Contracts for Schedulability Analysis

Philipp Reinkemeier, Albert Benveniste, Werner Damm, Ingo Stierand

Contracts for Schedulability Analysis ⋆

Philipp Reinkemeier[1], Albert Benveniste[2], Werner Damm[1 3], and Ingo Stierand[3]

[1] Offis, Oldenburg, Germany
[2] Inria-Rennes, France
[3] Dept of Computer Sciences, University of Oldenburg, Germany

Abstract. In this paper we propose a framework of Assume / Guarantee contracts for schedulability analysis. Unlike previous work addressing compositional scheduling analysis, our objective is to provide support for the OEM / supplier subcontracting relation. The adaptation of Assume / Guarantee contracts to schedulability analysis requires some care, due to the handling of conflicts caused by shared resources. We illustrate our framework in the context of AUTOSAR methodology now popular in the automotive industry sector.

1 Introduction and Related Work

The focus of this work is the integration phase of a design process, where software components are allocated to a hardware platform. We consider scenarios like the fol-

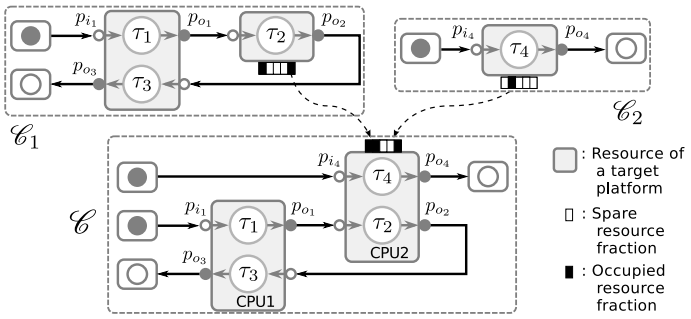

Fig. 1. Exemplary Integration Scenario using Resource Segregation

lowing. The bottom part of Figure 1 shows a target platform that is envisioned by, say, an Original Equipment Manufacturer (OEM). It consists of two processing nodes (CPU_1 and CPU_2). Suppose the OEM wants to implement two applications, characterized by contracts $\mathscr{C}_1$ and $\mathscr{C}_2$, on this architecture and delegates their actual implementation to two different suppliers. Both applications share a subset of the resources of the target

⋆ This work was partly supported by the Federal Ministry for Education and Research (BMBF) under support code 01IS11035M, *Automotive, Railway and Avionics Multicore Systems* (ARAMiS), and by the German Research Council (DFG) as part of the Transregional Collaborative Research Center *Automatic Verification and Analysis of Complex Systems* (SFB/TR 14 AVACS).

platform, e.g. tasks τ_2 and τ_4 are executed on CPU_2 after integration. Furthermore, we assume the system specification $\mathscr{C}$ shown in Figure 1 to be available from previous design phases. While some components together with their (local) contracts may also be known (e.g. in case of reuse), the OEM generally has to negotiate proper specifications with the suppliers, in our case the two contracts $\mathscr{C}_1$ and $\mathscr{C}_2$. In doing so, the scheduling of the software components delegated to each supplier must yield a satisfactory scheduling at system integration time, meaning that timing constraints are met given the performance characteristics of the computing and communication resources, even though the two designs compete for shared resources.

Quoted verbatim from [17] by Insup Lee et al.:[4] *Real-time systems could benefit from component-based design, only if components can be assembled without violating compositionality on timing properties. When the timing properties of components can be analyzed compositionally, component-based real-time systems allow components to be developed and validated independently and to be assembled together without global validation.* [17] develops a model of *scheduling interface* collecting the workloads, resources, and scheduling policy, addressing the above quoted objectives. Specific classes of hard real-time system scheduling problems are considered, namely periodic models and bounded-delay models. This group of authors has further developed the same track with the same techniques, making increasingly large classes of scheduling problems amenable for compositional analysis. This significant body of work is nicely summarized in the tutorial paper [3] and implemented through the CARTS tool for compositional analysis of real-time systems [14]. One interesting application case concerns the scheduling of ARINC partitions [10]. Compositional schedulability analyses have been proposed on top of the UPPAAL tool [6,5] with mixed scheduling policies and probabilistic evaluations. In a different direction, Lothar Thiele and co-workers have developed for real-time scheduling an algebraic framework called the *Real-Time Calculus* (RTC) [21,22,13]. Components of the RT Calculus are linear transfer functions in the max-plus algebra and interface behaviors are expressed as *arrival curves*, which specify lower and upper bounds for event arrivals. [21,20] considers real-time interfaces where assumptions and guarantees are expressed by means of arrival curves on inputs and outputs, respectively. Refinement of such interfaces is characterized and a parallel composition is defined; *adaptive interfaces* are proposed in which arrival curves are propagated throughout the network of components, compositionally. The above very elegant model captures timing and precedence constraints but does not consider conflicts due to shared resources. A blending of this model with timed automata is studied in [11] together with a mapping of RTC-based real-time interfaces to timed automata. The work [12] applies and develops similar techniques for distributed heterogeneous time-triggered automotive systems.

Our work in this section has its roots in [19,15,16,18], which in turn are based upon the ideas underlying the interfaces for control and scheduling proposed in [23,1,2]. Contrary to these scheduling interfaces, we consider multiple resources and take task precedences into account. Our aim is different from the previous set of references and complements it nicely. In our work we assume that a procedure performing global

[4] The reader is referred to this paper for further discussion on related work from the real-time scheduling community. We discuss here the references that are directly relevant to our work.

scheduling analysis is available, so our focus is not on specific classes of schedulability analyses. Our aim is rather to lift such procedures to a contract framework supporting OEM-supplier relations in a supply chain. We first provide support for decomposing a system-level scheduling contract into sub-contracts for suppliers, while guaranteeing safe system integration—this is different from the objectives of previous compositional schedulability studies. Second, our contract framework provides support for fusing different viewpoints on the system using contract conjunction. The model of *scheduling components* is presented in Section 2. Scheduling components capture implementations and environments for our *scheduling contracts,* developed in Section 3. Our approach is illustrated in Section 4 by an example in the context of the AUTOSAR methodology.

2 Scheduling Components

2.1 Our approach: Building on top of Assume/Guarantee contracts

Recall that Assume/Guarantee contracts (A/G-contracts) are pairs of assumption and guarantees: $\mathscr{C} = (A, G)$. In the basic A/G-contract framework [4], A and G are assertions, i.e., sets of traces for system variables.[5] Components capturing legal implementations or environments of contracts are also modeled by assertions. Component E is a legal environment for $\mathscr{C}$ if $E \subseteq A$ and component M is an implementation for $\mathscr{C}$ if $A \times M \subseteq G$. In this writing, $\subseteq$ is simply set inclusion and component composition $\times$ is by intersection of sets of traces (assuming that the underlying set of system variables is universal and thus fixed): $A \times M =_{\text{def}} A \cap M$. One key notion of A/G-contracts is *saturation*: Contracts (A, G) and $(A, G \cup \neg A)$, where $\neg$ is set complement, possess identical sets of legal environments and implementations, so we consider them equivalent. The second one is called *saturated* and is a canonical form for the class of equivalent contracts. Also, $M_{\mathscr{C}} = G \cup \neg A$ is the maximal implementation for this contract. Thus, we need the operations $\cup$ and $\neg$, or at least we need the operation $(A, G) \rightarrow G \cup \neg A$, which is to be interpreted as "A entails G". Then for $\mathscr{C}$ and $\mathscr{C}'$ two saturated contracts, *refinement* $\mathscr{C}' \preceq \mathscr{C}$ holds iff $A' \supseteq A$ and $G' \subseteq G$ hold. The result of the composition $\mathscr{C}_1 \otimes \mathscr{C}_2$ of two saturated contracts, is a contract $((A_1 \cap A_2) \cup \neg(G_1 \cap G_2), G_1 \cap G_2)$.

