# Introducing a Component Technology for Safety Critical Embedded Real-Time Systems

Kristian Sandström, Johan Fredriksson, and Mikael Åkerholm

Mälardalen Real-Time Research Centre,
Department of Computer Science and Engineering,
Mälardalen University, Box 883, Västerås, Sweden,
http://www.mrtc.mdh.se
kristian.sandstrom@mdh.se

**Abstract.** Safety critical embedded real-time systems represent a class of systems that has attracted relatively little attention in research addressing component based software engineering. Hence, the most widely spread component technologies are not used for resource constrained safety critical real-time systems. They are simply to resource demanding, to complex and to unpredictable. In this paper we show how to use component based software engineering for low footprint systems with very high demands on safe and reliable behaviour. The key concept is to provide expressive design time models and yet resource effective runtime models by statically resolve resource usage and timing by powerful compile time techniques. This results in a component technology for resource effective and temporally verified mapping of a component model to a commercial real-time operating system.

## 1 Introduction

The vehicle domain represents a class of embedded real-time systems where the requirements on safety, reliability, resource usage, and cost leaven all through development. Historically, the development of such systems has been done using only low level programming languages, to guarantee full control over the system behaviour. As the complexity and the amount of functionality implemented by software increase, so does the cost for software development. Therefore it is important to introduce software development paradigms that increase software development productivity. Furthermore, since product lines are common within the domain, issues of commonality and reuse is central for reducing cost as well as increasing reliability.

Component based software engineering is a promising approach for efficient software development, enabling well defined software architectures as well as reuse. Although component technologies have been developed addressing different demands and domains, there are few component technologies targeting the specific demands of safety critical embedded real-time systems. Critical for the safe and reliable operation of these systems is the real-time behaviour, where the timeliness of computer activities is essential. To be able to guarantee these

properties it is necessary to apply real-time systems theory. Thus, a component technology to be used within this domain has to address specification, analysis, and implementation of real-time behaviour.

A typical real-time constraint is a deadline on a transaction of co-operating activities. A transaction in these systems would typically sample information about the environment, perform calculations based on that information and accordingly apply a response to the environment, all within a limited time frame. Also important is the ability to constrain the variation in periodicity of an activity (jitter). The reason for this is that variations in periodicity of observations of the environment and responses to the same, will affect the control performance. Hence, a component technology for this domain should have the ability to clearly express and efficiently realize these constraints [1],[2],[3],[4].

The work described in this paper present a component technology for safety critical embedded real-time systems that is based on experience from our previous work with introducing state-of-the-art real-time technology in the vehicle industry. The benefits in development have been discussed in [5] and have also been proven by long industrial use. That real-time technology has been incorporated in the Rubus development suite and has been further developed [6]. Experience from the industrial application of the research reveals that a proper component model is not enough; success requires an unbroken chain of models, methods, and tools from early design to implementation and run-time environment.

The contribution of the work presented in this paper includes a component technology for resource effective and temporally verified mapping of a component model to a resource structure such as a commercial Real-Time Operating System (RTOS). This is made possible by introduction of a component model that support specification of high level real-time constraints, by presenting a mapping to a real-time model permitting use of standard real-time theory. Moreover, it supports synthesis of run-time mechanisms for predictable execution according to the temporal specification in the component model. Furthermore, in this work some limitations in previous work with respect to specification and synthesis of real-time behaviour are removed. These limitations are partially discussed in [5] and is mainly related to jitter and execution behaviour.

Many common component technologies are not used for resource constrained systems, nor safety critical, neither real-time systems. They are simply to resource demanding, to complex and unpredictable. The research community has paid attention to the problem, and recent research has resulted in development of more suitable technologies for these classes of systems. Philips use Koala [7], designed for resource constrained systems, but without support for real-time verification. Pecos [8] is a collaboration project between ABB and University partners with focus on a component technology for field devices. The project considers different aspects related to real-time and resource constrained systems, during composition they are using components without code introspection possibilities that might be a problem for safety critical applications. Rubus OS [6] is shipped with a component technology with support for prediction of real-

time behaviour, though not directly on transactions and jitter constraints and not on sporadic activities. Stewart, Volpe, and Khosla suggest a combination of object oriented design and port automaton theory called Port Based Objects [9]. The port automaton theory gives prediction possibilities for control applications, although not for transactions and jitter constraints discussed in this paper. Schmidt and Reussner propose to use transition functions to model and predict reliability in [10]; they are not addressing real-time behaviour. Wallnau et al. suggest to restrict the usage of component technologies, to enable prediction of desired run-time attributes in [11], the work is general and not focused on particular theories and methods like the work presented in this paper.

