MoDe II: Modeling and Analyzing Time-Constraints

Jewgenij Botaschanjan and Jan Jürjens
Systems & Software Engineering
TU München
85748 Garching, Germany
contact: botascha@in.tum.de

Abstract

The fulfillment of time requirements is one of the major acceptance criteria of safety-critical and real-time systems. They are dictated by the environment of these systems and are often well known at the early phases of the development process. At the same time, the time behavior is a cross-cutting concern, not bordered by the behavioral design units, e.g. components or classes.

This paper argues for consideration of the time behavior as an explicit model-based view on the system under development. It proposes a modeling language and an analysis method. The formal graphical modeling language allows the explicit notation of time requirements and the integration with the architectural and behavioral aspects of the system. The analysis method allows the developer to verify requirements in respect of their consistency and completeness as well as to validate design concerning the fulfillment of time requirements. It supports modular development, for which results are presented.

1 Introduction

A significant number of embedded (and in particular safety-critical) systems has not only to realize the demanded functionality correctly but also to fulfill a number of real-time constraints. The constraints are dictated by properties of the environment, which has to be monitored and/or controlled by this class of systems. These properties are firstly based on physical laws and secondly on protocols and commitments between different control units. The systems have to note changes within the environment in time and react/act within fixed delays.

The time behavior of the environment, as a part of the problem space, is studied, negotiated and captured within the specification. These constraints, available at the early stages of the development process, are important criteria for the acceptance of the system and should be analyzed for correctness as well as used to guide the system's design.

On the other hand, the analysis of the actual run time behavior of embedded systems needs primarily exact Best-Case Execution Time (BCET) and Worst-Case Execution Time (WCET) estimations. These can be achieved only at the assembler code level during the deployment phase. A discovered inconsistence within the demanded time behavior or miss of a demanded time constraint during the final stages of development process lead to an expensive and time-intensive redesign.

This motivates the modeling of the *demanded* time behavior as an explicit view on the system. The justification of design for the fulfillment of time constraints aims to (a) reduce the possibility of mismatch with the requirements (b) localize the cost of final verification/validation (c) localize/guide the possible refactoring measures. In order to achieve the listed goals, the model should be formally founded and support the notions of *composition* and *refinement*. These are preliminaries for the integration of the timing model with other aspects of the system, like the behavioral and architectural views.

This paper presents MoDe II (*Mo*del Based *De*ployment) — a flexible modeling approach for real-time requirements and describes a CLP-based (Constraint Logic Programming [11]) technique for their analysis. Furthermore, we discuss the integration of the timing model that is presented with other views on the system, such as behavioral and architectural views.

 DOI 10.1109/ECBS.2005.52

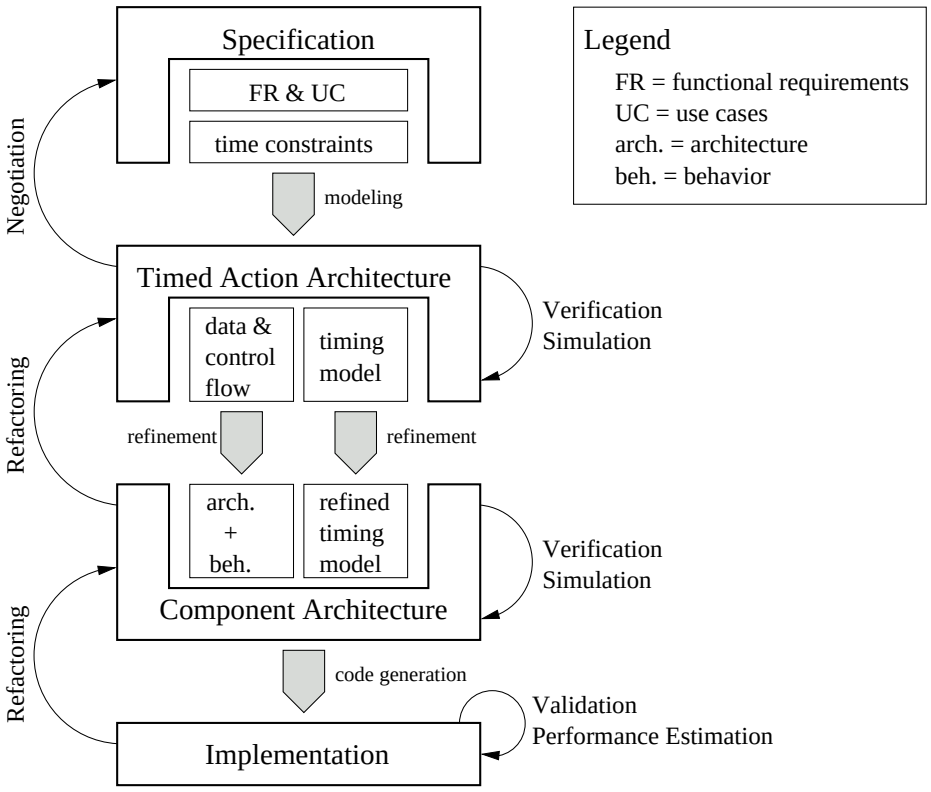

Figure 1. Modeling & Design Flow

2 Overview

2.1 Design Flow

Fig. 1 shows the design flow of our approach. Here the specification serves as the main input artifact. It consists of (a) functional requirements, (b) use cases and scenarios, which define a specific causal order on them, and (c) time constraints, which additionally demand specific time behavior from scenarios and functionalities.

The data and control flow between different functionalities (or *actions*) of the system are extracted from the specification and captured explicitly by a graph representation within the *timed action architecture (TAA)*. In addition, time constraints are assigned to the obtained flow relation. TAA serves as a base for the architectural and behavioral design of the system. Along this process, the timing model is being refined and adjusted, also. This is mainly done by associating and distributing the time constraints to the artifacts of the component architecture. The implementation is accomplished by code generation.

