PROJECT FINAL REPORT

Publishable Summary Report

Version 0.1, 2010-06-15 (Lisper, MDH)
Version 1.0, 2010-09-08 (Lisper, MDH)

Grant Agreement number: 215068

Project acronym: ALL-TIMES

Project title: Integrating European Timing Analysis Technology

Funding scheme: small or medium-scale focused research project

Period covered: from Dec. 1, 2007 to May 31, 2010

Name of the scientific representative of the project's co-ordinator, Title and Organisation::
Björn Lisper, Professor, Mälardalen University

Tel: +46-21-151709

Fax: +46-21-101460

E-mail: `bjorn.lisper@mdh.se`

Project website address: `www.all-times.org`

Contents

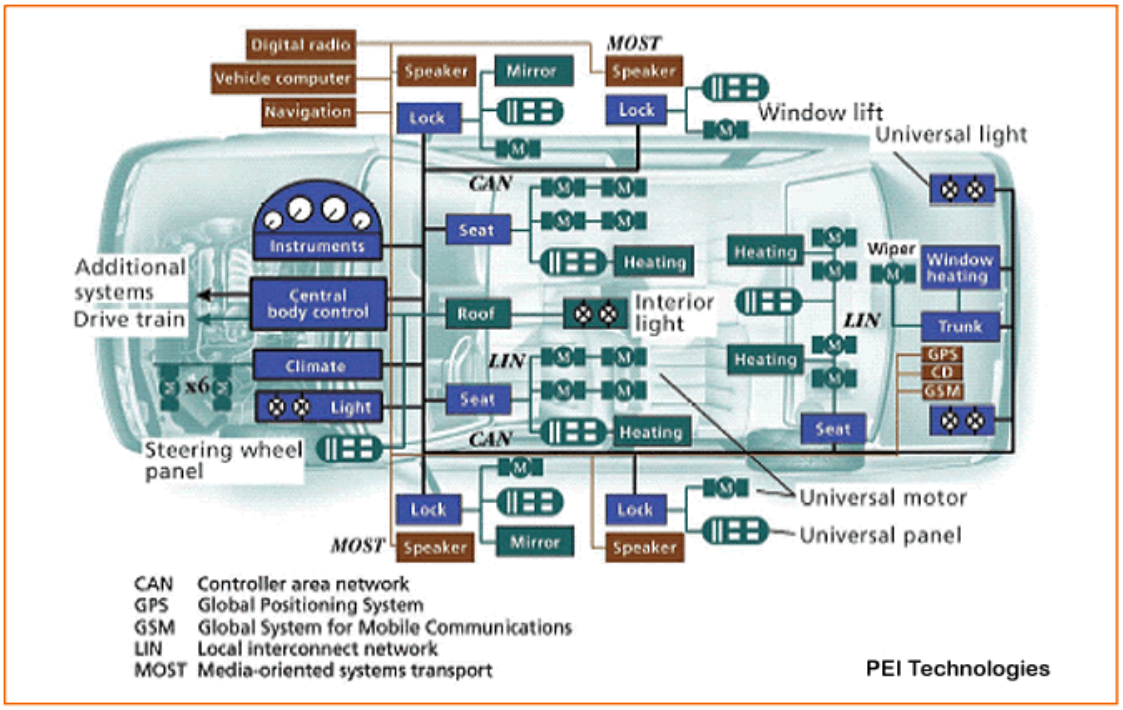

Figure 1: The electronics in a modern car.

1 Introduction

Embedded computer systems are omnipresent in modern life. Cars, telephones, washing machines, airplanes, and other products all contain embedded systems that monitor and control their functionality. A modern car can easily contain 40-60 processors that control different parts: ignition, fuel injection, powertrain, brakes, suspension, as well as entertainment and positioning systems, and lately advanced features like active collision avoidance systems and automatic parking assistance. See Fig. 1[1].

Obviously, it is very important that these embedded systems are reliable and always work as intended. A desktop computer crash can perhaps be annoying, but a failing embedded system can have fatal consequences. Many embedded systems are safety-critical, and if they fail then human lives can be at risk.

Another aspect is that an increasing share of the value for hi-tech products, such as cars, comes from the embedded systems. Time-to-market is becoming increasingly important. At the same time, the embedded systems are becoming more complex. For industry to stay competitive, development costs have to be curbed. Altogether, this implies that there is a need for improved methods and tools for embedded systems development, making it faster and more efficient, and where the resulting functionality is ensured.

For most safety-critical embedded systems, *correct timing* is an important part of the required functionality. Again, a car provides good examples: for instance, an ABS braking system must be able to react within a certain time to signals indicating that the tires are losing their grip. If the timing of the ignition is off then the car will run poorly, or not

[1]Picture by `photobucket.com`

at all. Obviously, the process of designing embedded systems to meet the required timing requirements, and to analyse the resulting systems to decide whether the requirements are met, needs ample support by methods and tools as well.

This is where ALL-TIMES comes in. Timing analysis has been the subject of academic study for a number of years, and some commercial tools have emerged. However, the tools have operated mostly in isolation, thus limiting their applicability and not utilizing their full potential. The objective of ALL-TIMES has been to create *integrated tool chains*, enabling different timing analysis tools to work together in a highly automated fashion. The project has also developed *methodologies* supporting the proper use of these tool chains in different situations. The tool chains have been enabled by the creation of *open interface formats* carrying different kinds of information, and enabling different *tool connections*. In one part of the project methods and tools for *early timing estimation* have been developed, which can direct the design work to provide a timing-correct system in shorter time. Altogether, the aim of ALL-TIMES has been to cut the design time for embedded systems, and provide more reliable embedded systems, by providing the appropriate tool support for timing analysis.

2 Objectives

The two principal project objectives for ALL-TIMES have been the following:

- O1: Integration of timing analysis tools, and
- O2: Increase of productivity of embedded systems development projects by 25% of the design time pertaining to timing issues.

These objectives have been refined into seven more technical project goals, which contribute to the project objectives according to the following table:

Goal	Description	O1	O2
g1	Enable source-code/binary code analysis integration	High	-
g2	Reduce number of manual annotations	-	High
g3	Minimise interference by instrumentation	-	High
g4	Open tool integration	High	-
g5	Improve accuracy of system-level timing verification	Low	High
g6	Enable timing estimation early in the design	Low	High
g7	Demonstrate results in industrial context	Low	High

3 Partners

ALL-TIMES has six partners: two academic partners, and four tool vendors:

Mälardalen University (SWE): coordinator, academic institution. It has developed the Worst-Case Execution Time (WCET) analysis tool SWEET, which is an academic prototype.

