A Process for Resolving Performance Trade-Offs in Component-Based Architectures

[1]Egor Bondarev, [1]Michel Chaudron, and [2]Peter de With

[1]Eindhoven University of Technology, System Architectures and Networking group
5600 MB, Eindhoven, The Netherlands
[2]LogicaCMG, 5605 JB, Eindhoven, The Netherlands
e.bondarev@tue.nl

Abstract. Designing architectures requires the balancing of multiple system quality objectives. In this paper, we present techniques that support the exploration of the quality properties of component-based architectures deployed on multiprocessor platforms. Special attention is paid to real-time properties and efficiency of resource use. The main steps of the process are (1) a simple way of modelling properties of software and hardware components, (2) from the component properties, a model of an execution architecture is composed and analyzed for system-level quality attributes, (3) for the composed system, selected execution scenarios are evaluated, (4) Pareto curves are used for making design trade-offs explicit. The process has been applied to several industrial systems. A Car Radio Navigation system is used to illustrate the method. For this system, we consider architectural alternatives, show their specification, and present their trade-off with respect to cost, performance and robustness.

1 Introduction

A major challenge in system development is finding the best balance between different quality requirements that a system has to meet. Time-to market constraints require that design decisions be taken as early as possible. To address this challenge, the architect should be able to solve a number of orthogonal issues: a) construct the component architecture satisfying the functional requirements, b) evaluate (predict) the extra-functional quality properties of the composed architecture, and c) identify several architecture alternatives that satisfy both types of requirements. Essentially, he needs a means to efficiently explore this architectural design space against multidimensional quality attribute scale.

A concurrent trend is the assembly of systems out of existing components (which can be both software and hardware), as this reduces development time and cost. Within this component-based approach, the challenge of early architecture assessment shifts to the evaluation of global system properties based on the properties of the constituent components. For this reason, the component-oriented society needs to develop techniques for modelling component properties such that these can be composed into a system model. Each model type usually addresses one attribute (performance, behaviour or cost). Upon component integration, models of the same type are composed into a system model of the corresponding system attribute. There has been a broad range of approaches towards this problem, known as *predictable assembly* ([1], [2] and [3]). The CB-SPE approach [4] provides a solid technique for evaluating the performance of component-based software systems. A prediction method based on formal specification of component non-functional properties is presented in [5]. Currently, these approaches focus on prediction of a single quality attribute (QA). To motivate various design trade-offs, assessment of multiple QAs is needed. The process, described in this paper, allows prediction of performance, robustness and cost QAs, and enables design space exploration with respect to these attributes.

The following methods feature multi-objective trade-off analysis. The method presented in [9] uses Petri nets with parameterized interfaces to assess performance and safety. For large system this method becomes computation expensive. Less calculation expensive PISA framework for design space exploration featuring QA prediction have been proposed in [10] for network processor architectures. It uses Real-Time Calculus that abstracts from the state space and has low calculation complexity. Recently proposed SESAME framework [11] uses simulation, application and architecture models to predict performance properties and explore design choices. The SPIE2 framework [12] adds the possibility to optimize the architecture using genetic algorithms. However, none of the last three methods supports designing a system out of conventional component with provides and requires interfaces. Instead, they define a component as an active entity (task or process). In [16] the authors propose compile-time framework that explores and optimizes performance properties of systems built out of active conventional components.

Contribution. In this paper we present the design space exploration (DSE) process that supports both active and passive COTS components. It allows software and hardware composition and mapping the components on the hardware nodes. The process enables accurate prediction of system performance attributes by composition of performance models of individual components. The component performance models are easy to construct and use, which speeds up the architecture assessment time. The supporting RTIE tool, developed by us, helps to construct multiple architecture alternatives and find the optimal solutions against multiple criteria. We illustrate the process by a case study on finding the optimal architecture for the Car Radio Navigation (CRN) system.

The paper is structured as follows. Section 2 explains the requirements of the CRN system to be designed. Section 3 describes our multidimensional DSE process in detail. Section 4 shows how we used the process to design and assess the architectures for CRN system, besides, it reveals the experimental results of the case study. Section 5 concludes the paper.

2 Car Radio Navigation System

We illustrate our process for resolving performance design trade-offs in CBA by designing a Car Radio Navigation system. This CRN system had to be built according to the component-based paradigm on a cost-limited (yet not predefined) hardware platform. However, the major challenge was to find at an early design stage an optimal system architecture in terms of the vital QAs like real-timeliness, robustness and cost. Technically speaking, the goal was the following: *given* a set of functional and extra-functional requirements, as well as a set of software and hardware components, to *determine* a set of architecture solutions, that are optimal with respect to the above-mentioned quality attributes.

Requirements. We divided the requirements into two categories: functional (Fn) and extra-functional (RTn). The main ones are summarized below:
F1: The system shall be able to gradually (scale = 32 grads) change the sound volume.
RT1: The response time of the operation F1 is less than 200 ms (per grade).
F2: The system shall be able to find and retrieve an address specified by the user.
RT2: The response time of the operation F2 is less than 200 ms.
F3: The system should be able to receive and handle Traffic-Message-Channel (TMC) messages.
RT3: The response time of the operation F3 for one message is less than 350 ms.

Functional decomposition. Requirement analysis led us to a conceptual software view depicted in Fig. 1.