Verification and simulation can be applied at the both modeling levels, as discussed in this paper. However, different problems are relevant on different levels of abstraction. At the stage of TAA the interesting questions for verification are the consistency and completeness of the time requirements. The simulation at the TAA level can give information about performance bottle necks. The component model has to be verified for the correctness of the refinement relation to the TAA.

During the implementation phase, the actual real-time behavior of the system can be measured or obtained using analytical approaches (such as abstract interpretation). The timing models can serve as a basis for test case generation for this purposes. If the actual behavior does not comply with the one demanded, the search for alternative design solutions has to be initiated, using refactoring. In case no satisfying solution could be found, the negotiation of time requirements is the logical consequence. Along this process, timing models are used to identify the affected parts of the system and to thus reduce the search space of the solution.

2.2 Reference Design Paradigm

The described development scenario applies to the design notation, with a simple and extensively supported formal semantics, AutoFOCUS [14]. In AutoFOCUS, systems are specified as hierarchical component networks, where components communicate via typed and directed channels (architectural view). A component may be refined by a network of sub-components. Leaf components in these hierarchies are defined by finite state machines extended by local variables and message send/receive statements.

On top of the formal semantics, the Quest framework [22] integrates AutoFOCUS models with verification tools (such as model checkers, SAT-solvers and theorem provers), as well as with test case and code generation tools. Furthermore, the simulation environment for AutoFOCUS models is also provided by the Quest framework.

Design principles, formal semantics and tool support make AutoFOCUS/Quest an adequate reference paradigm/integration environment for the presented approach.

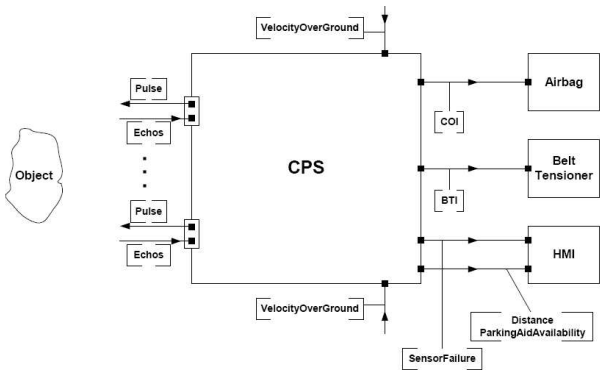

Figure 2. Context of the CPS System from [16]

2.3 Case Study

In the rest of this paper, we will illustrate the introduced concepts by a case study based on an example car periphery supervision system (CPS) whose specification was adapted from [16]. The example exhibits some typical properties of the targeted systems (in this case from the automotive domain), such as *real-time requirements*, *distribution* and *resource constraints*.

CPS is supposed to be the basis for many driver assistance services, such as parking assistance, pre-crash detection, blind spot supervision, lane change assistant etc. The CPS system is sending radar pulses out into the environment of the car and receives and processes the echoes which are reflected from objects in the environment (Fig. 2). Depending on number, position and relative velocity of the objects, CPS communicates with the airbag system, the belt tensioner, and a human machine interface (HMI). A number of real-time requirements are stated for the CPS system.

1. CPS must send the collision data to the airbag ECU, through the collision object interface (COI). The data must be delivered *at least 10 ms* before a predicted collision.
2. A further task of CPS is to activate the belt tensioner via the BTI interface in case of a predicted collision. The activation must take place *at least 110 ms* before the predicted time of collision.
3. In the normal case, *MeasurementControl* is operating the sensors in DistanceScan mode by sending out DScan commands *every 15 ms* to all sensors.
4. The task of the component SituationAnalysis (SA) is to decide whether the sensors shall be switched from DistanceScan mode to CVMeasurement mode. This is done when an object on collision course needs *less than 30 ms* before it will reach a specific safety area.

In the following sections, we will demonstrate how the approach allows flexible modeling of the real-time requirements of the CPS system and what results were obtained from their analysis.

3 Requirements Model

Specifications define use cases of the system and constrain their time behavior. In general, two types of time constraints are considered by the presented approach. (1) After an external event, the reaction of the system must come within a certain time. This type of time requirements demands specific response times from particular use cases/functionalities. (2) The events in the system's environment are often periodic and/or occur at particular points of time. The second type of requirements demands periodical reaction (or action) of the system on such events.

Both types of constraints refer to a specific functionality which must be provided by the system. At the highest level of abstraction (TAA), we model the functionality by so called *actions*.

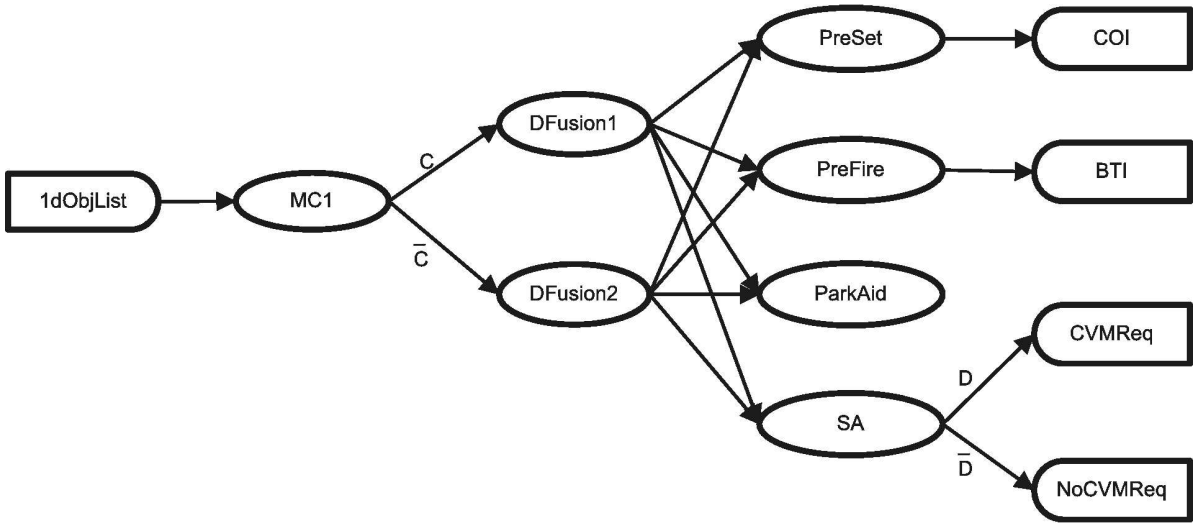

Figure 3. CPS Study: System Run *sr1*.

