Development environment for configuration and analysis of embedded and real-time systems [*]

Aleksandra Tesanovic, Peng Mu, and Jörgen Hansson
Department of Computer Science
Linköping University, Linköping, Sweden
{alete,jorha}@ida.liu.se

ABSTRACT

In this paper we present a tool framework that provides developers of real-time systems with support for building and configuring a system out of components and aspects, as well as real-time analysis of the configured system. This way a real-time system can efficiently be configured to meet functional requirements and analyzed to ensure that non-functional requirements are also fulfilled, e.g., available resources such as memory and CPU. We illustrate the usability of the tool by a case study of an embedded real-time database system.

1. INTRODUCTION

Modern real-time computing systems are faced with increasingly complex requirements in their development and operation. Successful usage of these systems in different applications strongly depends on the ability to tailor a real-time system to meet specific needs of the underlying application. Hence, there is a need for an efficient way of developing families of real-time systems, i.e., a product line architecture, suitable for a plethora of applications.

One way to ensure meeting these requirements is to adopt the component-based software development paradigm for real-time systems. This way systems are developed out of pre-defined software components, and can be tailored for a specific real-time application by adding or removing components to/from the architecture. However, approaches to developing families of real-time systems out of pre-defined components existing in a component library, e.g., [9, 15], do not provide efficient support for features that cannot cleanly be encapsulated into components, e.g., temporal constraints, scheduling policies, and synchronization. Aspectual component-based real-time system development, ACCORD [14], is an example of an effort to integrate the two software engineering techniques, aspect-oriented and component-based software development, into real-time system development. ACCORD enables building flexible product line architectures for real-time systems as it enables assembling the system both from components and aspects.

For a real-time system, it is essential that results produced by the system are both produced correctly and in a timely manner. To ensure timeliness, tasks[1] in a real-time system are associated with deadlines, and a number of real-time scheduling techniques has been developed ensuring that tasks meet their respective deadlines. These typically require the knowledge of the worst-case execution time (WCET) of a task, making the issue of determining the WCET essential for real-time systems. Furthermore, majority of real-time systems is also embedded, implying that they have sparse resources in terms of CPU and memory consumption or footprint. Thus, to be able to use a real-time system in a particular run-time environment and an application, analysis techniques need to be available to ensure that non-functional requirements, i.e., WCET and memory consumption needs, are met.

The contribution of this paper is a tool framework that provides real-time system developers with support for configuration and analysis of a real-time and embedded system composed of aspects and components, i.e., a system developed using ACCORD approach. The tool framework includes tools that support a designer in the composition process, as well as in the process of performing WCET and memory footprint analysis, and formal verification of the composed real-time system. We show the way the tool can be used for developing various configurations of a real-time database system, based on the specified functional and temporal requirements of the underlying run-time environment.

The paper is organized as follows. In section 2 we give background information on ACCORD and its main constituents. Then, in section 3, we describe the tool framework we developed to support the process of assembling and analyzing systems developed based on ACCORD. We illustrate the usability and applicability of the tool on the example of the real-time database system in section 4. Related work is discussed in section 5. Finally, the paper is concluded in section 6 with conclusions and directions for future work.

2. ASPECTUAL COMPONENT-BASED REAL-TIME SYSTEMS DEVELOPMENT

Within ACCORD aspects in real-time systems are classified in different categories [14]: (i) application aspects, (ii) run-time aspects, and (iii) composition aspects. The classification eases reasoning about different embedded and real-time related requirements, as well as the composition of the system and its integration into a run-time environment.

Application aspects can change the internal behavior of components to suit a particular application as they crosscut the code of components in the system, e.g., memory

[*]This work is supported by the Swedish Foundation for Strategic Research (SSF) via the SAVE project, the Center for Industrial Information Technology (CENIIT) under contract 01.07, and the Swedish National School in Computer Science (CUGS).

[1]Real-time systems are typically constructed out of concurrent programs, called tasks.

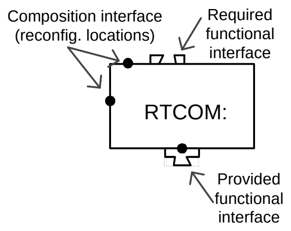

Composition interface (reconfig. locations)
Required functional interface
RTCOM:
Provided functional interface

Figure 1: Reconfigurable Real-Time Component Model (RTCOM)

optimization aspect and real-time policy aspect. Run-time aspects give information needed by the run-time system to ensure that integrating a composed real-time system would not compromise timeliness or available memory consumption. Composition aspects describe the version of the component, possibilities of extending the component with additional aspects, and a set of other components and application aspects with which this component can be combined.