Thus, as a first step, we need the counterpart of assertions for our scheduling component framework, with the associated algebra. Ingredients of scheduling problems are: *tasks* with their *precedence conditions* reflecting data dependencies and *resource allocation*. The sets of *timed traces* we are interested in are those satisfying the scheduling constraints, plus quantitative properties such as period, deadline conditions, etc. We call the resulting model *concrete scheduling components*. Unfortunately, no rich algebra with the requested operators $\subseteq, \times, \cup, \neg$ exists for concrete scheduling components.

By abstracting away part of the description of task activities in traces, we slightly abstract concrete scheduling components to so-called *abstract* ones. The idea is that we keep only what is essential for capturing interactions of scheduling problems, namely: 1) trigger and release events for tasks, and 2) busyness of resources. The abstraction map binds each concrete scheduling component to its abstraction and we will show that this binding is faithful with respect to composition and refinement. The framework

[5] These are typically specified using modeling tools such as Simulink/Stateflow.

of abstract scheduling components is simple enough so we manage to equip it with the wanted operations $\subseteq, \times, \cap, \cup, \neg$. A/G-contracts for abstract scheduling components follow then easily. The rest of this section is devoted to the introduction of concrete and abstract scheduling components. Then we study the relation between them.

2.2 Concrete Scheduling Components

For our model of scheduling components we assume the following:

- A slotted model of real-time, in which the real line $\mathbb{R}_+$ is divided into successive discrete time slots of equal duration. Successive slots are thus indexed by using natural numbers $1, 2, 3, \ldots, n, \ldots \in \mathbb{N}$.
- An underlying set $\mathcal{T}$ of *tasks*, generically denoted by the symbol τ. To describe events of interest for tasks, we consider the following alphabets: The *control alphabet* $\Sigma_c = \{i, o, io, aw, sl\}$ collects the *trigger*, *completion*, *trigger-and-completion*, *awake*, and *sleeping* events, for a task; this alphabet describes the triggering and completion of tasks; since we follow a slotted model of time, triggering and completion can occur within the same slot, which is indicated by the event *io*; The *busyness alphabet* $\Sigma_b = \{*, \bot\}$ collecting the *busy* and *idle* events; this alphabet indicates, for a task τ , its status busy/idle at a given time slot. On top of these alphabets, we build:

$$\Sigma_\tau =_{\text{def}} \{(c, b) \in \Sigma_c \times \Sigma_b \mid c{=}sl \;\Rightarrow\; b{=}\bot\} \tag{1}$$

  reflecting that task τ can only be busy when it is not sleeping. The status of a task τ in each time slot is expressed by using alphabet Σ_τ.
  In addition, each task τ comes equipped with a pair $(p^t(\tau), p^c(\tau)) \in \mathcal{P}{\times}\mathcal{P}$ of *trigger* and *completion ports*, where $\mathcal{P}$ is an underlying set of *ports.* For T a set of tasks, we will consider the set $P_T =_{\text{def}} \{p^t(\tau), p^c(\tau) \mid \tau \in T\}$.
- An underlying set $\mathcal{R}$ of *resources*, generically denoted by the symbol r. A resource can be either available or busy with executing a given task at a given time slot. Resources can run in parallel. Each resource $r \in \mathcal{R}$ is assigned the alphabet $\Sigma_r \subseteq \mathcal{T} \cup \{0\}$ of the tasks it can run, where the special symbol 0 indicates that r is idle.

Definition 1. *A* concrete scheduling component *is a tuple* $\mathbf{M} = (K, L)$*, where:*

- $K = (T, R, \rho)$ *is the* sort *of* $\mathbf{M}$*, where:* $T \subseteq \mathcal{T}$ *is the set of* tasks*,* $R \subseteq \mathcal{R}$ *is the set of* resources*, and* $\rho : T \to R$*, the* resource allocation map*, is a partial function satisfying* $\tau \in \Sigma_{\rho(\tau)}$.
  Say that tasks τ_1 *and* τ_2 *are* non-conflicting *if they do not use the same resource:* $\tau_1 \|_K \tau_2$ *if* $\rho(\tau_1)$ *is undefined, or* $\rho(\tau_2)$ *is undefined or* $\rho(\tau_1) \neq \rho(\tau_2)$.
- *For* $\tau_1, \tau_2 \in T$*, say that* τ_1 precedes *task* τ_2*, written* $\tau_1 \multimap\!\!\to \tau_2$*, if the completion port of* τ_1 *coincides with the start port of* τ_2*:* $p^c(\tau_1) = p^t(\tau_2)$.
  We require that this relation is cycle free and we denote by $\preceq$ *the partial order on* T *obtained by taking transitive closure of* $\multimap\!\!\to$ *and we call* $\preceq$ *the* precedence order.
  The dual order between ports will also be needed: for $p_1, p_2 \in P_T$*, say that* p_1 precedes p_2*, written* $p_1 \dashv\!\!\boxplus\!\!\to p_2$*, if there exists* $\tau \in T$ *such that* $p_1 = p^t(\tau)$ *and* $p^c(\tau) = p_2$*; relation* $\dashv\!\!\boxplus\!\!\to$ *is cycle free if so was* $\multimap\!\!\to$ *and, with no risk of confusion, we also denote by* $\preceq$ *the* precedence order *on* P_T *generated by* $\dashv\!\!\boxplus\!\!\to$.

– $L \subseteq \Sigma_T^\omega$ *is the* language *of* $\mathbf{M}$*, where* $\Sigma_T =_{\text{def}} T \to \Sigma_\tau$*, and* A^ω *denotes the set of all infinite words over alphabet* A*. Due to the decomposition* (1) *of* Σ_τ*, every word* $w \in L$ *can be equivalently seen as a pair of words* $w = (w_c, w_b)$ *describing the control and busyness history of* w*.*

Since a word $w \in L$ yields a history for each task, it induces, by picking the resource running that task, a corresponding *resource word* w^R, such that

$$\begin{array}{l} w^R \text{ is the tuple collecting the } w^r \text{ for } r \in R, \text{ such that, for every slot } n\text{:} \\ w^r(n) = \{\tau \in T \mid \rho(\tau) = r \text{ and } w(\tau, n) = (c, b) \text{ satisfies } b = *\} \end{array} \tag{2}$$

i.e., $w^r(n)$ returns the set of tasks that resource r runs at slot n. This set is not a singleton if and only if a conflict occurs at slot n regarding resource r.

Whenever convenient, we will denote by T_K or T^K the set of tasks of sort K, and similarly for the other constituents of a sort. The events of a task τ will be denoted by i_τ, o_τ, etc. For w a word of Σ_K^ω, $T' \subseteq T$, and $R' \subseteq R$, we denote by

$$w^{T'} \text{ the } T'\text{-word of } w, \text{ and by } w^{R'} \text{ the } R'\text{-word of } w, \tag{3}$$

obtained by projecting w to the sub-alphabet $\Sigma_{T'}$ and projecting the induced word w^R to the sub-alphabet $\Sigma_{R'}$, respectively.