3.1 Actions

There are *atomic* and *composite* actions. Atomic actions represent simple functionalities, which are described in the specification. These functionalities are referenced by use cases, which define data and control flow dependencies between them. Within our approach, these dependencies are captured by composite actions.

In our model of time behavior, atomic actions are the only units which consume time. No further information about the internal structure of atomic actions is known at the early modeling stages. However, by the integration with the behavioral view (see Sec. 3.2), the implementation details of atomic actions can be considered by the analysis of the system (Sec. 4).

Composite actions define the execution order of the atomic actions they consist of. The execution of an action can depend on a set of inputs, produced by execution of its predecessors. In addition, the execution can be triggered by the *execution history* of the system. The execution history is a sequence of actions executed so far. An occurrence of specific actions (in a specific order) in this sequence can determine the actions to be executed next. In order to express these circumstances, a *pre-* and a *postcondition* are assigned to every atomic action. An action can be executed if its precondition is true. The postcondition is evaluated at the end of the action's execution. It determines the further flow of the data and control.

Tab. 1 on page 10 lists a number of conditions from the CPS study. The first four lines define a non-deterministic choice between the execution of *DFusion1* and *DFusion2* after *MC1*, illustrated by Fig. 3. As a consequence of this choice, the action *SA* can be executed after either *DFusion1* or *DFusion2*. The rest of the table concerns the behavior of the environment (sensor), demanded by the following requirement: "*After receiving a CVMeasurement command, CVModeProcessing will start to deliver a time series of maximally 10 radial velocity values. The last value in this series is 0, no matter how long the sequence will be.*" This requirements is modeled by actions on Fig. 5.

The flow is captured by the presented approach on the basis of the following formal definition.

Definition 1 (data & control flow) *The data & control flow is defined as a partial order on atomic actions: $fl \subseteq A^{atom} \times A^{atom}$. Then $a \leq b$ is defined to be an abbreviation for $(a, b) \in fl$.*

Every atomic action $a \in A^{atom}$ has a pre- and a postcondition, defined as predicates over execution histories of the system, which are expressed in terms of atomic actions being already executed.

$$pr_a, ps_a \subseteq A^{atom} \times (A^{atom})^*$$

A precondition of an action must be stronger than the postconditions of all its predecessors: $pr_a \subseteq \bigcap_{a' \leq a} ps_{a'}$. Based in this definition, an action can be executed if and only if the associated precondition evaluates to true. Put another way, a postcondition of an action is satisfied if and only if the corresponding action has already been executed.

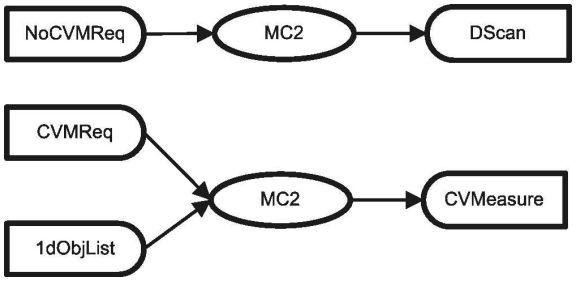

Figure 4. CPS Study: System Runs *sr2* **&** *sr3***.**

3.2 System Runs

Use cases and scenarios define the usage of the system. They describe (1) prerequisites of a particular usage (such as signals from the environment), (2) the affected functionalities and the order of their execution, (3) the observable results (e.g. outgoing signals), produced by the system, as well as (4) the reaction of the environment on them. Within the presented approach the different action types of the system/environment are captured by *system runs*.

System runs are *typed* connected composite actions, extracted from use cases and scenarios of the specification. They model the reactions and interactions of the system with its environment. The atomic actions of a system run are marked by one of the following four types. (1) *Input* and (2) *output* actions model *observable* events, which happen in the system's environment. An example is the arrival or transmission of a signal. These action types do not consume time during their execution. (3) *Internal* actions represent the functionality of the system. Their time consumption must be greater than 0. It is reasonable to model parts of the environment which are not directly observable by the system, since they also consume time. This is done by (4) *external* actions. The types listed above partition the set of atomic actions of a system run A_{sr}^{atom} into four disjoint subsets: IN_{sr}, OUT_{sr}, INT_{sr} and EXT_{sr} respectively.

Consider for example the system run *sr1* from Fig. 3. It has one input action *1dObjList*, which represents the interface from the sensor. The output actions *COI* and *BTI* describe interfaces to the airbag and the belt tensioner, respectively. The output actions *CVMREq* and *NoCVMReq* describe the feedback loop to the CPS system itself – depending on the situation in the outside world, it switches the internal action *SA* of the CPS into one of two different operational modes. The same actions appear as inputs in system runs *sr2* and *sr3*, Fig. 4. The external actions can be found in system runs *sr8*, *9.1* and *9.2*, represented as rectangles in Fig. 5. They model the time consumption of the sensor in its different operational modes.