Vienna University of Technology (AUT): academic institution. It has developed the academic prototype tool SATIrE for source-level program analysis and transformation.

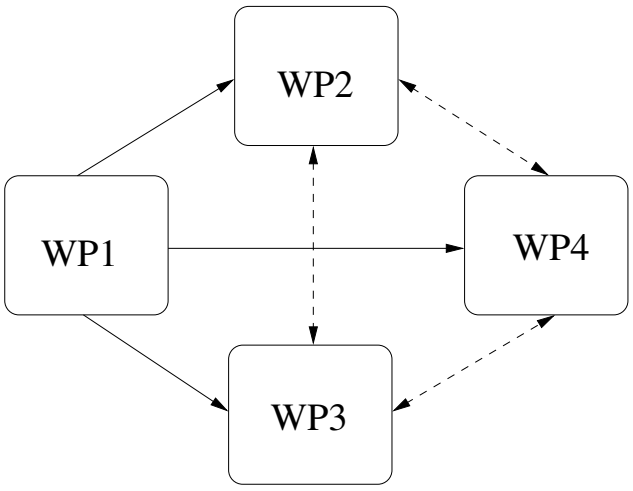

Figure 2: The technical work packages of ALL-TIMES.

AbsInt Angewandte Informatik GmbH (GER): tool vendor. AbsInt produces tools mainly for code-level analysis, including the WCET analysis tool aiT.

Gliwa GmbH (GER): tool vendor. Gliwa produces the tool T1 for measuring, visualising and analysing the timing of embedded software.

Symtavision GmbH (GER): tool vendor. The main product of Symtavision is the system-level timing analysis tool SymTA/S.

Rapita Systems Ltd (GBR): tool vendor. Rapita Systems provides the tool RapiTime, for WCET analysis, and performance debugging.

4 Project Activities, and Work Plan

The technical work in ALL-TIMES has been organized in four different work packages: WP1 *Requirements*, WP2 *System-level integration*, WP3 *Code-level integration*, and WP4 *Validation and dissemination*. Fig. 2 illustrates the dependencies between these work packages: a dashed double-pointed arrow indicates a mutual dependency for work packages that overlap in time. A fifth work package WP5 *Project management* concerned the project administration.

The Gantt chart of the work packages in Fig. 3 shows the timeline of the project, including the three reporting periods. (The project was extended with three months: therefore, the last period spans twelve months rather than nine.) The breakdown of work packages into work tasks, with timelines for these, is shown in Table 1. The work carried out in the technical work packages is described in Sections 4.1 - 4.4 below. The total effort (man months) spent by the partners on the different work packages is shown in Table 2.

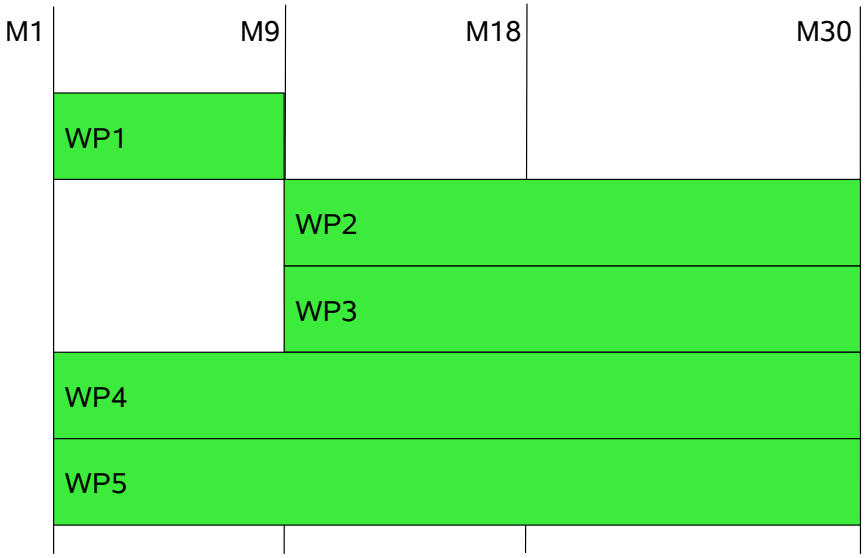

Figure 3: Gantt chart of work packages.

WP1: Requirements
- WT 1.1 Identification of use cases (M1 - M6)
- WT 1.2 Timing analysis tool requirements (M3 - M9)
- WT 1.3 Tool interface requirements (M3 - M9)

WP2: System-level integration
- WT 2.1 System-level tool interfaces (M10 - M30)
- WT 2.2 Early-stage system-level methodology (M10 - M30)
- WT 2.3 Late-stage system-level methodology (M10 - M30)

WP3: Code-level integration
- WT 3.1 Incorporating tool measurement data (M10 - M30)
- WT 3.2 Source-level analyses (M10 - M30)
- WT 3.3 Code timing analysis in early design stages (M10 - M30)

WP4: Validation and Dissemination
- WT 4.1 Case study plan (M1 - M6)
- WT 4.2 Validation using case studies (M7 - M30)
- WT 4.3 Plan for the use and dissemination of foreground (M1 - M6)
- WT 4.4 Dissemination (M7 - M30)

WP5: Project management
- WT 5.1 Project management (M1 - M30)

Table 1: The different worktasks.

	Mälard.	AbsInt	Vienna	Gliwa	Symtav.	Rapita	total
WP1	9.6	14	10	4.1	9.15	14.5	61.35
WP2	1.5	3	0	10	28.37	10	52.87
WP3	33.7	14.9	23.6	6.5	0.5	22	101.2
WP4	15.1	8.9	11.46	6.44	13.65	12.89	68.44
WP5	6.4	0	0.75	0.75	1.38	0.52	9.8
total	66.3	40.8	45.81	27.79	53.05	59.91	293.66

Table 2: Efforts spent in the project (man months).