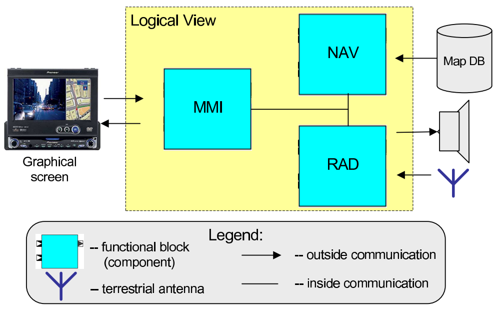


Fig. 1. Overview of the CRN system functionality.

The CRN logical view has three major functional blocks:

– The man-machine interface (MMI), that takes care of all interactions with the end-user, such as handling key inputs and graphical display output.
– The navigation functionality (NAV) is responsible for destination entry, route planning and turn-by-turn route guidance giving the driver visual advices. The navigation functionality relies on the availability of a map database and positioning information.
– The radio functionality (RAD) is responsible for tuner and volume control as well as handling of TMC traffic information services.

In the next section, we illustrate our DSE process that enables architecture comparison and supports resolving design trade-offs with respect to multiple performance attributes and cost.

3 Multidimensional Design Space Exploration Process

Fig. 2 depicts our DSE process that uses a component-based architecture as the skeletal structure, onto which the composition of QAs can be performed out of models of individual components. We developed an RTIE (Real-Time Integration Environment) toolset that supports all the steps in the DSE flow. A distinguishing feature of our process is that the analysis is based on the evaluation of a number of key execution *scenarios*. The use of scenarios enables efficient analysis while also enabling the architect to trade modelling effort (modelling multiple scenario's improves the 'coverage' of the behaviour of the system) against confidence in the results. The other cornerstones of the approach are:

- *modelling* of software components, processing nodes, memories and bus links,
- *composition* of system QAs out of these models,
- *prediction of a system timing behaviour and resource usage*, required for real-time system design,
- *pareto curves* for identification of optimal architecture alternatives.

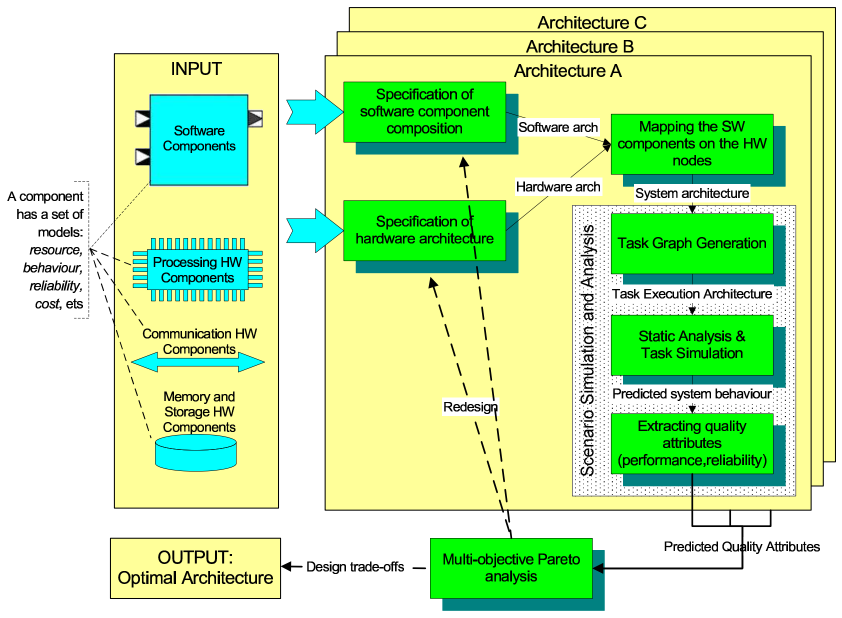

Architecture C
Architecture B
Architecture A
INPUT
Software Components
A component has a set of models: *resource, behaviour, reliability, cost*, ets
Processing HW Components
Communication HW Components
Memory and Storage HW Components
Specification of software component composition
Specification of hardware architecture
Software arch
Hardware arch
Mapping the SW components on the HW nodes
System architecture
Scenario Simulation and Analysis
Task Graph Generation
Task Execution Architecture
Static Analysis & Task Simulation
Predicted system behaviour
Extracting quality attributes (performance,reliability)
Redesign
Predicted Quality Attributes
OUTPUT: Optimal Architecture
Design trade-offs
Multi-objective Pareto analysis

Fig. 2. Multidimensional design space exploration process.

Let us outline the process phases. As input for an architecture, the system designer has various third-party hardware and software components (in a repository). Each component should be supplied with a set of models addressing important component attributes, like timeliness, cost, resource use. Relevant types of models for a software (SW) component are: functional, resource and behaviour models. Our example shows that these models can be made with comparatively little effort. Typical models for hardware (HW) components are: memory, communication and processing models.

The following steps are to be done for each architectural alternative (see Fig. 2). Considering more alternative solutions leads to more complete coverage of the design space.

Software Architecture Composition. The designer selects from the (RTIE) repository the software components that together satisfy the defined functional requirements and *may* satisfy extra-functional requirements. The component selection is done by checking the functional models of available components with respect to the functional requirements. The process assumes that the selected components are supplied along with corresponding set of models. By means of the RTIE graphical tool, the designer specifies component composition by instantiating and connecting components. The resulting composition is converted into XML-file with links to the individual component models stored in the repository.