Definition 2 (Semantics of concrete scheduling components). *Call* behavior *of sort* K *any infinite word* $w \in \Sigma_T^\omega$ *satisfying the following three* scheduling conditions*:*

1. *For each task* $\tau \in T$*, the control word* w_c *belongs to the language* $(sl^*.(io+i.aw^*.o))^\omega$*, where* $a^* =_{\text{def}} \epsilon+a+a.a+a.a.a+\ldots$ *is the Kleene closure starting at the empty word. Informally, the two events* i *and* o *alternate in* w*, with* i *occurring first;* io *is interpreted as the immediate succession of two* i *and* o *events at the same time slot.*
2. $\tau_1 \preceq \tau_2$ *implies that, for every* $n \geq 1$*, the* n*th occurrence of event* o_{τ_1} *must have occurred in* w *strictly before* i_{τ_2} *(in words,* τ_2 *can only start after* τ_1 *has completed);*
3. w *is* non-conflicting*: for any two conflicting tasks* τ_1 *and* τ_2 *belonging to* T *(cf. Definition* 1*), it never happens that* w^{τ_1} *and* w^{τ_2} *are non-idle at the same time slot.*

The semantics of $\mathbf{M}$ *is the sub-language* $[\![\mathbf{M}]\!] \subseteq L$ *consisting of all behaviors of* K *belonging to* L*. Say that* $\mathbf{M}$ *is* schedulable *if* $[\![\mathbf{M}]\!] \neq \emptyset$*.*

Due to the above Condition 2, tasks related by precedence conditions possess identical logical clocks—in particular, if they are specified periodic, their periods must be equal. This is not required for tasks not related by precedence conditions.

Comment 1 The pair $\mathbf{M} = (K, L)$ can be seen as the specification of a global scheduling problem. The sort K fixes the set of tasks and their precedence conditions, the set of resources, and the allocation of tasks to resources. The language L can serve to specify additional aspects of this scheduling problem, including task durations and/or minimum time interval between successive activation calls for a task. Semantics $[\![\mathbf{M}]\!]$ can be seen as the maximally permissive solution of the scheduling problem stated by $\mathbf{M}$.

Definition 3. *Say that* $\mathbf{M}_1$ *and* $\mathbf{M}_2$ *are* composable *if their allocation maps* ρ_1 *and* ρ_2 *coincide on* $T_1 \cap T_2$ *and the relation* $\multimap_1 \cup \multimap_2$ *on* $T_1 \cup T_2$ *is cycle free. If* $\mathbf{M}_1$ *and* $\mathbf{M}_2$ *are composable, their* composition $\mathbf{M}_1 \underline{\times} \mathbf{M}_2 =_{\text{def}} ((T, R, \rho), L)$ *is defined as follows:*

$$T = T_1 \cup T_2\ ,\quad R = R_1 \cup R_2\ ,\quad \forall \tau{\in}T : \rho(\tau) = \textbf{if } \tau \in T_1 \textbf{ then } \rho_1(\tau) \textbf{ else } \rho_2(\tau)$$
$$L = \mathbf{pr}^{-1}_{T \to T_1}(L_1) \cap \mathbf{pr}^{-1}_{T \to T_2}(L_2)$$

where $\mathbf{pr}_{T \to T_i}, i = 1, 2$*, denotes the projection from* T *to* T_i *and* $\mathbf{pr}^{-1}$ *is its inverse.*

Of course, the key to understand the meaning of composition $\underline{\times}$ is the construction of the semantics $[\![\mathbf{M}_1 \underline{\times} \mathbf{M}_2]\!]$. In the following lemma, for $\mathbf{M} = ((T, R, \rho), L)$ a scheduling component, we identify its semantics $[\![\mathbf{M}]\!]$ (which is a language) with the scheduling component $((T, R, \rho), [\![\mathbf{M}]\!])$. This gives a meaning to the expression $[\![\mathbf{M}_1]\!] \underline{\times} [\![\mathbf{M}_2]\!]$.

Lemma 1. *If* $\mathbf{M}_1$ *and* $\mathbf{M}_2$ *are composable, then* $[\![\mathbf{M}_1 \underline{\times} \mathbf{M}_2]\!] = [\![\, [\![\mathbf{M}_1]\!] \underline{\times} [\![\mathbf{M}_2]\!] \,]\!]$.

As announced in the introductory discussion of Section 2, the model of concrete scheduling components is too complex and detailed as a model of components on top of which contracts can be built. We are unable to define the operations we need on components, particularly $\subseteq$ and $\cup\neg$ (in turn, parallel composition $\underline{\times}$ was easy to define as we have seen). The notion of *abstract scheduling component* we develop in the forthcoming section will overcome these difficulties. Abstract scheduling components capture the architecture aspect of Figure 1, namely: ports carrying start and completion events of tasks, and resources—tasks themselves are, however, ignored.

2.3 Abstract Scheduling Components

Definition 4. *An* abstract scheduling component *is a language* $M \subseteq \mathcal{V}^\omega$*, where* $\mathcal{V} =_{\text{def}} (\{0,1\}^{\mathcal{P}}) \times (\prod_{r \in \mathcal{R}} \Sigma_r)$.

Recall that Σ_r is the alphabet of tasks that can be executed by resource r, see the beginning of Section 2.2. Symbol "1" indicates the occurrence of an event at the considered port. Abstract scheduling components come equipped with the following algebra:

- The Boolean algebra $\cap, \cup, \neg$, and the inclusion $\subseteq$ on sets;
- A *parallel composition* by intersection: $M_1 \times M_2 =_{\text{def}} M_1 \cap M_2$.

Thus, abstract scheduling components offer all the algebra required for a universe of components on top of which A/G-contracts can be built. It is therefore interesting to map concrete to abstract scheduling components.

Recall that, for $K = (T, R, \rho)$ a sort, we denote by $P_T =_{\text{def}} p^{\text{i}}(T) \uplus p^{\text{c}}(T) \subseteq \mathcal{P}$ the set of ports used by T, see the beginning of Section 2.2. Then, we set

$$\mathcal{V}_K =_{\text{def}} (\{0,1\}^{P_T}) \times (\textstyle\prod_{r \in R} \Sigma_r) \tag{4}$$

Definition 5 (Mapping concrete to abstract scheduling components). *Each concrete scheduling component* $\mathbf{M} = (K, L)$ *is mapped to a unique abstract scheduling component* $[\![\mathbf{M}]\!]^{\mathsf{A}}$ *called its* abstract semantics, *defined as follows:*

1. *Pick any behavior* $w \in [\![\mathbf{M}]\!]$*, see Definition 2;*
2. *Denote by* $\pi_T(w)$ *the word over* $\{0,1\}^{P_T}$ *obtained from* w *as follows. For every* $p \in P_T$*, define* ${}^\bullet p = \{\tau \in T \mid p^c(\tau) = p\}$ *and* $p^\bullet = \{\tau \in T \mid p^t(\tau) = p\}$*, the sets of anterior and posterior tasks of* p*. The nth event of* p *is put* nondeterministically *after the* $n-1$*st event of* p*, when or after every task belonging to* ${}^\bullet p$ *has completed for the nth time in* w*, and strictly before every task belonging to* $p^\bullet$ *has started for the nth time in* w*. If* ${}^\bullet p = \emptyset$*, then the first condition is not considered and similarly if* $p^\bullet = \emptyset$*.*
3. *Denote by* $\pi_R(w)$ *the word over alphabet* $\prod_{r\in R} \Sigma_r$ *defined as follows: For every time slot* n *and every resource* $r \in R$*, set*

   $\pi_R(w)(r,n) = \tau$ *if and only if* $w(\tau,n) = (c,b)$ *satisfies* $b = *$ *and* $\rho(\tau) = r$*.*

   This part of word $\pi_R(w)$ *represents the "positive history" of* w*, i.e., the use of the resources belonging to* R *by tasks belonging to* T*; We complement* $\pi_R(w)$ *by describing the "negative history" of* w*, consisting of a description of all the possibilities left, for tasks not belonging to* T*, in using resources from* R*:*

   in all slots of $\pi_R(w)(r,n)$ *that are still idle, we set* $\pi_R(w)(r,n) = \tau'$ *where* $\tau' \in \Sigma_r, \tau' \notin T$ *is chosen nondeterministically.*

   Then, with reference to the sort $K = (T,R,\rho)$ *of* $\mathbf{M}$*, we set:*

$$\eta_K(w) \ =_{\text{def}} \ (\pi_T(w), \pi_R(w)) \ \in \ \mathcal{V}_K^\omega$$

