improve the clarity of their requirements. This is needed to reduce the growing variety of technical solutions coming from different interpretations and levels of experience of rail automation system suppliers. Thus, it is expected that abstract modeling techniques will become a fundamental aspect for the international railway industry within Europe.

Although UML und SysML are quite good for abstract descriptions, they are not sufficient for the precise description of software and for the adaption of qualifiable or certifiable code generators and analysis tools. In this field, we can use SCADE with its strict formal semantics and its qualifiable code generator according to DO-178 B and certified code generator according to CENELEC.

The interconnection of models, design and implementation is a promising task to reduce manual error-prone translations between different abstraction layers. Especially the way back from lower levels of abstraction such as implementation to design is very important, because late changes are very often done at the implementation level during debugging and testing without fully changing the design documentation. Tool-supported round-trip engineering can mitigate this problem.

2.2 Market Requirements

The railway sector shares an increasing reliance on embedded software as well as growth rates with the automotive and avionics sectors. The costs for software testing and assessment usually have a share of up to 50% in the project budget. Compared to usual software development in the IT market, costs for the development of software for systems of the highest safety level are assumed to be 400% higher. Relevant regulations imposed by authorities require CENELEC 50126, 50128 and 50129, which provide requirements and recommended practices for development processes for safety-related software, systems, and the applied RAMS processes.

Another difference to the IT market is that product lifecycles of more than 20 years are usual. Sometimes, the lifecycle exceeds more than 30 years. Hence, special mechanisms for long-term maintenance are needed, including supporting tools and methods and an increase in the value of maintenance in overall product lifecycle costs. This might mean that "ancient" decisions or a poor architecture cannot easily be corrected by complete reimplementations, because this might easily lead to an incompatible product that no longer fits into its old environment. Therefore, decisions have to be made very carefully and a lot of circumstances have to be taken into account – again leading to another increase in overall development costs.

Frequent software updates are a usual way of dealing with errors or missing features in consumer electronics. Although the railway infrastructure is logically centralized, its physical distribution covers the whole operating area. ETCS, for example, would make all Europe one interoperable area. Updating software would require updating all affected subsystems without disrupting operation and without any additional risk to passengers using the railway system – under all operational and environmental circumstances. Usually updating safety-related systems is not done unattended, but a general roll-out would require having access to all affected subsystems at once so as not to interfere with the 24/7-operational requirement of railways.

Furthermore, the level of complexity of typical rail automation solutions – e.g. interlocking systems – is very high, even compared to solutions that are usual in the industry automation domain. Interlocking systems with more than 1,000 trackside elements such as points, signals and level crossings have to be handled, not considering the trackside equipment needed for track vacancy detection, communication components, and the installation of operation safety control computers. Systems that are capable of dealing with this level of complexity on the highest safety level are very challenging, especially as regards the architecture.

2.3 Process and Tools

For the European railway domain, the standards to be adhered to are published by CENELEC. The standard for the development of safety-related software is EN 50128, which defines a process for the development of software for safety-related systems. This process is based on a V-model describing all process steps and artifacts which need to be addressed during development and the methods to be applied.

In Siemens' Rail Automation Business Unit, the EN 50128 process has been applied to the definition of an internal development process as a basis, defining all artifacts and process steps with their responsibilities in more detail. Templates for artifacts are usually given as Word or Excel files. Transformation from one phase to the next is done manually from one given Word-based artifact to the next level. Thus, setting up a system architecture from the definition of the system's requirements, which is usually done from the perspective of model-based development, is not tool-supported at all. Quality assurance is by extensive analysis and review, checking whether all preconditions are met and that transformation has been done in the right way.

Transferring these ideas to the paradigm of model-based development, we support manual transformation and analysis by using tools. Besides an increase in efficiency especially for late changes, we expect a decrease in the number of errors found in late process steps of the overall development process. The propagation of transformation errors is expected to be reduced significantly.

A schematic diagram of the model-based software engineering approach is given below.

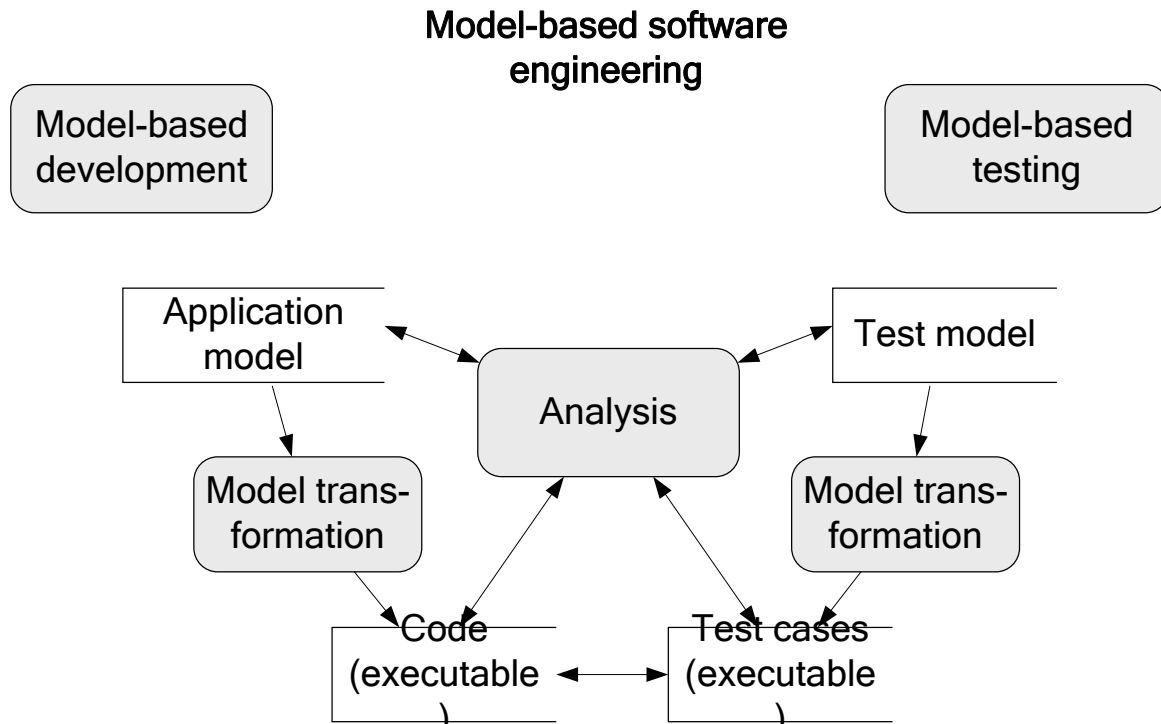


Figure 1: Model-based software engineering

The idea of tool-supported transformation and analysis is extended to the development part – left-hand side of the “V” – and to the testing part, which is covered by the right-hand side of the “V”. In-between, we have put an analysis phase for quality assurance. Analysis could be anything such as formal verification, test coverage, static analysis or timing analysis.

The INTERESTED tool chain contributes to model-based development and the analysis part of the model-based software engineering process. With this tool chain, we are able to extend the development part to another transformation step from SysML via SCADE to executable C-Code. Timing constraints or requirements can be analyzed with timing analysis tools provided by INTERESTED.

2.4 High-level Modeling Advantages

Dealing with systems of a very high level of complexity is usually done by a classic divide-and-conquer approach. The complex system is split into several parts of a lower level of complexity. This is usually repeated until the level of complexity is low and detailed enough for implementation. Thus, systems with a very high level of complexity have several levels of abstraction describing their purpose.

Traditional approaches in the railway industry have been using Word supported by graphics in Visio for representation of the architecture of a specific abstraction level. Neither the textual description itself nor the graphical language has any kind of standardization. Thus, documentation on the architectural level has a high variance. However, more abstract descriptions of interlocking systems use topological layouts of the tracks, whereas speed and operational profiles are used for the characteristics of train control components. This means that certain aspects are expressed in domain-specific languages using terminology which is common in the railway industry. Nevertheless, transformations from one level to the next need a lot of manual work for description, review and the adaption of any change that has been done during the system’s lifecycle.

Using SCADE for the design and component level as an software engineering environment has several advantages. The biggest advantage is that a much higher level of abstraction can be achieved by using SCADE compared to any other imperative programming language. Thus, the SCADE language has been designed especially for reactive embedded systems. The level of abstraction is sufficient for documentation as well, so that, except for general textual descriptions of modules, no special design documentation is

needed. Interfaces to requirements and test environments help a lot in reducing the implementation effort. Although SCADE is quite good on the design and module level, it has some limitations when describing high-level system architectures. On this level, domain-specific languages or representations in SysML with their sublanguages are more sufficient for higher abstraction levels. However, manual transformations from a higher abstraction level to a more concrete level can be quite tedious and a lot of review work is needed to be aware of transformational errors.

The INTERESTED tool chain bridges the gap between SysML and SCADE and can be seen as a contribution to more efficient and less error-prone transformation from one level of abstraction to the next.

2.5 Timing Analysis Advantages

Timing analysis is one part of the evaluation of non-functional requirements which has to be performed when setting up a completely new system. Especially for interlocking systems, since each interlocking system has an individual configuration according to the special track layout, an intensive analysis needs to be done on the target platform. At Siemens' Rail Automation Business Unit, every interlocking system is completely set up at the test center. An intensive analysis is performed including stress tests for the evaluation of its timing behavior. Customers are invited to attend factory acceptance tests before shipping.

Since hardware platforms for safety-related systems are very special due to the redundancy of hardware, there are variants of hardware devices which can be used as a replacement for less powerful and less expensive devices. Thus, the level of ability of reducing hardware costs is much lower than in the automotive business, since we are not dealing with a really large number of units. Usually, the hardware platforms are oversized due to safety reasons.

However, timing problems are well known in the railway domain, especially due to the high degree of distribution of components covering a large geographical area and the necessity of keeping the system in a safe state, which implies that the main controller always needs an actual image of the system's health in time. An analysis of the timing behavior might be very helpful in the design phase of system architectures to make sure that all timing requirements can be met by the architecture. This can help in reducing the number of test cases needed to ensure that timing is met during stress tests.

Since the overall budget for the timing analysis was limited by the amendment, we focused on setting up a very simple timing model for our domain and applied timing analysis tools developed within INTERESTED. Further evaluations should be done for the analysis of a more complex timing model of our system.

2.6 Cost-benefit Estimation

Looking at the historical data of software engineering, working on a higher level of abstraction always leads to an increase in productivity. Today, we spend about 40% of the overall project budget on documentation and implementation, whereas the documentation effort is higher than the implementation effort of the functionality. A more efficient way of documentation and implementation by tool-supported transformations should result in an increase in productivity by up to 30%. Besides, we expect to have costs for the integration of tools, training, etc., compensating the overall increase in productivity to an estimated level of 20% of the development costs.

Furthermore, we expect savings in validation, assessment and maintenance. Especially maintenance has a relatively high share of the overall product costs owing to very long product lifecycles. Due to our decision to use a research study for validation, we cannot provide savings for validation, assessment and long-term maintenance.

Although timing analysis is quite promising in our domain, we have not been able to use these tools due to our proprietary hardware platform. The main goal of the evaluation of timing analysis tools is to set up a timing model which is appropriate for our platform. Further evaluations are recommended to provide cost-benefit statements for timing analysis tools in the railway domain.

3 Technical Implementation of the SIEMENS ZLB Validator

In order to have a railway-specific platform for use in research projects, we chose to set up a realistic simulation environment which can be run on a standard Windows PC and therefore easily be given to research partners for evaluation without the need to rent a van and transport a whole lot of proprietary hardware, let alone the rooms needed at our partner's facilities. Choosing a simulation is in general no problem for the purpose of validating software development methods, because the main characteristic of safety-related software is correctness which, in a good system design, does not depend on specific hardware implementation due to its encapsulation and abstraction.

So as not to bother our research partners with too deep knowledge about operational details and to keep their training period short, we chose a simple operational system as validator, which is taken from German railway systems for regional low-demand railways and is called "Zugleitbetrieb" (ZLB, simplified traffic management on secondary lines). As this operational system is defined as a system without technical support, we extended this system by a functional overlay to support the driver and the operator. The basic requirement is that this overlay system shall ensure the collision-free operation of a regional railway. This artificial validator has already proved its suitability for this application in prior research projects, providing the advantage of having a history which we can reuse for effort estimation and metrics.

The software design approaches within the INTERESTED project are applied for the design of an automatic train operation platform to improve safety on regional railways along the lines of the German ZLB ([HFZG07]) system.

ZLB and its equivalents such as train order working in Australia or track warrant control in North America are based on a centralized operations management concept, with the control of all train movements being concentrated at a so-called dispatching center. Today, the system safety of operating procedures often depends exclusively on human activities.

A communication system is needed to exchange messages between the central dispatcher and train drivers or personnel at local stations. This could be achieved by using a mobile telephone network. The dispatcher records all the messages transmitted between the dispatching center and trains in a message log. The safety of the ZLB operating procedure depends on the dispatcher checking his records of previous communications to prevent conflicting routes. The figure below shows the principle of the centralized dispatching procedure in the ZLB system.

Investigations into current regional railways have exposed two major reasons for critical situations:

- the incorrect logging of messages, usually manual, resulting in unsafe train dispatching, and
- the incorrect execution of correct orders by train or station personnel.

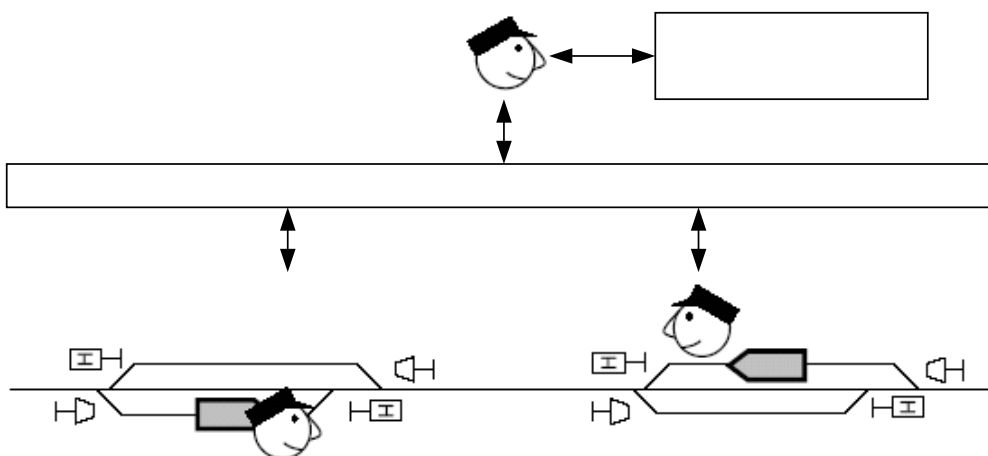


Figure 2: Schematic view of the ZLB system

The vast majority of activities in a railway system are extremely suitable for automation, and manual intervention should therefore be limited to deviations from normal operation. Thus, the logging of messages, the setting of train routes and train supervision in combination with an automatic train stop initiated in the event of unauthorized movements are all candidates for automation, which at the same time promise to improve safety and increase train frequency.