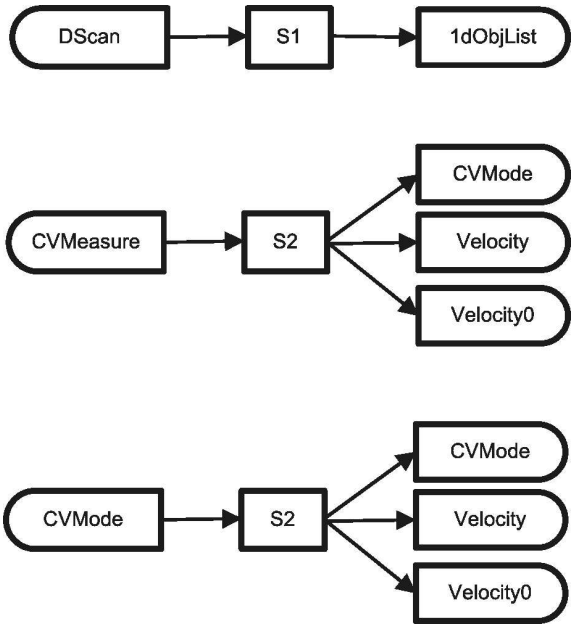

Figure 5. CPS Study: System Runs *sr8***,** *sr9.1* **&** *sr9.2***.**

3.3 System Model

The system is modeled by a set of system runs. Since system runs are defined as composite actions, the global data and control flow relation can be easily derived from the set of local ones. The global flow is the result of asynchronous composition of all system's runs. The common actions (that is, actions with the same name) within different system runs serve thereby as *synchronization points* within the flow. They are seen as the *same* instance of an action, so they are not allowed to happen in parallel. This restriction on the common input and output actions expresses the activation order of the system runs.

Definition 2 (internal & external interface) *A pair of system runs* $sr_1, sr_2 \in SR$ *have a common interface:* $I_{sr_1,sr_2} = (IN_{sr_1} \cap OUT_{sr_2}) \cup (IN_{sr_2} \cap OUT_{sr_1})$.

The remaining input and output actions build the interface to the environment: $I^{env} = \{a | a \in \bigcup_{sr} IN_{sr} \otimes a \in \bigcup_{sr} OUT_{sr}\}$, *where* $\otimes$ *stands for XOR-operator.*

Thus composition of system runs can be formally defined as the union of their flow relation functions.

Definition 3 (composition of system runs) *The parallel composition of system runs $SR = \{sr_1, \ldots, sr_n\}$ is specified in terms of an alphabet of atomic actions $A^{atom} = \bigcup_{sr \in SR} A^{atom}_{sr}$, as a global data and control flow relation $fl^g = \bigcup_{sr \in SR} fl_{sr}$.*

The global pre- and postconditions of every action are also defined as unions of all corresponding local system run conditions: $pr^g_a = \bigcup_{a \in A^{atom}_{sr}} pr_{sr.a}$, $ps^g_a = \bigcup_{a \in A^{atom}_{sr}} ps_{sr.a}$.

During the modeling and design phases of the system development, the behavioral model of the system is refined. The functionality is implemented by automata and associated to components (see Sec. 2.2 for the reference design paradigm this paper relies on). Consequently, actions have to be refined to execution traces of one or several automata. The input and output actions match with the presence of certain signals on certain input/output channels, since they model the interface with the environment. All automatas traces between the consumption of the input signals and emission of the output ones define the implementation of the system run. The data and control flow between internal actions can be mapped to the internal communication between components.

In terms of the presented approach, the behavior of a system is defined by the asynchronous composition of its system runs. According to this definition, the system is a composite action from Sec.3.1.

Definition 4 (system model) *The system s is defined as a tuple (A^{atom}, SR, fl^g) consisting of a set of system runs SR, as well as an alphabet and a global data and control flow relation, obtained by their composition.*

3.4 Time Constraints

System runs defined in the previous sections are constrained by certain *response times*. On the other hand some of them may have certain *activation times*, e.g. they have to be started periodically.

Response Times. A response time references a non-empty set of input actions and a non-empty set of output actions of a system run. It is defined as a span of time between the execution begin of all input actions and the execution end of all output actions.

For example the first requirement listed in Sec. 2.3 demands the response time between actions *1dObjList* and *COI* of *sr1* (Fig. 3) to lie within 0 and 100μs.

Activation Periods. The execution times of system runs depend on *activation periods* of their input and/or output actions. An activation period of an action is the lower bound of its execution begin. There are *sporadic* and *periodic* activations. The periodic ones occur within an interval $[a, b]$ with $a > 0$ and $b < \infty$, e.g. the time span between two subsequent activations is at least a and at most b. The sporadic activations define only the lower bound of the period: $[a, \infty)$, with $a > 0$.

For the systems which must provide a periodical output behavior, the activation periods can be associated with the output actions as well. For example consider the time requirement 3 listed in Sec. 2.3. During the modeling, the appropriate action (*DScan)* will get assigned a period of $[150\mu s, 150\mu s]$.

Time Constraints. After the treatment of system runs, response times and activation periods, we can introduce the notion of time constraint within the timed action architecture and the TAA itself formally.

Definition 5 (time constraint) *A time constraint (or time requirement) $rtreq$ consists of a system run sr and the corresponding Best-Case Response Time (BCRT) and Worst-Case Response Time (WCRT) values, denoted by the $resp$ function.*

$$rtreq = (sr, \{resp(sr, I, O) | I \subseteq IN_{sr} \wedge O \subseteq OUT_{sr}\})$$

Proposition 1 (composition of time constraints) *Given systems runs $SR = \{sr_1, \ldots, sr_n\}$ with associated time constraints, each time constraint on any of the runs induces a time constraint on the parallel composition of the system runs.*

Definition 6 (timed action architecture) *The timed action architecture of a system s is a tuple*

$$taa_s = (A^{atom}, SR, RESP, INP)$$

consisting of a set of system runs SR, built out of atomic actions A^{atom}, the demanded response times $RESP$ and activation periods for input and output actions INP.