ACCORD provides a real-time component model, denoted RTCOM, to support reconfigurability [14]. RTCOM components are "grey" as they are encapsulated in interfaces, while changes to their behavior can be performed in a predictable way using aspects.

Each RTCOM component has two types of functional interfaces: provided and required (see figure 1). Provided interfaces reflect a set of operations that a component provides to other components, while required interfaces reflect a set of operations that the component requires (uses) from other components. Composition interfaces define reconfiguration locations in the component code. Reconfiguration locations define the points in the component code where additional modification of components can be done by aspect weaving. These points can be used by the component user (or component developer) to reconfigure a component for a specific application or reuse context. Reconfiguration locations are also used analysis purposes [13, 12]. Note also that the operations declared in the provided functional interface can be used for aspect weaving and, thus, they represent also implicit reconfiguration locations.

ACCORD enables design of a system both when (1) the components and aspects are not available in the library, and also when (2) there is a pre-existing library of aspects and components developed for a family of real-time systems. In the first case, the design of a real-time system using ACCORD method is performed as follows. First, a real-time system is decomposed into a set of components. Decomposition is guided by the need to have functionally exchangeable units that are loosely coupled, but with strong cohesion. Then, a real-time system is decomposed into a set of aspects. After the design, components and aspects are implemented based on RTCOM. Further, components and aspect that provide specific functionality to the system are grouped into aspect packages. This is to ease the reuse of already developed software artifacts. The second case is illustrated in figure 2 where the composition of the system is done using software artifacts available in a library. Now, in the design phase of the system, the developer chooses appropriate aspects and components and forms a system configuration based on application requirements. Moreover, if the system needs to be modified during its lifetime to support new

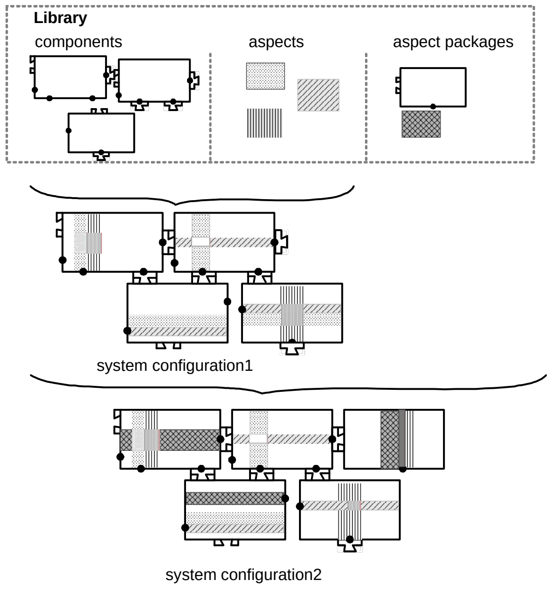


Figure 2: ACCORD-based design

functionality this can be achieved by adding aspect packages that provide exactly the functionality needed.

3. DEVELOPMENT ENVIRONMENT

The ACCORD development environment is a tool framework developed to provide tool support for the system designer when assembling and analyzing a real-time system for a particular application. We assume that, for a particular family of real-time systems, components and aspects are already developed and placed in the library, hence, providing support for case (2) of the ACCORD development process described in section 2.

The ACCORD development environment consists of (see figure 3): the ACCORD library of pre-developed software artifacts, the ACCORD Modeling Environment (ACCORD-ME), and a configuration compiler. In this section we describe each of the constituents of the ACCORD development environment and then discuss limitations and benefits of the environment.

3.1 ACCORD Library

The ACCORD library contains two types of artifacts, namely design-level artifacts and implementation artifacts (see figure 3). Design-level artifacts are: (i) models of components and aspects, (ii) run-time aspects of components and application aspects (as prescribed by ACCORD in section 2), and (iii) formal representations of the components and aspects. Design-level artifacts are used when designing, modeling, and analyzing the system. Implementation artifacts represent the implementation, i.e., source code, of the components and aspects. The implementation artifacts are used when producing the final product, i.e., a compiled code of the system that can be deployed in a specific run-time environment.