The major functional requirements for an automatic train operation system include:

- protection against overlapping routes
- prevention of movement authorities being overridden
- route setting, etc.

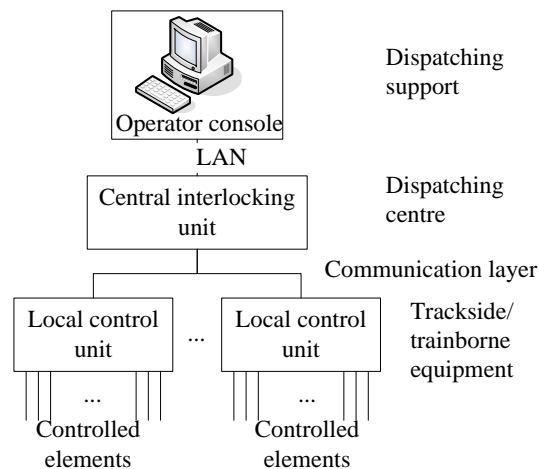


Figure 3: ZLB ATO system structure

Thus, the key idea is to implement an automatic control system for ZLB on the basis of dispatching information. On the one hand, such a system provides decision-making support for the dispatcher. On the other hand, combined with trackside equipment such as electronic signals or axle counters, automatic train protection (ATP) can be implemented. The common architecture of the system family for operational support for regional railways is referred to as an *ATO system*.

Since ZLB covers wayside signaling and train control systems, we have to deal with asynchronous subsystems – interlocking systems and trains – communicating with each other. Since the specific behavior of the interlocking system and the trains is supposed to be implemented in SCADE, we have to bridge the gap between synchronous SCADE models and the asynchronous world.

The ZLB project is used for different case studies und research activities. Thus, it is possible to reuse existing models and implementations and compare alternative modeling and implementation activities.

3.1 Current Approach

Reference implementation of the interlocking part of the ZLB example was done in C++. The design was able to deal with tracks, signals and points to set up a specific track layout. Routing was done by searching for the next signal within the track layout. Besides management for routes, signal aspects and point positions, we implemented a communication layer for communications to the train. Thus, a train driver was able to send a route request to the interlocking and received an answer from a movement authority.

Documentation was done in Word for early versions. Later on, we switched to an object design in Artisan Studio to document the software architecture of the C++ approach. Transformations from the architecture to the implementation level and vice versa were done manually.

3.1.1 Weaknesses of the Current Approach

Although implementation and documentation were done in several iterations, we found many deviations between implementation and documentation. Manual transformation is always error-prone and quality assurance accounts for a very high percentage of the effort to keep both descriptions in sync.

Although the C++ implementation was meant to use different kinds of track layouts by reading the desired layout from an xml file, we found that the implementation had some bugs which led to unsafe states of the interlocking when changing the track layout. Thus, flexibility of the implementation and quality had some restrictions which could not be checked offline but only by intensively testing different configurations. This large number of possible permutations and combinations makes it impossible to completely test a generic implementation. Here, an analytic approach could very much improve the correctness of the software, but is not available out of the box for conventional programming languages such as C++.

3.1.2 Strengths of the Current Approach

A strength that should not remain unmentioned is a psychological one. As the user of a modeling tool chain most often has a syntactic and partly semantic check supporting his or her work, he or she usually believes that he or she produces “good” results. In fact, tools do not prevent the user from making inefficient or wrong design decisions. So, as long as the tool chain does not complain about the artifacts, the willingness to discuss architecture and design is reduced.

In a Word- and Visio-based process, everyone is totally aware that no-one else than colleagues will review the artifact and give semantic feedback. As discussions about the basic structure of a software design leading to the necessity of a complete redesign are usually quite unpopular, people try to avoid these discussions at the end of their task. This usually leads to helpful early feedback, which also increases productivity and mitigates the risk of systematic errors.

3.2 INTERESTED Approach

Because we rely on using usage-proven libraries and hardware components for our system design and also have more abstract, rather large and distributed systems, our favorite and most practiced approach is a middle-out approach.

Using SysML and SCADE for implementation of the ZLB interlocking part, we started a combined top-down bottom-up approach to be as close as possible to a middle-out approach. This was needed since the interface between Artisan/Papyrus and SCADE first had to be developed in the INTERESTED project and complete round-trip engineering is not yet provided. We set up an overall architecture of the ZLB system including train control component, physical track behavior and interlocking and defined communication between these components. Communication between the interlocking and the train control component is naturally asynchronous.

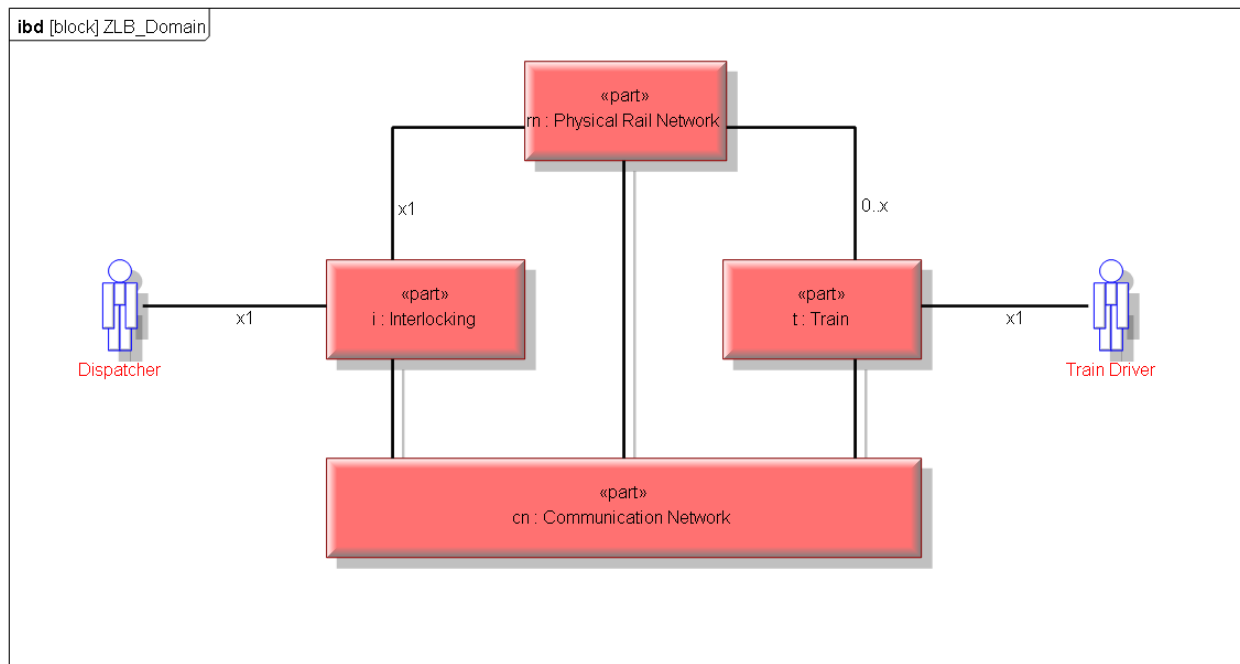


Figure 4: Top-level architecture of the ZLB system

The logical structure of the ZLB system is shown in Figure 4. The interlocking and the train have an HMI, allowing the dispatcher and the train driver to have control over the applications. Communication is done via the communication network. The interlocking and the train are connected to the physical rail network featuring all field elements such as points and signals. The interlocking system is intended to control all wayside field elements whereas the train will be able to receive signal aspects and point positions from the outside components.

Communication between the ZLB components is shown in Figure 5. Communication is done via a central broker component enabling the architecture to be extended by a diagnostic component or a juridical recorder if needed. All communication in the architecture is asynchronous. The component OBU is meant to be the graphical user interface for the train driver whereas the ATP system executes the safety functions, e.g. applying the brake, on the train. The component DHMI is the graphical user interface giving the dispatcher control over the interlocking. The interlocking itself features all safety functions for route setting and point and signal control. PhysicsEngine is a pure test environment simulating the physical behavior of the trains. It gives information on the current speed and position of the train to the ATP system and the interlocking.

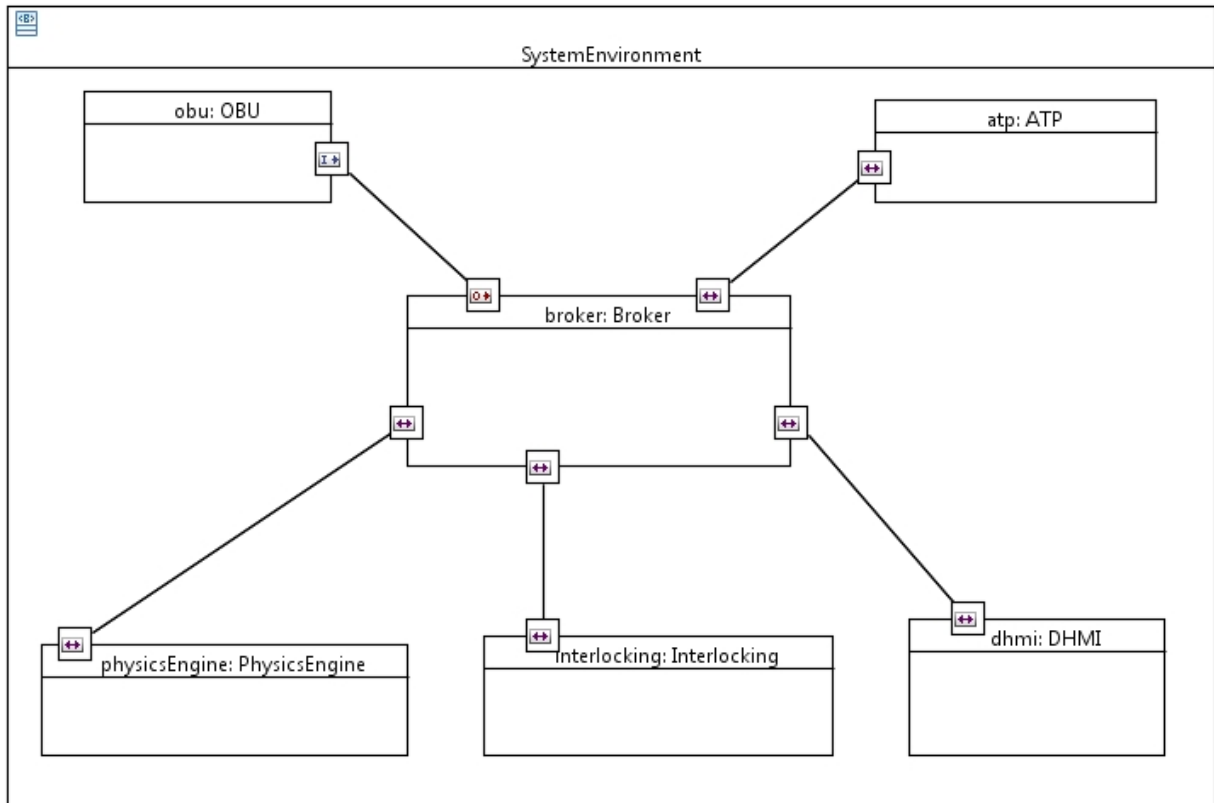


Figure 5: Top-level model of the system using a broker

When going to a more detailed level, we decided to use SCADE for design of the interlocking and the train control component. Since the components communicate asynchronously, they were set up as two different models running on different processors or within the simulation environment tasks.

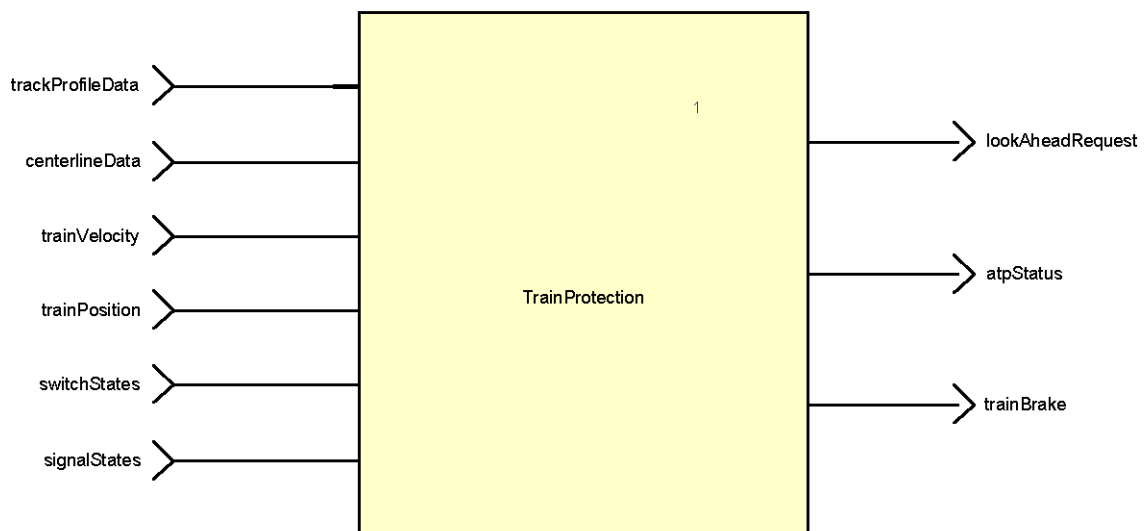


Figure 6: Top-level SCADE-model of the ATP system

Figure 7 shows the inputs and outputs of the ATP system. It has to be aware of the current velocity and position of the train. Besides, signal aspects and point positions need to be checked by the ATP system before being passed. The output gives an actual status and commands for controlling the brakes.