4.1 Work Package 1: Requirements

The work in this work package has concerned the following three aspects. First, a number of interesting so-called "use cases" (tool couplings) were identified. The purpose of these use cases is to enable the formation of tool chains, where different timing analysis tools interact to make the timing analysis process more efficient. 20 potential use cases were identified at an early stage of the project. Of these, 16 were finally implemented. They are shown in Fig. 4.

Second, an analysis was made what kind of development of the timing analysis tools that would be required to support the selected use cases. The third part of the work was to similarly identify the requirements on the tool interfaces to support the use cases. This part of the work formed the basis for the continued work in work packages 2 and 3.

4.2 Work Package 2: System-level Integration

System-level timing analysis takes the complete system into account. The analysed system consists of a single processor, or several processors and communication buses. Each processor hosts a number of activities (so-called *tasks*), which together carry out the duties of the system. The analysis obtains timing bounds for the tasks from the code-level analysis, and subsequently determines the *schedulability* given requirements on the response times of the system. If the system is schedulable, then its timing requirements are met. The single system-level analysis tool in ALL-TIMES is SymTA/S from Symtavision.

A key factor in the system-level analysis is what information to pass from the code-level analysis tools, and how to pass it. For this purpose, two open interface formats were specified and implemented: XTC 2.0 ("eXtensible Timing Cookies"), for passing information like WCET bounds on tasks, and ATF for communicating traces. These interfaces carry five of the use cases identified in work package 1, between the code-level analysis tools aiT, T1, RapiTime, and the system-level tool SymTA/S.

Visualisation is a useful tool to help the engineer understand the system-level timing behaviour. For that purpose, requirements for a *Common Trace Viewer* were specified. These requirements specify what a tool for visualising trace data must be able to do. Two Common Trace Viewer prototype implementations have been made in the project. A screenshot of the Symtavision trace viewer is shown in Fig 5.

An important part of the work has been the development of methodologies for using the tool combinations enabled by the new tool connections. The methodologies developed in the project cover several design situations. This includes both early design phases, where estimations of time consumption are made to direct the system design, and late verification

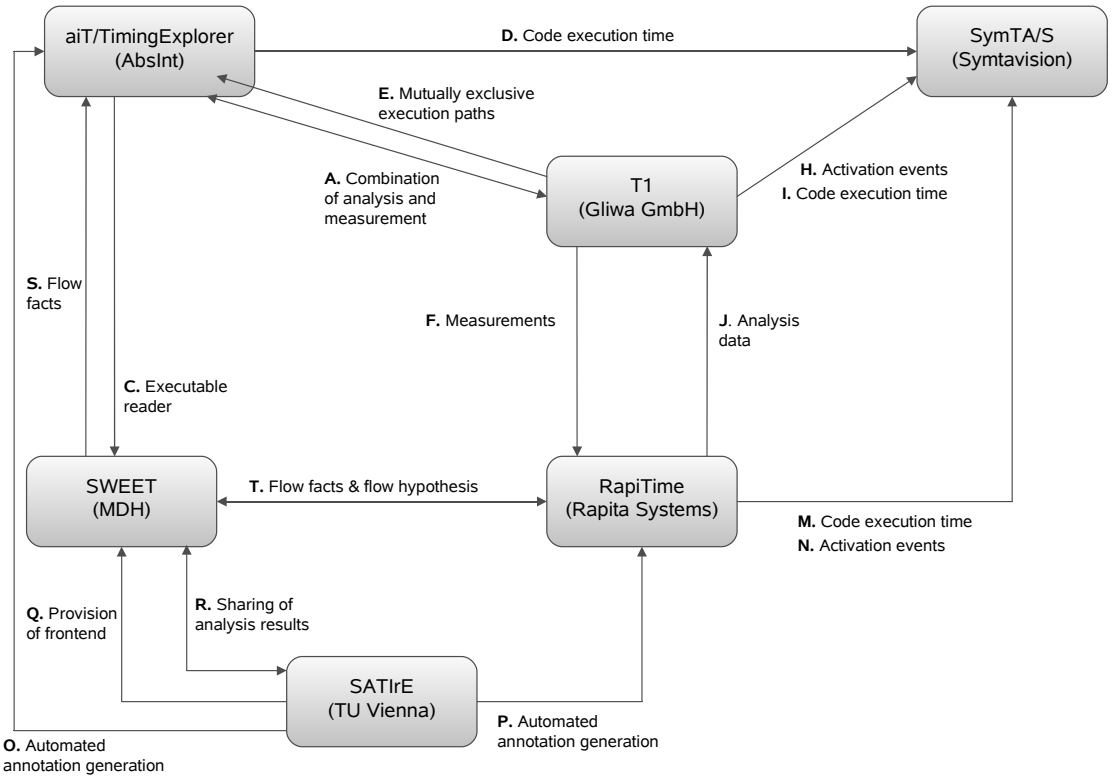

Figure 4: The use cases finally implemented in ALL-TIMES.

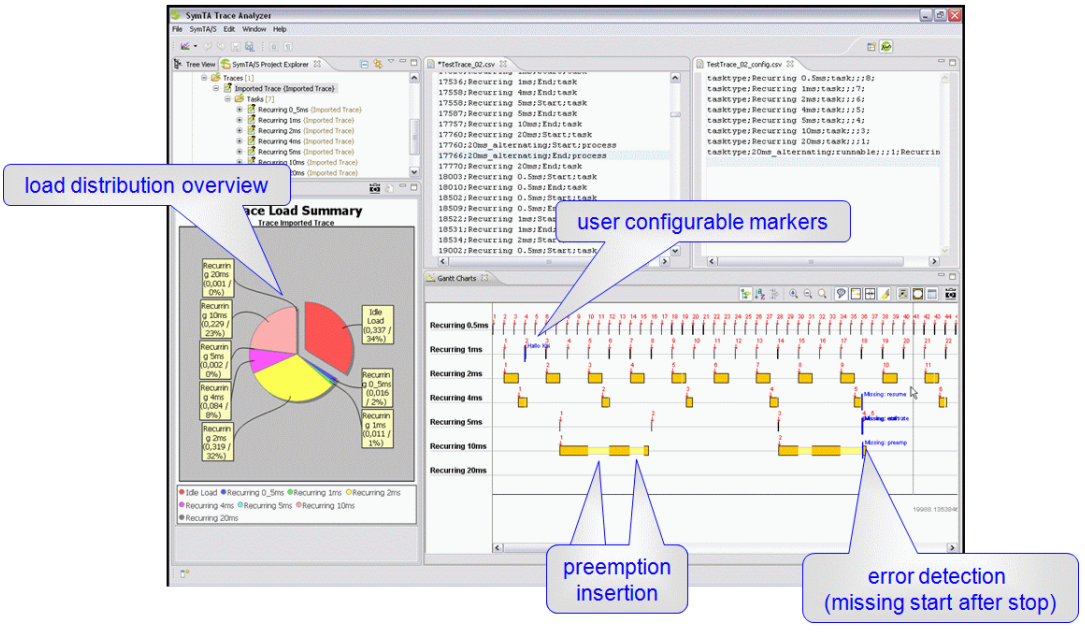

Figure 5: A trace viewer screenshot.

where the timing properties of the final system are verified. The final methodologies are *integrated*, and cover both the code-level and system-level parts of the integrated tool chains.