Hardware Architecture Specification. The hardware architecture specification can be done in parallel. In most of the cases, a hardware platform is pre-specified. If not, the designer can select available hardware components from a repository and choose a specific topology, number of processing nodes, types of memory, communication means and scheduling policy. Then, he puts these together on a design canvas, thereby specifying the hardware architecture. The architecture is also represented in XML-file with references to the models of hardware nodes.

SW/HW Mapping. Once the software and hardware architecture are specified, the mapping of the software components on the hardware nodes is made. The mapping shows on which processing node each software component should be executed. Efficient mapping is required to distribute the load of hardware resources in an optimal way. However, at the first mapping iteration, it is not clear how to deploy the software components to achieve the optimal load distribution. Various mapping alternatives are possible at this stage. Each alternative represents a system architecture.

Model Synthesis and Scenario Simulation. Some system attributes like *cost* can be found analytically given a static architecture. However, for prediction of other important system attributes (performance and robustness) the behaviour of a system needs to be found. In our process, we obtain these attributes through the *scenario simulation method* [6]. This method synthesize a model of the *task execution architecture* by composing *resource* and *behaviour* models of individual software components, *performance* models of hardware nodes and *scenario* model of the constructed software composition. All these models enable parameter-dependent specification. For the details of parameter-dependent modeling the reader is referred to [15]. The following two paragraphs specify models in more detail.

The *resource model* contains parameter-dependent processing and memory requirements of each operation implemented by the component. The resource requirements can be obtained by profiling of each individual component on a reference processor. The reference processor is also specified in the model in order to scale the operation resource requirements to any other processor. The *behaviour model* specifies for each implemented operation a parameter-dependent sequence of external calls to operations offered by other interfaces. The external call is a (synchronous or asynchronous) invocation of other interface's operation made inside the implemented operation. The data for the behaviour model can be obtained by the source-code analysis. The *performance model* of a hardware block specifies its capabilities. A performance model for a processing core defines its instruction type (RISC, CISC or VLIW) and execution frequency. A model for a memory IP block describes a memory type (SRAM, SDRAM, etc), a memory size in MBytes and addressing type. A bus performance model specifies the scheduling protocol (TDMA, CDMA, fare use, etc) and bandwidth size. The data for performance models can be obtained by measurements or from supplier data sheets.

For software composition architecture, the designer defines a set of resource-critical scenarios and for each of them specifies an application *scenario model*. Critical scenarios are the application execution configurations that may introduce processor, memory or bus overload. In the scenario, the designer may specify environmental stimuli (events or thread triggers) that influence the system behaviour. For a stimulus, the designer may define the burst rate, minimal inter-arrival time, period, deadline, offset, jitter, task priority, and so on. By defining the stimuli, the designer specifies autonomous behaviour of the system, or emulates an environmental influence (interrupts, network calls) to the system.

The scenario, resource, behaviour and performance models are synthesized by the RTIE tool. The objective of the synthesis is to reconstruct (generate) the tasks running in the application. Prior to the synthesis, the task-related data is spread over different types of models. For instance, the *task periodicity* may be specified in an application scenario model, whereas the information about the *operation call sequence* comprising the task is spread over corresponding component behaviour models. The compiler combines these two types of data in the task information containing period, jitter, offset, deadline and operation sequence call graph. The synthesis results in the *task execution architecture* that contains parameter-dependent data on the tasks running in the designed system and data on the allocation of these tasks on the software and hardware architectures. An example of this allocation is given in Fig. 3. Here, the system executes three tasks using two processors and five deployed service instances. The

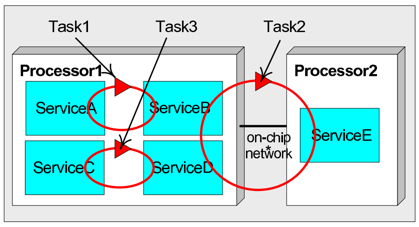


Fig. 3. Task allocation on the component and hardware architecture.

Task1 executes on Processor1 and consists of operations offered by ServiceA and ServiceB. The Task3 execution is spread over both processors and includes a communication via the on-chip network. The task executes operations offered by three service instances: ServiceB, ServiceD and ServiceE.

The obtained task execution architecture is a subject for virtual scheduling (simulation). A simulation-based analysis employs virtual schedulers that simulate the execution of the tasks specified in the system model for some period of time. The selection of a scheduling algorithm is dictated by the types of communication lines and operating system used for the designed system. The RTIE tool provides the following virtual schedulers: rate monotonic (RM), deadline monotonic (DM), earliest deadline first (EDF), constant bandwidth server (CBS), time division multiple access (TDMA) and fare-use algorithms. The simulation techniques feature both processing and communication resources scheduling. An example of the simulation results is given in Fig. 4.

The diagram shows the execution timelines of the three processors and the bus-load timeline. For each processor timeline, the tasks executing the operations of the services that are mapped on the processor are shown. For each task instance, its initiation and completion times are given. Beside this, the diagram reflects the time slots when a task instance misses its deadline. The bus-load timeline represents the timed bus utilization done by the communicating operations in these three tasks. The statistics, generated from the simulation timelines, gives the overall data on the predicted task properties and load of the resources.