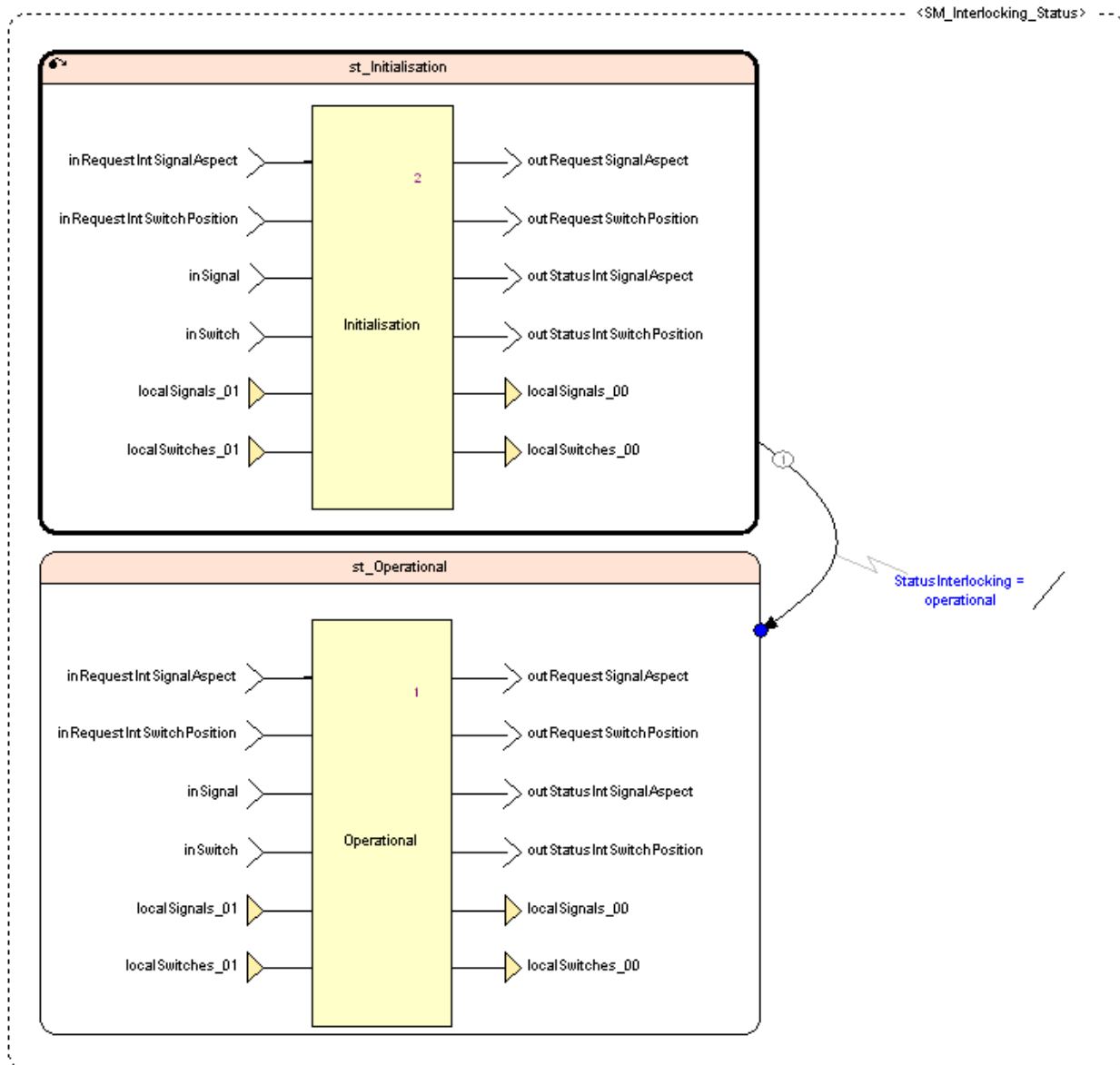


Figure 7: Top-level SCADE model of the interlocking

The top-level behavior of the interlocking is given in Figure 7. First, there has to be an initialization phase checking all point positions, signal aspects and train positions. After this normal operation such as route setting, the route can be canceled by the dispatcher.

Transformation from the architecture level in SysML to SCADE has been done using the interfaces provided by Artisan, Papyrus and SCADE.

3.2.1 Evaluating the Use of SysML/SCADE

When using SysML and SCADE, we were able to cover the development lifecycle from description of the architecture down to implementation for the complete ZLB example. With the classic approach, we were not able to describe the architecture to the same level of precision or reuse components from the C++ project. One major reason for this might be that adaption of the interface is not as easy as in the integrated SysML and SCADE approach.

In the C++ documentation, we did not describe the abstract architecture. Thus, we focused on the interlocking part of the ZLB system. With the combined SysML/SCADE approach, we were able to reach a better level of abstraction of the ZLB architecture, making it easier to introduce new employees into the project and to cut down on their necessary training periods. Thus, we were able to increase overall maintainability of the product.

The total number of injected errors of both approaches was not evaluated by metrics, but we recognized that the number of errors had been reduced right from the beginning. The integration phase was cut down significantly in time. Besides, flexibility of the interlocking, which was achieved with the SCADE approach, is much higher than in the version done in C++. Flexibility means that changes can be done faster, especially when changing the track layout with a higher level of reliability. So we observed an increase in quality when using SysML and SCADE.

However, we had to deal with engineering tools to support the development process. Since these tools have been developed within the INTERESTED project, we had to deal with (pre-)beta versions for evaluation. This means that using the interface and changing parts in our SysML design sometimes resulted in compatibility problems with the already existing SCADE design – e.g. eliminating all changes that had been made beforehand. When looking at the overall tool integration costs, we tried to filter these effects, since we expect that they will be corrected in a productive release.

Besides some minor bugs, we think that the vertical transformation – coming from a higher level of abstraction in SysML and going to a more detailed design level in SCADE – can be improved as well. Right now, transformation is limited to flowcharts only. Transformation of state machines could be helpful as well. Furthermore, integration of sequence diagrams could help especially when designing asynchronous communication behavior.

3.2.2 Strengths of the INTERESTED Approach

- Coverage ranges from software architecture to software implementation.
- Increase in quality
- Reduction of errors
- Increase in flexibility
- Increase in maintainability and understandability

3.2.3 Weaknesses of the INTERESTED Approach

- Only data flow diagrams can be transformed.
- Conceptual work is needed for the transformation of other diagrams.
- Some minor bugs that are expected to vanish in the release version

Besides, the integration of test tools and further test concepts based on test models should be the next step for an integrated tool chain of model-based development and analysis tools.

3.3 Metrics

For the ZLB example, we have a reference implementation done in C++ during a former research study. From that project, we have a cost structure of the lifecycle steps for requirements engineering, architecture description, implementation and documentation. For the INTERESTED approach, we reused the requirements phase for the implementation based on SysML and SCADE.

All values are given as a percentage of the C++ reference implementation. We used different scenarios, e.g. one scenario including the requirements phase and one scenario excluding the requirements phase. Thus, we are able to represent numbers for savings achieved using the INTERESTED tool chain.

3.3.1 Engineering Costs

	C++	SysML/SCADE
Requirements engineering	20 %	20 %
Architecture	27 %	17 %
Implementation	20 %	13 %
Documentation	33 %	Implicit
Tool integration	0 %	17 %
Training	0 %	13 %
Overall	100 %	80 %

Table 1 Development costs: reference implementation vs. INTERESTED tool chain

Table 1 gives an overview of the total development costs for the reference implementation and the INTERESTED tool chain. It covers all costs starting from requirements engineering up to tool integration and training. Tool integration costs are derived from the total costs of integrating all prototype versions that have been developed during the INTERESTED project. Thus, we had several integration cycles.

The overall effort needed for tool training is expected to be lower than the figure given in Table 1, since training material has improved a lot during the INTERESTED project. The first version of the training material was given as a set of PowerPoint slides. We expect less training effort for the productive versions of tools and training material.

	C++	SysML/SCADE
Requirements engineering	0 %	0 %
Architecture	33 %	21 %
Implementation	25 %	17 %
Documentation	42 %	Implicit
Tool integration	0 %	21 %
Training	0 %	0 %
Overall	100 %	58 %

Table 2: Total costs excluding requirements engineering and tool training

To give a more detailed view on the costs and savings that can be achieved when using the INTERESTED tool chain, we excluded the requirements and tool training phase. This can be achieved starting from the same requirements with experts in using SysML and SCADE. Most of the savings are achieved due to the implicit documentation of architecture and implementation done in SysML and SCADE. The C++ approach uses tedious Word and Visio documentation with lots of review steps.

	C++	SysML/SCADE
Requirements engineering	0 %	0 %
Architecture	33 %	21 %
Implementation	25 %	17 %
Documentation	42 %	Implicit
Tool integration	0 %	10 %
Training	0 %	0 %
Overall	100 %	48 %

Table 3: Costs with expected tool integration costs

Since we had to use a prototype version of the tools with several integration cycles, we expect tool integration costs to be cut by 50 % as a realistic estimation of the tool integration costs, which is needed in a productive environment. Even with an operative engineering tool chain, adaptations to changing the environment are needed. These overall costs will increase as the number of integrated tools will increase. On the other hand, we expect a significant reduction of costs by using an integrated tool chain.

3.3.2 Qualitative Evaluation

Savings discovered in the previous section are derived through

- **Readability:** Especially the graphical notation of architecture and design leads to an increase in readability of the system description.
- **Understandability:** SysML and SCADE representations became a basis for discussion between different roles involved in the project. This leads to a common understanding of the system.
- **Maintainability:** Although testing was not part of the INTERESTED tool chain, we were able to speed up the localization of errors introduced during the development process and found during testing and integration.
- **Expandability:** Especially when changing the architecture which might be necessary due to late changes to the requirements, it became easier when using the INTERESTED tool chain. Due to integration of the tools and code generation, even more comprehensive changes can be done more easily than in the classic approach.
- **Product quality:** Although we did not have metrics for the total number of errors found during testing and integration within the INTERESTED project, we felt a reduction of errors in comparison with other rail automation projects already using the model-based development approach.

4 Technical Implementation of the SIEMENS Timing Validator

4.1 Timing Analysis – Current Approach

Rail automation systems typically have to fulfill timing requirements.

An existing Siemens track vacancy detection (TVD) system has been chosen as validator for evaluation of the INTERESTED timing analysis tools.

Target TVD (track vacancy detection) system description

As its main task, a track vacancy detection system provides an interlocking system with information about whether a track section is clear or occupied by a train.

The TVD system is based on wheel detectors located along the track. The detectors rely on an electromagnetic principle and register train wheels entering or leaving a track section. The TVD system counts the wheels and reports a track section to the interlocking system as clear when the number in a section is zero.

The TVD system consists of several distributed computer components linked via communication lines and buses as shown in the example below.

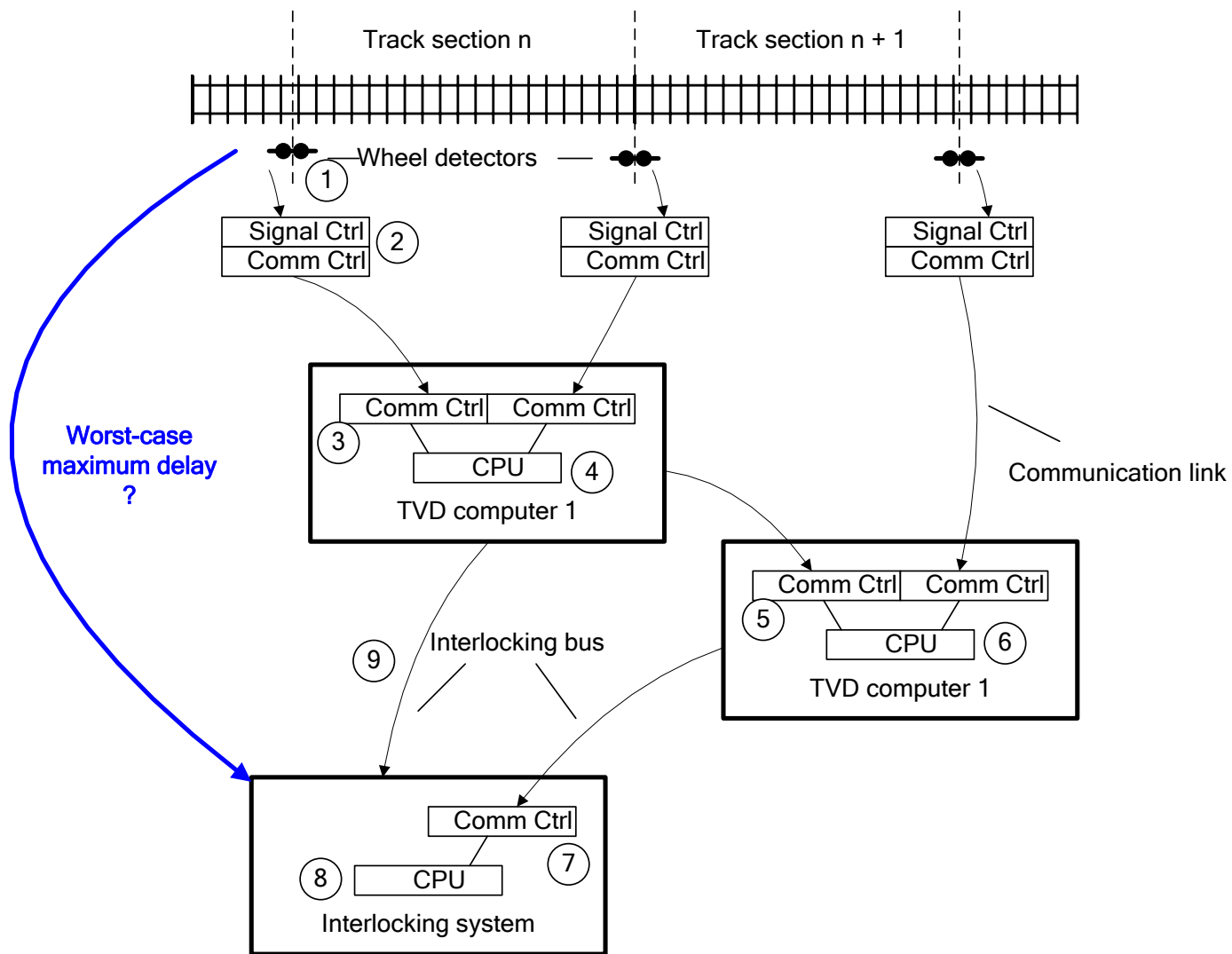


Figure 8: Track vacancy detection system as an evaluation object

When a train’s wheel passes a detector (1), the wheel’s iron mass affects the detector’s electromagnetic field. This influence is recognized by the detector’s electronics (2) and converted into a digital data stream. The detector’s electronics consist of a signal processing component and a communication controller that transfers the signal information to the corresponding TVD computer. The TVD computer receives the detector information with its own communication controller (3) and – within its main CPU (4) – derives the track section-related vacancy information from all detectors on either side of the section. The vacancy information is transferred to the interlocking system (7) (8) via the interlocking bus (9). In larger system configurations, more than one TVD computer is needed. In these cases, not all sensors of a track section can be connected to the same TVD computer; instead, the detector information has to be transmitted to another TVD computer (5) (6).

The following figure shows a more detailed view of the target system from the detector to the TVD computer.