3.2 ACCORD-ME

The ACCORD-ME part of the development environment is implemented using the generic modeling environment (GME),

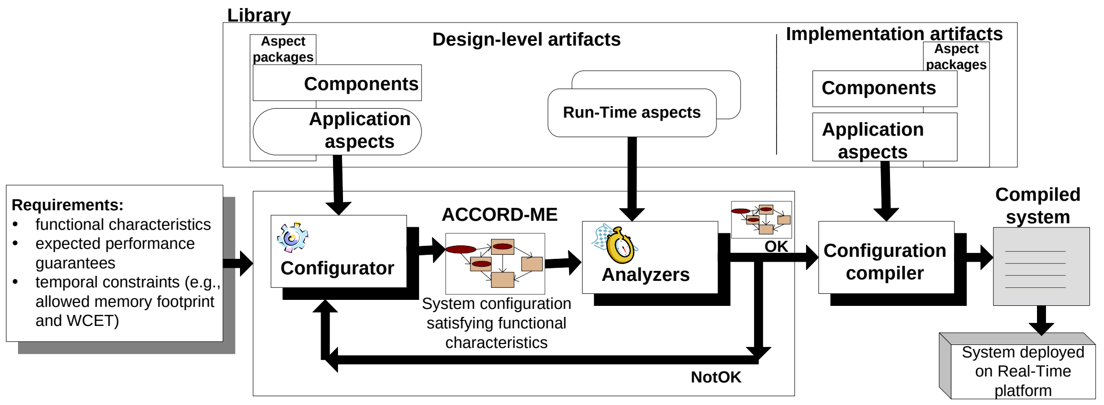


Figure 3: ACCORD Development Environment

a toolkit for creating domain-specific modeling environments [3]. The creation of a GME-based tool is accomplished by defining metamodels that specify the modeling paradigm (modeling language) of the application domain. The GME environment also enables specifying different tool plug-ins, which can be developed independently of the environment and then integrated with the GME to be used for different modeling and/or analysis purposes.

The input to the ACCORD-ME are requirements that are placed on the system. This includes functional and non-functional, e.g., performance guarantees, and temporal and memory constraints, requirements a system should fulfill when used in a specific run-time environment. This tool also uses the design-level artifacts for configuration and analysis of the system. The output of the tool is a configuration file that contains information about the system configuration that fulfills the specified functional and non-functional requirements.

ACCORD-ME is developed with a number of sub-tools that are plugged into ACCORD-ME, namely the configurator, memory and WCET (M&W) analyzer, and formal verifier.

The *configurator* helps the designer to assemble the system configuration by suggesting a subset of suitable aspects and components. This tool provides three levels of support to the system configuration based on the expertise and preferences of the developer.

Expert option is used by developers familiar with the library of components and aspects. The expert option enables the developers to create a number of custom made configurations of the system. This is useful in cases when a comparison of the performance of different configurations is of interest.

Configuration-based option gives a list of possible configurations of the system. This option is intended for developers that can directly express the requirements of the system in terms of available configurations.

Requirement-based option provides the system developer with the list of requirements from which the developer can choose a relevant subset of requirements for a particular application. Thus, developers do not need to have knowledge of what components and aspects exist in the library.

The *M&W analyzer* analyzes a configuration with respect to WCET and memory requirements (the algorithm employed for WCET calculations is presented in [13]). This tool takes as input (i) a configuration file that resulted from the configuration process aided by the configurator; and (ii) the run-time aspects. These run-time aspects contain the run-time information the tool needs to calculate the impact of aspect weaving and system composition with respect to WCET or memory consumption.

We employ symbolic techniques [4] for calculating WCETs and memory requirements of operations. This implies that WCETs in the run-time aspects are expressed in terms of symbolic expressions. Hence, they are a function of one or several parameters that in turn abstract the properties of the underlying run-time environment. The symbolic expressions need to be re-evaluated for each run-time environment, i.e., parameters in symbolic expressions should be instantiated in the analysis. Hence, the M&W analyzer provides a list of symbolic expressions and the list of necessary parameters that need to be configured. Since it is possible to assemble a number of configurations in ACCORD-ME, e.g., when using the expert option in the configurator, the M&W analyzer detects the configurations and enables developers to choose which configuration she/he would like to analyze. Moreover, it is possible to choose whether WCET or memory analysis of the system should be performed.