The outline of the rest of this paper is as follows; section 2 gives an overview of the component technology. In section 3 the component model is described and its transformation to a real-time model is explained in section 4. Section 5 presents the steps for synthesis of real-time attributes and discusses run-time support. Finally, in section 6, future work is discussed and the paper is concluded.

## 2 Component Technology

In this section we will give an overview of the component technology facilitating component based software development for safety-critical embedded real-time systems. We will hereafter refer to this component technology as the AutoComp technology. A key concept in AutoComp is that it allows engineers to practise Component Based Software Engineering (CBSE) without involving heavy run-time mechanisms; it relies on powerful design and compile-time mechanisms and simple and predictable run-time mechanisms. AutoComp is separated into three different parts; component model, real-time model and run-time system model. The component model is used during design time for describing an application. The model is then transformed into a real-time model providing theories for synthesis of the high level temporal constraints into attributes of the run-time system model. An overview of the technology can be seen in Fig. 1. The different steps in the figure is divided into design time, compile time, and run-time to display at which point in time during development they are addressed or used.

During design time, developers are only concerned with the component model and can practise CBSE fully utilizing its advantages. Moreover, high level temporal constraints in form of end-to-end deadlines and jitter are supported. Meaning that developers are not burdened with the task of setting artificial requirements on task level, which is essential [12], [5]. It is often natural to express timing constraints in the application requirements as end-to-end constraints.

The compile time steps, illustrated in Fig. 1, incorporate a transition from the component based design, to a real-time model enabling existing real-time analysis and mapping to a RTOS. During this step the components are replaced by real-time tasks. Main concerns in this phase are allocation of components to tasks, assignment of task attributes, and real-time analysis. During attribute assignment, run-time attributes that are used by the underlying operating system are assigned to the tasks. The attributes are determined so that the high level

constraints specified by the developer during the design step are met. Finally, when meeting the constraints of the system, a synthesis step is executed. It is within this step the binary representation of the system is created, often the operating system and run-time system are also included with the application code in a single bundle

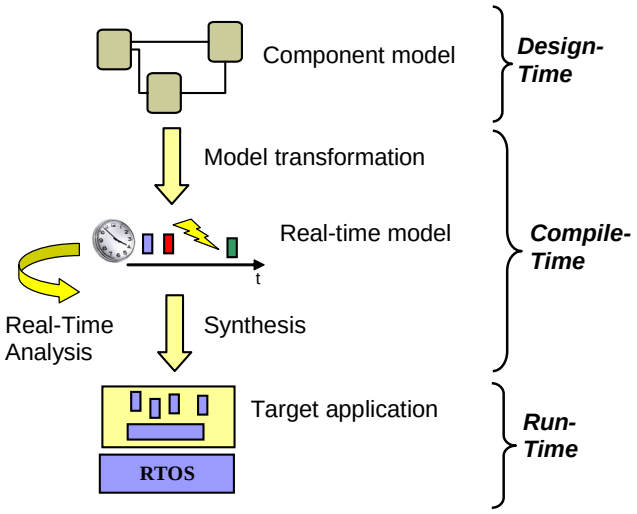


**Fig. 1.** The AutoComp component technology

The run-time system is assumed to be a traditional RTOS with Fixed Priority Scheduling (FPS) of tasks. Most commercial RTOS can be classified into this category; furthermore they are simple, resource efficient and many real-time analysis techniques exist. In some cases a layer providing run-time support for the tasks has to be implemented in order to fully support FPS models used in real-time theory.

