A Design Framework for End-To-End Timing Constrained Automotive Applications

Friedhelm Stappert, Jan Jonsson, Jürgen Mottok, Rolf Johansson

A Design Framework for End-To-End Timing Constrained Automotive Applications

Friedhelm Stappert*, Jan Jonsson†, Jürgen Mottok‡, and Rolf Johansson§

*Continental, Systems and Technology Automotive, Regensburg, Germany
†Chalmers University of Technology, Göteborg, Sweden
‡Regensburg University of Applied Sciences, Germany
§Mentor Graphics, Göteborg, Sweden

I. Introduction

In modern cars more and more algorithms are implemented as distributed systems. For example, an ACC-System (*Adaptive Cruise Control*) today requires a minimum of 5 ECUs (*Electronic Control Units*): Engine ECU, Gearbox ECU, Breaking ECU, the MMI-Interface, and an ECU operating the radar system. Mastering the overall timing behaviour of such a distributed system is a fundamental challenge during design. The so-called *end-to-end timing* from a sensor to an actuator must meet a certain deadline, also claimed by functional safety regulations like IEC 61508 and ISO DIS 26262. In order to fulfil such requirements, the timing on the bus, the ECU-timing, and the timing of the communication controller have to be taken into account.

Control engineering and body electronics are two important domains in automotive systems. Both domains use multi-rate functions and rely on correct end-to-end timing, but they essentially differ in the meaning of end-to-end delays. Control systems that continuously drive external actuators shall ensure that these driving signals do not exceed a maximum age. 'Data age' is a concept in the heart of control engineering theory. Clearly, if the same signal is consumed twice, the second consumption is critical because the (unchanged) signal at the time of the second consumption is older. In body electronics, the situation can be very different. In a door lock system, the first arriving signal will command the consuming device to lock the door. Any later signal duplicate can not lock the door 'more'. This shows that there exist at least two different semantics of end-to-end timing. In addition, constraining timing is not always about delays between stimuli and responses. An important class of constraints deals with the synchronization between either stimuli or responses, respectively. Referring again to the door lock system, the reaction time between button pressed (stimuli) and door locked (response) could typically have a span between fastest and slowest reaction of several hundreds of milliseconds. However, the tolerated difference between when the different doors are locked is perhaps just some tens of milliseconds.

There is, consequently, a need for classification of the semantics of end-to-end timing constraints in terms of the treatment of duplicate data and the synchronization of input or output data. Also, when applications are composed of different subsystems, it is important to know how the effects of duplicated or purged signal data propagate over subsystem interfaces and what the net effect of them are on the application itself.

The main goal of the TIMMO[1] project [1] is to define a predictable development process that is able to handle timing in all design phases and able to verify as well as validate the timing behaviour of a real-time system throughout the process.

II. Project Context

The increasing importance and complexity of electronic functions in the electric/electronic (E/E) systems in modern cars often requires the integration of widely different subsystems like powertrain, chassis, and infotainment. Furthermore, multiple suppliers are involved in the design of such systems, which requires the need for unambiguous exchange of engineering data among all involved parties. In response to this trend, the automotive industry is establishing standards that support the interoperability among the different stakeholders. AUTOSAR (*'AUTomotive Open System ARchitecture'*) [2] is the most important project in this area. It is a development partnership for automotive electronics, with the purpose of developing an open industry standard for automotive software architectures. Furthermore, AUTOSAR defines a methodology which supports the distributed development of these electronics systems.

However, the scope of the AUTOSAR standardization does not cover all aspects of automotive E/E system design. Things like requirements engineering and feature modeling are outside the scope. Also things like abstract architectures for identifying commonalities in the realization of different features are outside the scope of AUTOSAR. To get a more complete modeling framework for automotive E/E system design, the EC funded research projects ATESST and ATESST2 [3] have developed an architecture description language called EAST-ADL2, which complements AUTOSAR.

[1]This document is based on the TIMMO project in the framework of the ITEA2, EUREKA cluster programme Σ! 3674. The work has been funded by the German Ministry for Education and Research (BMBF) under the funding IDs 01IS07002(A-K), and the Swedish Agency for Innovation Systems (VINNOVA) under the funding IDs P30619-1/2. The responsibility for the content rests with the authors.