The *formal verifier* is a tool that performs the formal verification of the composed real-time system. Formal analysis is done based on the formal framework for modular verification of reconfigurable components presented in [12]. The behavior of the system configuration is checked using formal models of aspects and components represented as augmented timed automata with reconfiguration and verification interfaces (stored in the ACCORD library).

3.3 Configuration Compiler

This tool is part of the development environment that aids the designer in compiling the final product. The configuration compiler takes as input: (i) the information obtained from ACCORD-ME about the created configuration that satisfied functional and non-functional requirements, and (ii) the implementation level artifacts. Based on this input it generates a compilation file. This file is then used for compiling source code of needed aspects and components into the final system. Hence, the output of this tool is the compiled system configuration, which is ready to be deployed on a specific run-time environment. The configuration compiler also provides necessary documentation about the generated configuration which can later be used for maintaining the system.

3.4 Benefits and Limitations

The ACCORD development environment offers benefits for system developers in terms of automated support during the composition process (via the configurator tool), analysis of the system (via various analysis tools), and the compilation and deployment (via the configuration compiler tool). With this automation the overall development time for a real-time system decreases dramatically.

The tool environment in general, and ACCORD-ME in particular, treats both components and aspects as first-class constituents of a real-time system. Moreover, the uniqueness of the environment is the set of tools encompassed by the ACCORD-ME for analysis of a real-time system woven with aspects (via components' reconfiguration points).

For each family of real-time systems developed using ACCORD, i.e., each new application domain, specific implementation of components and aspects that constitute the domain should be made, and these should be placed in the ACCORD library. Moreover, parts of the ACCORD development environment should be extended to embrace a particular domain (section 4 shows the tool applied to the domain of real-time databases). Namely, the configurator should be extended with a set of requirements and composition rules for this particular domain, and the compilation rules in the configuration compiler tool need also to be updated to contain the rules for compilation of the newly developed components and aspects.

4. EXAMPLE APPLICATION

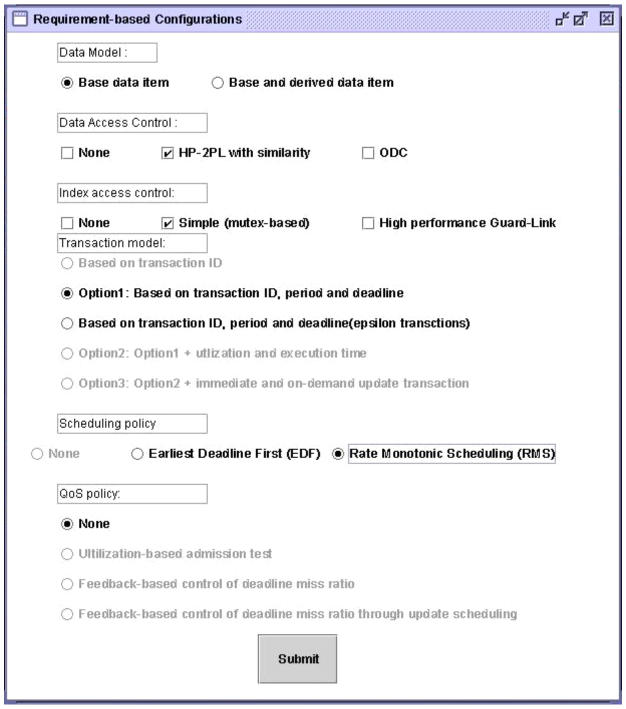
We illustrate the applicability of the tool on an example of the COMET real-time database system. We first briefly introduce COMET components and aspects, and then show how to develop a COMET database configuration using the ACCORD development environment.

4.1 COMET Aspects and Components

The ACCORD library contains, in this case, a set of COMET components and aspects developed for the domain of embedded real-time databases. COMET components are [8, 11]: a user interface component, a transaction management component, an index management component, a memory management component, a locking manager component, and a scheduling manager component. COMET aspects include concurrency control policies, scheduling policies, transaction model, and QoS policies. Depending on the application with which the database is to be integrated, aspects, components or specific aspect packages e.g., the concurrency control aspect package and the quality of service aspect package [11], can be used in the system composition.