4. *Finally, we define*

$$[\![\mathbf{M}]\!]^{\mathsf{A}} \ =_{\text{def}} \ \mathbf{pr}^{-1}_{\mathcal{V}^\omega \to \mathcal{V}_K^\omega}\Big(\big\{\eta_K(w) \,\big|\, w \in [\![\mathbf{M}]\!]\big\}\Big) \ \subseteq \ \mathcal{V}^\omega$$

   where the quantification ranges over $w \in [\![\mathbf{M}]\!]$ *and all instances of nondeterministic choices in step 2, and* $\mathbf{pr}_{\mathcal{V}^\omega \to \mathcal{V}_K^\omega}$ *denotes the projection, from* $\mathcal{V}^\omega$ *to* $\mathcal{V}_K^\omega$*.*

Step 2 is sound since w is a behavior in the sense of Definition 2. Step 2 is the key step since it transforms a max-plus type of parallel composition (every task waits for all its preceding tasks having completed before starting) into a dataflow connection where data are communicated through the shared ports. The data communicated are the events carried by the ports. These events occur nondeterministically after all preceding tasks have completed for the nth time and before all succeeding tasks start for the nth time.

Step 3 complements the actual history of each task of $\mathbf{M}$ by an explicit description of all possibilities that are left to other scheduling components in using resources shared with $\mathbf{M}$. The reason for doing this is that this allows to capture the interleaved use of shared resources by different components, by a simple parallel composition by intersection.

The above construction is illustrated in Figure 2. When hiding the tasks sitting inside the boxes, the architecture shown on Figure 1 is a dataflow representation of $[\![\mathbf{M}]\!]^{\mathsf{A}}$: in interpreting this figure, one should consider that each task is free to start any time after it has received its triggering event, and free to wait for some time before emitting its completion event.

Lemma 2. *The mapping* $\mathbf{M} \to [\![\mathbf{M}]\!]^{\mathsf{A}}$ *satisfies the following properties:* 1) *Schedulability is preserved in that* $[\![\mathbf{M}]\!] \neq \emptyset$ *if and only if* $[\![\mathbf{M}]\!]^{\mathsf{A}} \neq \emptyset$*;* 2) *For every* $r \notin R$*, the set* $\{v(r) \,\big|\, v \in [\![\mathbf{M}]\!]^{\mathsf{A}}\}$ *is the free language* $(\mathcal{T} - T)^\omega$*.*

The special property 2) is not preserved under the Boolean set algebra. Therefore, the mapping $\mathbf{M} \to [\![\mathbf{M}]\!]^{\mathsf{A}}$ is not surjective.

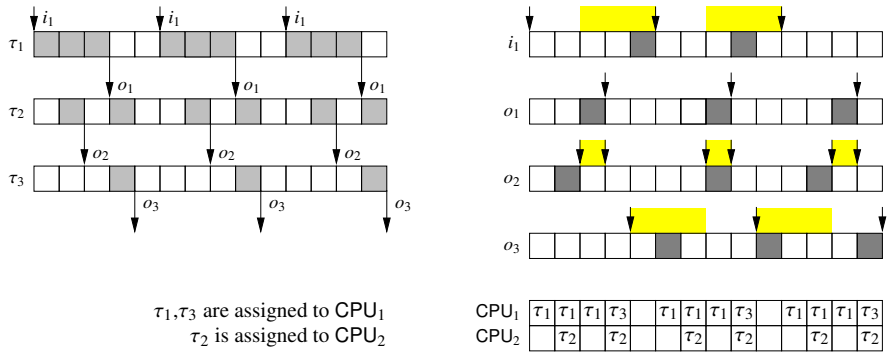

Fig. 2. Showing a concrete behavior of $\mathbf{M}$ (left, with reference to Figure 1) and a corresponding abstract behavior of $[\![\mathbf{M}]\!]^{\mathsf{A}}$ (right), by using $\mathcal{P} = \{i_1, o_1, o_2, o_3\}$ as underlying alphabet of ports. On the second diagram, blanks figure the slots left free for any external task to run on the referred resource. The yellow rectangles indicate the room for nondeterministic choices; bounds of these rooms are figured by pointing arrows; where such arrow is missing, the corresponding rectangle is unbounded on that side.

2.4 Faithfulness of the mapping

Consider two concrete scheduling components $\mathbf{M}_1$ and $\mathbf{M}_2$, where $\mathbf{M}_i = (K_i, L_i)$ and $K_i = (T_i, R_i, \rho_i)$. It is difficult and generally undecidable to compare their abstract semantics: $[\![\mathbf{M}_1]\!]^{\mathsf{A}} \subseteq [\![\mathbf{M}_2]\!]^{\mathsf{A}}$. We will, however, need such checks in the sequel. We thus propose some effective sufficient conditions ensuring the inclusion of abstract semantics.

Lemma 3. *The following conditions on the pair* $(\mathbf{M}', \mathbf{M})$ *imply* $[\![\mathbf{M}']\!]^{\mathsf{A}} \subseteq [\![\mathbf{M}]\!]^{\mathsf{A}}$*:*

1. *There exists a surjective total map* $\psi : T' \to T$*, such that:*
   (a) *For every* $\tau \in T$*:* $p^t(\tau) = \min_{\psi(\tau')=\tau} p^t(\tau')$ *and* $p^c(\tau) = \max_{\psi(\tau')=\tau} p^c(\tau')$*, where min and max refer to the order* $\preceq'$ *generated by the precedence relation* $\multimap$ *on ports of* $\mathbf{M}'$*, see Definition 1;*
   (b) *The following holds, for every 4-tuple of tasks* $(\tau'_1, \tau'_2, \tau_1, \tau_2) \in T'^2 \times T^2$*:* $\psi(\tau'_1){=}\tau_1$ *and* $\psi(\tau'_2){=}\tau_2$ *together entail* $\tau'_1 \parallel' \tau'_2 \Rightarrow \tau_1 \parallel \tau_2$*.*
2. *For each task* $\tau \in T$*, there exists an injective total map* $\chi_\tau : \Sigma_\tau \to \biguplus_{\tau' \in \psi^{-1}(\tau)} \Sigma_{\tau'}$*, where the* $\Sigma_{\tau'}$ *are copies, for each referred task* τ'*, of the alphabet defined in* (1)*; set* $\chi =_{\text{def}} \biguplus_{\tau \in T} \chi_\tau$*. The language* L *is defined through some temporal property* Timing_Prop *on the events from alphabet* $\biguplus_{\tau \in T} \Sigma_\tau$*, and, replacing, in* Timing_Prop*, every event* e *by its image* $\chi(e)$ *defines a language* L'' *such that* $L'' \supseteq L'$*.*

Say that $\mathbf{M}' \sqsubseteq \mathbf{M}$ *when the above three conditions hold.*

Observe that Conditions 1 involve only the sorts K_1 and K_2 of $\mathbf{M}_1$ and $\mathbf{M}_2$. Condition 2 formalizes the situation in which the language L_2 is specified through timing properties relating certain events of interest for tasks of $\mathbf{M}_2$ (duration between trigger and completion, end-to-end duration when traversing a set of successive tasks, etc.). The considered events are then mapped to some events of $\mathbf{M}_1$ and the timing property remains the same or is strengthened. The following results hold:

Lemma 4. *If* $\mathbf{M}_1$ *and* $\mathbf{M}_2$ *are composable, then* $[\![\mathbf{M}_1 \preceq \mathbf{M}_2]\!]^{\mathsf{A}} = [\![\mathbf{M}_1]\!]^{\mathsf{A}} \times [\![\mathbf{M}_2]\!]^{\mathsf{A}}$.