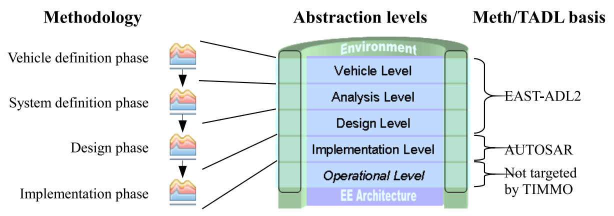

Fig. 1. Phases of the methodology and their relation to abstraction levels

The objective of the EAST-ADL2 is to define an information model capturing all relevant engineering information in a standardized way. It defines several abstraction levels (as shown in Figure 1 that reflect different views and details of the architecture and implicitly different phases of an engineering process.

By means of these abstraction levels, an automotive system is modelled from a very abstract 'feature model' down to the concrete implementation in hardware and software. The AUTOSAR architecture covers the two lower abstraction levels, i.e. implementation and operational level.

The TIMMO language and methodology define timing information on all abstraction levels of the EAST-ADL2 architecture, thereby extending it with a formal timing model, which is also compatible with AUTOSAR. Therefore, it is now possible to model timing information on all abstraction levels of the AUTOSAR and EAST-ADL2 architectures.

III. The TIMMO Design Framework

The first technological challenge of TIMMO was to develop a common, standardised approach to handle timing constraints throughout the entire development process. The objective was to find a solid theoretical base for addressing the timing aspects of distributed embedded automotive systems. This also includes the definition of a common specification language covering semantically sound timing aspects. The second main challenge was to develop an accompanying methodology which explicitly includes cross-company exchange aspects.

A major result of the TIMMO project is the timing augmented description language (TADL), which allows expressing timing information, such as requirements and properties, of automotive electronic systems. TADL is specified as a UML metamodel and extends corresponding EAST-ADL2 and AUTOSAR metamodels, with clear references to relevant elements in the extended models. In order to facilitate the language integration with AUTOSAR and EAST-ADL2, the TADL Domain Model is done according the AUTOSAR Meta Modeling Guidelines. This means that the AUTOSAR Template Profile, the so-called atp stereotypes, is used in the definition of the domain model. As also the EAST-ADL2 domain model is done according to this guideline, all these three languages are built on the same meta meta model thus facilitating a smooth integration.

On the higher abstraction levels (vehicle, analysis, and design) structural modeling is performed as defined within EAST-ADL2. On the implementation level AUTOSAR modeling is performed. The TADL constraints are defined separately from the structural modeling and refer to structural elements via Events and Event Chains. The events are tailored to refer to specific elements in the structural model, i.e., different sets of events are available for the EAST-ADL model and the AUTOSAR model. Additionally, all TADL constraints carry a timing bound and a mode.

However, in order to boost its effectiveness and, in particular, in order to facilitate communicating and exchanging timing information between two or more contracting parties, it is necessary to put TADL into a development context. This is the purpose of the methodology. The methodology is based on a development process induced by EAST-ADL2 and AUTOSAR, where timing issues are particularly highlighted and emphasized. Other, non-timing related, issues are described with less detail.

The issue of unambiguous communication between parties becomes quite evident in the light of component-based development. In early phases of the development process, the vehicle manufacturer partitions the system (vehicle) into subsystems. Each subsystem is, in the general case, developed by a different supplier. Afterwards, the subsystems have to be integrated into the full system again. Such distribution of work needs to be thoroughly specified and with little room for alternative interpretations, in particular with respect to timing requirements and constraints. The purpose of the methodology in this context is to state what timing information could be expected at which point in the development process. It answers questions like what information is needed to perform a certain analysis, like timing budgeting, and where does it come from. This relates to the issue of traceability. With this mechanism it becomes clear on which other types of timing information the information currently under discussion depends. Such dependency information could moreover prove useful when two parties negotiate about which information to share.

On the way down to the implementation, the software components are typically first designed by control engineers who use tools such as Matlab/Simulink or ASCET. They define 'function blocks' that are later realized as AUTOSAR 'runnables' (by software developers) and distributed to tasks that are finally put under operating system control (by the ECU integrators). If a software component only maps to one 'code block' or runnable, it can be mapped to a single task. Software components, whose realization is distributed to several runnables with different execution rates (also defined by the function developer), will most likely be mapped to several tasks with different periods. This consequently leads to under- and over-sampling in the communication among tasks and between tasks and communication frames with several important implications for the meaning of end-to-end timing (as shown in the next section).