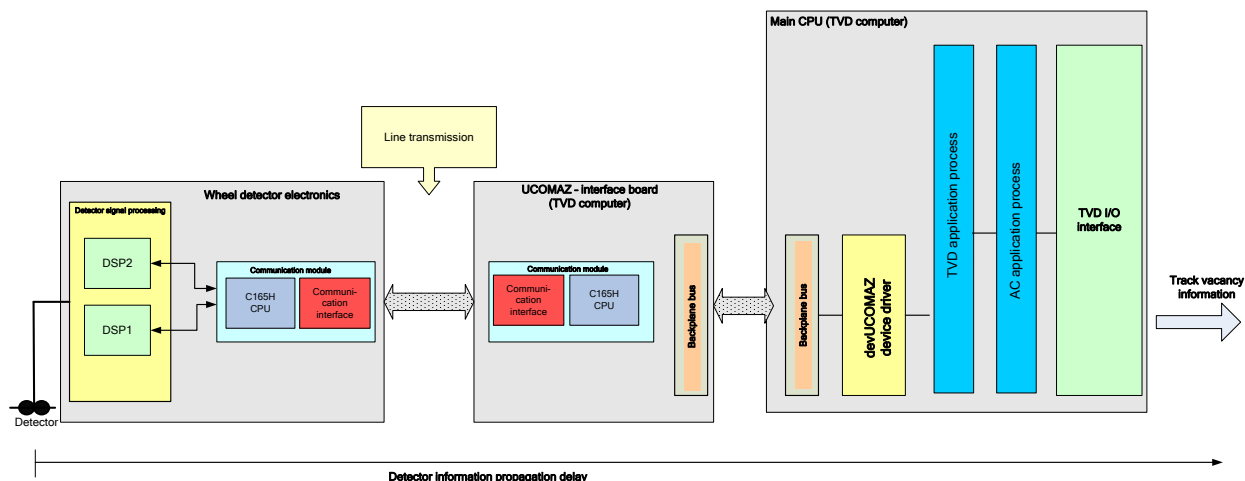


Figure 9: Detailed view of a TVD system

Critical timing requirement

The interlocking system has to be informed as soon as the front of the train’s first wheel passes a detector and changes the track section status from "clear" to "occupied". This worst-case maximum propagation delay from the detector to the interlocking system has to be guaranteed and is currently defined with 300 ms.

$$Constraint\ 1: \Delta t_{delay} = t_{received} - t_{occupied} \leq 300ms$$

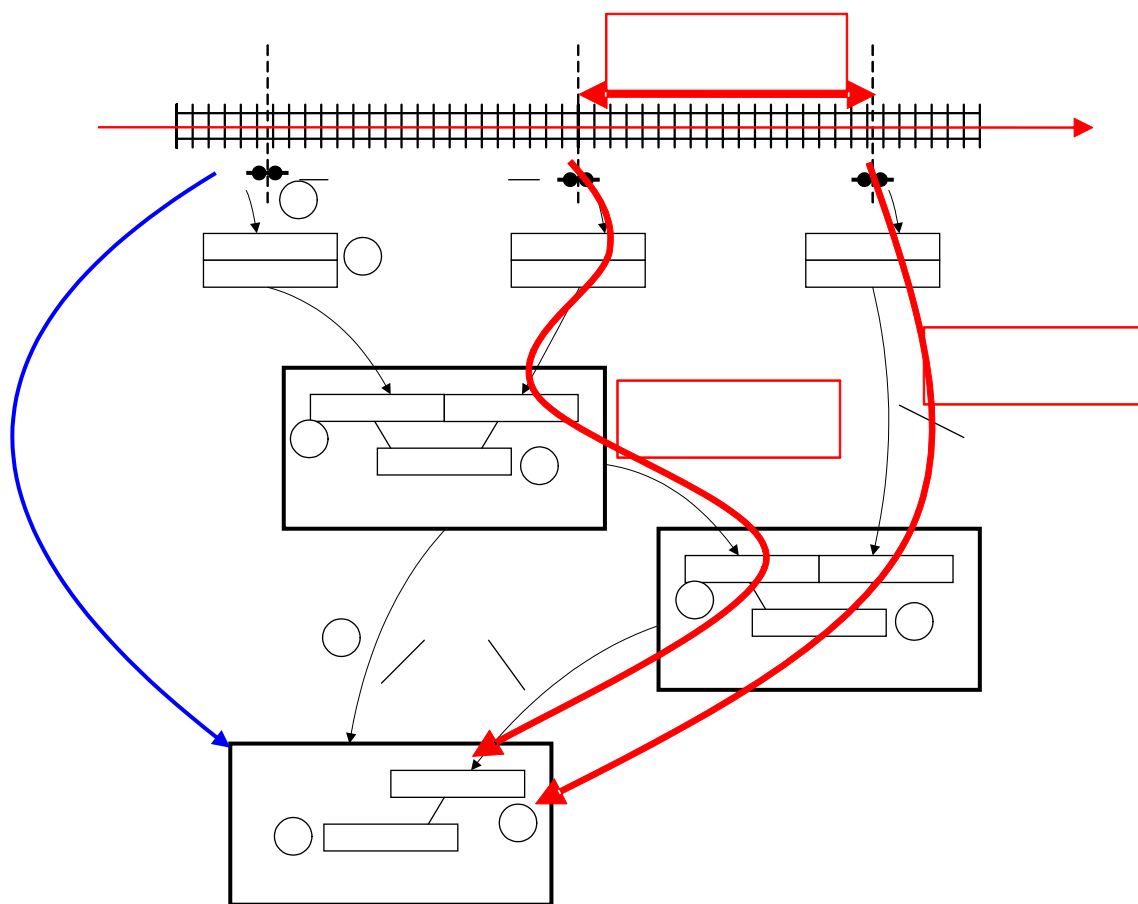


Figure 10: Timing constraint for path delays derived from the wheel detector “occupied” status

Focusing the system from the above diagram, this means that it might be possible that data read at a first wheel detector might be taken over by the data from second occupied wheel detector. This situation leads to misinterpreted train positions and can disrupt the central system up to a track lock where no train is located physically.

In detail, the time difference from occupying two neighboring wheel detectors must be compared with the delay for two different data paths. As a formula, it appears as follows:

$$\text{Constraint 2: } \frac{\Delta t_{\text{sensor1}}}{2} = \frac{\text{distance}_{\text{min}}}{\text{speed}_{\text{max}}} > t_{\text{data-path1}} - t_{\text{data-path2}}$$

The specification requires a minimum distance between two sensors of 100 m at a maximum train speed of 250km/h. This information leads to the time difference of

$$\frac{100\text{m}}{250\frac{\text{km}}{\text{h}}} = \frac{3600\text{s} * 100\text{m}}{250 * 1000\text{m}} = 1.44\text{s} = 1440\text{ms} > t_{\text{data-path1}} - t_{\text{data-path2}}$$

$$\text{Constraint 2: } \frac{\Delta t_{\text{sensor1}}}{2} = \frac{\text{distance}_{\text{min}}}{\text{speed}_{\text{max}}} > t_{\text{data-path1}} - t_{\text{data-path2}} < 1440\text{ms}$$

This means that the difference between the delays of path 1 and path 2 should not exceed ~1500 ms, where the data of path 1 is routed via two TVD CPUs and the data of path 2 is routed via only one TVD CPU.

Current timing solution

Today, the worst-case maximum propagation delay is determined by measurement. The target system is subjected to several heavy load conditions in a system test environment. Under these circumstances, the propagation delays are measured.

The maximum measured interval has to be significantly lower than the required maximum worst-case delay.

In addition, system watchdogs will detect when a maximum propagation delay violation occurs in an actively running system.

4.1.1 Strengths and Weaknesses of the Current Approach

The maximum propagation time can be expected in a peak-load situation: peak activities at the detectors, maximum communication traffic, maximum CPU load. In addition, cyclical internal processes add dependencies from the sample point of time and phase.

Therefore, it is nearly impossible to exactly reproduce the peak-load condition. This situation can only be approximated by producing various load situations, observing the target system for a longer period of time and capturing the interesting timing values.

The watchdogs prevent unsafe consequences of maximum delay violations. However, when a watchdog triggers, it may suspend TVD system operation unintentionally with obstructive impacts for the overall railway system, of course.

In summary, the current timing analysis methods are time-consuming, cost-intensive and imprecise.

4.2 INTERESTED Approach

The idea of the INTERESTED approach is to replace the existing timing determination with measurement and testing by more exact timing analysis methods.

In particular, the following two methods have been evaluated:

1. Static target software analysis based on a model of the target CPU, its memory and buses. This method leads to approximated worst-case timing values and is the working principle of the "aiT" tool from AbsInt Angewandte Informatik GmbH (Section 4.3).
2. Abstract modeling of the timing interaction of system components, communication links and processes. It requires a known timing behavior of the system components themselves. These timing values are determined by measurement. This method is evaluated with the tools SymTAS and TraceAnalyzer from Syntavision GmbH (Section 4.4).

4.3 Worst-case Timing Analysis (Code Level) with aiT

A worst-case timing analysis on the code level was performed with the product "aiT" from AbsInt Angewandte Informatik GmbH. It performs a static code analysis of the target source code transformed into an executable binary value.

For calculation of the execution time, aiT makes use of an internal model of the target CPU. This model reflects the timing behavior of the CPU's caches and pipelines, buses, internal and external memory accesses and instruction execution. Therefore, aiT necessarily requires models of the target system CPUs.

The TVD target system utilizes three different CPU types (Figure 9):

- a digital signal processor (DSP) for wheel detector signal processing
- a C165 communication controller for field detector data transmission
- an Intel CPU as the TVD computer's main CPU.

Of these types, the C165 communication controller is the only one to be supported by an aiT CPU model.

By reason of this restriction and due to time and budget constraints, the communication module software (Figure 9) was chosen for a static code timing analysis with aiT.

The communication module software is a complex system without an underlying operating system. Therefore, it must first initialize the hardware. Also, the remote connection is controlled by the hardware. In the event of an error, the error cause is logged and the system is restarted by a watchdog. The communication module software also frequently performs self-tests. The most timing-critical part of the software is communication with the remote interface. Therefore, this interesting part where messages are sent to the communication channel is chosen for analysis.

As described in Section 4.3.2, aiT requires some input specifications such as file names and CPU settings. In addition, some user annotations must be given in case the value analysis cannot compute the correct values of the registers. User annotations can also be used to exclude paths from the analysis in case these are not part of normal program execution.

In the case of a communication error, the hardware is reset and the communication channels must be instantiated. In this case, it is clear that the system will exceed its timing constraints and log this malfunction. Therefore, the error handlers and exception routines as well as the logging routines are excluded from the analysis by a user annotation (see Section 4.3.3.3).

The messages that are sent via the communication channel are copied from memory areas. Therefore, the communication module software makes use of the **memcpy** and **strncpy** functions which come with the used compiler. These routines contain loops for which the number of iterations depends on the input. In the case that the value analysis cannot compute the boundaries of the loop due to external constraints, a

default user annotation (see Section 4.3.3.3) is given. The value for the number of iterations is derived from the maximum length of the input data.

Calls to functions via function pointers are commonly used in C programs. aiT can handle such calls if the value analysis can determine the target of the function call. However, if the target is computed by a program, aiT may not compute the call target automatically. Two function calls of the communication module software calls could not be determined by aiT's value analysis and were therefore annotated by user annotations (see Section 4.3.3.2). The targets of the calls which can be either a number of functions, an array of functions or parts of a C-struct, were looked up in the source code.

High overestimations may stem from imprecise value analysis results for memory accesses. For safe computation of the worst-case execution time, the analyzer must assume that these memory accesses target memory regions with the worst possible timing behavior such as buses for example. The analysis can be supported by user annotations for certain areas (see Section 4.3.3.3). The scope of these annotations can be adjusted from the instruction level to a global scope, with instruction level annotations at the highest priority. The memory areas to which accesses should be mapped could be derived from the source code and the layout of the executable binary value. As only a few memory accesses target the bus, the limit of the worst-case execution time could be lowered significantly by these annotations.

4.3.1 AbsInt's WCET Analyzer aiT

aiT is AbsInt's timing analyzer which can find upper limits for the worst-case execution times (WCETs) of sequential tasks. For a precise computation of WCET, aiT operates on the executable binary file. The tool exists in different versions, depending on the target architecture.

4.3.2 aiT Usage

aiT features a comprehensive GUI (Figure 11) to specify the memory architecture of the target, the location of source files, the name of the executable binary file, the name of a separate parameter file called .ais file, the name of the report files to be written, etc., and the starting point of the analysis (a routine name or an address). All this information can be stored in a project file (.apx file). The .ais file may contain the clock rate of the target processor, the upper boundaries for the iteration numbers of loops, possible targets of computed calls, etc. (see Section 4.3.3).

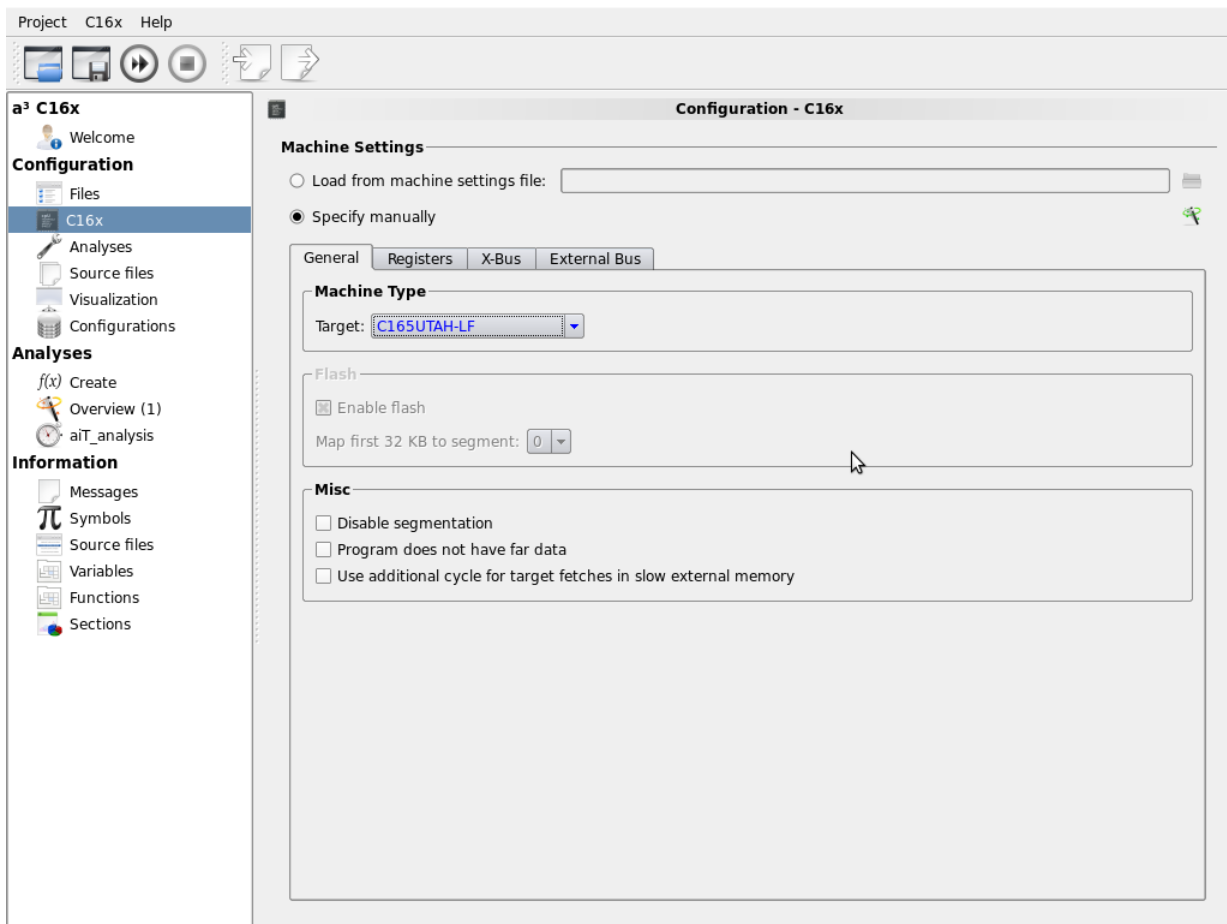


Figure 11: Graphical user interface of aiT

The analyses of aiT are mainly based on the executable binary file. If available, aiT can also read the source files for further information. The association between addresses in the executable binary file and positions in the source files is obtained from the debug information in the executable binary file.

aiT can be started in three ways:

- 1) When started for interactive usage, the GUI depicted in Figure 1: the graphical user interface of aiT is opened. The fields in the GUI can be filled with appropriate values which may be stored in a project file (.apx file). Alternatively, an existing project file can be loaded. Multiple analysis tasks with different analysis entry points can be defined.
- 2) aiT can also be started in batch mode with a project file. In this case, the project file is loaded and then aiT behaves as if the “Start all analyses” button has been clicked.