Quality attribute extraction. The *throughput*, *latency* and *resource consumption* QAs are extracted in a straightforward way from the generated task simulation timeline. For other attributes, like *robustness* additional computation is needed. Robustness can be calculated as performance sensitivity to stimuli rate increase. For this, the designer changes the stimuli rate in each of the scenario system models and redo the simulations. Comparison of the new task latencies or resource use with the old values answers the question on how sensitive is the architecture against the input event rate changes.

Multi-objective Pareto analysis. At this stage, having defined a number of alternative architectures and predicted multiple QAs for each of them, we look for an optimal design alternative. Pareto analysis is a powerful means for

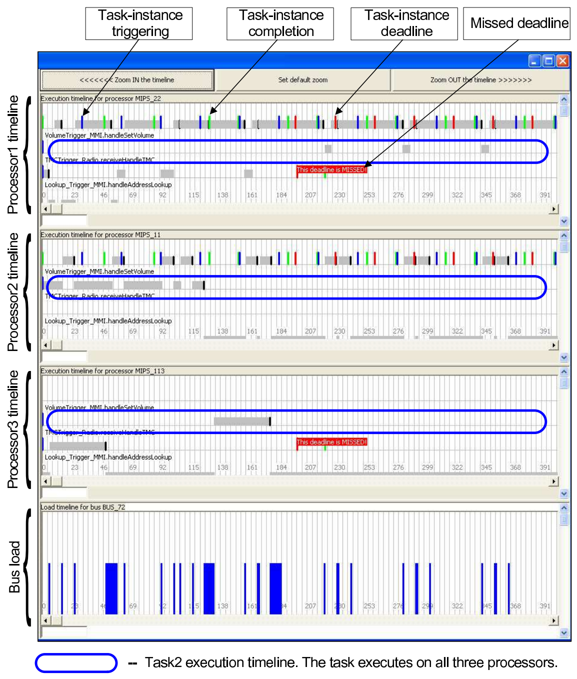

Task-instance triggering
Task-instance completion
Task-instance deadline
Missed deadline
<<<<<< Zoom IN the timeline
Set default zoom
Zoom OUT the timeline >>>>>>
Processor1 timeline
Execution timeline for processor MIPS_22
VolumeTrigger_MMI.handleSetVolume
TMCTrigger_Radio.receiveHandleTMC
Lookup_Trigger_MMI.handleAddressLookup
This deadline is MISSED!
Processor2 timeline
Execution timeline for processor MIPS_11
Processor3 timeline
Execution timeline for processor MIPS_113
Bus load
Load timeline for bus BUS_72
– Task2 execution timeline. The task executes on all three processors.

Fig. 4. Execution timelines for tasks on three processors obtained by RMA simulation.

resolving conflicting objectives [7]. The multi-objective optimization problem does not yield a unique solution, but a set of solutions that are Pareto-optimal. An example of the Pareto analysis is shown in Section 4.4.

4 The Quest for an Optimal CRN Architecture

For this case study, we implemented and packaged three Robocop software components: RAD, MMI and NAV, which correspond to above-mentioned CRN functional blocks. The Robocop component model [8] supports modelling and composition of a wide spectrum of component attributes and is targeted to embedded systems domain. The Robocop component is an open set of models (see Fig. 5).

For example, the functional model specifies the component functionality, while resource model (see Fig. 6.B) specifies resource utilization of the component operations. The executable entity (.dll file) is also considered as a special type of model. A Robocop component can be downloaded from a common repository as a black-box and used for third-party binding. A component developer is responsible for specification of the models. The executable component may include a number of executable entities called services. A service may have provides and requires interfaces. The provides interfaces specify and give access to operations implemented by the service.

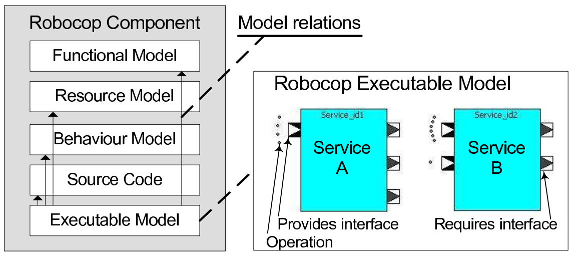


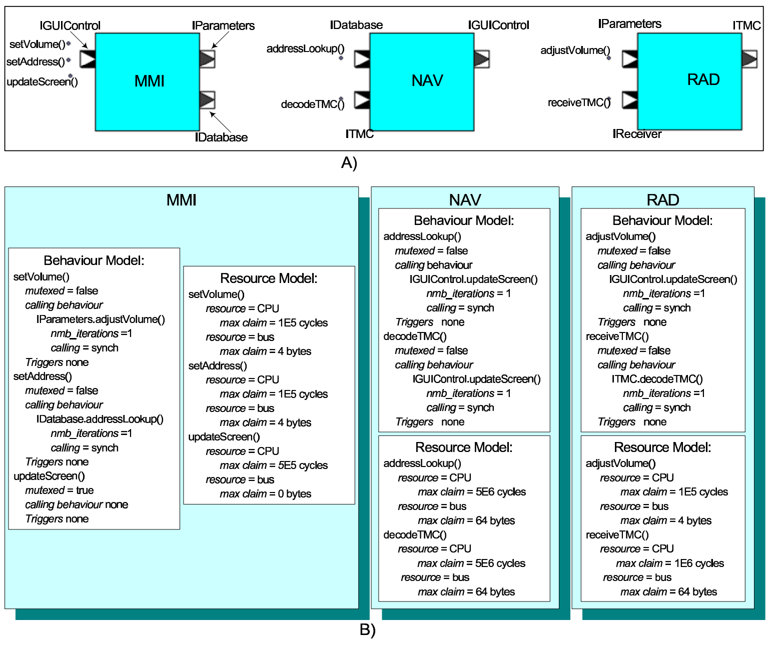
Fig. 5. Robocop component model.