3.5 Deployment

The term deployment in the context of system runs represents the a *partial sequentialization* function on independent actions. A pair of actions is independent if and only if there exists no flow dependency between them in the transitive closure of the global flow relation. The deployment function reduces the number

of possible runs of a system, by establishing flow relationships between independent actions. These relationships are generally motivated by the following four types of activities. (1) The *distribution* assigns actions onto physical components. (2) *Scheduling* specifies the execution order of (independent) actions on a specific component. The assignment of actions to (3) *tasks* allows the modeler to bundle sets of actions and compute schedules upon them. The independent actions within one task are executed in arbitrary sequential order without delays in-between. (4) Communication protocols restrict the *communication* order and bring further sequentializations: The bus communication allows at most one message transmission at the same time between connected physical components. Sequentialization is formally defined as:

Definition 7 (sequentialization) *A system s' defined by a tuple (A^{atom}, SR', fl') is a* sequentialization *of a system $s = (A^{atom}, SR, fl)$, iff $fl \subseteq fl'$. s' is a* valid sequentialization *of s, iff $fl \subseteq fl' \wedge SR \models taa_s \Rightarrow SR' \models taa_s$, e.g. if s' satisfies the specification of s. A* maximal valid sequentialization *of s is a valid sequentialization s', so that no further valid sequentialization s'' exists, with $fl_{s''} \subsetneq fl_{s'}$.*

The maximal valid sequentialization demonstrates the deployment strategy, which fulfils the timing constraints and involves a minimal number of physical resources (e.g. ECUs):

Fact 1 *If a system s satisfies a time constraint $rtreq$ and the system s' is a valid sequentialization of s, then s' satisfies the time constraint $rtreq$.*

Distribution. The set of atomic actions is distributed onto a set of physical components. Within the presented mathematical model this is accomplished by a total partitioning function $dep : A^{atom} \rightarrow \mathbb{N}^+$, which assigns a specific number, representing a computation unit (e.g. an ECU) to every atomic action.

Scheduling. Members of every subset of the partition, given by the dep-function, are executed sequential. The order of their execution is defined by (a) the original flow relation and (b) the scheduling policy.

A *cyclic non-preemptive* scheduling policy, considered here, defines following constraint for all actions, deployed to the same computation unit:

$$\forall a_1, a_2 : dep(a_1) = dep(a_2) \Leftrightarrow a_1 \leq a_2 \vee a_2 \leq a_1$$

Communication. Actions deployed on different units, which stand in the $\leq$-relation to each other, have to communicate through a bus line. This consumes time and brings additional sequentialization of the system's model. The reference architecture, this papers relies on, assumes that the all computation nodes are bound by one bus link with broadcast communication (e.g. like CAN). The bus communication is modeled by a special set of actions A^{bus}. The sequentialization constraint for the set of bus actions says:

$$\forall a_1, a_2 \in A^{bus} : a_1 \leq a_2 \vee a_2 \leq a_1$$

The bus actions augment the flow relation, spread between different computation units, using the $bmap$ function, which maps a bus action to a flow transition:

$$\begin{aligned} \forall a_1, a_2 : \quad & (dep(a_1) \neq dep(a_2) \wedge a_1 \leq a_2 \\ \Leftrightarrow \quad & \exists a_b \in A^{bus} : bmap(a_1, a_2) = a_b) \end{aligned}$$

The global flow relation enriched by the bus communication is obtained by

$$\begin{aligned} fl^{bus} = fl^g \setminus \{(a_1, a_2) | dep(a_1) \neq dep(a_2) \wedge a_1 \leq a_2\} \\ \cup \{\{(a_1, a_b), (a_b, a_2)\} | bmap(a_1, a_2) = a_b\} \end{aligned}$$

The deployment at the level of components and their implementation by automata within AutoFOCUS models (see Sec. 2.2) was considered by the predecessor project MoDe [7].

4 Analysis

The analysis of the modeled time behavior consists of (a) translation of the data and control flow relationships into a set of constraints, (b) formulation of the problem as a goal constraint, and (c) proof of the satisfiability of the obtained constraint set. Next subsection describes the schematic translation of the system runs and time requirements into a constraint logic satisfaction problem. Sec. 4.2 lists problems which can be analyzed by the presented approach.

4.1 Translation into CLP($\mathcal{FD}$)

A constraint logic programming (CLP) problem [11] defines relationships (constraints) between a set of variables in a specific domain (for instance in the finite (integer) domain ($\mathcal{FD}$)). The solution of a CLP problem is a valuation of these variables, that satisfies all constraints. The following paragraphs describe the representation of the concepts introduced in the previous sections as CLP constraints.

$$
\begin{array}{l}
preC(DFusion1, [(MC1, s, d, r, [DFusion1])], Hist) \Leftarrow \\
\quad fre(DFusion1, (MC1, s, d, r, [DFusion1]), Hist). \\
preC(DFusion2, [(MC1, s, d, r, [DFusion2])], Hist) \Leftarrow \\
\quad fre(DFusion2, (MC1, s, d, r, [DFusion2]), Hist). \\
preC(SA, [(DFusion1, s, d, r, [SA])], Hist) \Leftarrow SA \in N_a. \\
\quad fre(SA, (DFusion1, s, d, r, [SA]), Hist),\ !. \\
preC(SA, [(DFusion2, s, d, r, [SA])], Hist) \Leftarrow SA \in N_a. \\
\quad fre(SA, (DFusion2, s, d, r, [SA]), Hist). \\
\\
postC(MC1, [(DFusion1, \_, \_, \_, \_)], Hist). \\
postC(MC1, [(DFusion2, \_, \_, \_, \_)], Hist). \\
postC(DFusion1, [(SA, \_, \_, \_, \_), \ldots], Hist).
\end{array}
$$

Figure 6. Generated Pre- & Postconditions from the CPS Study

Atomic Actions. The interesting informations about an atomic action are its start time, its *demanded duration*, the physical component (resource), it runs on and the set of its successors, which evaluate the associated postcondition to true. These informations preceded by the corresponding action name are kept in a tuple (a, s_a, d_a, r_a, N_a). We write ι_a as an abbreviation of this tuple and ι_A for a set of them. The demanded duration of an action is the time, it can spend for its execution. It can be boarded by experience values, estimates, and/or experiments.