4.3 Work Package 3: Code-level Integration

Code-level analysis derives bounds on the execution time (typically WCET) for single tasks. In contrast to system-level analysis, which mainly deals with numbers representing timing bounds, code-level analysis must analyse the program code as well as the influence of the hardware on the timing. This makes the analysis complex. In practice, the user often has to provide information manually that the tools are not able to derive automatically. To reduce the need for such manual intervention has been an explicit goal of the work in ALL-TIMES, and this problem has been tackled in different ways in this work package.

Code-level analysis can be done by a variety of techniques, ranging from timing estimations based on various measurements to pure static program analysis. It can be done both for executables and source code. Different approaches have different pros and cons. All the main code-level analysis techniques are represented in ALL-TIMES by different tools.

In order to enable the best combination of analysis techniques for different design situations, a number of connections have been realized between the code-level analysis tools. One group of connections effectuates the communication of measured data, and involves the tools T1, aiT, and RapiTime. The data can be measured times for program fragments as well as information on program flow, like the observed maximal number of iterations of a loop. These connections use the XTC or ATF interface formats. Measurement of data yields a high level of automation, but the accuracy of the results depends on the quality of the test data.

Another part of the work has been the development of supporting source-level analyses. Information about program flow, like the maximal number of loop iterations, or possible values of function pointers, is typically easier to find from the source code than from the executable. A static analysis of the source code gives high confidence in the results. The tools SATIrE and SWEET are able to do source-level analysis: SATIrE analyses the source code directly, while SWEET uses its multilevel code interface ALF. An ALF translator from SATIrE has been developed, enabling SATIrE to act as a frontend for SWEET. In addition, the "ARAL" interface has been defined and implemented: it enables the tools to exchange program analysis information. Both tools have furthermore been equipped with interfaces to export their analysis results to aiT and RapiTime, which both can use the information when calculating WCET bounds. The benefits are a higher level of automation, and/or better precision in the resulting WCET bounds.

The third part of this work package concerns code-level timing analysis in early design stages. This is useful for design space exploration, typically in situations where a migration to more recent hardware is desired. The analysis can then be used to direct the selection of hardware. For this purpose, the TimingExplorer tool has been developed. This is a version of aiT that trades precision for speed in order to allow a fast design space exploration with different hypothetical hardware configurations. TimingExplorer has the same interfaces as aiT, and can thus directly replace aiT in the tool chains. In particular, it can use information both from measurements and source-level analyses through the interfaces developed in the project.

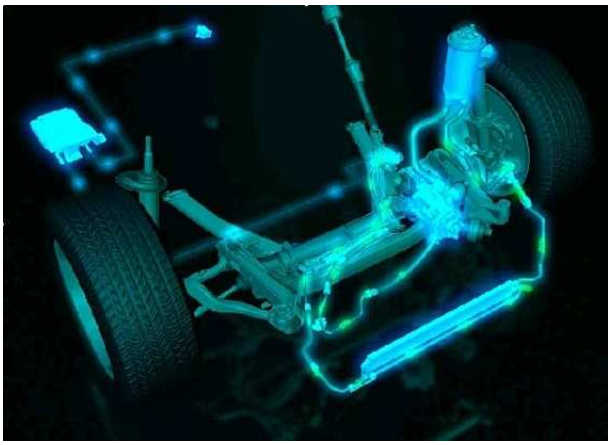

Figure 6: Active Front Steering (AFS).

4.4 Work Package 4: Validation and Dissemination

4.4.1 Validation

The results of the project have been validated using two case studies. Both case studies come from the automotive domain. The first case study concerns the "VECU" (vehicle control unit) in an experimental fuel-cell-powered system. This unit calculates the torque necessary to fulfil the driver's request in the current situation, as well as handling the buffer battery in the system. The second case study is an active front steering system ("AFS"), which changes the steering ratio gradually according to the vehicle speed and also implements a number of stability-enhancing functions. See Fig. 6.

VECU was used as the main case study, complemented by AFS. Validation was done both on an individual basis, per use case, and for full tool chains. In both cases the reduction in design effort, resulting from the improved workflows made possible by the project results, was estimated.

The tool chain validation was done using two scenarios: an "early stage" scenario, assuming a situation where a processor upgrade is to be done, and a "late stage" scenario where the schedulability of an existing system is to be decided. For each scenario a typical workflow was considered, and the reduction in effort for each step in the workflow was estimated. The result of this was an estimate of the overall reduction in design effort for each selected scenario. The tool chains for the early and late stage scenarios are shown in Figures 7 and 8, respectively.