Lemma 5. *Let* $(\mathbf{A}, \mathbf{G})$ *be a composable pair of concrete scheduling components such that* $R_{\mathbf{A}} = \emptyset$. *Then, the following formulas define a concrete scheduling component* $\mathbf{M} = ((T, R, \rho), L)$ *such that* $[\![\mathbf{M}]\!]^{\mathsf{A}} = [\![\mathbf{G}]\!]^{\mathsf{A}} \cup \neg[\![\mathbf{A}]\!]^{\mathsf{A}}$:

$$T = T_{\mathbf{A}} \cup T_{\mathbf{G}}\ ,\ R = R_{\mathbf{G}}\ ,\ \rho(\tau) = \textbf{if } \tau \in T_{\mathbf{A}} \textbf{ then } \rho_{\mathbf{A}}(\tau),$$
$$L = \mathbf{pr}^{-1}_{T \to T_{\mathbf{G}}}(L_{\mathbf{G}}) \cup \mathbf{pr}^{-1}_{T \to T_{\mathbf{A}}}(\neg L_{\mathbf{A}})$$

3 Scheduling Contracts

As recommended in Section 2.1, we first define what components are, and then we define contracts. Regarding components, the notations used here refer to the operations $\subseteq, \cap, \cup, \neg, \times$ introduced for abstract scheduling components in Section 2.3.

Definition 6 (Scheduling contracts). *A* scheduling contract *is a pair* $\mathscr{C} = (A, G)$ *of abstract scheduling components, called the* assumptions *and the* guarantees.

The set $\mathcal{E}_{\mathscr{C}}$ *of the legal environments for* $\mathscr{C}$ *collects all abstract scheduling components* E *with non-empty semantics such that* $E \subseteq A$. *The set* $\mathcal{M}_{\mathscr{C}}$ *of all implementations of* $\mathscr{C}$ *consists of all abstract scheduling components* M *with non-empty semantics such that* $A \times M \subseteq G$.

Each scheduling contract can be put in its equivalent saturated form $\mathscr{C} = (A, G \cup \neg A)$, *possessing the same sets of legal environments and implementations. Scheduling contract* $\mathscr{C}$ *is compatible if and only if* $A \neq \emptyset$ *and consistent if and only if* $G \cup \neg A \neq \emptyset$. *Say that scheduling contract* $\mathscr{C} = (A, G)$ *is* schedulable *if* $A \cap G \neq \emptyset$.

The background theory of A/G-contracts applies. Note that $A \cap G = A \cap (G \cup \neg A)$, hence checking schedulability does not require the contract to be saturated. The justification of this notion of schedulability for contracts is given in the next section.

In practice the designer will specify scheduling contracts using concrete, not abstract, scheduling components:

Definition 7 (Concrete scheduling contracts). *Call* concrete *scheduling contract (or concrete contract) a pair* $\mathbf{C} = (\mathbf{A}, \mathbf{G})$ *of composable concrete scheduling components called its* assumptions *and* guarantees.

See Definition 3 for the notion of composability. To contrast with concrete contracts, we will sometimes call *abstract scheduling contracts*, or *abstract contracts*, the scheduling contracts of Definition 6. The mapping from concrete to abstract scheduling components developed in Section 2.3 allows mapping concrete scheduling contracts to abstract ones:

$$\mathbf{C} = (\mathbf{A}, \mathbf{G}) \ \mapsto\ \mathscr{C}_{(\mathbf{A},\mathbf{G})} =_{\text{def}} ([\![\mathbf{A}]\!]^{\mathsf{A}}, [\![\mathbf{G}]\!]^{\mathsf{A}})$$

Say that $\mathbf{C}$ is *consistent, compatible, schedulable,* or *in saturated form,* if so is $\mathscr{C}_{(\mathbf{A},\mathbf{G})}$.

Lemma 6 (Checking for contract refinement). *The following conditions imply refinement* $\mathscr{C}_{(\mathbf{A},\mathbf{G})} \preceq \left(\bigwedge_{j \in J} \mathscr{C}_{(\mathbf{A}_j,\mathbf{G}_j)}\right)$: *for every* $j \in J$, $\mathbf{A}_j$ *is composable with* $\mathbf{G}$ *and the following two conditions hold:* $\mathbf{A}_j \preceq \mathbf{G} \sqsubseteq \mathbf{G}_j$ *and* $\mathbf{A}_j \sqsubseteq \mathbf{A}$.

3.1 Getting sub-contracts in the AUTOSAR development process

In this section we develop techniques in support of the following design steps, which are advocated by the AUTOSAR methodology [7,8]:

Process 1 (AUTOSAR development process)

1. Start with a top-level, system wide, contract. At this level, only functions are considered whereas computing resources are ignored. Functions are abstracted as systems of tasks with their precedence constraints.
2. To prepare for subcontracting to different suppliers, decompose this functional top-level contract into functional sub-contracts.
3. At this step the computing resources are now taken into account. Perform system wide (global) schedulability analysis, thus inferring resource budgets.
4. Derive resource aware sub-contracts and submit them to the supplier.

This process is rather informal. It is thus tempting to interpret the above tasks as refinement steps, for scheduling contracts. With this in mind, Steps 1 and 2 exhibit no particular difficulty. Step 3, however, raises a problem. Adding the consideration of resources to a resourceless contract cannot be a refinement step. This can be seen from Lemma 3, which gives sufficient conditions for concrete contract refinement: referring to this lemma, there is no way that the resulting contracts $\mathbf{C}' = (\mathbf{A}', \mathbf{G}')$ can refine $\mathbf{C}$ since $[\![\mathbf{A}']\!]^{\mathsf{A}} \supseteq [\![\mathbf{A}]\!]^{\mathsf{A}}$ is not possible when resources are added, from $\mathbf{A}$ to $\mathbf{A}'$. This is no surprise in fact, since one cannot independently add *shared* resources to different contracts, and at the same time expect to be able to develop independently.

Of course, from a theoretical standpoint, there is an easy solution to this problem. One could argue that not considering resources and budgeting them from the very beginning is a mistake and cannot work. Following this argument we would need to consider resources already in the top-level contracts, and address budgeting right from the beginning. Unfortunately, this is in total disagreement with the AUTOSAR approach, which advocates at early stages the specification of software architectures consisting of software components, regardless of resources.

To overcome this difficulty, our approach is: 1) to precisely characterize the "illegal" development steps we perform that violate contract refinement, and 2) to precisely identify the resulting risks for later system integration. To this end we will use the weaker notion of *port-refinement*, for concrete contracts. Decompose the alphabet $\mathcal{V}$ introduced in Definition 4:

$$\mathcal{V} \;=\; (\,\{0,1\}^{\mathcal{P}}\,) \times (\,\textstyle\prod_{r\in\mathcal{R}} \Sigma_r\,) \;=\; \mathcal{V}^{\mathcal{P}} \times \mathcal{V}^{\mathcal{R}}$$

For $\mathbf{M} = ((T, R, \rho), L)$ a concrete scheduling component, define

$$[\![\mathbf{M}]\!]^{\mathcal{P}} \;=_{\text{def}}\; \mathbf{pr}_{\mathcal{V}^{\mathcal{P}}}\Big([\![\mathbf{M}_{\rho/\epsilon}]\!]^{\mathsf{A}}\Big), \text{ where } \mathbf{M}_{\rho/\epsilon} =_{\text{def}} ((T, \emptyset, \epsilon), L),$$

and ϵ is the allocation map with empty domain. In words, we first ignore the possible conflicts due to shared resources (replacing $\mathbf{M}$ by $\mathbf{M}_{\rho/\epsilon}$), we then take the abstract semantics $[\![\mathbf{M}_{\rho/\epsilon}]\!]^{\mathsf{A}}$, and we finally project the resulting abstract semantics over the ports only (taking $\mathbf{pr}_{\mathcal{V}^{\mathcal{P}}}(...)$). $[\![\mathbf{M}]\!]^{\mathcal{P}}$ captures the scheduling aspect of $\mathbf{M}$ while discarding the resource aspect of it. Observe that $[\![\mathbf{M}]\!]^{\mathcal{P}}$ contains the language obtained by projecting $[\![\mathbf{M}]\!]^{\mathsf{A}}$ over the ports; this inclusion is generally strict. For $\mathbf{C} = (\mathbf{A}, \mathbf{G})$ a concrete scheduling contract, define

$$[\![\mathbf{C}]\!]^{\mathcal{P}} =_{\text{def}} ([\![\mathbf{A}]\!]^{\mathcal{P}}, [\![\mathbf{G}]\!]^{\mathcal{P}}) \text{ the } \textit{port-contract} \text{ associated with } \mathbf{C}.$$