The execution of an atomic action is denoted by the following CLP clause.

$$
\begin{array}{l}
execA(a, Hist, (a, s_a, d_a, r_a, N_a)) \Leftarrow \\
\quad preC(a, P_a, Hist), \\
\quad getR(r_a), seqA(s_a, r_a, P_a, Hist), \\
\quad postC(a, N_a, Hist).
\end{array}
$$

$getR$ and $seqA$ determine the computation node assignment and the starting point of the action respectively. $preC$ and $postC$ are the results of translating the pre- and postconditions into constraints. The precondition clause determines a set of predecessors, which enable the execution of the action. The translated postcondition specifies the followers of the action, which get the data and/or control. The pre- and postcondition clauses have in general the following form:

$$
\begin{array}{l}
preC(a, [], Hist). \\
preC(a, [(p, s_p, d_p, r_p, N_p)|\iota_P], Hist) \Leftarrow \\
\quad fre(a, (p, s_p, d_p, r_p, N_p), Hist),\ a \in N_p, \\
\quad precondA(a, Hist),\ preC(a, \iota_P, Hist). \\
postC(a, N_a, Hist) \Leftarrow postcondA(a, Hist).
\end{array}
$$

Thereby, is the $fre(a, \iota_b, Hist)$ predicate true, iff there is a postfix T of the execution history $Hist$, with $\iota_b \in T$ and $\nexists \iota_a : \iota_a \in T$. The $precondA$ and $postcondA$ predicates stand for a history dependant part of the clauses. Some pre- and postconditions from the CPS study listed in Tab.1 are shown in Fig. 6.

If an atomic action is being refined by an automaton, the pre- and postconditions are replaced by its representation as a CLP clause. The translation of AutoFOCUS automata is discussed in [23].

Using the $seqA$ predicate different sequentialization strategies can be realized. The *minimal* sequentialization, (i.e. all independent actions are executed in parallel) is given by the following clause.

$$
\begin{array}{l}
seqA(s_a, r_a, [], Hist). \\
seqA(s_a, r_p, [(p, s_p, d_p, r_p, \_)|\iota_P], Hist) \Rightarrow \\
\quad s_p + d_p \leq s_a, seqA(s_a, r_p, \iota_P, Hist).
\end{array}
$$

The maximal sequentialization, as defined in Sec. 3.3, is given by the following implementation of the $seqA$ clause.

$$
\begin{array}{l}
seqA(s_a, r_a, \iota_P, []). \\
seqA(s_a, r_a, \iota_P, [(p, s_p, d_p, r_a)|Hist]) \Leftarrow !, \\
\quad s_p + d_p \leq s_a, seqA(s_a, r_a, \iota_P, Hist). \\
seqA(s_a, r_a, \iota_P, [(p, s_p, d_p, r_p)|Hist]) \Leftarrow \\
\quad (p, s_p, d_p, r_p) \in \iota_P, !, s_p + d_p \leq s_a, \\
\quad seqA(s_a, r_a, \iota_P, Hist). \\
seqA(s_a, r_a, \iota_P, [\iota_p|Hist]) \Leftarrow seqA(s_a, r_a, P_a, Hist).
\end{array}
$$

Control. Actions can be executed either by using the global flow relation or by executing the system runs. The implementation of the both cases is rather straight forward and lies beyond the scope of this paper.

The top-level clause $sysRun$ produces an execution run of the system.

Proposition 2 (composition of time constraints) *Suppose we are given the parallel composition of the systems runs* $SR = \{sr_1, \ldots, sr_n\}$ *with associated time constraints* $rtreq_s$ *for the resulting system* s. *Suppose*

that for each of the system runs sr we have associated time constraints $rtreq_{sr}$ such that, for each time constraint $resp(sr, I, O)$, each $i \in I$ and $o \in O$, the part of the control flow graph between receiving an input i and the next response with output o, the sum of the demanded durations of the involved system runs is bounded by the time constraint $resp(sr, I, O)$. Then, if the CLP translation of each of the system runs sr satisfies the relevant time constraints $rtreq_{sr}$, the CLP translation of the parallel composition SR of the system runs satisfies the composition of the time constraints $rtreq_s$.

Note that it is possible to refine this result, for example by taking into account that during parallel executions, the synchronization delay may be reduced, but this is beyond scope here.

4.2 Properties

The $sysRun$ predicate from the previous section describes the set of all possible system execution traces. The demanded response times and activation periods prune a number of traces, resulting in the set of traces which fulfil the time requirements.

$$sysRun([(a, s, d, r)|Hist]), filterTAA(Hist).$$

The optimization of this strategy is the propagation of properties to the trace segments which they affect. For example the response time property affects only the execution of one particular system run. If it is evaluated directly after its execution, the set of invalid traces is reduced more efficiently. Further strategies for the performance improvement of the presented analysis method are discussed in Sec. 5.

Except of invariants like response times and activation periods, an interesting task is to find the maximal valid sequentialization (see appropriate definition in Sec. 3.5). It shows the minimum demand on physical components (e.g. ECUs), which fulfils the time requirements. The execution order of the actions generated for each component delivers a valid schedule table. The sequentialization strategy is controlled by the clause $seqA$.