After a successful analysis, aiT reports its results in several ways:

- 1) aiT can produce a graphical output showing the call graph of the analyzed part of the application, depicting the routines and their calling relationships (see Figure 12). The routine boxes can be opened to show their control flowcharts with WCET results or stack levels for basic blocks (see Figure 13). Technically, aiT writes a description of these charts into a GDL file which can be visualized by AbsInt’s graph browser aiSee.
- 2) aiT can write a text report which is meant to be human-readable and a more formal XML report (see Section 4.3.3.4.2). These reports contain detailed results for all analyzed routines in all calling contexts, including specific results for the first few iterations of loops versus a result for the remaining iterations.

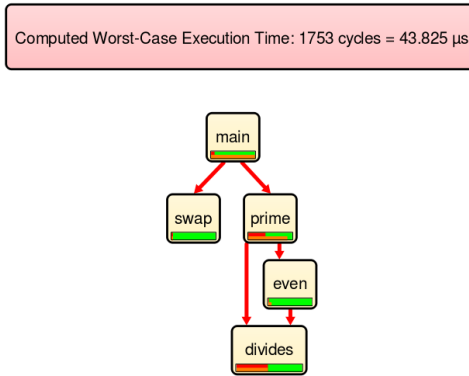


Figure 12: Call chart with WCET result

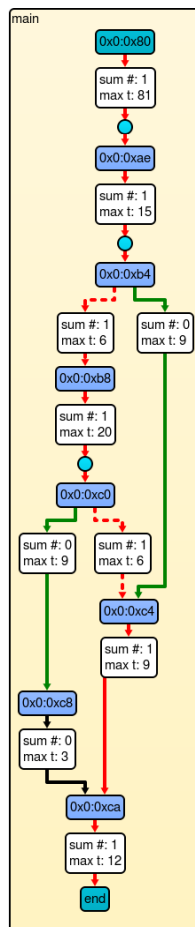


Figure 13: Control flowchart with basic block timings

4.3.3 AIS Annotations

To perform a successful analysis, aiT requires some information that is not explicit in the executable binary file , e.g. upper boundaries for the iteration numbers of loops. It employs some static analyses to obtain this information by itself, but does not succeed in all cases (the underlying problems are usually undecidable). Therefore, aiT offers a way for users (or partner tools) to specify the missing information. This can be done either in separate parameter files (.ais files) or as special AIS comments in the source code.

In this document, we describe the most important kinds of AIS annotations only. A complete description is contained in the user manual of aiT for the various target architectures. We start with a description on how to specify program points in AIS notation.

4.3.3.1 Specification of Program Points

Information in AIS format needs to refer to program points for various reasons, e.g. as points of reference or as call targets.

- The simplest way to refer to a program point is by its absolute address in the executable binary file. Yet this kind of program point specification is not very convenient. If the application is modified and then recompiled, the absolute addresses will change and need to be adapted. This is true even for unmodified routines since they may move to a different location in the memory.
- These problems can be avoided to some extent by relative addressing, e.g.
`"main" + 0x20 bytes`
describes a program point by an address relative to the entry of the routine main. If the application is modified and then recompiled, only the relative addresses in the modified routines must be adapted.
- Program points may also be specified by symbolic descriptions. For instance,
`"prime" + 2 loops`
is the beginning of the second loop in the routine prime,
`"find" + 3 reads`
is the third instruction reading from memory in the routine find, and
`"watch" + 1 call`
is the first call instruction in watch.
- Further examples of symbolic descriptions will be presented in the next few sections.
- `file 'Name' line Number` refers to line *Number* in source file *Name*. The translation of this line into an address in the executable binary file relies on the line information that is part of the executable binary file.
- The program point description `here` can only be used in source code annotations and then refers to the line where the annotation starts. This line information is translated to an executable address in the same way as the explicit line numbers presented above.

The subsequent sections describe the most important AIS annotations to be generated by the partner tools.

4.3.3.2 Targets of Computed Calls

Here, "computed call" means a call instruction where the target address is computed from register contents. Non-trivial applications may contain some computed calls and branches (in hand-written assembly code) that cannot be resolved by the decoder module in aiT; these unresolved computed calls and branches require user annotations. If no such annotations are given, appropriate warning messages are issued. Such annotations have to list the whole set of possible targets of computed calls and branches:

```
INSTRUCTION ProgramPoint CALLS Target1, ..., Targetn;  
INSTRUCTION ProgramPoint BRANCHES TO Target1, ..., Targetn;
```

The *ProgramPoint* which refers to the computed call or branch instruction and the *Targets* are points as defined by the previous section; they may be absolute or relative addresses or symbolic descriptions. A program point description particularly suited for **CALLS** and **BRANCHES** specifications is

```
"R" + n COMPUTED
```

which refers to the *n*th computed call or branch in routine *R* - counted statically in the sense of increasing addresses, not dynamically following the control flow.

If the application contains an array *P* of function pointers, then a call *P*[*i*](*x*) may branch to any address contained in *P*. aiT tries to obtain the list of these addresses automatically: If the array access and the computed call in the executable binary file are part of a small code pattern as typically generated by the compiler, aiT notices that the computed call is performed via this array. If furthermore the array contents are

defined in a data segment so that they are statically available, and the array is situated in a ROM area so that its contents cannot be modified, then aiT automatically considers the addresses in the array as possible targets of the computed call. If array access and the computed call are too far apart or realized in an untypical way, aiT cannot recognize that they belong together. The array belonging to the computed call can be declared by the user. The declaration starts like the ones described above:

```
INSTRUCTION ProgramPoint CALLS VIA ArrayDescriptor ;
```

Here, the ***ArrayDescriptor*** describes the address and the format of the table that contains the call targets. These targets are extracted from the table according to the given format rules.

4.3.3.3 Loop Boundaries

WCET analysis requires that upper boundaries for the iteration numbers of all loops be known. aiT tries to determine the number of loop iterations by loop boundary analysis, a combination of value analysis and pattern matching which looks for typical loop patterns. In general, these loop patterns depend on the code generator and / or compiler used to generate the code that is being analyzed.

Despite all sophistication integrated into the loop boundary analysis, there may be some loops that are too complicated for automatic analysis. Boundaries for such loops must be provided by user annotations.

Loop boundary specifications can be classified in two different ways. On the one hand, there is a distinction between local loop boundary specifications applicable to a single loop and global annotations applicable to all loops in a set of routines. On the other hand, there is a distinction between absolute specifications with a higher priority than automatic results and default specifications with a lower priority. All global specifications are default specifications, and so all absolute specifications are local specifications.

The various ways to obtain loop boundaries interact as follows:

1. Absolute loop boundary specifications have the highest priority. They supersede any automatic results and default specifications.
2. Automatic loop boundary detection is applied to all loops, but its results are only used at loops without absolute loop boundary specifications. More exactly, this means that aiT tries to run the loop as often as the user has specified, but this may prove impossible so that the automatic boundary seems to prevail or lead to contradictions if the user specification is wrong.
3. A default local specification for a loop only takes effect if there is neither an absolute specification nor any automatic result for that loop.
4. Global loop annotations apply only to loops for which there are neither local loop annotations nor results of the automatic loop boundary analysis.

4.3.3.3.1 Local Loop Boundary Annotations

A fixed maximum iteration number of *j* is specified as follows:

```
LOOP ProgramPoint Qualifier MAX j ;
LOOP ProgramPoint Qualifier MAX j BY DEFAULT;
```

The first form is an absolute local loop boundary and the second a default local loop boundary (cf. above).

A ***ProgramPoint*** description particularly suited for this purpose is

```
"R" + n LOOPS
```

which means the *n*th loop in routine ***R*** counted from 1.

Qualifier is optional information. It may be one of the following: **begin** indicates that the loop test is at the beginning of the loop, as for C's while-loops. **end** indicates that the loop test is at the end of the loop, as for C's do-while-loops. If the qualifier is omitted, aiT assumes the worst case of the two possibilities, which is **begin**. This means that the loop test is executed one more time. The begin / end information refers to the executable binary file, not to the source code; the compiler may move the loop test from the beginning to the end, or vice versa.

Example:

```
loop "_TDDLL_CheckTxFrameByID" + 2 loop end max 255 ;
```

specifies that the second loop in `_TDDLL_CheckTxFrameByID` has the loop test at the end and is executed 255 times at the most.

4.3.3.3.2 Global Loop Boundary Annotations

Global loop boundary specifications provide information about all loops in a set of routines or even the entire executable binary file. They exist in different forms. The most important one for INTERESTED is the following:

```
LOOP-ITERATION DEFAULT Routines IS Qualifier MAX j;
```

In this form, *Routines* stands for a comma-separated list of routine names (enclosed in double quotes as usual). The annotation applies to the listed routines (including all loop routines extracted from these routines). As indicated by the word **DEFAULT**, this global loop boundary specification applies to all loops in the listed routines that are not limited by other means (loop boundary analysis or local loop boundary specifications (absolute or default)).

4.3.3.3.3 Various Annotations to Improve the Precision of the WCET Analysis

With the following two annotations, the precision of WCET analysis can be improved if the value analysis results in the report show that there are unknown memory accesses in the analyzed routines, or if certain code snippets are not part of the normal control flow.

Using the values of the registers, value analysis tries to determine the addresses of memory accesses. These addresses are important for an analysis of the data cache and for determining the duration of the memory accesses. Value analysis usually works so well that only a few indirect accesses cannot be determined exactly. Address ranges for these accesses may be provided by user annotations of the form

```
INSTRUCTION ProgramPoint ACCESSES Range;
```

Useful *ProgramPoint* formats for such specifications are simple instruction addresses and symbolic descriptions of the forms

`"R" + n READS` and `"R" + n WRITES`

meaning the *n*th instruction in routine *R* reading from memory and the *n*th instruction writing to memory, respectively.

A *Range* may be a single position in the memory, or a range specified by a start and end position, or an array name meaning the memory area covered by that array (this means that the debug information in the executable binary file has to contain the start address and the end address or the length of the array). A position may be an absolute address in the memory or an address relative to the beginning of an array (this means that the debug information in the executable has to contain the start address of the array).

Value analysis can find certain conditions to be always true or always false for some contexts. Yet value analysis cannot predict all register values, in particular if they depend on input data. Conditions can be provided by user annotations of the form

```
CONDITION ProgramPoint IS ALWAYS false EXCLUDE;
CONDITION ProgramPoint IS ALWAYS true MAKE INFEASIBLE;
```

The *ProgramPoint* should be a conditional branch. *Exclude* excludes the path which is not taken from being decoded and analyzed. *Make infeasible* lets aiT decode the path of the branch which is not taken, but, by marking it as “infeasible”, aiT is aware that the path does not contribute to the worst-case execution time.

4.3.3.4 XML Result Files and XML Report Files

Besides a textual report file meant to be read by humans, aiT offers two kinds of XML output to report on its analysis results: XML result files and XML report files

4.3.3.4.1 XML Result Files

XML result files describe the result of a single timing analysis action. This result basically is a single number. Consequently, result files have a very simple form, for instance

```
<!DOCTYPE XmlResults>
<results>
<result type="aiT" id="aiT_0">
<cycles>103731</cycles>
<time>2.922 ms</time>
</result>
</results>
```

The unit of measurement is processor cycles in the case of timing analysis.

XML result files are used to communicate analysis results from aiT/StackAnalyzer back to the aiT driver which implements the XTC interface for the AbsInt analyzers.

4.3.3.4.2 XML Report Files

XML report files are much more complicated. They may report on several analysis actions at once and include all information necessary to reproduce the analysis results, all warnings and error messages produced during the analyses, and a host of intermediate and auxiliary results.

The root element of an XML report file is `<a3>`. It consists of the following subelements:

- `<start>` contains a time stamp of the analysis start.
- `<project>` contains the relevant input of aiT (name of the analyzed executable binary file(s) and of the global parameter files, values of some flags, processor-specific configuration information, etc.) and the analysis tasks.
- `<wcet_analysis_task>` is a list of `<wcet_analysis_task>` elements. A `<wcet_analysis_task>` is a bunch of analysis actions. The bundling of analysis actions into tasks is performed automatically in such a way that all actions in a task can be performed after a single decoding step. Consequently, a `<wcet_analysis_task>` element starts with a subelement `<decode>` containing a description of the decoded part and messages from the decoder. Then, the list of decoded routines follows (element `<routines>`), and information on the mapping of code addresses to source code locations (element `<locations>`). Finally, the analyses belonging to the task are listed as a sequence of `<analysis>` elements (see below).
- `<computations>` is a list of `<computation>` elements describing computations. Computations can combine the results of analyses and of other computations by means of operations such as addition and maximum formation.

- `<end>` concludes the report of the entire run of aiT having a time stamp of the analysis end.

Each analysis action performed during the aiT run is documented as an `<analysis>` element. Such an `<analysis>` element is a sequence of elements corresponding to the various steps of a timing analysis (loop analysis, value analysis, pipeline analysis, etc.) The elements corresponding to these analysis steps contain messages issued by the steps (if any) and auxiliary results produced (if any). The last element of a timing analysis is `<wcet_analysis>` which documents the WCET contributions of the various routines in element `<wcet_results>` and the overall WCET in element `<wcet>` which may, for instance, look like this: `<wcet>103731 cycles = 2.922 ms</wcet>`

4.3.4 Strengths of the aiT Approach

The aiT tool provides timing analysis capabilities for the executable code on a dedicated CPU type based on static code analysis.

The prerequisites for this technology are:

- The target source code and the resulting executable binary file must exist.
- The target programming language, compiler and executable data format must be supported by aiT.
- The target CPU type and performance-relevant characteristics of the hardware platform such as cache and memory access rates must be known in detail. The running target hardware itself is not required.
- aiT relies on an internal software model of the CPU hardware. Therefore, the target CPU type must be supported by aiT.
- The static code analysis is, in principle, unable to determine the program control flow at the run time
 - if the control flow depends on events and data from external interfaces
 - if the control flow depends on decisions computed elsewhere in the code

This information has to be provided by the user otherwise and added to the analysis manually.

- The program flow is not affected by the activities of an underlying operating system such as task scheduling or interrupt handling at the run time.

In practice, analysis precision is mainly influenced by the quality and amount of the user-supplied add-on information.

With the prerequisites fulfilled, the worst-case execution times for software routines and functions can be determined in conjunction or separately.