Despite the boldface notation used, port-contracts are abstract contracts. For $\mathbf{C}$ and $\mathbf{C}'$ two concrete contracts, say that

$$\mathbf{C}' \textit{ port-refines } \mathbf{C}, \text{ written } \mathbf{C}' \preceq_{\mathcal{P}} \mathbf{C} \quad \text{if} \quad [\![\mathbf{C}']\!]^{\mathcal{P}} \leq [\![\mathbf{C}]\!]^{\mathcal{P}}.$$

We will restrict the illegal steps of Process 1 to the following situation, which does not contradict the AUTOSAR methodology. Assume, from early design stages on, prior knowledge of the following property about a given set T of tasks—this does not require detailed knowledge of the computing resources:

Definition 8 (T-closed contracts). *Say that a set $T \subset \mathcal{T}$ of tasks is* segregated *if the set $\mathcal{R}$ of all resources partitions as follows: $\mathcal{R} = R \cup \overline{R}, R \cap \overline{R} = \emptyset$, and $T \subseteq \Sigma_R$ whereas $\mathcal{T} - T \subseteq \Sigma_{\overline{R}}$. For any segregated set of tasks T, say that concrete contract $\mathbf{C} = (\mathbf{A}, \mathbf{G})$ is* T-closed *if $T_{\mathbf{G}} \subseteq T$ and $T_{\mathbf{A}} \cap T = \emptyset$.*

If $\mathbf{C} = (\mathbf{A}, \mathbf{G})$ is T-closed, then $\rho_{\mathbf{G}}(T_{\mathbf{G}}) \cap \rho_{\mathbf{A}}(T_{\mathbf{A}}) = \emptyset$ holds. *Illegal steps* are performed on T-closed contracts only. An illegal step consists in replacing T-closed contract $\mathbf{C}$ by another T-closed contract $\mathbf{C}'$ port-refining it: $\mathbf{C}' \preceq_{\mathcal{P}} \mathbf{C}$.

Port-refinement being not a refinement, replacing $\mathbf{C}$ by $\mathbf{C}'$ *won't* ensure that any implementation of $\mathbf{C}'$ meets the guarantees of $\mathbf{C}$ under any legal environment for $\mathbf{C}'$—it should ensure this if it was a true refinement. Still, the following result holds, which precisely bounds the risks at system integration time:

Lemma 7. *Let be $\mathbf{C}' \preceq_{\mathcal{P}} \mathbf{C}$ satisfying the following conditions: $\mathbf{C}$ and $\mathbf{C}'$ are T-closed for a same segregated set T of tasks, $\mathbf{C}'$ is schedulable, $[\![\mathbf{A}']\!]^{\mathcal{P}} = [\![\mathbf{A}]\!]^{\mathcal{P}}$, $\mathbf{A}$ and $\mathbf{A}'$ both have their tasks pairwise non-conflicting, and $\mathbf{G}$ is resourceless. Then: $\emptyset \neq \mathbf{A} \underline{\times} \mathbf{G}' \sqsubseteq \mathbf{G}$.*

Lemma 7 expresses that $\mathbf{G}'$ is an implementation of $\mathbf{C}'$ that, when put in the context of the most permissive environment of $\mathbf{C}$, meets the guarantee $\mathbf{G}$ and is schedulable. That $\mathbf{G}$ is met will remain valid for any legal environment of $\mathbf{C}$ and any implementation of $\mathbf{C}'$. Schedulability, however, is only ensured by the most permissive environment of $\mathbf{C}$ and implementation of $\mathbf{C}'$. This restriction is not surprising since schedulability is a liveness property whereas A/G-contracts support only safety properties.

We are now ready to explain how the AUTOSAR development process (Process 1) can be made safe by implementing the illegal development steps safely.

Process 2 (AUTOSAR development process made safe) We assume a segregated subset T of tasks.

1. Start with a top-level, T-closed, contract $\mathbf{C}_{\text{top}}^{\text{func}} = (A_{\text{top}}, G_{\text{top}})$. At this level, only functions are considered while computing resources are ignored. Functions are abstracted as systems of tasks with their precedence constraints. The top-level contract may be the conjunction of several viewpoints, and/or it may be specified by means of requirement tables.
   - *Comment*: No change with respect to Process 1 besides T-closedness.

2. To prepare for subcontracting to different suppliers, decompose the above functional contract $\mathbf{C}_{\text{top}}$ into functional, resource agnostic, sub-contracts in such a way that

$$\mathbf{C}_{\text{ref}}^{\text{func}} = (\mathbf{A}_{\text{ref}}, \mathbf{G}_{\text{ref}}) = \ \leq_{i\in I} \mathbf{C}_i \ \text{ satisfies } \begin{cases} \mathbf{A}_{\text{top}} \leq \mathbf{G}_{\text{ref}} \sqsubseteq \mathbf{G}_{\text{top}} \\ \mathbf{A}_{\text{ref}} \sqsupseteq \mathbf{A}_{\text{top}} \end{cases} \tag{5}$$

where the $\mathbf{C}_i$ are T-closed subcontracts for the different suppliers. In addition, we require that $\mathbf{A}_{\text{ref}}$ and $\mathbf{A}_{\text{top}}$ possess identical sets of tasks, i.e., map ψ of Lemma 3 is the identity. By Lemma 6, (5) ensures $\mathscr{C}_{\text{ref}}^{\text{func}} \preceq \mathscr{C}_{\text{top}}^{\text{func}}$. So far resources were not considered.

   – *Comment*: No change so far, with respect to Process 1, besides naming contracts and making refinement step precise through (5). The first two steps make no reference to semantics, meaning that no scheduling analysis is required, cf. Comment 1. From the next step on, this process deviates from Process 1.

3. At this step the computing resources are now taken into account. Allocate a resource to each task of $\mathbf{A}_{\text{ref}}$ and $\mathbf{G}_{\text{ref}}$, in such a way that all tasks of $\mathbf{A}_{\text{ref}}$ are pairwise non-conflicting, see Definition 1. Precedence constraints between tasks are not modified. This yields a resource aware T-closed contract $\mathbf{C}_{\text{ref}}^{\text{res}}$ such that

$$\mathbf{C}_{\text{ref}}^{\text{res}} = (\mathbf{A}_{\text{ref}}^{\text{res}}, \mathbf{G}_{\text{ref}}^{\text{res}}) \ \preceq_{\mathcal{P}} \ \mathbf{C}_{\text{ref}}^{\text{func}} \tag{6}$$

Since $\mathbf{A}_{\text{ref}}^{\text{res}}$ is free of conflict, only $\mathbf{G}_{\text{ref}}^{\text{res}}$ requires a non-trivial scheduling analysis, which result is specified through the semantics $[\![\mathbf{G}_{\text{ref}}^{\text{res}}]\!]$, cf. Comment 1. At this point, resources have been globally budgeted and scheduling analysis globally performed.

   – *Comment*: This is the illegal step, which is protected by Lemma 7.