The three implemented components, their provides/requires interfaces and operations are depicted in Fig. 6.A. The MMI component provides IGUIControl interface and requires to be bound to IParameters and IDatabase interfaces. The GUIControl interface provides access to three implemented operations: setVolume (handles the volume rotary button request from the user), setAddress (handles the address keyboard request from the user) and updateScreen (updates the GUI display). The NAV component provides IDatabase, ITMC interfaces and requires operations from the IGUIControl interface. The IDatabase interface gives access to addressLookup() operation, which queries the address in the database and finds a path to this address. The ITMC interface provides an access to decodeTMC() operation. The RAD component provides IParameters, IReceiver interfaces and requires ITMC interface. The two operations implemented by this component are adjustVolume() and receiveTMC().

Each component is accompanied by *resource*, *behaviour* (see Fig. 6.B), and *cost* models. The resource model specifies resource requirements per individual operation. The behaviour model describes the operation's underlying calls to operations of other interfaces. Besides, the model may specify a periodic thread triggers (like Posix thread with a periodic timer), if they are implemented inside the component. Both resource and behaviour models are composable, i.e. from a number of behaviour models of constituent components one can generate a system behaviour model. The composition principles are explained in detail in [6]. The resource requirements (CPU claim) has been obtained by profiling of each individual component on a reference RISC processor. The operation behaviour data has been generated from the component source code. For example, the RAD behaviour model describes that the operation adjustVolume() synchronously calls once the IGUIControl. updateScreen() operation. This model also shows the bus usage of the adjustVolume() operation: 4 bytes. That means the operation sends outside (as an argument of updateScreen()) 4 bytes of data.

4.1 Defining Architecture Alternatives

Following the process, we composed a component assembly (see Fig. 7.A) from the available components. We were able to design only one software architecture alternative due to a limited number of available software components. These



```
MMI
Behaviour Model:
setVolume()
   mutexed = false
   calling behaviour
      IParameters.adjustVolume()
         nmb_iterations =1
         calling = synch
   Triggers none
setAddress()
   mutexed = false
   calling behaviour
      IDatabase.addressLookup()
         nmb_iterations =1
         calling = synch
   Triggers none
updateScreen()
   mutexed = true
   calling behaviour none
   Triggers none

Resource Model:
setVolume()
   resource = CPU
      max claim = 1E5 cycles
   resource = bus
      max claim = 4 bytes
setAddress()
   resource = CPU
      max claim = 1E5 cycles
   resource = bus
      max claim = 4 bytes
updateScreen()
   resource = CPU
      max claim = 5E5 cycles
   resource = bus
      max claim = 0 bytes

NAV
Behaviour Model:
addressLookup()
   mutexed = false
   calling behaviour
      IGUIControl.updateScreen()
         nmb_iterations = 1
         calling = synch
   Triggers   none
decodeTMC()
   mutexed = false
   calling behaviour
      IGUIControl.updateScreen()
         nmb_iterations = 1
         calling = synch
   Triggers   none

Resource Model:
addressLookup()
   resource = CPU
      max claim = 5E6 cycles
   resource = bus
      max claim = 64 bytes
decodeTMC()
   resource = CPU
      max claim = 5E6 cycles
   resource = bus
      max claim = 64 bytes

RAD
Behaviour Model:
adjustVolume()
   mutexed = false
   calling behaviour
      IGUIControl.updateScreen()
         nmb_iterations =1
         calling = synch
   Triggers   none
receiveTMC()
   mutexed = false
   calling behaviour
      ITMC.decodeTMC()
         nmb_iterations =1
         calling = synch
   Triggers   none

Resource Model:
adjustVolume()
   resource = CPU
      max claim = 1E5 cycles
   resource = bus
      max claim = 4 bytes
receiveTMC()
   resource = CPU
      max claim = 1E6 cycles
   resource = bus
      max claim = 64 bytes
```

Fig. 6. (A) Components used for the case study; (B) Behaviour and resource models of the selected components.

three components were instantiated and bound together via pairs of their provides/requires interfaces. This assembly satisfies the three defined functional requirements: F1, F2 and F3.

The next phase is to define a set of hardware architectures and map the software components onto hardware. We reused five feasible alternative hardware architectures with different mapping schemas proposed in [13] (see Fig. 7.B). For instance, in Architecture A there are three processing nodes connected with a single bus of 72 kbps bandwidth. The MMI_Inst component is executed (mapped) on a 22-MIPS processor, the NAV_Inst component is mapped on a 113-MIPS processor, and RAD_Inst component executes on a 11-MIPS processor. The capacity of the processing nodes and communication infrastructure was taken from the datasheets of several commercially available automotive CPUs. The multi-objective DSE process has been performed for these five solutions.

4.2 Scenarios and Task Generation

For our case study, we selected three distinctive execution scenarios to assess the architecture against the six defined requirements. These scenarios should impose the highest possible load on the hardware resources for accurate evaluation of the real-time requirements RT1, RT2 and RT3.