4.2 Developing COMET configuration(s)

The following example illustrates how the COMET configuration can be tailored for a particular real-time system. We focus here on using ACCORD-ME for configuring a product line architecture for a specific electronic control unit (ECU) in vehicles. The ECU at hand is used for controlling the


Requirement-based Configurations

Data Model :
[✓ Base data item] Base and derived data item

Data Access Control :
None [✓ HP-2PL with similarity] ODC

Index access control:
None [✓ Simple (mutex-based)] High performance Guard-Link

Transaction model:
Based on transaction ID
[✓ Option1: Based on transaction ID, period and deadline]
Based on transaction ID, period and deadline(epsilon transctions)
Option2: Option1 + utlization and execution time
Option3: Option2 + immediate and on-demand update transaction

Scheduling policy
None Earliest Deadline First (EDF) [✓ Rate Monotonic Scheduling (RMS)]

QoS policy:
[✓ None]
Ultilization-based admission test
Feedback-based control of deadline miss ratio
Feedback-based control of deadline miss ratio through update scheduling

Figure 4: Requirement-based configuration of the real-time database system

engine in a vehicle, and has the following set of data management requirements [7].

R1: The application performs computations using data obtained from sensors; these data items are typically referred to as base data items.

R2: Sensor data should reflect the state of the controlled environment implying that transactions used for updating data items should be associated with real-time properties, such as periods and deadlines.

R3: Values of data should be updated only if they are stale[2] to save computation time of the CPU.

R4: The tasks in the ECU should be scheduled according to priorities.

R5: Multiple tasks can execute concurrently in an ECU. This in turn implies that the same data items can be read and written by different tasks (which could result in inconsistent data values in the system).

R6: The memory footprint of the system should be within the *memBound*, which typically is obtained from the ECU specifications.

When developing a configuration for such a system we start by specifying the requirements using the configurator in ACCORD-ME. Given that we know the system requirements, then the requirement-based option can be chosen in configurator to guide the system composition. Now, based on the requirements R1-R5 we can choose options in the

[2]A data item is stale if its value does not reflect the current state of the environment.

requirement-based form (shown in figure 4) provided by the configurator as follows. The configuration of the COMET database system that is suitable for the ECU at hand should contain only base data items (R1). Furthermore, it should provide mechanisms for dealing with concurrency such that conflicts on data items are resolved (R5) and data items that are stale are updated (R3). This can be achieved using one of the similarity techniques, e.g., HP-2PL with similarity [5]. The transaction model should be chosen such that transactions are associated with periods and deadlines (R2). We choose the rate monotonic scheduling policy [6] to enforce priority-based scheduling of tasks (R4). Performance guarantees in terms of levels of quality of service are not required.

When these requirements are submitted, the configurator loads those components and aspects that could satisfy them into ACCORD-ME. For more efficient composition process one can use help provided in the descriptions of components and aspects in terms of composition rules. Moreover, the relevant components and aspects are grouped in aspect packages for easier system composition.

Observe that if one wants to make several different configurations satisfying a broad spectrum of functionalities, then the expert option in the configurator can be chosen which loads all possible components, aspects, and aspect packages so that the developer can configure the system. A system configuration satisfying functional requirements is shown in the upper part of figure 5. In ACCORD-ME, ovals are used as the graphical representation of aspects, while squares represent components. When the composition of the system is made, it should be analyzed to get the memory and WCET needs of the configuration and contrast these to the prescribed available values of memory and WCETs. Hence, when the configuration part of the system development is finished then the obtained configuration can be analyzed using the M&W analyzer tool (see figure 5 for the snapshot of the analysis process). When the M&W analyzer is invoked, it detects the configuration(s) one might have made in ACCORD-ME and prompts for the choice of configuration. In our example, we created only one configuration and this one is detected by the M&W analyzer (see figure 5). After the configuration is chosen, the appropriate files describing run-time aspects of components and aspects are loaded for analysis. Since run-time properties of aspects and components are described in terms of symbolic expressions with parameters, the values of these parameters should be provided in the analysis. If a parameter is needed for a symbolic expression of a WCET or a memory of a component or an aspect, we say that this component/aspect should be configured for a run-time environment. The list of components that require configuration is shown during analysis, so one can make an inspection of the symbolic expressions, and input the values of parameters in the expressions. Note that advice that modify the components are included in the component run-time description as shown in the lower right corner of figure 5. Once the values of parameters are set for this particular ECU, the tool outputs the resulting WCET and/or memory consumption values which can be compared with the values given in requirement (R6).

If the obtained configuration satisfied the requirements of the engine ECU, the next step is to compile the system and deploy it in the run-time environment, which is done using the configuration compiler. As mentioned previously, the configuration compiler also provides documentation of the composed configuration of COMET.