The TIMMO methodology is being developed based on the following four principles:

- Focus on timing

- Align with EAST-ADL2 and AUTOSAR
- Assume a system being built from scratch
- Ideal process flow

The methodology focuses on timing by making everything timing related explicit. This timing information often, if not always, refers to one or more EAST-ADL2 or AUTOSAR entities. If such a dependency has been identified, it has been incorporated in the methodology, otherwise it has been left out or hidden in other activities or work products. The aim has been to find a smallest 'timing related closure'. As TADL itself interfaces the de-facto standards EAST-ADL2 and AUTOSAR, and thereby inherits most of their properties, there is a natural implication to apply this alignment also to the methodology. Systems are developed according to a vast number of different scenarios depending on the current situation. It would not be feasible to cover all of them. The methodology therefore focuses on the scenario where a vehicle is built completely from scratch. Other scenarios can, to a large extent, be seen as being partial specializations or variants of this scenario. Real-life development is normally full of iterations. They can occur from virtually any activity to any other activity. Including every such occurrence would inevitably hide the main process flow. Therefore, only iterations that constitute a natural part of an idealistic fault-free development are included. Re-iterations due to a faulty design are left implicit.

A. Methodology Overview

The TIMMO methodology consists of four main phases: vehicle definition, system definition, design, and implementation. These map one to one with the abstraction levels of EAST-ADL2, as depicted in Figure 1. The first three phases are based on the implicit methodology induced by the main work products of EAST-ADL2: vehicle feature model (VFM), functional analysis architecture (FAA) and functional design architecture (FDA). The implementation phase is, on the other hand, based on the AUTOSAR methodology. The following paragraphs summarize the content of each phase, and exemplify its main deliverables on an ABS system (anti-lock brake system) with focus on the handling and refinement of an end-to-end timing requirement.

B. Vehicle definition phase

The vehicle definition phase defines interacting features and use cases, and high-level timing requirements on these. The ABS system is itself a feature, and has certain interactions with the speedometer feature. Timing requirements in this phase are mostly concerned with end-to-end delays and synchronisation. A few examples are listed below:

- End-to-end delay: The vehicle must start decelerating within 'driver's reaction time' (500ms) after the driver has indicated his wish to do so.
- End-to-end delay: The wheels may never lock for a 'continuous period of time' ($>$ 100ms).

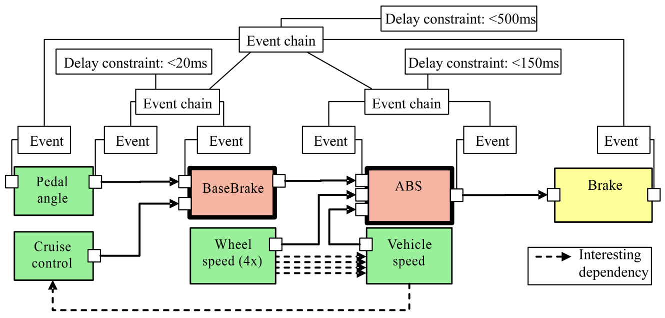

Fig. 2. End-to-end-requirements on a functional architecture of an ABS

- Inter-feature end-to-end delay: Data from the speedometer feature must reach the ABS 'instantly' (within 50ms).
- Synchronisation: The speed information coming from different wheels must arrive 'simultaneously' (difference $<$ 50ms) to the ABS.

The requirements are intentionally made fuzzy, and written from an outside perspective. These will be refined and made more accurate in later phases. If a more accurate deadline is stated in the system specification, or if the development team can provide a precise value based on experience, that value might be stated in a refined version of the vehicle level requirements with traceability to the original requirement. At this high level of abstraction, there is no infrastructure to perform any deep analysis. However, the methodology indicates that timing requirements could be validated with respect to feasibility.

C. System definition phase

The features are then refined in the system definition phase into analysis functions. These do not need to be a strict refinement of the features, but one function may realise parts of several different features. Figure 2 presents the part of the vehicle system related to the ABS. For the sake of the example, we say that the Base brake function produces a message advocating a brake force linear to the pedal angle, or to a similar input message received from, e.g., the cruise control system. The linear brake message can then be modified by the ABS function depending on the values of the wheel speeds and the estimated vehicle speed. The resulting brake force message is forwarded to the brake actuator. It should be noted that there are a few interesting data dependencies outside the scope of the ABS, which could influence overall system timing behaviour and which should be taken into account when analysing the system. The cruise control function is clearly dependent on speed information, in order to compute an appropriate engine torque, or ask for deceleration from the braking system. The vehicle speed is in turn computed from the four individual wheel speed values.