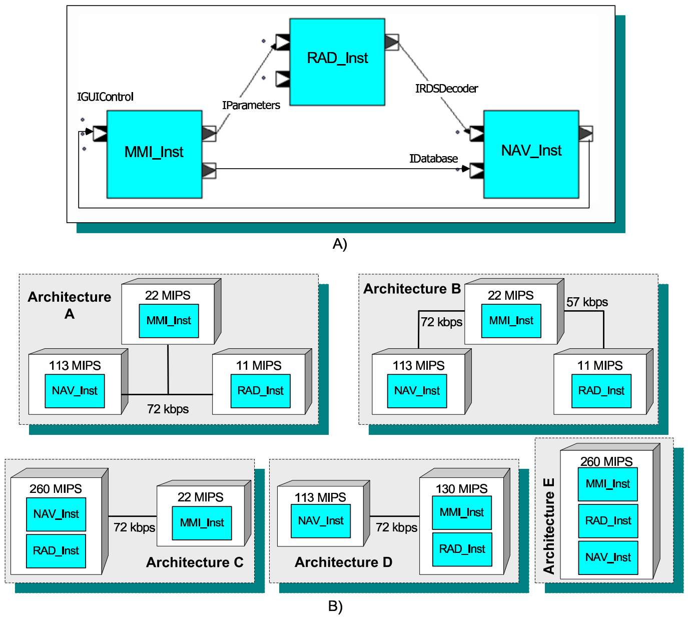


Fig. 7. (A) Software component assembly of the CRN system. (B) Five alternative system architectures to explore.

"Change Volume" scenario. The user turns the rotary button and expects instantaneous audible and visual feedback from the system. The maximum rotation speed of the button is 1 sec from lowest to highest position. For emulating this user activity, we introduced a VolumeStimulus task trigger, which initiates execution of the IGUIControl.setVolume() operation. The trigger parameters are defined in the following way: the event period is set to 1/32 sec, as the volume button scale contains 32 grades. The task deadline is set to 200 ms, according to R1. The trigger and component assembly resemble a scenario model.

For this scenario, the RTIE tool generated (from the behaviour models of participating components) the message sequence chart (MSC) of operation calls involved in the task execution. The scenario model and obtained MSC are shown in Fig. 8.A. The task is executed periodically (31 ms) and passes through MMI_Inst and RAD_Inst.

"Address Lookup" scenario. Destination entry is supported by a smart typewriter style interface. The display shows the alphabet and the user selects the first letter of a street. By turning a knob the user can move from letter to letter; by pressing it the user selects the currently highlighted letter. The map database is searched for each letter that is selected and so on. We assume that the worst-case rate of the letter selection is 1 time per second. This user activity was emulated with a LookupStimulus trigger, which initiates execution of the IGUIControl.setAddress() operation. The trigger period was set to 1000 ms. The deadline for the address lookup task is 200 ms, according to RT2.

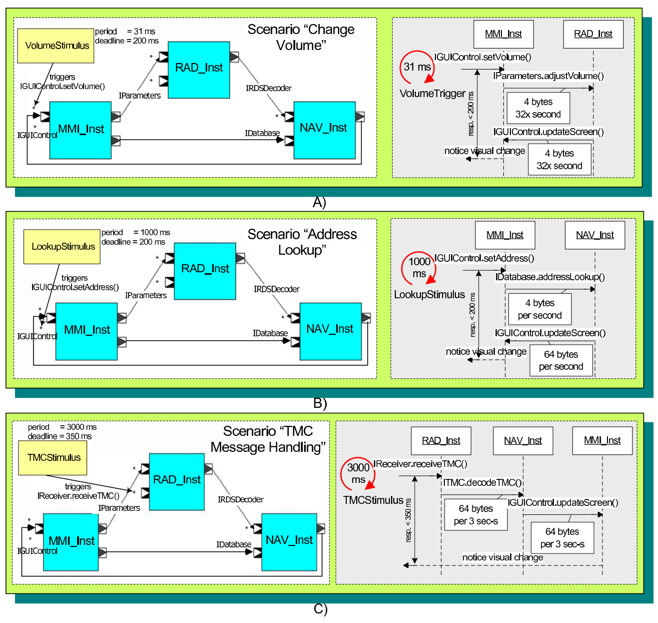


Fig. 8. Model and message sequence chart for scenarios: (A) Change Volume; (B) Address Lookup, and (C) TMC Message Handling.

The task-generation procedure outputs the task MSC for this scenario. The obtained scenario model and MSC are shown in Fig. 8.B. The task is executed periodically (1000 ms) and passes the MMI_Inst and NAV_Inst components.

"TMC Message Handling" scenario. RDS TMC is a digital traffic information that enables automatic replanning of the route in case of traffic jam. Traffic messages are received by the RAD component (in the worst case 1 time per 3 seconds). We introduced a TMCStimulus trigger emulating these TMC messages. The trigger initiates execution of the IReceiver.receiveTMC() operation. The period is set to 3000 ms. The deadline for the TMC handling task is set to 350 ms, according to RT3.

The task-generation procedure resulted in the task MSC for this scenario. The obtained scenario model and task are represented in Fig. 8.C. The task is executed periodically (3000 ms) and passes through three component instances: RAD_Inst, MMI_Inst and NAV_Inst. The fully decoded messages are forwarded to the user.