4.4.2 Dissemination

The dissemination of the project results has taken place through a number of channels. One journal publication, 18 papers in reviewed scientific conferences and workshops, 14 papers in branch magazines, user conferences, and similar, and 22 other publications have resulted from the work in the project. The project has been represented at nine conferences or trade shows with posters, demos, or similar. A number of press releases, flyers, and other advertising material has been produced. ALL-TIMES partners AbsInt, Gliwa, and Symtavision have formed the *Real-Time Experts Alliance*[2] for joint promotion

[2] `www.real-time-experts.com`

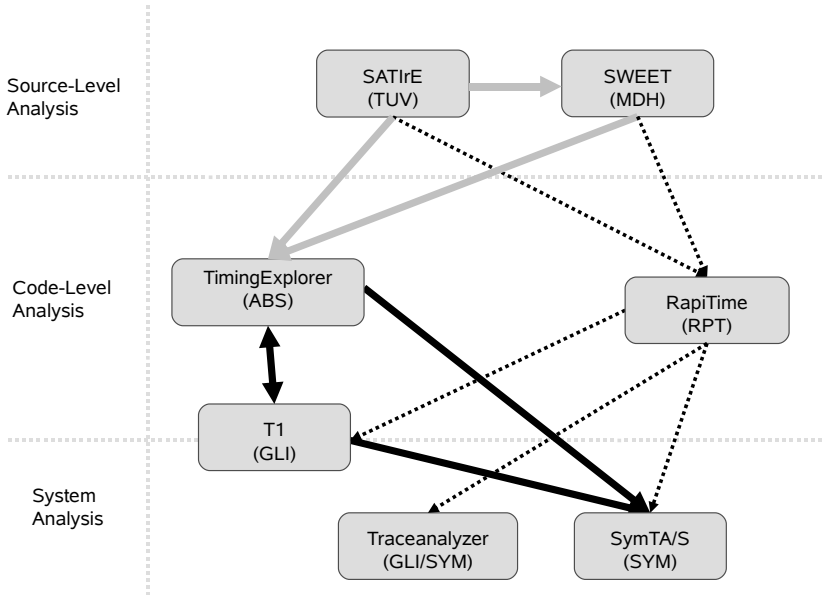

Figure 7: Early stage validation scenario.

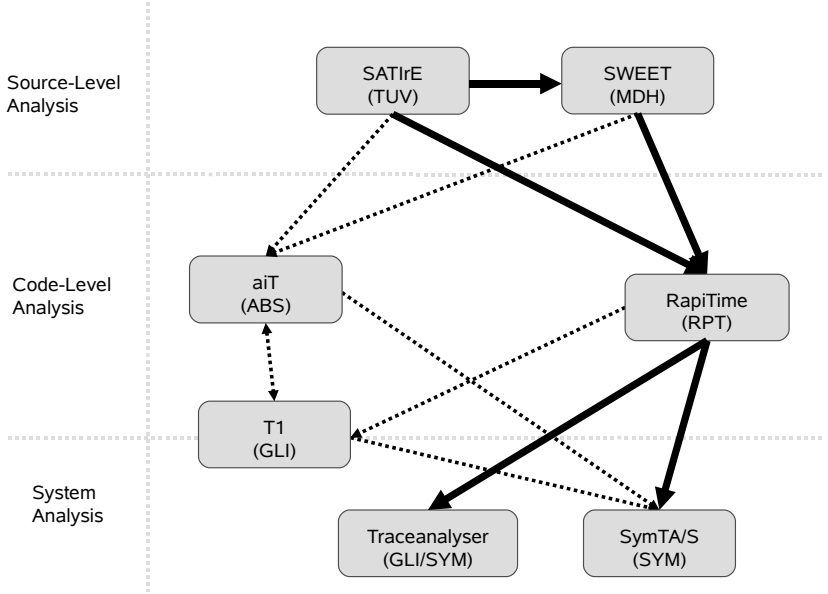

Figure 8: Late stage validation scenario.

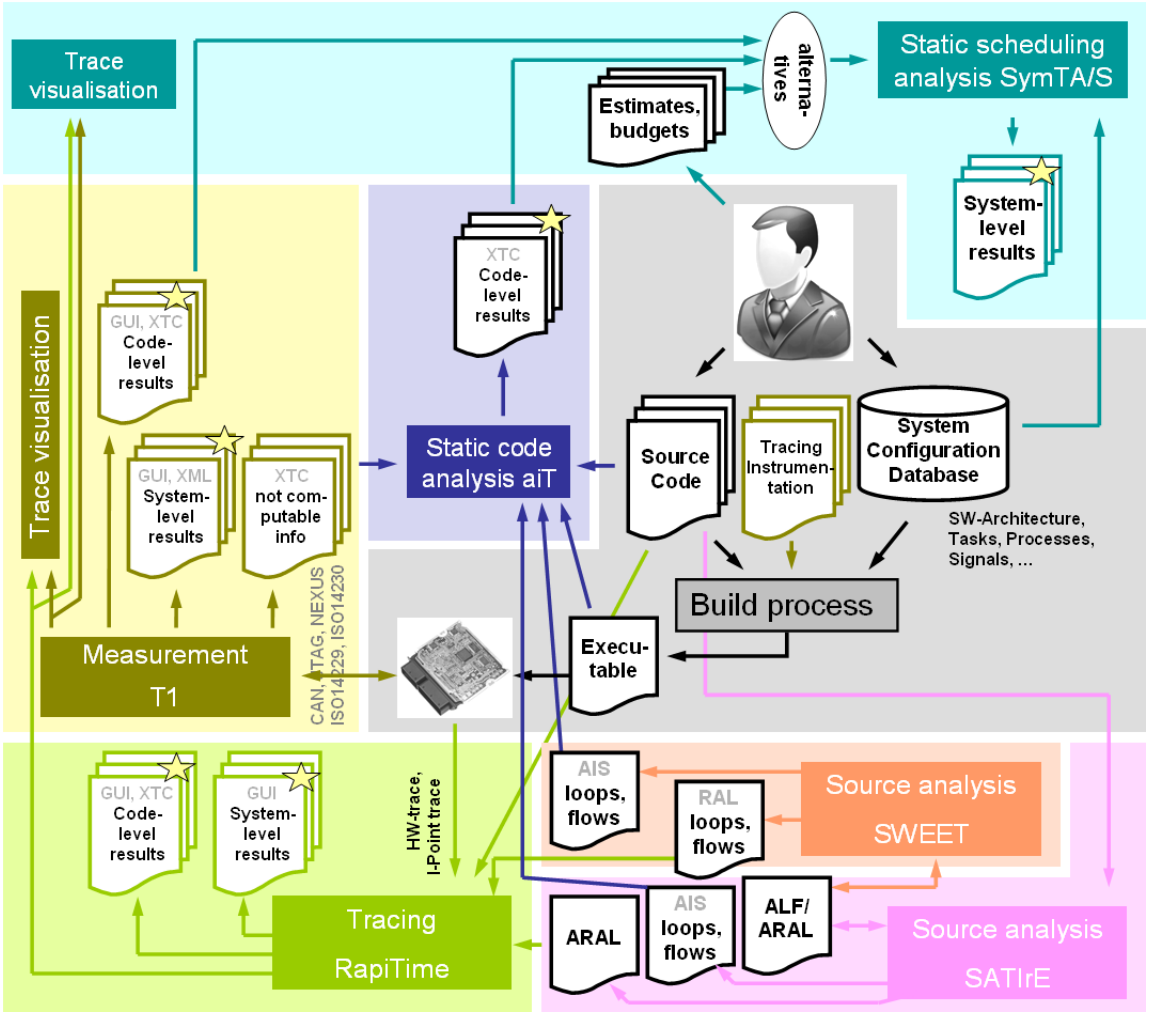

Figure 9: Tools, analysis techniques, tool connections, formats.