## 3 Component Model

Vehicles present a heterogeneous environment where the interaction between the computer system and the vehicle take different forms. Some vehicle functionality requires periodic execution of software, e.g., feedback control, whereas other functionality has a sporadic nature, e.g., alarms. Although vehicle control plays a central role, there is also an abundance of other functionality in vehicles that is less critical and has other characteristics, e.g., requires more flexibility. Although less critical, many of these functions will still interact with other more critical parts of the control system, consider for example diagnostics. We present a model that in a seamless way allows the integration of different functionality,

by supporting early specification of the high level temporal constraints that a given functionality has to meet. Moreover, the computational model is based on a data flow style that results in simple application descriptions and system implementations that are relatively straightforward to analyse and verify. The data flow style is commonly used within the embedded systems domain, e.g., in IEC 61131 used for automation [13] and in Simulink used for control modelling [14].

The definition of the AutoComp component model is divided into components, component interfaces, composition, the components invocation cycle, transactions and system representation. In Fig. 2 the component model is illustrated using UML2, which could be a possible graphical representation during design.

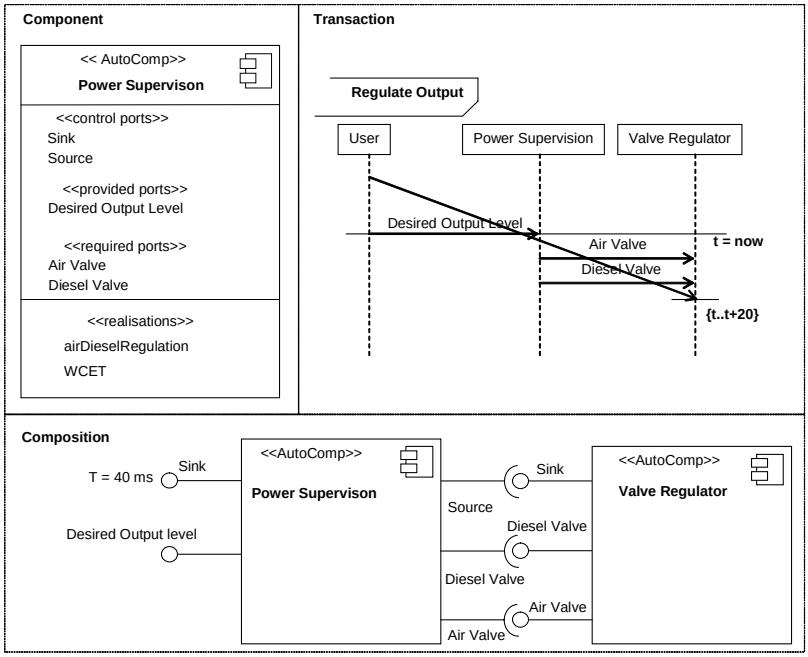


**Fig. 2.** In the upper left part of the figure there is a UML 2 component diagram for modelling of a component. The lower part of the figure is a composition diagram showing a composition of two components. Finally the upper right part of the figure is a sequence diagram with a timing constraint that is used to express the end-to-end deadline for a transaction

The components are defined as *glass box*, meaning that a developer can see the code of a component for introspection purposes. It does not mean that a developer has to look into a component during normal composition, and not

that it is allowed to modify a component. The introspection possibility is a requirement during verification of safety critical applications in order to gain complete knowledge about components behaviour. Furthermore, the components can only exchange data with each others through data ports. A component can be a composite containing a complete subsystem, or a basic component with an entry function. Composite components can be treated as any other component during composition, but it is also possible to enter a composite and change timing requirements and other properties. The entry function provided by non-composite components can be compared to the entry function for a computer program, meaning that the contained number of functions of the component can be arbitrary.

The interfaces offered by a component can be grouped into the two classes data and control interfaces. The data interfaces are used to specify the data flow between components, and consist of data ports. Data ports have a specified type and can be either provided or required. Provided ports are the ports provided by components for input, i.e., the ports a component reads data from. Required ports are the ports a component writes data to. A component also has a control interface with a mandatory control sink, and an optional control source. The control interface is used for specifying the control flow in the application, i.e., when or as a response to what component should be triggered. The control sink is used for triggering the functionality inside the component, while the control source is used for triggering other components.

During composition the developer has three main techniques to work with. The data flow is specified through connection of provided and required data ports. The rules are as follows; required ports must be wired to provided ports with a compatible type. It is possible to make abstractions through definition of composite components. Composite components can be powerful abstractions for visualizing and understanding a complex system, as well as they provide larger units of reuse. The control flow is specified through binding the control sinks to period times for periodic invocation, to external events for event invocation, or to control sources of other components for invocation upon completion of the other components.