4.3 Simulation and Attribute Extraction

The scenarios sketched above have an interesting property: they can occur in parallel. TMC messages must be processed while the user changes the volume or enters a destination address at the same time. Therefore, we combined these three scenarios into two in order to get worst-case load on the system resources during simulation. We defined *ScenarioA* as a combination of the SetVolume and TMCHandling scenarios, and *ScenarioB* as a combination of the Address-Lookup and TMCHandling scenarios. From the processing point of view, both new scenarios have two tasks executing in parallel.

Table 1. Experimental data of the predicted quality attributes.

Attribute	**Arch. A**	**Arch. B**	**Arch. C**	**Arch. D**	**Arch. E**
Max. task latency against RT1 (RT1=200ms)	37.55 ms	37.55 ms	30.52 ms	9.18 ms	3.58 ms
Max. task latency against RT2 (RT2=200ms)	86.51 ms	86.51 ms	61.49 ms	63.79 ms	21.05 ms
Max. task latency against RT3 (RT3=350ms)	325.05 ms	395.05 ms	101.71 ms	114.12 ms	46.02 ms
Performance sensitivity (latency increase for TMC handling)	57.6%	51.1%	3.2%	3.1%	0.0%
Cost, euro	290	305	380	335	340

Following our DSE process, we simulated the execution of these two scenarios for each of the five system architectures. Before simulation, the following pre-processing of the computation and communication time data is performed. For each of the processing nodes, the execution times of all operations to be executed on the node are calculated from the *component resource* and *node performance* models (execution_time = *CPU_claim*_value * processor_speed). The communication time of the operation calls made through the processor boundaries is calculated by dividing the *bus claim* value of an operation on a bus bandwidth value, defined in a bus performance model.

The scenario simulation by preemptive RM algorithm (other policies can also be used) resulted in (a) predicted system timing behaviour description and (b) resource consumption of a system for each scenario and task worst-case latencies. First, we analyzed the predicted *task latencies* against the real-time requirements RT1, RT2 and RT3 for each of the five architectures (see Table 1).

Analyzing the table data, we concluded that except for Architecture B, the rest of the four architectures satisfy the given real-time requirements. The Architecture B does not satisfy the requirement RT3, because it has TMCHandling task latency higher than 350 ms. Architecture A can be considered fast enough; architecture E is the fastest solution. Then, we analyzed the architecture robustness as a performance *sensitivity* to the changes in the input event rates (arrival period of the three stimuli). We increased the data rate of the three stimuli by 5% (i.e. VolumeStimulus to 33.6 events/s, LookupStimulus to 1.05 events/s and

TMCStimulus to 0.35 events/s). Afterwards, we re-simulated the adjusted scenarios and obtained new task latencies. The fourth row in Table 1 describes the increase of the latency of the TMC handling task as percentage of the normal latency per architecture. For instance, the end-to-end delay of the TMC message handling task for architecture A increased by 57.6%! This happened due to a high overload of the 22-MIPS processor in this scenario.

The system cost attribute was calculated as a cumulative *cost* of the system hardware and software components. The software component cost has been defined with correlation to the component source code complexity (in reality, the cost of a third-party component is defined by the component producer). The cost of the hardware components was calculated from the available market prices. The total calculated cost for each architecture is given in Table 1. The most expensive architecture was number C due to the costly high-performance processing nodes.

4.4 Analysis of Architecture Alternatives

The performance, robustness and cost attributes were selected as main objectives for our design space exploration. Using the RTIE Pareto analysis tool, we obtained several two-dimensional Pareto graphs. Two of them, *robustness vs. cost* and *performance vs. cost* are depicted in Fig. 9.

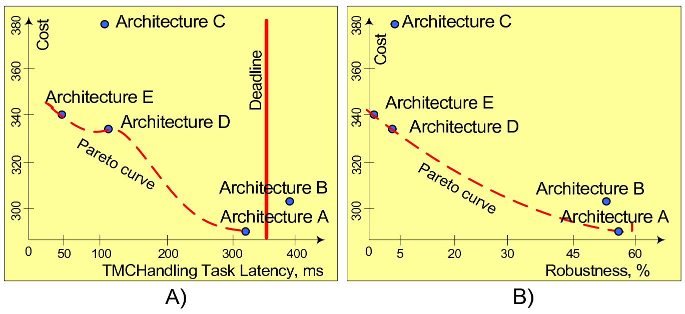


Fig. 9. Pareto exploration graphs based on A) performance vs. cost, B) robustness vs. cost quality attributes

The graphs can be evaluated as follows. The Pareto curve is drawn by connecting the alternatives that are closest to the origin. This curve defines a set of optimal alternatives. With respect to the cost-robustness trade-off (see Fig. 9.B), the optimal architectures are E, D and A, because they create the curve closest to the null-coordinate point. The alternatives C and B are non-optimal. The choice from the three alternative architectures depends on a weighting function (priority) for the cost and robustness attributes. If cost has higher priority then Architecture A should be selected. If performance sensitivity is a critical factor, then the Architecture A is not the best candidate. Moreover, looking at the cost-performance trade-off (see Fig. 9.A), we can observe that TMC task latency for Architecture A is close to its deadline. Thereby, low robustness (57.6%) of Architecture A cannot be tolerated.