In this way, worst-case software execution times can be calculated when the appropriate hardware is not yet available or to ascertain that a planned hardware will be powerful enough.

Considering the special characteristics of the static code analysis approach, aiT is suitable to compute fine-grained worst-case software execution times on a detailed software level.

4.3.5 Weaknesses of the aiT Approach

The most important limitation for aiT results from its prerequisites. In the case of the TVD target system, only the communication module was analyzable, simply caused by the CPU type restriction. Also, research for and preparation of the necessary user-supplied information can be time-consuming work.

aiT analyzes software execution only. Therefore, the timing of hardware components, system resources such as communication buses, operating systems and distributed multi-computer systems are not in the aiT focus.

For an entire system timing analysis, aiT could be combined with tools on the system level where aiT provides the low-level software execution times.

4.4 Worst-case Timing Analysis (System Level) with SymTA/S

The following multi-step approach was focused on in this project:

1. identification of timing-relevant system architecture parts to identify the important system resources (CPUs, buses, communication paths, I/O devices, etc.)
2. identification of internal scheduling elements (signals, frames, tasks, processes, etc.)
3. specification of timing requirements and parameters (scheduler algorithm, process priority, task runtimes, activation behavior, etc.)
4. set-up of the entire system timing model
5. refinement of the model based on system log data
6. performance analyses for resource internals (load, delays, required buffer sizes, etc.)
7. analysis for maximum end-to-end delays starting from wheel detectors and ending up in the TVD computer
8. comparison of worst-case analysis results with constraints
9. identification of optimization potential and strategies

The system timing analysis is provided by modeling the target systems with **SymTA/S** where the SymTA/S timing model approximates and reflects the real target system timing behavior. The timing model is mainly derived from configuration information obtained from the developer's knowledge.

The timing behavior is identified from real systems by using trace or log files which are imported into the Symtvision **TraceAnalyzer**. Log files from the TVD computers are imported into the TraceAnalyzer and examined for dedicated timing behavior. The results can directly be used within SymTA/S and support analyzing for realistic worst-case results.

Based on the abstract system model and those measured timing values, SymTA/S is used to determine the required worst-case timing of the TVD system.

4.4.1 Identification of the System Architecture

The main information about the entire system architecture is obtained from the TVD target system description (see Section 4.1). The main parts are:

- wheel detector (WD)
- WOM interface board (reading wheel detector data)
- TVD computer (processing wheel detector data)
- interlocking system (processing all system data)

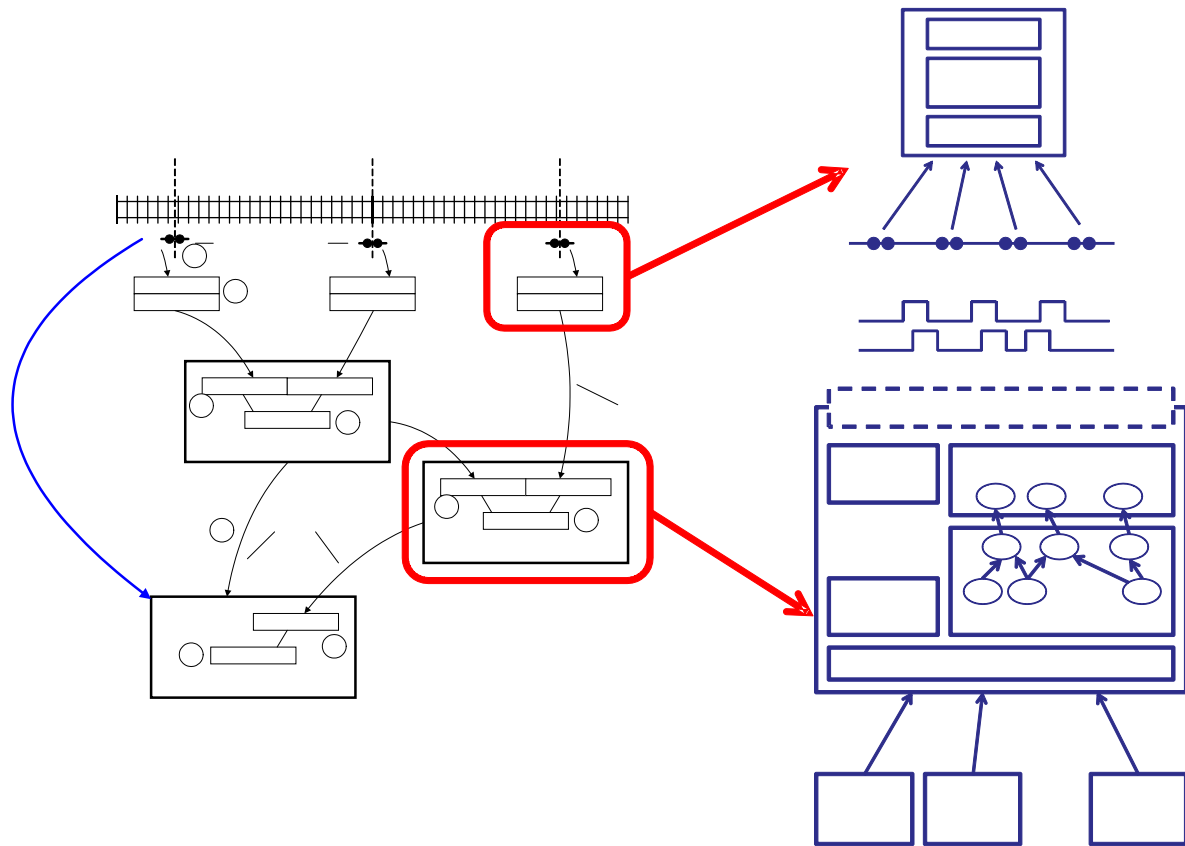


Figure 14: Approximated internals incl. simplification for the first timing model

It was identified that the main focus should lie on the delays starting from the wheel detector and ending at the central interlocking system.

4.4.2 Identification of Internal Scheduling Elements

This step concentrates on the components that are relevant for performing scheduling analyses. Some physical existing elements need not be reflected in the SymTA/S timing model; thus, some approximation and simplification have been done.

Wheel detector

The wheel detector is directly connected to a WOM interface board that reads and prepares wheel data for further processing. Internally, the WOM has a DSP which scans the wheel cyclically at intervals of 50 μ s. Thus, the timing-relevant part uses signal data at intervals of 50 μ s at the TVD computer.

Communication link between WOM and TVD computer

The communication link is a physical direct connection without significant delays due to scheduling effects.

TVD computer

The TVD computer uses an operating system which schedules tasks by priorities. Most tasks can be preempted by higher-priority tasks. The main tasks to focus on are the TVD task and the AzAnw task. These tasks are responsible for processing wheel detector data and have a priority somewhere in the middle range. Their scheduling details are:

- TVD: period of 80 ms, core execution time [4 ms – 32 ms]
- AzAnw: period of 100 ms, core execution time [1 ms – 8 ms]

The higher- and lower-priority tasks are again approximated using modeling possibilities from SymTA/S, where all high-priority tasks in sum produce a load of ~5% and the low-priority tasks a load of ~30%.

Interlocking bus between TVD and central interlocking system computer

The interlocking bus is an in-house Siemens solution and uses round-robin scheduling for transmitted frames. The frames have a period of typically 100 ms.

4.4.3 Specification of Timing Requirements and Parameters

The challenge is that the architecture supports the cascading of TVD computers. Due to complex architecture constellations and different ways taken by the wheel detector data, it might be possible that wheel data read from a detector physically before wheel data from another detector, the wheel information from the second detector arrives earlier possibly due to scheduling effects.

SymTA/S should be used for identifying those insufficiencies where different scheduling effects are taken into account.

Detailed scheduling behavior is taken from real system log files where the TraceAnalyzer analyzes for main timing data. This information is used within SymTA/S for refining timing model data.

4.4.4 Set-up of the Entire System Timing Model

In the first step, no log file information was available. Due to this initial situation, a rough timing model inclusive estimated timing data was taken for setting up the SymTA/S model.

4.4.4.1 First SymTA/S Model

At first, the architecture model from above (see **Figure 8**) was taken into account. It was translated into:

- two TVD CPUs
- one BUREP bus
- one auxiliary CPU per TVD CPU required for detailed modeling support in SymTA/S

The interlocking system receiver side was ignored, because only the delays on the way to the central system are of interest, not the internals. Thus, the following architecture was set up:

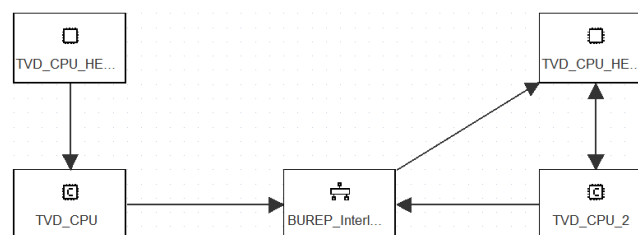


Figure 15: SymTA/S architecture model incl. communication and activation directions

In addition to the architecture, the elements incl. the estimated core execution times from Section 4.4.2 were considered. The TVD CPU internals consist of:

- TVD task
- AzAnw task
- high-priority tasks
- low-priority tasks
- internal communication between tasks

- inter-resource communication between two CPUs using bus communication

The BUREP bus was approximated with frames sent by the round-robin communication schedule. The following frames were added:

- one receiver frame transmitting data from the first TVD CPU directly to the central interlocking system
- one receiver frame transmitting data from the second TVD CPU to the central interlocking system
- one intermediate frame transmitting data from the first to the second TVD CPU
- two additional frames representing additional traffic on the BUREP bus

TVD CPU-internal and BUREP interlocking bus scheduling was approximated as follows:

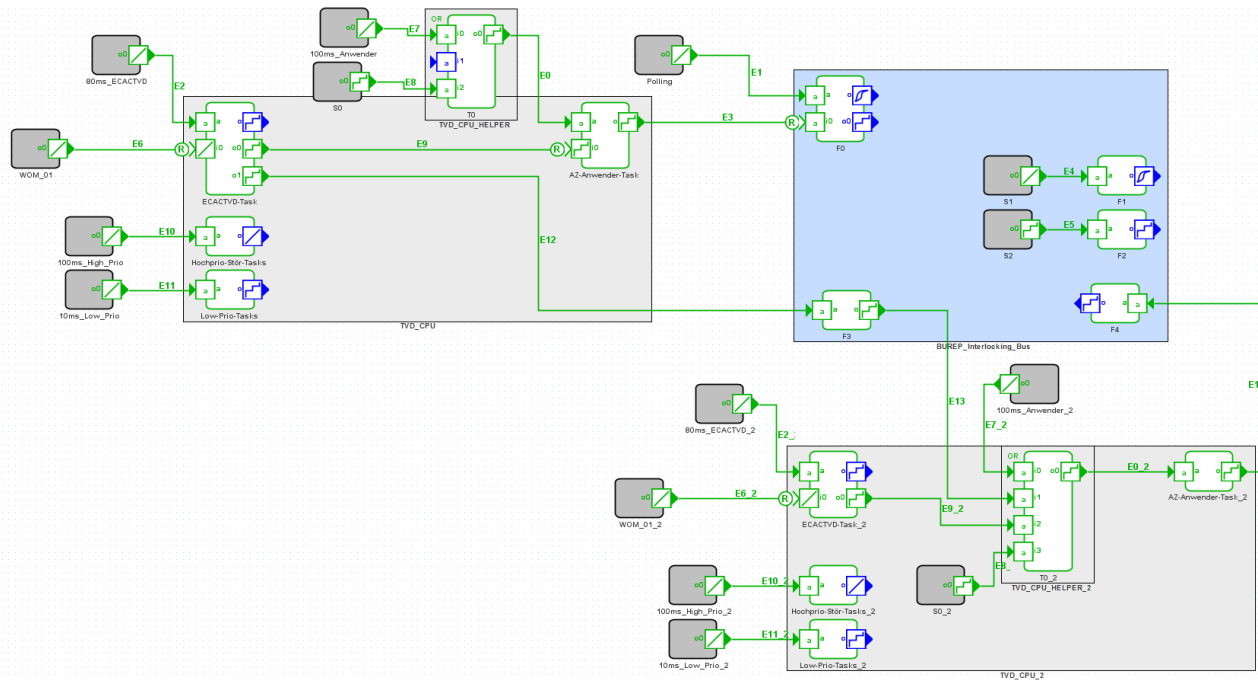


Figure 16: TVD CPU and BUREP interlocking bus details

The large gray areas represent the TVD CPU-internal timing model. The blue area represents the BUREP interlocking communication bus.

The white bubbles within the gray areas represent the scheduled tasks and, within the blue areas, the scheduled frames.

The gray small bubbles represent external signal data or external events that are responsible for activating tasks.

Green borders and edges indicate a successful analysis status.

4.4.4.2 Results from Set-up of the First Timing Model

The first model was identified as a good starting point for discussions about the following details:

- What does scheduling analysis take into account?
- Which architecture details must be considered?
- Which elements influence the analysis?
- Which parameters are available and which can only be estimated?

- What kind of communication must be considered?
- Which parts can be ignored or roughly approximated?

The discussion results were partly reflected in the previous section. The most important result was to obtain a much better understanding about the system internals.

One of these highlights was that scheduling analysis does not need detailed information in the first step. The timing model can be refined during the project runtime after obtaining more and more knowledge about the system at all.

Another advantage was identified when using budgets or estimates instead of real runtime data. This was heavily used for the first timing model due to unavailable tracing / log files.

Finally, as a third advantage, it was identified that not every system detail must be modeled in detail. So the WOM interface boards were approximated by using only an element providing signal data in a dedicated period (50 μ s). Another approximation was used at the TVD CPUs or the BUREP bus internally. There, the high-priority tasks or additional bus traffic was approximated with single timing elements enriched with dedicated timing parameters.

Details about the analysis results are skipped here, because refined timing data and modeling details were able to be achieved as described by the following sections.

4.4.5 Refined SymTA/S Model based on TraceAnalyzer Results

One of the most important timing parameters for CPU schedules is the core execution time (CET) which represents the net runtime of a task or an executable binary file without any preemption. One way to obtain this value is by using tools from AbsInt such as a3 (formally known as aiT) which performs static code analysis based on the target object code. Another approach is to use trace data where any task or executable binary file executed by the CPU is logged with four main statuses:

- Activate
- Start
- End
- Terminate

In addition to these statuses, additional statuses can be logged:

- Preempt
- Resume
- Wait
- Release

Symtvision's TraceAnalyzer is able to consider all these statuses and finally analyzes for the following relevant timing data:

- CET (core execution time): net runtime without preemptions
- EET (effective execution time): runtime from start to end incl. preemptions
- RT (response time): runtime from activation to end incl. preemptions
- Load (overall and bin load over time)
- Burst duration (occupied lengths of a CPU or bus)
- Event distances (distances from activate to activate, start to start, end to end, terminate to terminate)
- Interrupt event models (activation model for non-periodic tasks or frames)

The analyzed results from traces might not reflect the worst-case values, because the trace reflects only the system behavior based on external stimulation. Thus, if the activation patterns used are not able to force the system going towards border areas, it cannot be guaranteed that the timing model using this data predicts

the real worst case at all. Keeping this information in mind and comparing it with the initial situation using budgeting or estimation, using trace files is much more detailed and is comparable with simulation results.