A components invocation cycle can be explained as in the following sentences. Upon stimuli on the control sink, in form of an event from a timer, an external source or another component; the component is invoked. The execution begins with reading the provided ports. Then the component executes the contained code. During the execution, the component can use data from the provided ports and write to the required ports as desired, but the writes will only have local effect. In the last phase written data become visible on the required ports, and if the control source in the control interface is present and wired to the control sink of another component stimulus is generated.

Transactions allow developers to define and set end-to-end timing constraints on activities involving several components. A transaction in AutoComp can be defined as:

A transaction $Tr_i$ is defined by a tuple $< C, D, J_s, J_c >$ where:

$C$ - represent an ordered sequence of components;
$D$ - represent the end-to-end deadline of the transaction;
$J_s$ - represent the constraint on start jitter of the transaction;
$J_c$ - represent the constraint on completion jitter of the transaction.

The end-to-end deadline is the latest point in time when the transaction must be completed, relative to its activation. Jitter requirements are optional and can be specified for transactions involving time triggered components. Start jitter is a constraint of the periodicity of the transactions starting point, while completion jitter is a constraint on the periodicity of a transactions completion point. Both types of jitter are expressed as a maximum allowed deviation from the nominal period time. A restriction, necessary for real-time analysis, is that components directly triggered by an external event can only be part of a transaction as the first component.

A system can be described with the UML class diagram in Fig. 3. A system is composed of one or several components, each with a data interface, a control interface and a realization as a subsystem or an entry function. A system also has zero or more data couplings, describing a connected pair of required and provided data ports. Furthermore, systems have zero or more control couplings which describe a connected pair of control sink and source. Finally, the last part of a system is zero or more transactions with the included components, an end-to-end deadline and the possibility to specify jitter requirements.

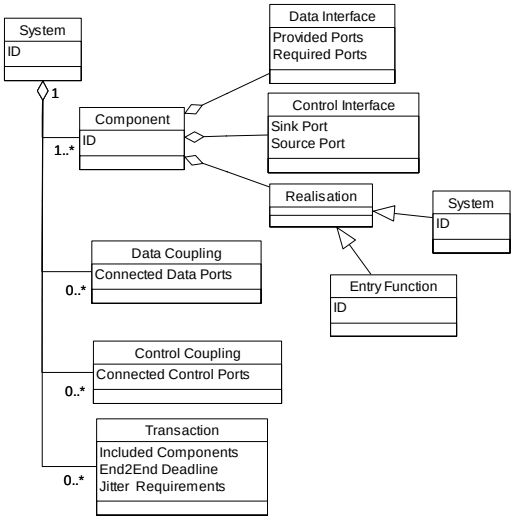


**Fig. 3.** UML class diagram showing the static view of the component model

## 4 Model Transformation

Model transformation involves the steps necessary in order to transit from the component model allowing an efficient and powerful design phase, to a run-time model enabling verification of temporal constraints and usage of efficient and deterministic execution environments. As previous stated in section 2 we assume a FPS run-time model. The FPS model defines a system as a set of tasks with the attributes period time, priority, offset, and WCET. Hence, it is necessary to translate the component model with its temporal constraints in to tasks holding these attributes. The translation is performed in two separate steps; the first step is to make a transformation between components and task (task allocation), the second step is to assign attributes to the tasks (attribute assignment). To assign the FPS model attributes in such a way that the high level temporal constraints on transactions are met is non-trivial and has been addressed in research by e.g., [1], [3].

### 4.1 Task Allocation

The easiest approach for task allocation is a one to one relationship between components and tasks, but that is not necessarily optimal. In fact the task allocation step has a lot of different tradeoffs. Such tradeoffs can be found between reliability and run time overhead; few tasks reduce run time overhead at the cost of memory protection (usually at task level) between components. Testability and schedulability are examples of other properties that are affected by the allocation scheme.

In this paper we introduce a task allocation strategy that strives to reduce the number of tasks considering schedulability and reliability. Components are not allocated to the same task if schedulability is obviously negatively affected and structurally unrelated components are not allocated to the same task in order to cater for memory protection and flexibility.