and marketing of project results. Project partners have arranged four scientific workshops related to the project. ALL-TIMES has interacted with the European projects TIMMO, MERASA, and INTERESTED, and project results will be further disseminated through collaborations in future projects: project partners are currently involved in the proposed European projects TIMMO-2-USE (ITEA2), ReComp (ITEA2, already running), and SafeCer (ARTEMIS).

The project website, `www.all-times.org`, hosts general information about the project, public documents, and open interface specifications.

5 Main Project Results

The main project results of ALL-TIMES are new and improved *tools*, *tool connections*, *open interface formats*, and *methodologies* helping the user use the tool chains enabled by the results brought by the project. Fig. 9 illustrates the tools and tool connections, and the different formats and analysis techniques that have been developed and knit together in the project.

5.1 Tools

Existing tools have been enhanced and extended in the project, and a number of tool connections have been established. The following tools have been created, or enhanced, in the project:

SymTA/S has been extended with support for the new XTC 2.0 format, and for the ATF format for exchanging trace information. It has been further extended with a Distribution-Analysis prototype, in order to capture both worst-case schedules as well as typical cases, their distribution and the probability of worst cases.

Common Trace Viewers Symtavision and Gliwa have developed a prototype tool based on the ALL-TIMES agreed trace viewer requirements. They have also added support for multicore ECUs.

T1 Gliwa has extended their T1 to support ATF and XTC 2.0. The measurement resolution has been improved to the granularity of machine instructions. An on-target timing analysis component has been developed which allows to determine loop bounds and call targets of indirect function calls. Additionally, the on-target analysis captures max/min core execution times and the max/min CPU load.

TimingExplorer The TimingExplorer tool has been developed within the project. It is a version of aiT for static WCET analysis, which is tuned to suit design space exploration in early design phases. Therefore it trades precision for analysis speed, which is more important in design space exploration. TimingExplorer is fully compatible with aiT as regards interfaces, and both TimingExplorer and aiT have been provided with new interfaces for the XTC 2.0 format.

SWEET A new version of SWEET has been implemented, which uses the new multilevel code format ALF as input format for its program flow analysis. It can now generate a number of new program flow constraints, and it has been enhanced with a new format for input value annotations to allow a more precise analysis. Its Flow Fact format for representing program flow constraints has been extended to be more expressive. SWEET has also been extended with backends that generate flow facts in the ARAL, AIS[3], and RapiTime annotation formats.

SATIrE is a source program analysis/transformation tool built on top of the Rose compiler framework. It has been extended in various ways to support source level analysis, including advanced analyses to compute function pointer information, and a flow-sensitive interval analysis. A tool for translating Rose's internal format into ALF has been implemented: together with Rose's C frontend this provides a C frontend for SWEET. SATIrE has also been extended to export function pointer sets to the AIS and RapiTime annotation formats.

[3] AIS is the annotation format for aiT/TimingExplorer.

RapiTime has been extended with new interfaces for the XTC 2.0 and ATF formats. RapiTime can now also export ATF directly. The ATF support in RapiTime has enhanced RapiTime "Rewind" v2.3, allowing it to relate wall-clock times to events in the trace. RapiTime's annotation format has been extended to use the Abstract Source Location (ASL) format to specify program locations, and an ARAL reader for importing results from SATIrE.

5.2 Tool Connections

The following tool connections (use cases) have been effectuated through the implementation of the various interfaces and tool enhancements done in the project. See also Fig. 4:

A. Combination of analysis and measurement (aiT ↔ T1) T1 can export measurement-based timing information, and loop iteration bounds, to aiT through the XTC 2.0 format. The connection also enables comparisons between measured end-to-end execution times from T1, and estimates done by the static analysis of aiT.

C. Executable reader (aiT → SWEET) A translator for binaries, stored in aiT's internal representation, into ALF has been implemented for the NEC V850 and PowerPC instruction sets. This connection enables SWEET to perform program flow analysis on binaries, using aiT as a "frontend".

D. Code execution time (aiT → SymTA/S) Through XTC 2.0, aiT exports WCET estimates for uninterrupted tasks to SymTA/S. This connection existed in earlier versions of XTC, and has been maintained into XTC 2.0.

E. Mutually exclusive execution paths (T1 → aiT) T1 can export information about unreached program points, and function-pointer destinations, to aiT. The format is XTC 2.0.

F. Measurement input (T1 → RapiTime) Traces are exported from T1 to RapiTime through the ATF common trace format. RapiTime can then perform a WCET analysis using the traces, and generate a detailed WCET report that can be viewed in the RapiTime report viewer.

H. Activation events (T1 → SymTA/S) Traces containing activation events are exported from T1 to SymTA/S using ATF. SymTA/S can then perform a schedulability analysis based on this information, and the trace viewer of SymTA/S can visualize the trace data.

I. Code execution time (T1 → SymTA/S) Execution time bounds extracted from traces are exported from T1 to SymTA/S through the XTC 2.0 format.

J. Analysis data (RapiTime → T1) This use case allows the export of traces from RapiTime to T1 using ATF. It allows T1 users to reuse the RapiTime instrumentation.

M. Code execution time (RapiTime → SymTA/S) RapiTime exports WCET estimates to SymTA/S, using the XTC 2.0 format.

N. Activation events (RapiTime → SymTA/S) RapiTime provides traces containing runtime events such as scheduler events, task/thread activation events, etc., using ATF. SymTA/S imports the trace data to show its visual representation.

O. Automated annotation generation (SATIrE → aiT) SATIrE exports function pointer sets to aiT/TimingExplorer using the AIS annotation format. This reduces the need for manual annotations.