5. RELATED WORK

A number of GME-based tools exist that address modeling and development of software systems. Here we review only those tools that provide support for development of component or aspect-oriented systems; the full list of GME-based tools can be found in [3]. C-SAW [2] is an example of a GME-based tool that enables weaving of aspect models providing a constraint weaver for nor-real-time systems. VEST [10] is a GME-based tool that supports building component-based real-time and embedded systems with support for run-time aspects. In contrast to our approach and the ACCORD development environment, VEST does not provide support for system configuration (out of components and application aspects), nor does it support the designers in suggesting a relevant subset of components during system composition.

The Koala tool suite [15], which is a tool environment for composition of product line architectures in industry, provides similar support for the developer when assembling an embedded system out of components only.

6. SUMMARY

In this paper we have presented the ACCORD development environment for building embedded and real-time product-line architectures using components and aspects available in the library. In this environment, we provide the support for real-time system developers in all phases of the system development, from requirement specification to configuration, system analysis, and compilation and deployment of the system. The ACCORD development environment is suited both for developers with extensive knowledge of available artifacts as well as ones with little or no knowledge of available components and aspects in the library by having a tool called configurator as the part of the environment. The configurator suggests the subset of aspects and components suitable for the underlying system. Moreover, the analysis tools enable developers to efficiently analyze the system with respect to CPU and memory needs. This analysis is essential in the real-time domain. The tool environment outputs the compiled system configuration as a final product with documentation to ease maintenance of the system.

Our ongoing work focuses on the implementation of the formal analyzer, which provides mechanisms for formal analysis of the system, and its integration into the development environment.

7. REFERENCES

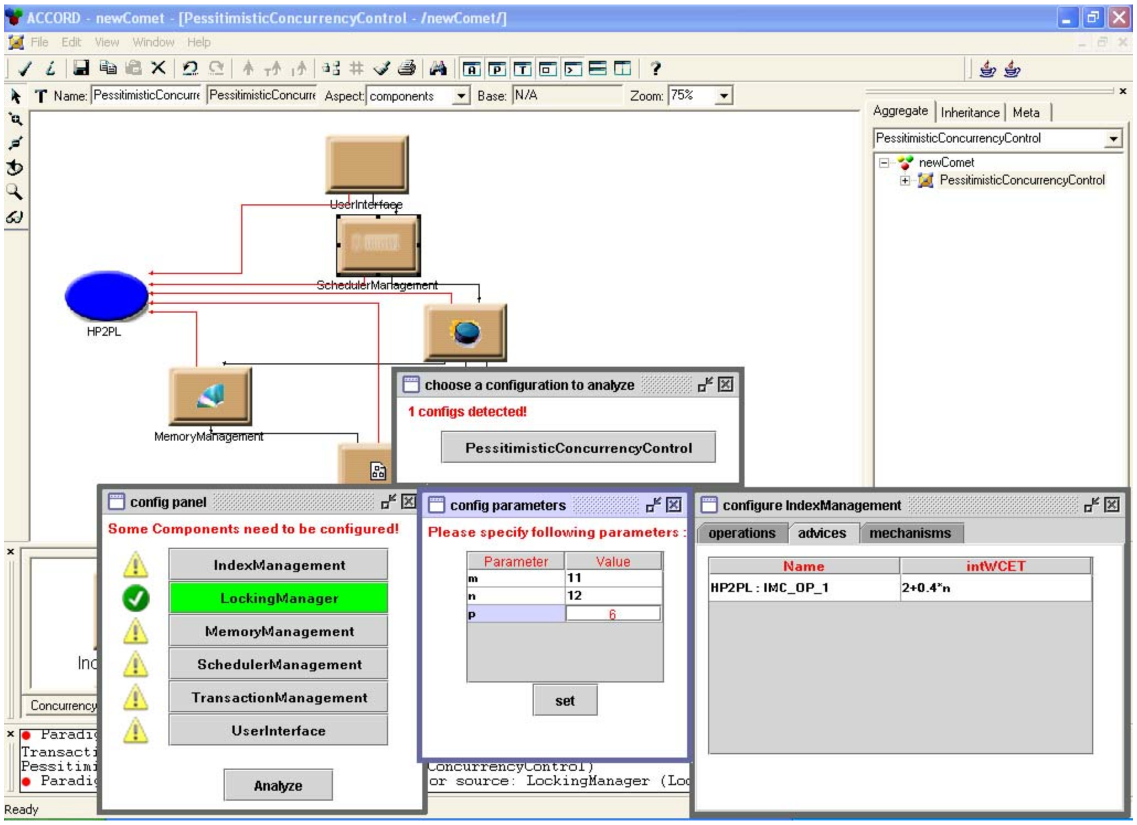
[1] Uppaal tool. http://www.uppaal.com.