The first step in the allocation process is to convert all composite components to a flat structure of the contained basic components. Secondly the following rules are applied:

1. All instances of components are allocated to separate tasks, Worst Case Execution Time (WCET) is directly inherited from a component to the corresponding task
2. The start jitter Js corresponding to a transaction with jitter requirements is set as a requirement on the task allocated for the first component in the ordered sequence C, while the completion jitter Jc is set to the task allocated for the last component in the sequence
3. Tasks allocated for components with connected pairs of control sink and sources, where the task with the source do not have any jitter requirements, and both tasks are participating in the same and only that transaction are merged. The resulting WCET is an addition from all integrated tasks WCET

4. Tasks allocated for time triggered components that have the same period time, not have any jitter constraints and are in a sequence in the same and only that transaction are merged. The resulting WCET is an addition from all integrated tasks WCET

The situation after application of the allocation rules is a set of real-time tasks. The high level timing requirements are still expressed in transactions, but instead of containing an ordered set of components a transaction now contain an ordered set of tasks. The rest of the attributes, those that cannot be mapped directly from the component model to the real-time model are taken care of in the following attribute assignment step. In Fig. 4, given the two transactions $Tr_1 =< C, D, J_s, J_c >=< A, B, C, 60, -, 25 >$ and $Tr_2 =< C, D, J_s, J_c >=< D, E, F, 40, 5, - >$ the task allocation step for the components in Table 1 is shown. The resulting task set is in Table 2.

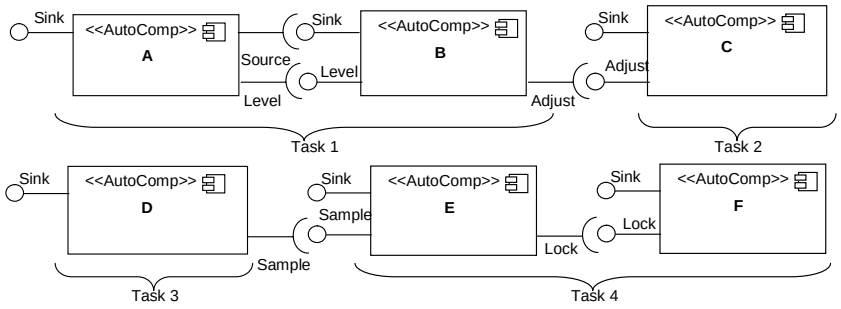


**Fig. 4.** Task allocation example

| | Sink Bound To | WCET |
|---|---|---|
| A | T = 100 | 5 |
| B | A.Source | 10 |
| C | T = 60 | 5 |
| D | T = 40 | 5 |
| E | T = 40 | 6 |
| F | T = 40 | 9 |

**Table 1.** A component set

| | Trigger | Jitter | WCET |
|---|---|---|---|
| Task 1 | T = 100 | | 15 |
| Task 2 | T = 60 | 25 | 5 |
| Task 3 | T = 40 | 5 | 5 |
| Task 4 | T = 40 | | 15 |

**Table 2.** The resulting task set

### 4.2 Attribute Assignment

After the components have been assigned to tasks, the tasks must be assigned attributes so that the high level temporal requirements on transactions are met. Attributes that are assigned during task allocation are WCET for all tasks, a period time for periodic tasks and a Minimum Interarrival Time (MINT) for event triggered tasks.

The scheduling model that is used throughout this paper is FPS, where tasks have their priorities and offsets assigned using an arbitrary task attribute assignment methodology. Examples of existing methods that can be used for priority assignment are Bate and Burns [1], Sandström and Norström [3] or by combination of Yerraballi [15] or Cheng and Agrawala [16] with Dobrin, Fohler and Puschner [17]. In this paper it is assumed that task attributes are assigned using the algorithm proposed by Bate and Burns [1], and it is showed that the component model described in this paper is applicable to their analysis model. Weather the tasks are time triggered or event triggered is not considered in the Bate and Burns analysis but is required during the mapping to the FPS model, where periodic and event triggered (sporadic) tasks are separated. The attributes that are relevant, considering this work, in the Bate and Burns approach are listed below.

*For tasks:*

**T (Period)** - All periodic tasks have a period time that is assigned during the task allocation. Sporadic tasks have a MINT that analytically can be seen as a period time;

**J (Jitter)** - The jitter constraints for a task is the allowed variation of task completion from precise periodicity. This type of jitter constraint is known as completion jitter. Jitter constraints can be set on the first and last task in a transaction;

**R (Worst Case Response time)** - The initial Worst Case Response time for a task is the WCET for the task, i.e., the longest time for a task to finish execution from its starting point in time.