P. Automated annotation generation (SATIrE → RapiTime) Similarly to use case O, SATIrE exports function pointer sets to RapiTime using RapiTime's annotation format and ASL (abstract source code locations).

Q. Provision of frontend (SATIrE → SWEET) Through a translator from Rose's internal format to SWEET, SATIrE can act as a C frontend to SWEET. This enables SWEET to perform source-level program flow analysis.

R. Sharing of analysis results (SATIrE ↔ SWEET) The ARAL format has been designed to exchange program analysis information. API's have been created that can be used to read and write analysis results in ARAL. These API's are used by SWEET and SATIrE to export program analysis results.

S. Flow facts (SWEET → aiT) "Flow facts" (program flow constraints) automatically generated by SWEET are exported to aiT/TimingExplorer in the AIS format. The provision of these can reduce the need for manual annotations.

T. Flow facts & flow hypotheses (SWEET ↔ RapiTime) Similarly to use case S, SWEET can also export flow facts to RapiTime in the RapiTime format using ASL to specify program locations. An experimental version of SWEET that can read traces and generate flow facts from these (so-called "flow hypotheses") has also been implemented.

5.3 Interface Formats

The following open interface formats have been developed within the project. The formats are open, and can be freely used also by other tools than the ones in ALL-TIMES. Specifications for these formats are available at `www.all-times.org`.

XTC 2.0 XTC stands for "eXtensible Timing Cookies", and is an XML-based format used to exchange different kinds of timing analysis data. XTC supports a request/reply style of communication, where different tools successively add information to the cookie. XTC 2.0 can transfer information about worst-case execution times, maximum stack usage, iteration bounds of loops, response times, activation patterns aligned with the upcoming AUTOSAR 4.0 and TIMMO event model descriptions, and scheduling overheads.

ATF This is an XML-based trace format. It allows the exchange of trace information, including metadata. Similarly to XTC, it has a "cookie" function allowing different tools to add information.

ALF ALF is a program code format. It is designed to be easy to analyze, as well as to be able to faithfully represent code on different levels, ranging from source level to binary level. Therefore it includes both high-level constructs, such as function calls, as well as low-level constructs such as jumps with computed targets.

ARAL This is a format to exchange program flow analysis information, which has been developed within ALL-TIMES. It allows to express contexts such as call strings, and it has a language for expressing analysis results.

ASL ASL is a format to express locations in source code. This information is needed to express source code analysis results.

5.4 Methodology

The ALL-TIMES methodology helps the user choose the best combination of timing analysis tools and techniques for a given situation. It covers both *early exploration* and *late verification*, as well as *code level* and *system level* analysis.

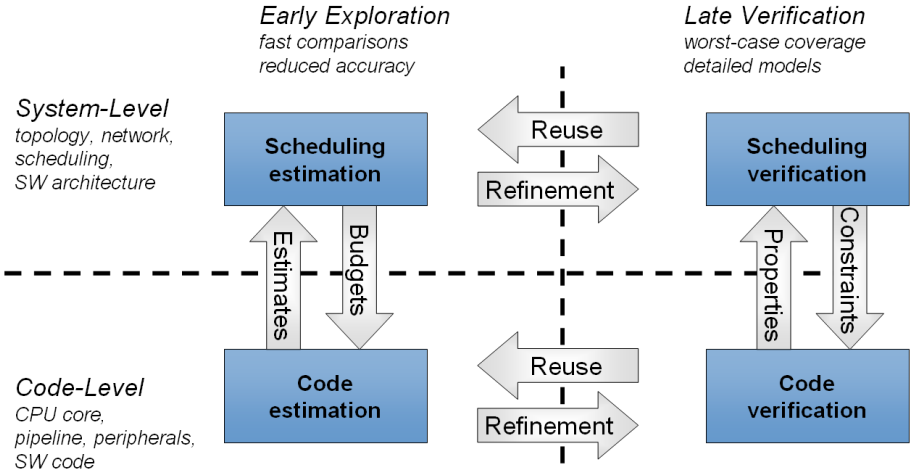

Figure 10: The four quadrants of the ALL-TIMES methodology and their relationships

Figure 10 summarizes the four aspects of the ALL-TIMES methodology and their relationships. As development progresses, early phase models for design space exploration are refined into late phase models for timing verification. In the opposite direction, late phase models of a system are reused as the basis for early phase models of new versions of the system. In the early phase, system-level models are used to derive budgets for code-level timing. In the opposite direction, code-level analysis provides timing estimates for consideration on the system level: for instance, the timing estimates can be checked against the timing budgets. In the late phase, system-level models are used to specify constraints for code-level timing. In the opposite direction, code-level analysis provides timing properties for consideration on the system level. An example is a system-level schedulability analysis based on WCET bounds obtained from code-level analysis.

The methodology specifies a number of workflows in different situations. An example is shown in Fig. 11, which specifies a workflow for early stage code-level analysis using source-level analysis. These workflows involve different tool chains, like SATIrE – SWEET

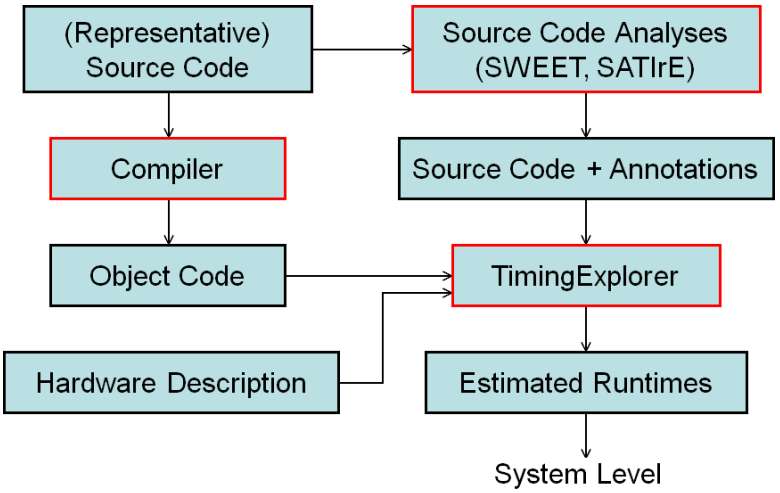

Figure 11: Workflow for source-code analysis and static timing analysis.