With respect to the cost-performance trade-off, again the optimal alternatives are E, D and A, though C is not positioned on the hypothetical ideal Pareto curve. The Architecture B gets out of competition because its TMC task latency is higher than task deadline. Despite of its low cost, the Architecture A with low performance and insufficient robustness can be also omitted.

Concluding, the Architectures E and D can be considered as optimal alternatives. If the cost weighting function is higher than performance or robustness weighting function, the architecture E can be adopted for further development and *vice versa*. In addition, we may also re-iterate the DSE process to achieve acceptable performance for less costly Architecture A. For instance, we can add a new software component TMCHandler, which reduces TMCHandling task latency, or re-dimension one of the processing node. Another optimization technique would be to reduce the cost of the Architecture E, by sacrificing (within acceptable range) its performance and robustness.

5 Conclusion

The proposed DSE process includes the steps of designing, predicting quality attributes and evaluating the architectural alternatives. The accuracy of performance attribute prediction has been previously validated by a case study on a Linux-based MPEG-4 player [14]. The prediction accuracy on the general performance proved to be higher than 90%. In all case studies, the modelling effort required from application designer was fairly small - in the order of hours. The most of the modelling work goes to the component developer, because he should provide the component models. Thereby, the application developer may relatively easily model a system out of 100 components (scalability), because necessary models are already supplied within these components. The process enables early identification of the bottlenecks of individual alternatives and leads to selection of optimal solutions. In this paper we address strictly performance attributes, however the proposed DSE process enables targeting other important QAs, like reliability and availability. This extensibility is realized by open component model structure, in which new model types can be easily added.

Limitations. There are certain limitations of the process. Firstly, the component behaviour model represents an abstraction of the component source code, leaving out implementation details. That eases the assessment of the component and system behaviour, but limits the specification of all aspects of the source code, like complex parameter-dependent loops and condition forks implemented inside a component operation. Secondly, to explore the design space of a system, a designer can *only* select the components that already contains required cost, resource and behaviour models. Moreover, the QAs that are system-wide, like safety and security, cannot be easily localized and modeled at the component level. Thirdly, introduction of scenarios requires that the designer has a good understanding of the system-environment interaction aspects, and has some analytical skills in identifying the scenarios. The scenario identification criteria is our ongoing work. Finally, the RTIE tool does not facilitate generation of complete design space. Instead, the designer is responsible for identification of the architecture alternatives.

In our future plans, we focus on development of automated optimization algorithms as a back-end for this exploration process. Genetic algorithms can be used to generate better alternatives by varying the topology, mapping and scheduling-policy of an architecture.

References

1. I. Crnkovic and M. Larsson. *Building Reliable Component-based Software Systems*, Artech House, 2002
2. K.C. Wallnau, "Volume III: A Technology for Predictable Assembly from Certifiable Components", *CMU/ESI-2003-TR-009 report*, April 2003.
3. S.A. Hissam, *et al.*, "Packaging Predictable Assembly with Prediction-Enabled Component Technology", *CMU/ESI-2001-TR-024 report*, November 2001.
4. A. Bertolino, R. Mirandola, "CB-SPE Tool: Putting Component-Based Performance Engineering into Practice", *Proc. 7th Symp. on CBSE*, Edinburgh, UK. Vol. 3054 of LNCS, Springer (2004) 233-248.
5. S. Zschaler, "Towards a Semantic Framework for Non-functional Specifications of Component-Based Systems", *Proc. 30th EUROMICRO Conf.*, France, Sep. 2004.
6. E. Bondarev, *et al*, "Predicting Real-Time Properties of Component Assemblies: a Scenario-Simulation Approach", *Proc. 30th Euromicro Conf.*, Sep. 2004.
7. C.A. Mattson and A.Messac, "A Non-Deterministic Approach to Concept Selection Using s-Pareto Frontiers", *Proc. ASME DETC 2002*, Canada, Sep. 2002.
8. "Robocop: Robust Open Component Based Software Architecture", http://www.hitech-projects.com/euprojects/robocop/deliverables.htm
9. Schmidt, H.W. *et al*, "Modelling Predictable Component-based Distributed Control Architectures", *Proc OORTDS workshop*, 2003, 339-346
10. L. Thiele *et al*, Design Space Exploration of Network Processor Architectures, *Network Processor Design: Volume 1*, Morgan Kaufmann Publishers, 2002.
11. A. D. Pimentel *et al*, "A Systematic Approach to Exploring Embedded System Architectures at Multiple Abstraction Levels", *IEEE Trans. on Computers*, Vol. 55), Feb. 2006.
12. M. Zitzler *et al*, "SPEA2: Improving the performance of the strength pareto evolutionary algorithm", *Technical Report TIK-Report 103*, ETH, Zurich, May 2001.
13. E. Wandeler, L. Thiele, M. Verhoef, "System Architecture Evaluation Using Modular Performance Analysis - A Case Study", *Proc. 1th ISOLA Symposium*, 2004.
14. E. Bondarev *et al* "On Predictable Software Design of Real-Time MPEG-4 Video Applications", *SPIE Proc. VCIP 2005*. China. July, 2005.
15. E. Bondarev *et al.*, "Modelling of Input-Parameter Dependency for Performance Predictions of Component-Based Embedded Systems", *In Proc. of 31th Euromicro Conference; CBSE Track*, Porto, September 2005.
16. J. Fredriksson *et al*, "Optimizing Resource Usage in Component-Based Real-Time Systems", *Proc 8th CBSE Symposium*, May, 2005.