4. Continue by decomposing contract $\mathbf{C}_{\text{ref}}^{\text{res}}$ into resource aware sub-contracts $\mathbf{C}_i^{\text{res}}$, following the architecture specified in Step 2, in such a way that $\bigotimes_{i\in I} \mathscr{C}_i^{\text{res}} \preceq \mathscr{C}_{\text{ref}}^{\text{res}}$. The results of the next section can be used for this.

4 An example in the context of AUTOSAR

To illustrate the practical use of the framework of scheduling contracts in AUTOSAR, we consider as an example an excerpt of an exterior light management system for an automobile.[6] Regarding modeling methodology and notations, we will be using both concrete contracts (for the specification of contracts at early steps of the design) and abstract contracts (when using the contract algebra). We will use the symbols $\mathbf{C}$ and $\mathscr{C}$ to distinguish between them. The duration of the time slot is 1μs.

Step 1 of Process 2: In this step a view of the Virtual Functional Bus System is created. It shows how the system functions interact regardless of any network topology or deployment across multiple ECUs. Step 1 of Process 2 is performed by considering resourceless contracts for this Virtual Functional Bus (*VFB*) view. All contracts created in this step have to be T-closed to prepare for the later steps where resources are added. In

[6] A case-study from the German SPES2020 project

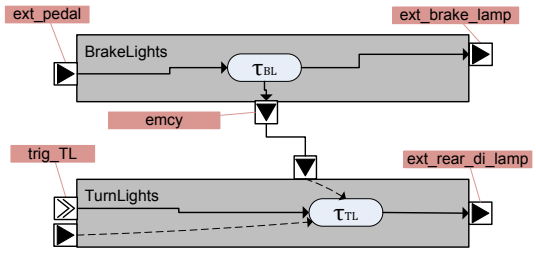

Fig. 3. Virtual Functional Bus (*VFB*) architecture

these contracts, functions provided by software components, are represented by means of tasks with precedence constraints. Typical constraints imposed by the language of contracts would be latency intervals, synchronization of events and event models. Figure 3 shows the VFB architecture of the exterior light management. The system shall control the brake lights in accordance with the driver pressing the brake pedal. The TurnLights component controls the direction indicator lights according to the position of the turn signal lever and the warn lights button. The system shall also implement an emergency stop signal, where warn lights flash in case of severe braking. The graphical notations in Figure 3 distinguishes pure data flows (the dashed lines) from control flows (the solid lines), where the latter may also carry data items.

The languages $L_{\mathbf{A}}$ and $L_{\mathbf{G}}$ of a (concrete) scheduling contract can be specified by means of the AUTOSAR timing extensions [9]. The concept of *observable events* allows to derive sorts of scheduling components, as well as ports of their tasks. Precedence order $\multimap\!\!\!\!\rightarrow$ follows from the interconnection of ports of software components.

To avoid heavy textual notations, in the following we denote by the expression $\delta(\mathsf{X},\mathsf{Y})$ the latency between occurrence of an event at port X and occurrence of an event at port Y. Further $\mathcal{S}(\mathsf{X},T,J)$ denotes a periodic event model for occurrences of events at port X, where T is the period and J is the jitter. We will also write $\mathcal{S}(\mathsf{X},T)$, if $J=0$. We use boolean operators to combine such expressions.

In the case-study there is a maximum allowed latency between brake sensing and activating the brake lights. The same applies to flashing the warn lights in case of an emergency brake situation. The resulting top-level contract for the VFB is as follows:

$$\mathbf{C}_{\mathsf{top}} = (\mathbf{A}_{\mathsf{top}}, \mathbf{G}_{\mathsf{top}}) = \begin{pmatrix} \mathcal{S}(\mathsf{ext\_pedal}, 20ms) \wedge \\ \mathcal{S}(\mathsf{trig\_TL}, 20ms) \end{pmatrix}, \begin{array}{l} \delta(\mathsf{ext\_pedal}, \mathsf{ext\_brake\_lamp}) \leq 25ms \wedge \\ \delta(\mathsf{ext\_pedal}, \mathsf{ext\_rear\_di\_lamp}) \leq 60ms \end{array}$$

$\mathbf{A}_{\mathsf{top}}$ makes explicit an assumption about the frequency of sensor samples of the brake pedal position. These assumptions were not part of the requirements.[7]

Step 2 of Process 2: Assuming that components BrakeLights and TurnLights are implemented by two different suppliers, we propose sub-contracts specifying a time budgeting for them. Thereby a clear assignment of responsibilities to the suppliers is achieved.

[7] It is actually not uncommon that some critical assumptions are implicit in requirements documents, which may, at times, become a problem.

This activity is step 2 of Process 2. In our case two subcontracts are specified:

$$\mathbf{C}_{\mathsf{BL}} = \left(\mathcal{S}(\mathsf{ext\_pedal}, 20ms) \quad , \begin{array}{l} \delta(\mathsf{ext\_pedal}, \mathsf{ext\_brake\_lamp}) \leq 25ms \wedge \\ \delta(\mathsf{ext\_pedal}, \mathsf{emcy}) \leq 5ms \end{array} \right)$$

$$\mathbf{C}_{\mathsf{TL}} = \left(\begin{array}{l} \mathcal{S}(\mathsf{emcy}, 20ms, 5ms) \wedge \\ \mathcal{S}(\mathsf{trig\_TL}, 20ms) \end{array} \quad , \delta(\mathsf{emcy}, \mathsf{ext\_rear\_di\_lamp}) \leq 50ms \right)$$

These contracts are still resourceless. To ensure that $\mathbf{C}_{\mathsf{BL}}$ and $\mathbf{C}_{\mathsf{TL}}$ are correct with respect to the top-level contract, we must prove the refinement

$$\mathscr{C}_{\mathsf{BL}} \otimes \mathscr{C}_{\mathsf{TL}} \quad \preceq \quad \mathscr{C}_{\mathsf{top}}$$

This can be performed by invoking Lemma 6.

Steps 3 and 4 of Process 2: The next step in the AUTOSAR methodology consists in developing system and sub-system views, where the network topology and deployment of software components to ECUs is defined. Resource aware contracts are now considered. A resource is allocated to each task of the contracts defined for the VFB description. Precedence constraints between tasks are not modified. If tasks of the assumption are pair-wise non-conflicting (see Definition 1), this yields a resource aware *T*-closed contract, that port-refines the contract of the VFB.

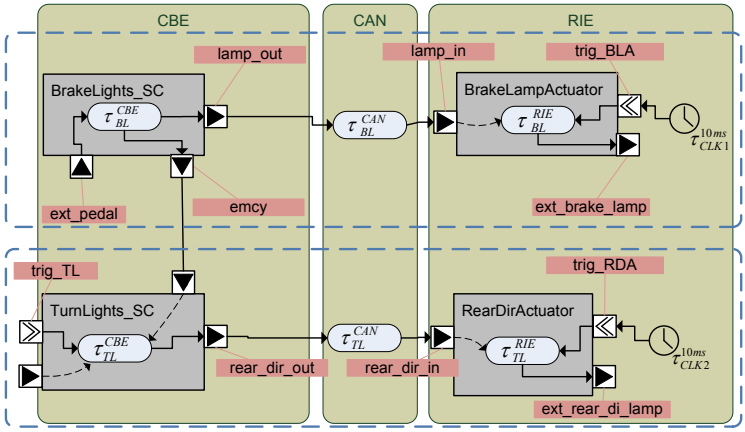

Fig. 4. Deploying *VFB* on computing and communication resources.