*For transactions:*

**T (Period)** - The period of a transaction is the least common multiple of the period times of the participating tasks of the transaction;

**End-to-End deadline** - Transactions have a requirement that all tasks have finished their execution within a certain time from the transactions point of start in time.

In Bate and Burns approach additional attributes, such as deadline and separation for tasks and jitter requirements for transactions are considered. In this paper those attributes are disregarded since there are no such requirements in the previously described component model. It is trivial to see that from the component model, the period and jitter constraints match the model proposed by Bate and Burns. The initial worst case response time R is assigned the WCET value in the component model. For the transaction the end-to-end deadline requirements match the transaction deadline of the Bate and Burns model. The period time of the transaction is derived from the least common multiple of the period of the tasks participating in the transaction.

The next step is naturally to assign the FPS model with run-time and analysis attributes. The new attributes priority and offsets will be derived through existing analysis methods [1]. The new parameters for the FPS model are described below.

**P (Priority)** - The priority is an attribute that indicates the importance of the task relative to other tasks in the system. In a FPS system tasks are scheduled according to their priority, the task with the highest priority is always executed first. All tasks in the system are assigned a priority;

**O (Offset)** - The offset is an attribute that periodic tasks with jitter constraints are assigned. The earliest start time is derived by adding the offset to the period time.

In Table 3 it is summarized what attributes belonging to time triggered and event triggered tasks in the FPS model.

| Attribute | Time triggered | Event triggered |
|---|---|---|
| Period | X | |
| MINT | | X |
| Priority | X | X |
| Offset | X (Upon Jitter Constraints) | |
| WCET | X | X |

**Table 3.** Attributes associated with time and event triggered tasks

Applying the Bate and Burns algorithm determines task attributes from the tasks and transactions described in Table 2. The resulting run-time attributes priority, offset period and WCET are shown in Table 4. The attributes offset and priority are determined with the Bate and Burns analysis, whilst the period and WCET are determined in the task allocation.

| | Priority | Offset | Period | WCET |
|---|---|---|---|---|
| Task 1 | 2 | 0 | 100 | 15 |
| Task 2 | 1 (Lowest) | 35 | 60 | 5 |
| Task 3 | 4 (Highest) | 0 | 40 | 5 |
| Task 4 | 3 | 0 | 40 | 15 |

**Table 4.** Assigned task attributes

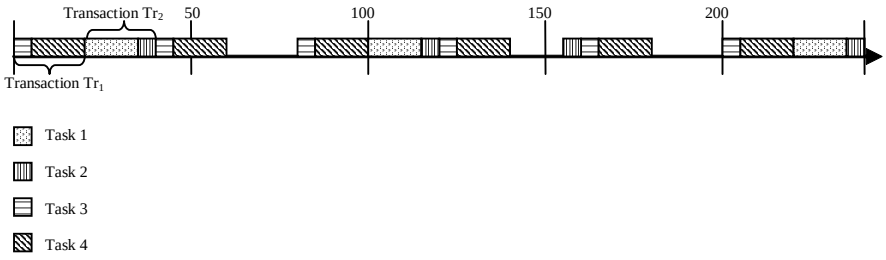


**Fig. 5.** Trace of an FPS schedule

In Fig. 5 a run-time trace for an FPS system is shown and the transactions $Tr_1$ and $Tr_2$ are indicated.

When the FPS model has been assigned its attributes it has to be verified. The verification of the model is performed by applying real-time scheduling analysis to confirm that the model is schedulable with the assigned parameters. This is necessary since attribute assignment does not necessarily guarantee schedulability, but only assigns attributes considering the relation between the tasks.

### 4.3 Real-Time Analysis

To show that the FPS tasks will meet their stipulated timing constraints, schedulability analysis must be performed. Much research has been done with respect to analysis of different properties of FPS systems, and all those results are available for use, once a FPS model has been established. The temporal analysis of an FPS system with offsets, sporadic tasks and synchronization has been covered in research by e.g., Palencia et al. [18], [19] and Redell [20].