Scheduling has been proven to be an NP-complete problem [25] in even simpler contexts than those characteristic to distributed systems represented as TAA. Hence, it is essential to work out tactics, which aim to produce good quality results in a reasonable time. These tactics guide the search for (maximal) valid sequentialization by introducing the additional constraints, which (a) fix the number of target nodes for distribution, (b) state a fixed assignment of particular actions to particular nodes, (c) introduce additional causal relationships between independent actions.

5 Results

Within the case study described along this paper, specification of the CPS system (see Sec. 2.3 and [16]) was modeled as a set of system runs, saved as a XML file. The CLP problem was generated automatically out of this file according to the schema presented in Sec. 4.1.

During the analysis of the system, problems from Sec. 4.2 were solved with the ECLiPSe system [24]. In particular, the consistency and the completeness of the timing specification was proved. Further on the deployment on one ECU with the corresponding schedule was computed.

6 Related Work

As a classic precursor to this work, in [15], a set of criteria is defined to help find errors in software requirements specifications. Based on these, analysis procedures are defined for state-machine based modeling languages that provide a semantic analysis of real-time process-control software requirements.

Similar to the problem of deployment and preparing the ground for composition-based reasoning, [2] considers problems with respect to real-time specification relating to inheritance with the goal to reuse programs. Here the problem is that changing real-time specifications in sub-classes may require substantial redefinitions. To solve the problem, real-time composition filters are introduced which concern the real-time characteristics of messages that are received or sent by an object. The filters can be used to specify real-time constraints and to reuse these without inheritance anomalies.

With respect to tool-support, [1] presents an extensible framework of the software tools to support model-based development of complex, parallel, real-time, computer systems. [5] considers a development environment for the development of complex instrumentation systems where the performance requirements force the instrumentation into a parallel real-time implementation. It allows one to model multiple aspects of the parallel system, representing resources, requirements, and algorithms. The instrumentation system is then automatically synthesized

Run	Action	Precondition	Postcondition
sr1	MC1	$MC1 \in N_{1dObjList}$	$N_{MC1} \equiv \{DFusion1\} \otimes N_{MC1} \equiv \{DFusion2\}$
sr1	DFusion1	$DFusion1 \in N_{MC1}$	$N_{DFusion1} \equiv \{SA, PreFire, PreSet, ParkAid\}$
sr1	DFusion2	$DFusion2 \in N_{MC1}$	$N_{DFusion2} \equiv \{SA, PreFire, PreSet, ParkAid\}$
sr1	SA	$SA \in N_{DFusion1} \cup N_{DFusion2}$	$N_{SA} \equiv \{CVMReq\} \otimes N_{SA} \equiv \{NoCVMReq\}$
sr9.2	S2	$S2 \in N_{CVMode}$	$(count(CVMode, tail(CVMeasure, Hist)) \leq 9 \wedge N_{S2} \equiv \{CVMode, Velocity\}) \otimes (N_{S2} \equiv \{Velocity0\})$
sr9.2	CVMode	$CVMode \in N_{S2}$	$true$
sr9.2	Velocity	$Velocity \in N_{S2}$	$true$
sr9.2	Velocity0	$Velocity0 \in N_{S2}$	$true$

Table 1. CPS Case Study: Pre- and Postconditions of atomic Actions

from high-level system models. [21] also considers constraint-based model development for embedded systems. Precisely, the paper describes a tool suite supporting the construction of such models. The tool suite can be linked to existing analysis frameworks.

On the modeling side, related to our UML-like AutoFOCUS model is the UML-based approach of [4]. A particular feature of that approach is the possibility to also describe a system's physical environment, which enables the derivation and analysis of real-time constraints from resp. against the environment dynamics.

A number of approaches, which extract (formal) models out of specifications, mainly deal with functional requirements [13], [26]. They address the time requirements as annotations or as a temporal logic formulas, operating in terms of logical system steps. The presented approach allows the developer to capture and model the timing information explicitly as well as to relate it with the terms and objects of design and implementation, like components, channels, schedules, tasks *etc.*

There exists a number of approaches, which abstract an *existing* task architecture to a graph, representing data and/or control dependencies between tasks [20], [18], [10]. Some of them provide a rudimentary support for conditional flow dependencies based on non-deterministic choices [10], [19]. Due to the high level of abstraction, coarse granular modeling as well as rather poor expressiveness, these approaches, whose main goal is schedulability, cannot find a valid schedule in many cases. On the other hand model checking based approaches [8], which analyze the complete behavior of the system, guarantee to find a schedule (if exists). The drawback of these approaches is their inefficiency for big problems, as they have to deal with a lot of information, not relevant for the scheduling problem. The "gray-box" view on the system, delivered by the presented approach allows the modeler on the one hand to hide the non-relevant implementation information within atomic actions and on the other hand to describe the time behavior with necessary level of details.

Timed automata [3] are a powerful instrument for describing time constraints. There exist model checking-based algorithms and verification tools for them [6], [9]. The drawback this is the absence of integration with the other views on a system (behavior, architecture), which makes timed automata to a standalone formalism.

7 Conclusion and Future Work

We have presented a flexible, modular approach for modeling and analysis of real-time requirements. The central idea of the approach is to model the demanded time behavior, which is aimed at guiding the creation of the system's architectural and functional design. The suggested methodology integrates the modeled demanded time behavior seamlessly with the model-based development process, associating it to the behavior and architecture models of the system. The CLP-based analysis method allows the user to verify the consistency and completeness of the timing specification as well as to investigate the possible valid deployment variants. Moreover the approach provides tracing of timing informations between specification, design and implementation documents.