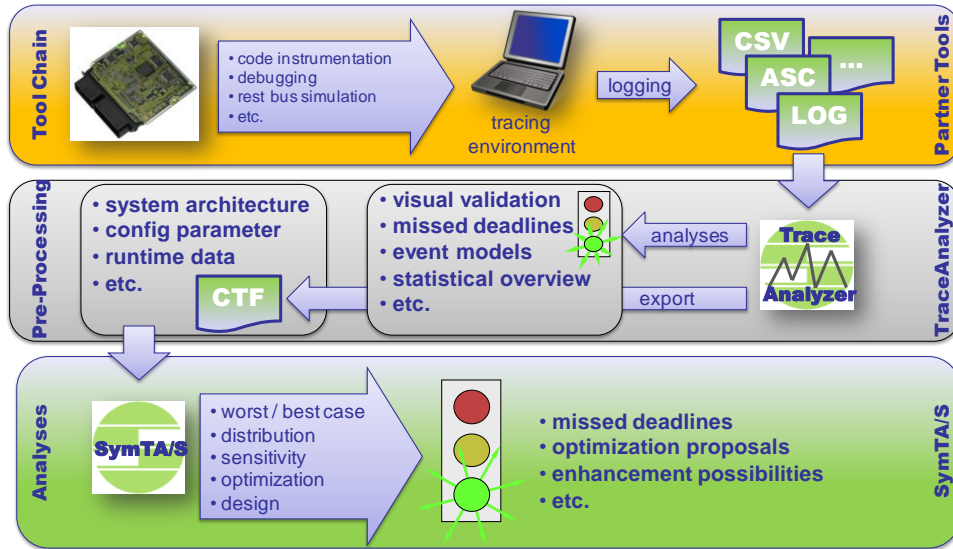


Figure 17: Typical TraceAnalyzer workflow

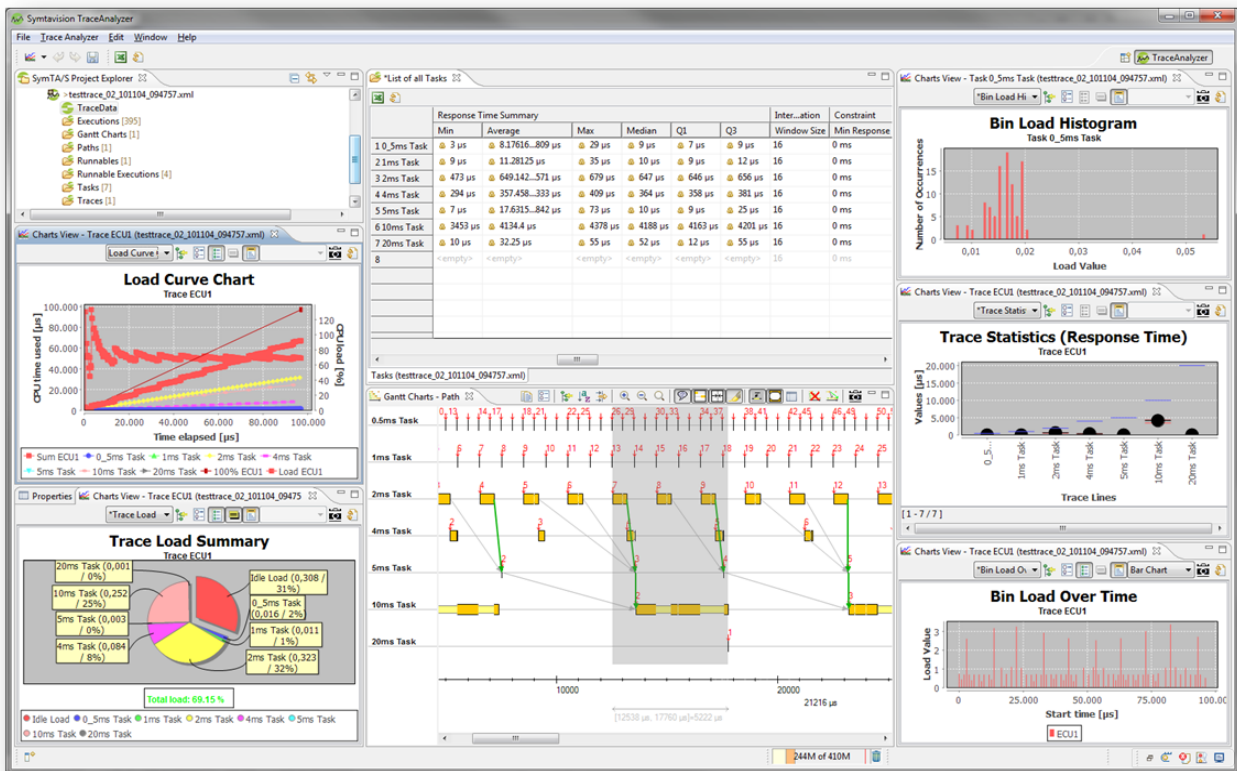


Figure 18: TraceAnalyzer GUI overview

With the help of Symtvision’s TraceAnalyzer by examining TVD computer log files, executed tasks inclusive related timing could be identified. In detail, it was possible to analyze for the following data:

- unknown tasks

- task names
- task activation
- task core execution time (net runtime without preemptions)
- task priorities

With the help of this information, the SymTA/S timing model was refined and the following details were added:

Task name	Priority	Core execution time summary (min.)	Core execution time summary (max.)
System	26	10.648 ms	607.312 ms
interruptcontroltask	27	0.297 ms	19.425 ms
Burep_Sendprozess1	30	2.701 ms	6.719 ms
Burep_Empfangsprozess1	31	0.814 ms	2.088 ms
Burep_Sendprozess2	32	4.523 ms	2.2911 ms
Burep_Empfangsprozess2	33	0.898 ms	2.615 ms
checker	37	2.660 ms	3.557 ms
tango	38	0.570 ms	9.187 ms
devUCOMAZ receiver task	40	2.172 ms	32.274 ms
diagmon	42	2.963 ms	9.652 ms
Cygly_Control_Task	46	0.385 ms	6.490 ms
Secly_Control_Task	47	0.750 ms	6.646 ms
Icio	48	1.141 ms	2.274 ms
AzAnw_task	49	0.941 ms	9.105 ms
TVD_Application	50	0.270 ms	16.842 ms
Abzpb_task	52	1.971 ms	10.456 ms
SWUpload	53	1.056 ms	1.056 ms
babtransfer	57	0.645 ms	3.548 ms
Background	126	948.977 ms	948.977 ms

The following sections reflect the results from the refined model based on the TraceAnalyzer data.

4.4.6 Performance Analyses for Resource Internals

SymTA/S supports analyzing for worst-case scenarios based on tracing results. This means that, even if within the trace the worst-case scenario is never found due to incomplete test scenarios, SymTA/S is able to synthesize a worst-case situation based on trace / log data.

Coming back to the scheduled tasks, the focus must be set on the task response times. The following maximum delays are identified for the TVD and AzAnw tasks:

- TVD: [min;max] response times = [0.27 ms ; 789.906 ms]
- AzAnw: [min;max] response times = [0.941 ms ; 780.748 ms]

Details about the worst-case schedules for the TVD and AzAnw tasks are shown in the following figures.

The red vertical arrows represent task activation, light yellow areas represent time spans where a task should be executed but could not due to preemption from high-priority tasks, and finally the dark yellow areas represent time spans where the task is executed. The red horizontal arrow shows the 'worst-case response time' (wcrt) for the focused task. Figure 19 nicely shows that all higher-priority tasks may preempt the TVD

task for a long time (wcr = 1540.094 ms), because, in the case of unsynchronized situations, all tasks may be activated at the same time. A similar situation occurs for the AzAnw task.



Figure 19: Worst-case scenario for the TVD task

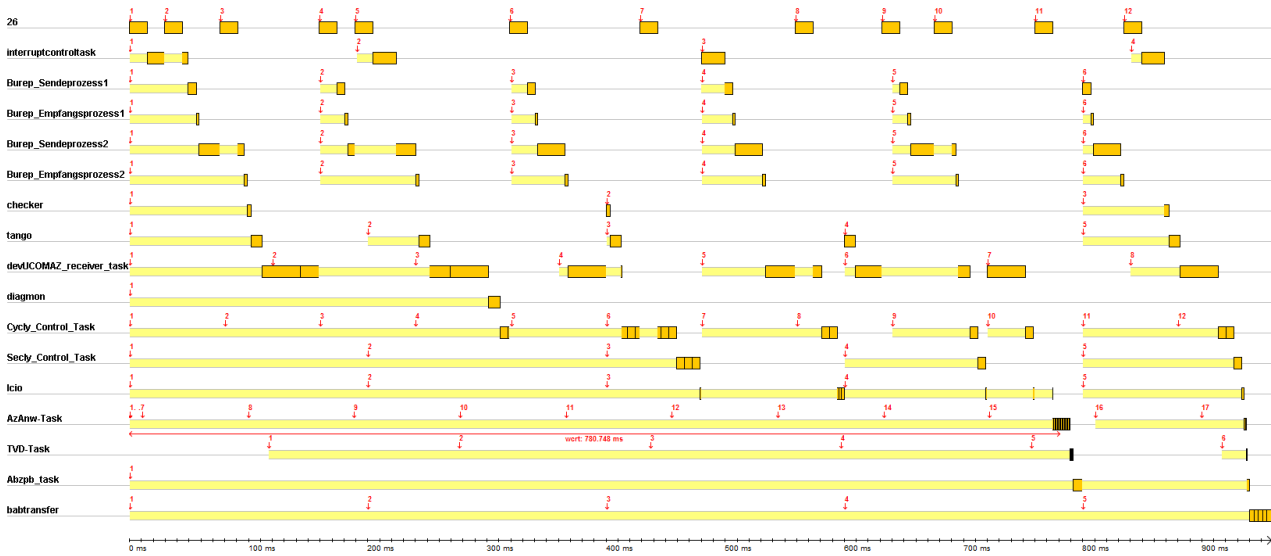


Figure 20: Worst-case scenario for the AzAnw task

Comparing these results with 'Constraint 1' in Section 4.1 show that this constraint is violated by more than a factor of 2:

- TVD: max. response time = 789.906 ms > 300ms
- AzAnw: max. response time = 780.748 ms > 300ms

The reason for this heavy constraint violation might have multiple sources. One reason is often that the timing model is not detailed enough regarding internal correlations. For example, it is currently not considered which task execution depends on which other. Offset scenarios are also not taken into account here, which has a main impact on the preemptions for lower-priority tasks, such as TVD and AzAnw.

Finally, summarizing these modeling weaknesses, SymTA/S provides a conservative worst-case analysis result here which has to be refined for the purpose of obtaining more realistic results.

4.4.7 Analyzing for Maximum End-to-end Delays

There exist two main data paths in the focused system. The first one transfers wheel detector data via only one TVD bus and one BUREP bus directly to the central interlocking system. The second path transfers wheel detector data via two TVD buses and one BUREP bus, but bus transfer is used twice.

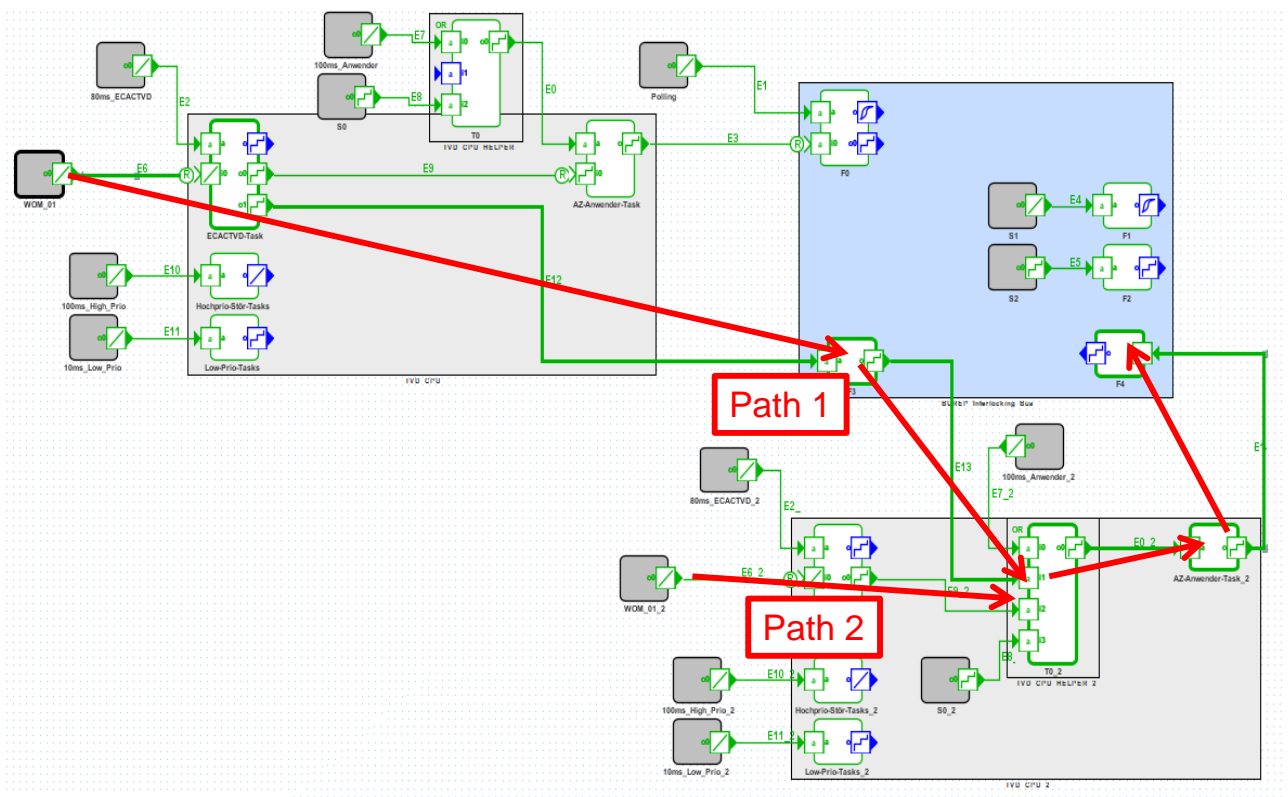


Figure 21: Focused data paths

Analyzing for path delays has the results as shown in Figure 23 and Figure 22. The green vertical arrows represent the data flow between tasks and frames. The horizontal red arrows stand again for scheduling delays due to preemptions and, especially in the case of register-based communication, for register delays due to oversampling and undersampling. As can be easily seen, bus communication is the lowest part of the overall delay. The longest delay times are the results of preemptions and register delays.