For the example, the VFB view is further refined and then deployed as the architecture shown in Figure 4. The blue dashed boxes denote the previous components BrakeLights and TurnLights from the architecture shown on Figure 3. The brown boxes labeled CBE, CAN and RIE, indicate resources allocated to tasks. Deployment is driven by the separation of the sensing and control parts from the actuation part. In addition to allocating resources, execution budgets are specified per task. Mirroring the decomposition of the VFB description, the contract of the system view is the composition of the resource aware sub-contracts $\mathscr{C}_{\mathsf{BL}'}$ and $\mathscr{C}_{\mathsf{TL}'}$. Since contracts created in this step and in previous steps 1 and 2 are *T*-closed, Lemma 7 applies and we can bound the risks for later system integration.

5 Conclusion

We have developed a framework of Assume / Guarantee contracts for schedulability analysis. The methodological step of AUTOSAR suggesting a transition from a *Virtual Function Bus* view, which is independent of the target platform, to a system view where network topology and deployment across ECUs is considered, was particularly challenging. A strict contract based approach offering independent development was not feasible, since task scheduling is a resource allocation problem, which, by essence, can only be solved globally. However, our approach allows to properly bound the development steps of the AUTOSAR methodology that do not comply with the rules of contract based design, while avoiding risks at system integration with a clear and limited additional discipline regarding resource segregation. Within resource segregated subsystems, our contract framework enables compositional reasoning about scheduling of applications distributed over several resources.

References

1. Alur, R., Weiss, G.: Regular Specifications of Resource Requirements for Embedded Control Software. In: Proceedings of the 14th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2008. pp. 159–168. IEEE Computer Society (2008)
2. Alur, R., Weiss, G.: RTComposer: A Framework for Real-Time Components with Scheduling Interfaces. In: Proceedings of the 8th ACM & IEEE International conference on Embedded software, EMSOFT 2008. pp. 159–168. ACM (2008)
3. Anand, M., Fischmeister, S., Lee, I.: A Comparison of Compositional Schedulability Analysis Techniques for Hierarchical Real-Time Systems. ACM Trans. Embedded Comput. Syst. 13(1), 2 (2013)
4. Benveniste, A., Caillaud, B., Ferrari, A., Mangeruca, L., Passerone, R., Sofronis, C.: Multiple Viewpoint Contract-Based Specification and Design. In: Proceedings of the Software Technology Concertation on Formal Methods for Components and Objects, FMCO'07. Lecture Notes in Computer Science, vol. 5382, pp. 200–225. Springer (October 2008)
5. Boudjadar, A., David, A., Kim, J.H., Larsen, K.G., Mikucionis, M., Nyman, U., Skou, A.: Widening the Schedulability of Hierarchical Scheduling Systems. In: Formal Aspects of Component Software - 11th International Symposium, FACS 2014, Bertinoro, Italy, September 10-12, 2014, Revised Selected Papers. pp. 209–227 (2014)
6. Boudjadar, A., Kim, J.H., Larsen, K.G., Nyman, U.: Compositional Schedulability Analysis of An Avionics System Using UPPAAL. In: Proceedings of the 1st International Conference on Advanced Aspects of Software Engineering, ICAASE 2014, Constantine, Algeria, November 2-4, 2014. pp. 140–147 (2014)
7. AUTOSAR consortium: 10 years AUTOSAR. Tech. rep., AUTOSAR(2013), available from http://www.autosar.org/fileadmin/files/events/10yearsautosar/ATZextra_AUTOSAR_-_THE_WORLDWIDE_AUTOMOTIVE_STANDARD_FOR_EE_SYSTEMS.pdf
8. AUTOSAR consortium: Methodology (Release 421, 2014), available from http://www.autosar.org/fileadmin/files/releases/4-2/methodology-templates/methodology/auxiliary/AUTOSAR_TR_Methodology.pdf
9. AUTOSAR consortium: Specification of Timing Extensions (Release 421, 2014), available from http://www.autosar.org/fileadmin/files/releases/4-2/methodology-templates/templates/standard/AUTOSAR_TPS_TimingExtensions.pdf

10. Easwaran, A., Lee, I., Sokolsky, O., Vestal, S.: A Compositional Scheduling Framework for Digital Avionics Systems. In: 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2009, Beijing, China, 24-26 August 2009. pp. 371–380 (2009)
11. Lampka, K., Perathoner, S., Thiele, L.: Component-based system design: analytic real-time interfaces for state-based component implementations. STTT 15(3), 155–170 (2013)
12. Lukasiewycz, M., Schneider, R., Goswami, D., Chakraborty, S.: Modular Scheduling of Distributed Heterogeneous Time-Triggered Automotive Systems. In: Proceedings of the 17th Asia and South Pacific Design Automation Conference, ASP-DAC 2012, Sydney, Australia, January 30 - February 2, 2012. pp. 665–670 (2012)
13. Marimuthu, S.P., Chakraborty, S.: A Framework for Compositional and Hierarchical Real-Time Scheduling. In: 12th IEEE Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2006), 16-18 August 2006, Sydney, Australia. pp. 91–96 (2006)
14. Phan, L.T.X., Lee, J., Easwaran, A., Ramaswamy, V., Chen, S., Lee, I., Sokolsky, O.: CARTS: A Tool for Compositional Analysis of Real-Time Systems. SIGBED Review 8(1), 62–63 (2011)
15. Reinkemeier, P., Stierand, I.: Compositional Timing Analysis of Real-Time Systems Based on Resource Segregation Abstraction. In: Embedded Systems: Design, Analysis and Verification - 4th IFIP TC 10 International Embedded Systems Symposium, IESS 2013, Paderborn, Germany, June 17-19, 2013. Proceedings. pp. 181–192 (2013)
16. Reinkemeier, P., Stierand, I.: Real-Time Contracts - A Contract Theory Considering Resource Supplies and Demands. Reports of SFB/TR 14 AVACS 100, SFB/TR 14 AVACS (July 2014), http://www.avacs.org
17. Shin, I., Lee, I.: Compositional Real-Time Scheduling Framework. In: Proceedings of the 25th IEEE Real-Time Systems Symposium (RTSS 2004), 5-8 December 2004, Lisbon, Portugal. pp. 57–67 (2004)
18. Stierand, I., Reinkemeier, P., Bhaduri, P.: Virtual Integration of Real-Time Systems Based on Resource Segregation Abstraction. In: Formal Modeling and Analysis of Timed Systems - 12th International Conference, FORMATS 2014, Florence, Italy, September 8-10, 2014. Proceedings. pp. 206–221 (2014)
19. Stierand, I., Reinkemeier, P., Gezgin, T., Bhaduri, P.: Real-Time Scheduling Interfaces and Contracts for the Design of Distributed Embedded Systems. In: 8th IEEE International Symposium on Industrial Embedded Systems, SIES 2013, Porto, Portugal, June 19-21, 2013. pp. 130–139 (2013)
20. Stoimenov, N., Chakraborty, S., Thiele, L.: Interface-Based Design of Real-Time Systems. In: Advances in Real-Time Systems (to Georg Färber on the occasion of his appointment as Professor Emeritus at TU München after leading the Lehrstuhl für Realzeit-Computersysteme for 34 illustrious years). pp. 83–101 (2012)
21. Thiele, L., Wandeler, E., Stoimenov, N.: Real-Time Interfaces for Composing Real-Time Systems. In: Proceedings of the 6th ACM & IEEE International conference on Embedded software, EMSOFT 2006, October 22-25, 2006, Seoul, Korea. pp. 34–43 (2006)
22. Wandeler, E., Thiele, L.: Interface-Based Design of Real-Time Systems with Hierarchical Scheduling. In: 12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2006), 4-7 April 2006, San Jose, California, USA. pp. 243–252 (2006)
23. Weiss, G., Alur, R.: Automata Based Interfaces for Control and Scheduling. In: Hybrid Systems: Computation and Control, 10th International Workshop, HSCC 2007. Lecture Notes in Computer Science, vol. 4416, pp. 601–613. Springer (2007)