[2] Constraint-specification aspect weaver (C-SAW). http://www.gray-area.org/Research/C-SAW/, December 2004.

[3] The generic modeling environment (GME). http://www.isis.vanderbilt.edu/Projects/gme/, December 2004.

[4] G. Bernat and A. Burns. An approach to symbolic worst-case execution time analysis. In *Proceedings of the 25th IFAC Workshop on Real-Time Programming*, May 2000.

[5] K.Y. Lam and W.C. Yau. On using similarity for concurrency control in real-time database systems.


ACCORD - newComet - [PessitimisticConcurrencyControl - /newComet/]
File Edit View Window Help
Name: PessitimisticConcurre
Aspect: components
Base: N/A
Zoom: 75%
Aggregate
Inheritance
Meta
PessitimisticConcurrencyControl
newComet
PessitimisticConcurrencyControl
UserInterface
SchedulerManagement
HP2PL
MemoryManagement
Ready

choose a configuration to analyze

1 configs detected!

PessitimisticConcurrencyControl

config panel

Some Components need to be configured!

- IndexManagement
- LockingManager
- MemoryManagement
- SchedulerManagement
- TransactionManagement
- UserInterface

Analyze

config parameters

Please specify following parameters :

Parameter	Value
m	11
n	12
p	6

set

configure IndexManagement

operations | advices | mechanisms

Name	intWCET
HP2PL : IMC_OP_1	2+0.4*n

Figure 5: Snapshot of ACCORD-ME when doing WCET analysis on a real-time system configuration

The Journal of Systems and Software, 43(3):223–232, 1998.

[6] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in hard real-time traffic environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, January 1973.

[7] D. Nyström, A. Tešanović, C. Norström, J. Hansson, and N-E. Bankestad. Data management issues in vehicle control systems: a case study. In *Proceedings of the 14th IEEE Euromicro International Conference on Real-Time Systems*, June 2002.

[8] Dag Nyström, A. Tešanović, M. Nolin, C. Norström, and J. Hansson. COMET: A component-based real-time database for automotive systems. In *Proceedings of the Workshop on Software Engineering for Automotive Systems at 26th International Conference on Software engineering (ICSE'04)*, May 2004. IEEE Computer Society Press.

[9] H. Schmidt. Trustworthy components-compositionality and prediction. *The Journal of Systems and Software*, pages 215–225, 2003.

[10] J. Stankovic, R. Zhu, R. Poornalingam, C. Lu, Z. Yu, M. Humphrey, and B. Ellis. VEST: an aspect-based composition tool for real-time systems. In *Proceedings of the 9th Real-Time Applications Symposium 2003*, May 2003. IEEE Computer Society Press.

[11] A. Tešanović, M. Amirijoo, M. Björk, and J. Hansson. Empowering configurable QoS management in real-time systems. In *Proceedings of the Fourth ACM SIG International Conference on Aspect-Oriented Software Development (AOSD'05)*. ACM Press, March 2005.

[12] A. Tešanović, S. Nadjm-Tehrani, and J. Hansson. chapter Modular Verification of Reconfigurable Components. *Component-Based Software Development for Embedded Systems-An Overview on Current Research Trends* Springer-Verlag, 2005.

[13] A. Tešanović, D. Nyström, J. Hansson, and C. Norström. Aspect-level worst-case execution time analysis of real-time systems compositioned using aspects and components. In *Proceedings of the 27th IFAC/IEEE Workshop on Real-Time Programming (WRTP'03)*, May 2003. Elsevier.

[14] A. Tešanović, D. Nyström, J. Hansson, and C. Norström. Aspects and components in real-time system development: Towards reconfigurable and reusable software. *Journal of Embedded Computing*, 1(1), October 2004.

[15] R. van Ommering. Building product populations with software components. In *Proceedings of the 24th International Conference on Software Engineering*, pages 255–265, May 2002. ACM Press.