The output from the analysis is whether the system is feasible or not in the worst case. If the analysis shows that the system is infeasible, the parts that can not keep its requirements are either changed and reanalysed or emphasised for the developer to make changes.

## 5 Synthesis

The next step after the model transformation and real-time analysis is to synthesise code for the run-time system. This includes mapping the tasks to operating system specific task entities, mapping data connections to an OS specific communication, modifying the middleware, generating glue code, compiling, linking and bundling the program code (see Fig. 6).

The synthesis is divided into two major parts. Given a task set and necessary information about the run-time system, the synthesis generates code considering communication, synchronization.

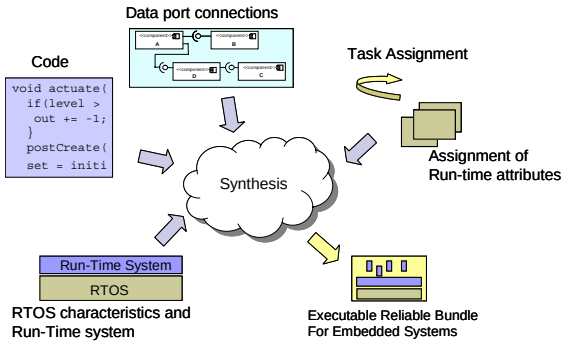


**Fig. 6.** The steps of synthesizing code for the run-time system

- The first part in synthesis is to resolve the communication within and between tasks. Two communicating components that are assigned to different tasks will form an Inter Task Communication (ITC) while communication between components assigned to the same task are realized with shared data spaces within the task. The ITC is later mapped to operating system specific communication directives.
- The other part in the synthesis is to resolve the control couplings, i.e., the sink and source. If a tasks starting point is dependent on the former tasks finishing point the tasks have to be synchronized. The synchronization is solved through scheduling. The synthesis will generate code for scheduling periodic tasks, handle the control flow between tasks and consider offsets. The code generated for the periodic scheduling and offsets is dependent on the middleware and can be realized as a configuration file or actual code in each task. Invocations of sporadic tasks are mapped to event handlers in the middleware or the operating system.

It is assumed that a middleware is present as shown in Fig. 7, for each platform and that it provides functionality that the component model needs but the

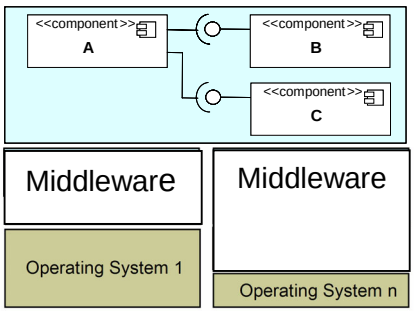


**Fig. 7.** A component model with adjustments for different operating systems to promote platform independence

operating system does not provide. The more functionality the operating system provides, the smaller the middleware has to be. The middleware encapsulates core communication and concurrency services to eliminate many non-portable aspects of developing and is hence platform specific in favour of a platform independent component model. Typical functionality that is not provided by most commercial RTOS is periodicity and support for offsets. The middleware also need to support sink and source couplings since task coupled with its source need to be able to invoke the corresponding task. The run-time system conforms to FPS and hence the run-time task model is similar to the previously described FPS model with some exceptions. The worst case execution time is merely an analysis attribute and is not needed in the run-time model. The MINT is usually a requirement on the environment rather than a task attribute, and is thus also analytical and unnecessary. Hence the run-time task model is for periodic tasks Period time, Priority, Offset and for sporadic tasks Priority.

## 6 Conclusions and Future Work

In this paper we show how to use component based software engineering for low footprint systems with very high demands on safe and reliable behaviour. The key concept is to provide expressive design time models and yet resource effective run-time models by statically resolve resource usage and timing by powerful compile time techniques.

The work presented in this paper introduces a component technology for resource effective and temporally verified mapping of a component model to a resource structure such as a commercial RTOS. This is made possible by introduction of a component model that support specification of high level real-time constraints, by presenting a mapping to a real-time model, permitting use of standard real-time theory, and by synthesis of run-time mechanisms for predictable execution according to the temporal specification in the component model.