– TimingExplorer in the example, and the methodoloqy explains how to use the different tools and exchange formats in the different situations.

5.5 Validation

The goal of the validation was to decide whether the project had met its main objectives O1 (integration of timing analysis tools), and O2 (increase of productivity of embedded systems development projects by 25% of the design time pertaining to timing issues). O1 was validated on a per use case basis, where a use case (tool integration) was considered validated when used for at least one case study. 13 out of the finally implemented 16 use cases were O1 validated.

As explained in Section 4.4.1 the O2 validation was done for two different design scenarios involving two tool chains, an early stage design exploration scenario and a late stage verification scenario. For each scenario, the overall reduction in effort brought by the ALL-TIMES technologies was estimated. For the early stage scenario the estimation indicates a ten-fold reduction in effort, and for the late stage scenario a reduction by factor of two. Taking into account that the scenarios cover only the timing analysis part and not the full design effort pertaining to timing, these estimates still provide strong evidence that O2 is met.

5.5.1 Experiences, and Lessons Learned

The validation provided valuable experience regarding the use of the integrated tool chains, and the parts that are likely to contribute the most to the improvement of the timing analysis process. For both validation scenarios, the main improvement comes from the automated generation of program flow annotations. In particular the automatic derivation of program flow information from traces turned out to be very successful. High speedups were also recorded for the automation of information exchange through the XTC and ATF formats, which enables a much faster interaction between code- and system-level analysis tools. This information exchange also yields new opportunities to visualise different kinds of timing information.

The source-level analysis validation encountered some problems, mainly pertaining to source code missing for some parts of the system, and inability to parse parts of the source code due to the code stretching the C standard. It also turned out that detailed knowledge of the build process was necessary to guide the source analysis to the proper parts of the source code. These issues seem to be common for "deep", semantics-based source-level analysis in general. If industry wants to take full advantage of source-level analysis then care has to be taken regarding coding standards, and source code will have to be provided by third-party software providers to a larger extent. More research is also clearly needed how to handle situations with incomplete source code.

Finally, a lesson learned was the importance of making sure early that the selected case studies really will support the validation of all the techniques and tool developed in the project. ALL-TIMES initially aimed to have three case studies. This was narrowed down to two case studies, AFS and VECU, since only these case studies contained source code. Quite late into the project it was discovered that the AFS source code had been stripped of almost all the interesting contents due to IP reasons, and therefore was unsuitable for validating the source level analyses. This left the project with only the VECU case study being possible to use throughout the project, with AFS restricted to the parts that would not involve source-level analysis. If this had been found earlier, the consortium would have had more time to look for alternative case studies giving better support for the validation of the source-level analyses.

6 Potential Impact and Use

The commercial partners all plan to integrate the ALL-TIMES results into their next generation of products. More specifically all these partners plan to integrate the interface support, developed within ALL-TIMES, into their commercially offered tools. This particularly applies to the XTC and ATF interfaces, which support a large number of use cases. Rapita Systems will also incorporate the ASL notation into their next line of products: this will effectively port the integration with the source-level tools to the commercial version of RapiTime. Symtavision will develop its Common Trace Viewer into a product, the Symtavision TraceAnalyzer. AbsInt will evaluate TimingExplorer with a selected set of customers, and if the outcome is positive then a series of commercial TimingExplorers, for different processor families, will be made available. This then means that basically all the improvements brought in the project by the commercial partners will be available quite soon to end-users, and that they will benefit from the expected productivity increase resulting from their use. For instance, the early stage validation scenario (Fig. 7) will then be fully supported. It is foreseen that the primary markets will be in the automotive, avionics, and train areas. The impact will be an increased competitiveness of these sectors of the European industry, due to higher quality of the embedded software and reduced development costs.

The academic partners, which have provided the source-level analysis tools to the project, plan to give their tools free distribution, possibly as open source. This then means that end-users who want to try out the source-level analysis extensions of the tool chains, and possibly integrate these into their development processes, will be able to do so.

The project results also provide an infrastructure for future research. The ability to connect academic prototypes with industrial-strength tools makes it possible to perform

more realistic studies of different analysis methods and algorithms, where they are applied to realistic systems rather than simple academic benchmarks. The academic partners plan to use this infrastructure in their further research. Furthermore, future research collaborations are facilitated by the tool connections established within ALL-TIMES. This will be used in future research projects like TIMMO-2-USE, ReComp, and SafeCer.

7 Conclusions

ALL-TIMES has been a successful project. The benefits brought by the combined use of tools, which has been made possible by the work carried out in the project, will be available to end-users and will strengthen their competitiveness. Future research collaborations between academic and commercial partners will be facilitated by the infrastructure built in the project. Europe will continue to be in the lead as regards timing analysis.

8 Contact Info

- Mälardalen University (coordinator)
  Contact person: Björn Lisper
  Address: School of Innovation, Design, and Engineering, Mälardalen University, P.O. Box 883, SE-721 23 Västerås, Sweden
  Email: `bjorn.lisper@mdh.se`

- Vienna University of Technology
  Contact person: Jens Knoop
  Address: Institut für Computersprachen, Fakultät für Informatik, Technische Universität Wien, Argentinierstraße 8 / 4 / E185.1, A-1040 Wien, Austria
  Email: `knoop@complang.tuwien.ac.at`

- AbsInt Angewandte Informatik GmbH
  Contact person: Christian Ferdinand
  Address: AbsInt Angewandte Informatik GmbH, Science Park 1, D-66123 Saarbrücken, Germany
  Email: `ferdinand@absint.com`

- Gliwa GmbH
  Contact person: Peter Gliwa
  Address: GLIWA GmbH embedded systems, Dollmann Str. 4, D-81541 München, Germany
  Email: `peter@gliwa.com`

- Symtavision GmbH
  Contact person: Marek Jersak
  Address: Symtavision GmbH, Frankfurter Straße 3b, D-38122 Braunschweig, Germany
  Email: `jersak@symtavision.com`

- Rapita Systems Ltd
  Contact person: Nicholas Merriam
  Address: Rapita Systems Ltd., IT Centre, York Science Park, York, YO10 5DG, United Kingdom
  Email: `nmerriam@rapitasystems.com`

Project website: `www.all-times.org`