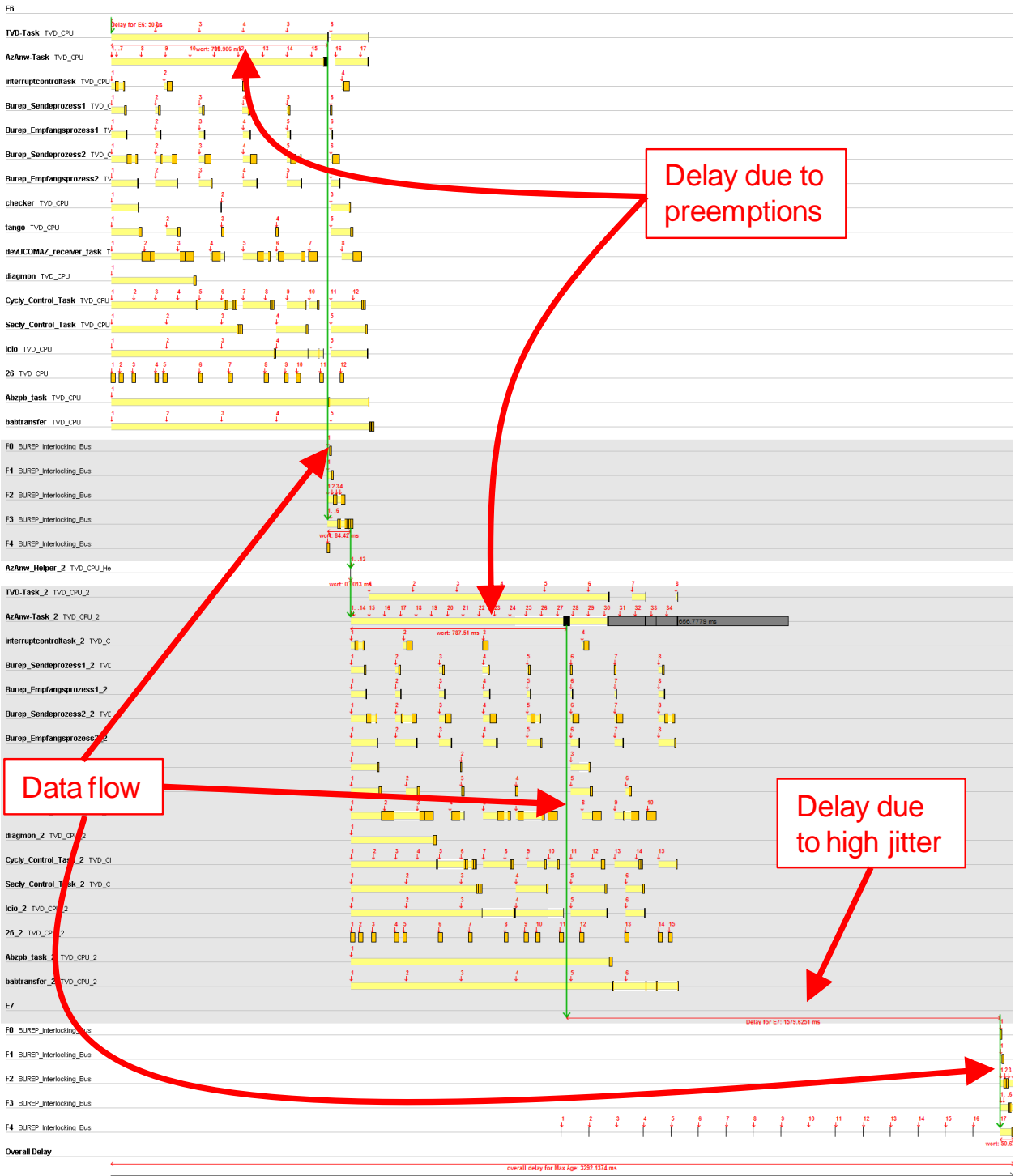


Figure 22: Worst-case scheduling along data path 1

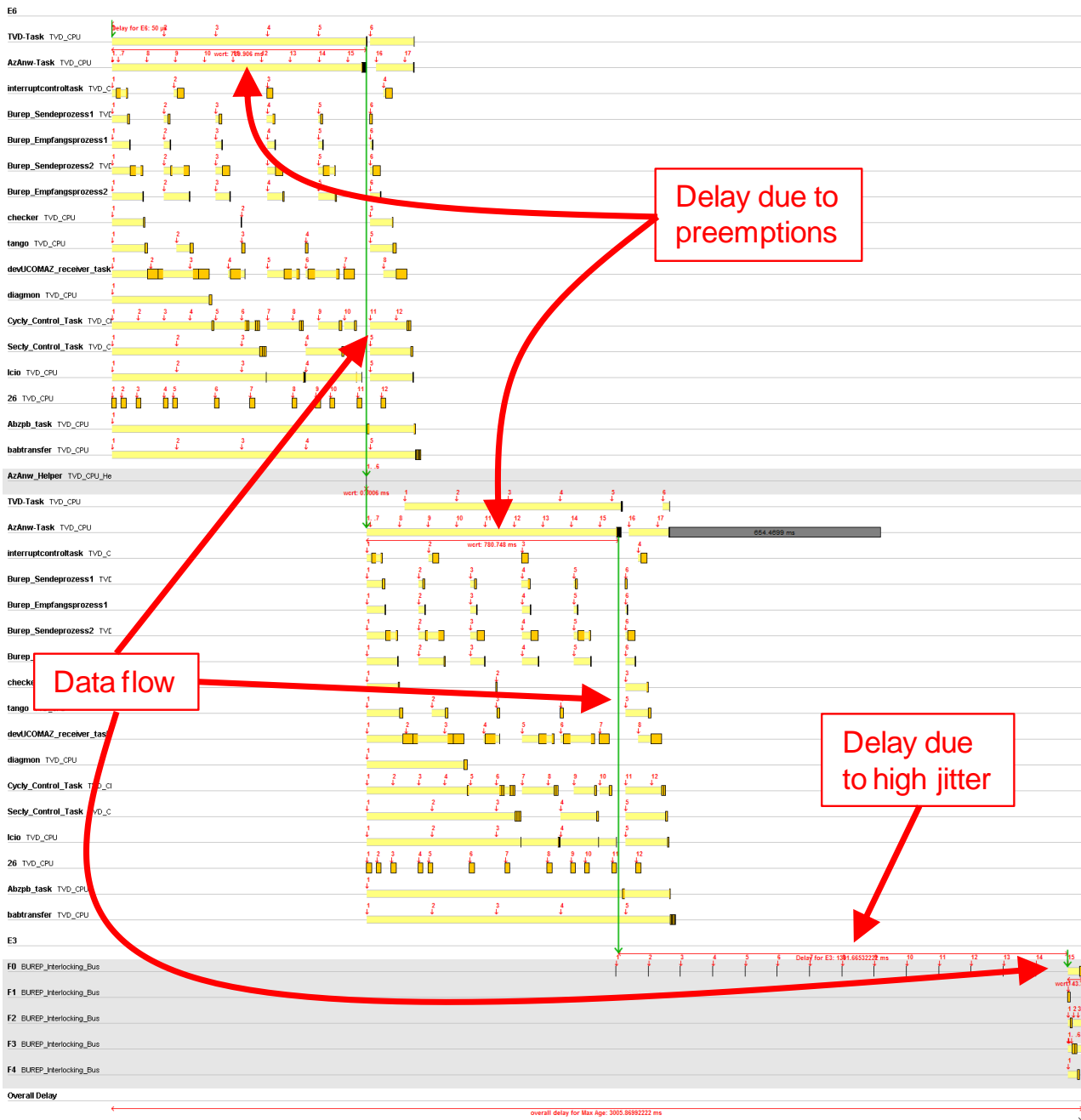


Figure 23: Worst-case scheduling along data path 2

The analyzed delays for both paths are:

- Path 1: [min;max] delays = [3.21 ms ; 3292.14 ms]
- Path 2: [min;max] delays = [2.21 ms ; 3005.87 ms]

As already introduced in the previous section, there exist the same and further weaknesses of the timing model which have to be taken into account:

1. The used timing model did not consider several important configuration details. The main one is a higher architecture complexity. The system design supports more cascading configurations than focused on here. Thus, the worst-case cascade must be identified and analyzed.
2. A second detail which is not considered is synchronization between periodic tasks which is often used for reducing preemptions by exploiting communication gaps during periodic activation.
3. Another configuration detail is task chaining. A task is able to activate another or multiple other tasks, which themselves activate other tasks, etc. This might also reduce preemptions due to other scheduling behavior.
4. The fourth weakness is related to the worst-case schedule of each individual task. Each is used during the worst analysis of the focused paths, even if the tasks are chained and are running on the same CPU. In the real world, there again exist dependencies for both tasks which again will improve the analysis result if it can be considered. Here, it is not the case, due to unavailable detailed information.

4.4.8 Comparison of Worst-case Analysis Results with Constraints

The challenge is now to identify delays on both data paths and compare them, where the minimum distance between two wheel detectors in combination with the maximum speed of a train also results in a delay value which is directly comparable with the delay difference of both paths.

In addition, it has to be considered that the second path might follow best-case communication where data path 1 is related to the worst case. This effect strengthens the situation that wheel detector data is able to overtake other wheel detector data and leads to very high path delay differences.

As a result, the transfer delay should never be larger than the time span of reading wheel detector data from two adjacent wheel detectors.

$$t_{sensor_2} - t_{sensor_1} > t_{data-path_1-max} - t_{data-path_2-min}$$

$$1440ms > 3292.14ms - 2.21ms = 3289.93ms$$

It seems that there problems might occur due to communication delays!

If the weaknesses identified in previous sections regarding the timing model are reflected in this result, it is obvious that it is too conservative and thus has to be refined after enhancing the model.

4.4.9 Identification of Optimization Potential and Strategies

Optimization potential was identified on both sides: modeling and implementation.

4.4.9.1 Model Improvements

All mentioned improvement aspects from Section 4.4.7 were not considered during this project due to less time, but it can be anticipated that a refined model will lead to better local delays (at CPU level) and even worse path delays due to higher cascades (more TVD CPUs in series incl. more BUREP buses connecting them). A follow-up project might pick up these details again.

4.4.9.2 Implementation Improvements

This example impressively shows possible delays as a result of scheduling effects. Reducing these effects is the solution for avoiding deadline violations. A wide range of possible arrangements can be taken into account for scheduling improvements.

One of the most intuitive solutions is changing the priority of heavily preempted tasks. Using a higher priority leads to less preemptions and thus to lower delays.

Due to less time during this project, no optimization was finally focused upon. This might be done in a further project.

4.4.10 Strengths and Weaknesses of the SymTA/S Approach

The SymTA/S tool chain focuses timing analysis on the system level in an abstract way by modeling the timing behavior of components, resources, communication pathways and scheduling policies, regardless of whether they are composed of software, hardware or mechanics. It constructs worst-case scenarios synthetically based on the modeled timing behavior of the single system components. The real target system does not need to exist at that time. It can best be used during a technical systems planning process and architecture phase and for spitting up timing requirements to system components or for choosing communication strategies.

The abstract modeling of timing behavior implies an approximation between the model and target systems. This leads to a balance between effort and timing analysis precision. With more effort, more precision can be gained. The effort can be concentrated wherever precision is needed.

Later on in the development process, SymTA/S can be used to verify the fulfillment of timing constraints. Therefore, the SymTA/S timing model must be fed with realistic timing values from the target system. The SymTA/S TraceAnalyzer supports this process by extracting timing values from target system log files.

With the exception of the TraceAnalyzer, the target system does not require any special preparation or interfaces for SymTA/S. In this way, SymTA/S fits existing real targets as well as planned systems.

Due to the fact that SymTA/S requires a separate abstract timing model of the target system, the target system as it is cannot be taken over into the timing analysis directly. Construction of the timing model requires additional effort and results in a more or less tight approximation to the real target, but will never represent the target exactly in detail. On the other hand, leaving out a tool- and model-based timing analysis means ensuring the required timing constraints in an alternative way, and this also incurs costs.

The timing model is limited to working with planned or estimated timing properties of the target system base components as long as these components do not exist. It is not until then that the properties can be measured and fed into the model.

The target system scheduling and arbitration policies need to be supported by SymTA/S.

Due to the fact that designing an abstract timing model is an intellectual process, SymTA/S is more suitable for high-level system approaches than digging down into fine-grained hardware or software details.

Besides providing worst-case times, SymTA/S and the TraceAnalyzer in general lead to a deep understanding about the interaction of internal functions and processes within the target system and the impact of resource conflicts and scheduling policies and, in this way, assist in discovering options to enhance target system timing.

4.5 Timing Analysis Summary

In real systems, worst-case timing scenarios usually cannot be generated by test cases due to asynchronous events; simple timing measurements do not provide worst values. This is the domain of timing analysis technologies.

The objects of validation were the two different technologies of timing analysis as shown in Sections 4.3 and 4.4.

Both approaches showed their specific strengths: while SymTA/S predominantly covers the system top level and system architecture in hardware and software with abstract timing models, aiT focuses on fine-grained

timing analysis for concrete software executable binary files on specific CPU platforms. Therefore SymTA/S and aiT are not competitive but complementary solutions. In addition, the SymTA/S TraceAnalyzer serves to feed the timing model with measured basic values from an existing target system.

The capabilities of both methods could be validated successfully and lead to probable results for an existing complex target system. The tools proved as stable and mature. Moreover, working with them created a deep understanding for timing aspects and relationships and that they facilitate a targeted timing behavior as part of the development process. Very important is the ability to identify the causes for time-consumption and to investigate the effects of changes of the target system.

Validation also showed that target systems have to fulfill some technical preconditions to be compliant with timing analysis tools. These preconditions could not be met completely for the investigated target system with respect to limited effort and may be a cause of deviations of analysis results. This limitation will not exist for targets at the beginning of their lifecycle when the prerequisites can be easily implemented.

The model- and tool-based timing analysis was found as a significant improvement of the development process. In comparison to the verification of timing constraints by costly measurements, a reduction of costs can be expected.

With more effort, for instance for a more detailed timing model, the analysis results can be refined and furthermore approximated to the real system behavior.

5 Summary of Validation Results

The INTERESTED tool chain as validated by Siemens Rail Automation fulfills most of our requirements to the extent expected from a research prototype. In general, prototypic implementation of the INTERESTED tool chain already shows maturity very close to a final product. In comparison to stand-alone tools or even conventional tools without integrated semantic checking as sometimes still used, the tool chain shows a significant increase in productivity and developer satisfaction due to an earlier response to some semantic mistakes which are immediately disclosed by the tool chain or later by colleagues during reviews.

INTERESTED successfully showed that having a seamless model-based development from system design to implementation is not only ideal but can be successfully transferred into real-life development. The tool vendors involved demonstrated that seamless tool interaction is possible. The idea of using SysML as modeling for systems fits very well with SCADE's programming paradigm.

The INTERESTED tool chain has the potential to be a key player in the market of safety-related reactive systems.

6 References

- [HFZG07] M. Huhn, B. Florentz, A. Zechner, and S. Gerken. Using Architecture Exploration in the Development of an Automated Train Operation Platform. Formal Methods for Automation and Safety in Railway and Automotive Systems (FORMS/FORMAT 07), 2007, 326-340