For evaluation purposes, a prototypical implementation of the introduced concepts was made, which generates out of a system model, represented as a XML document, a CLP problem. Afterwords the problem can be executed by any Prolog-based CLP interpreter. Using this implementation the model of the CPS system was analyzed.

The presented formal model is supposed to be a basis for adapting wide accepted graphical notation techniques. A good candidate seem to us the members of the sequence chart family, like LSCs or MSCs. There exists a number approaches which model the (functional) requirements using sequence charts [17], [12]. An important prerequisite is the funding by formal semantic, which is also provided by MSCs and LSCs.

The computation of the communication schedule, as well as the consideration of heterogeneous bus topologies (e.g. computation nodes distributed over several connected bus lines, and/or event- vs. time-triggered bus paradigms) are considered as further promising directions of the investigation.

References

[1] B. Abbott and M. Joshi. Tools for model-based real-time system synthesis. In *ECBS 1997*, pages 65–72. IEEE Computer Society, 1997.

[2] M. Aksit, J. Bosch, W. van der Sterren, and L. Bergmans. Real-time specification inheritance anomalies and real-time filters. In M. Tokoro and R. Pareschi, editors, *ECOOP 1994*, volume 821 of *LNCS*, pages 386–407. Springer, 1994.

[3] R. Alur. Timed automata. In *Summer School on Verification of Digital and Hybrid Systems*, NATO-ASI, 1998.

[4] J. Axelsson. Unified modeling of real-time control systems and their physical environments using uml. In *ECBS 2001*, pages 18–. IEEE Computer Society, 2001.

[5] T. Bapty and J. Sztipanovits. Model-based engineering of large-scale real-time systems. In *ECBS 1997*, pages 467–. IEEE Computer Society, 1997.

[6] G. Behrmann, A. David, and K. G. Larsen. A tutorial on UPPAAL. In M. Bernardo and F. Corradini, editors, *Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004*, LNCS 3185, pages 200–236. Springer–Verlag, September 2004.

[7] J. Botaschanjan, J. Romberg, and O. Slotosch. *Formal Methods and Models for System Design – A System Level Perspective*, chapter MoDe: A Method for System-Level Architecture Evaluation. Kluwer Academic Publishers, 2004.

[8] S. Campos, E. Clarke, W. Marrero, M. Minea, and H. Hiraishi. Computing quantitative characteristics of finite-state real-time systems. In *Proceedings of IEEE Real-Time Systems Symposium*, 1994.

[9] C. Daws and S. Tripakis. Model checking of real-time reachability properties using abstractions. In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS'98*, LNCS 1384, 1998.

[10] P. Eles, K. Kuchcinski, Z. Peng, A. Doboli, and P. Pop. Scheduling of conditional process graphs for the synthesis of embedded systems. In *Proceedings of the Design Automation and Test in Europe*, pages 132–139, 1998.

[11] T. Frühwirth and S. Abdennadher. *Essentials of Constraint Programming*. Springen-Verlag Berlin Heidelberg New York, 2003.

[12] D. Harel and M. Rami. Specifying and executing behavioral requirements: the play-in/play-out approach. *Journal on Software and Systems Modeling, SOSYM*, 2(2):82–107, 2003.

[13] P. Heymans and E. Dubois. Scenario-based techniques for supporting the elaboration and the validation of formal requirements. Technical report, Technical Report of the University of Namur, October 1998.

[14] F. Huber, B. Schätz, and G. Einert. Consistent Graphical Specification of Distributed Systems. In J. Fitzgerald, C. Jones, and P. Lucas, editors, *Industrial Applications and Strengthened Foundations of Formal Methods (FME'97)*, LNCS 1313, pages 122–141. Springer Verlag, 1997.

[15] M. S. Jaffe, N. G. Leveson, M. P. E. Heimdahl, and B. E. Melhart. Software requirements analysis for real-time process-control systems. *IEEE Trans. Software Eng.*, 1991.

[16] Kowalewski and Rittel. Real-time service allocation for car periphery supervision. Technical report, Robert Bosch GmbH. Research and Development – FV/SLD, 2002.

[17] I. Krüger. *Distributed System Design with Message Sequence Charts*. PhD thesis, Technische Universität München, 2000.

[18] K. Kuchcinski. Embedded system synthesis by timing constraint solving. In *Proceedings of the International Symposium on System Synthesis*, pages 50–57, 1997.

[19] K. Kuchcinski. Constraints driven design space exploration for distributed embedded systems. *Journal of Systems Architecture*, 47:241–261, 2001.

[20] E. A. Lee and T. M. Parks. Dataflow process networks. In *Proceedings of the IEEE*, volume 83, pages 773–801, May 1995.

[21] S. Neema, J. Sztipanovits, G. Karsai, and K. Butts. Constraint-based design-space exploration and model synthesis. In R. Alur and I. Lee, editors, *EMSOFT 2003*, volume 2855 of *LNCS*, pages 290–305. Springer, 2003.

[22] J. Philipps and O. Slotosch. The quest for correct systems: Model checking of diagrams and datatypes. In *APSEC'99: Asian Pacific Software Engineering Conference*, pages 449 – 458. IEEE Computer Society, 1999.

[23] A. Pretschner and H. Lötzbeyer. Model Based Testing with Constraint Logic Programming: First Results and Challenges. In *Proc. 2nd ICSE Intl. Workshop on Automated Program Analysis, Testing and Verification (WAPATV'01), Toronto*, 2001.

[24] H. Simonis. Developing applications with ECLiPSe. Technical report, IC-Parc, Imperial College London, 2003.

[25] D. Ullman. NP-Complete Scheduling Problems. *Journal of Computer Systems Science*, 10:384–393, 1975.

[26] A. van Lamsweerde. From system goals to software architecture. In M. Bernardo and P. Inverardi, editors, *Formal Methods for Software Architectures*, LNCS 2804, pages 25–43. Springer-Verlag, 2003.