Although the basic concept has been validated by successful industrial application of previous work [5], it is necessary to further validate the component technology presented here. In order to facilitate this, a prototype implementation of the component technology is under development where the core part has been completed. The prototype will enable evaluation of different technology realisations with respect to performance. Moreover, parts of the model transformation need additional attention, foremost the strategies for allocation of components to tasks. Furthermore, we will make efforts in extending the component model making it more expressive and flexible while still keeping the ability for real-time analysis. Interesting is also to investigate trade-offs between run-time foot print and flexibility with respect to e.g., adding functionality post production. Finally, the component technology will be evaluated in a larger, preferably industrial, case.

## References


1. Bate, A., Burns, I.: An approach to task attribute assignment for uniprocessor systems. In: Proceedings of the 26th Annual International Computer Software and Applications Conference, IEEE (2002)
2. Mok, K., Tsou, D., Rooij, R.C.M.D.: The msp.rtl real-time scheduler synthesis tool. In: In Proc. 17th IEEE Real-Time Systems Symposium, IEEE (1996) 118–128
3. Sandström, K., Norström, C.: Managing complex temporal requirements in real-time control systems. In: In 9th IEEE Conference on Engineering of Computer-Based Systems Sweden, IEEE (2002)
4. Würtz, J., Schild, K.: Scheduling of time-triggered real-time systems. In: In Constraints, Kluwer Academic Publishers. (2000) 335–357
5. Norström, C., Gustafsson, M., Sandström, K., Mäki-Turja, J., Bånkestad, N.: Experiences from introducing state-of-the-art real-time techniques in the automotive industry. In: In Eigth IEEE International Conference and Workshop on the Engineering of Compute-Based Systems Washington, US, IEEE (2001)
6. Arcticus: (Arcticus homepage: http://www.arcticus.se)
7. van Ommering, R., van der Linden, F., Kramer, J.: The koala component model for con-sumer electronics software. In: IEEE Computer, IEEE (2000) 78–85
8. Nierstrasz, G., Arevalo, S., Ducasse, R., Wuyts, A., Black, P., Müller, C., Zeidler, T., Genssler, R., van den Born, A.: Component model for field devices. In: Proceedings of the First International IFIP/ACM Working Conference on Component Deployment, Germany. (2002)
9. Stewart, D.B., Volpe, R.A., Khosla, P.K.: Design of dynamically reconfigurable real-time software using port-based objects. In: IEEE Transactions on Software Engineering, IEEE (1997) 759–776
10. Schmidt, W.H., Reussner, R.H.: Parameterised contracts and adaptor synthesis. In: Proc. 5th International Workshop of Component-Based Software Engineering (CBSE5). (2002)
11. Hissam, S.A., Moreno, G.A., Stafford, J., Wallnau, K.C.: Packaging predictable assem-bly with prediction-enabled component technology. Technical report (2001)
12. Hammer, D.K.., Chaudron, M.R.V.: Component-based software engineering for re-source-constraint systems: What are the needs? In: 6th Workshop on Object-Oriented Real-Time Dependable Systems, Rome, Italy. (2001)

13. IEC: International standard IEC 1131: Programmable controllers (1992)
14. Mathworks: (Mathworks homepage : http://www.mathworks.com)
15. Yerraballi, R.: Scalability in Real-Time Systems. PhD thesis, Computer Science Department, old Dominion University (1996)
16. Cheng., S.T., K., A.A.: Allocation and scheduling of real-time periodic tasks with relative timing constraints. In: Second International Workshop on Real-Time Computing Systems and Applications (RTCSA), IEEE (1995)
17. Dobrin, R., Fohler, G., Puschner, P.: Translating off-line schedules into task attributes for fixed priority scheduling. In: In Real-Time Systems Symposium London, UK, December. (2001)
18. Palencia, J.C., Gonzalez Harbour, M.: Schedulability analysis for tasks with static and dynamic offsets. In: Proc. of the 19th Real-Time Systems Symposium. (1998)
19. Palencia, J.C., Gonzalez Harbour, M.: Exploiting precedence relations in the schedulabil-ity analysis of distributed real-time systems. In: Proc. of the 20th Real-Time Systems Symposium. (1999)
20. Redell, O., Törngren, M.: Calculating exact worst case response times for static priority scheduled tasks with offsets and jitter. In: Proc. Eighth IEEE Real-Time and Embedded Tech-nology and Applications Symposium, IEEE (2002)