End-to-end delays on the feature model can in this phase be budgeted with respect to the function structure. An example of a budgeted end-to-end requirement is also shown in Figure 2, based on how such requirements are represented in the TADL metamodel. The overall requirement (500ms) refers to an event chain, which in turn refers to one ’stimulus’ event (left in the figure) and one ’response’ event (right in the figure). Each event is associated with a port on a function. Though not explicitly stated in the figure, the events in this example refer to a value being changed on the associated port. The overall requirement shall consequently be read as ’from change of pedal angle until the brake force changes, there shall at most elapse 500ms’. This overall event chain is then divided into segments in the following budgeting process. The segments are recursively also event chains. Two segments are shown in the figure: one related to communication between functions (left hand side), and one related to the execution time of a function (right hand side). Events and event chain segments can be shared between several high-level event chains. If draft high-level behavioural models (e.g. Matlab/Simulink or ASCET) are provided, the budgeting of segments can be based an early assessments of communication and execution times.

In general, the following timing information is dealt with in the system definition phase:

- End-to-end delay: From change of pedal angle until the brake force changes, there shall at most elapse 500ms.
- Timing budgets for event chain segments.
- Synchronisation: The speed information coming from different wheels must arrive to the ABS function with a maximal difference of 50ms between the first and last messages.

Given that high-level behavioural models are available, early control performance analysis with respect to timing can be performed. By doing this, many control related mistakes will be caught early in the development process. Ergonomic aspects related to timing can also be assessed at this stage. This assessment focuses on how the user/driver perceives the system. In our case, 500ms between pressing the pedal until the brakes start actuating might be perceived too long resulting in annoyance and repeated pedal pressing. It might neither be accepted from safety point of view. However, for the sake of the example, we retain this deadline.

D. Design Phase

The design phase refines the analysis functions into design functions. The phase also defines hardware and communication architectures onto which the design functions are mapped. The level of detail of the functions is moreover higher than in the system phase, and embryos of hardware drivers emerge in the form of ’Local device managers’.

The design phase typically contains the same type of timing requirements as in the system definition phase, but with a higher degree of accuracy based on the more detailed models. This refined level allows for making good estimations and simulations on the timing performance, such as CPU utilisation and bus load. Other analysis that can be performed in this phase are early response time analysis and schedulability analysis. All these high-level estimates provide a good foundation for finding a reasonable system configuration in the implementation phase, thus avoiding many unnecessary iterations.

E. Implementation phase

The implementation phase provides all low-level details to the design, as specified by AUTOSAR. The design functions are further refined into software components. Detailed timing properties such as task periods and bus frame intervals become evident. Analysis performed in this phase have the potential of being extremely accurate as all interfering details can be modelled. The implementation phase is subdivided into the following four sub-phases (called ”views” in AUTOSAR) which represent the main activities to do when implementing a system.

- VFB View, where SWCs are interconnected over a virtual bus. It focuses on the communication patterns, and abstracts away all hardware related aspects.
- System view, where the network is defined, the software components are distributed to the ECUs, and the ECUs are configured.
- Component view, where the software components with their runnables are implemented.
- ECU-View, where the different ECU details (operating system, runtime environment, COM-stack etc.) are configured and the ECU executable is generated.

We will now take a closer look at the system view, considering the ABS example (Figure 2) and its end-to-end delay property.

The implementation sub-phase consists of two major development paths, which represent the network- and the ECU-development. On the network development path, the communication matrix or, in case of a FlexRay bus, the bus schedule is defined. The development of the bus timing is restricted by several constraints from different points of the methodology, from the system view or other views of the implementation phase. The ECU-development path, on the other hand, leads over to the ECU view of the implementation phase. The development steps on this path have also influence as well as they are influenced by the network development path. The system view of the implementation phase describes the different ECUs in the system and the SWC mapping to them and in the end, together with the communication matrix, the overall system architecture. Timing analysis can be performed at the end of each path and after the join of the paths. In the paths timing load is analysed with respect to communication and computation resources respectively, whereas the third analysis focuses on the mapped system with respect to schedulability and simulation. Applying the ABS example to the implementation phase, this leads, amongst other things, to a refinement of the event chain.

Taking a look at the overall system architecture, a distribution of the different SWCs to hardware entities, namely ECUs, sensors, actuators, and buses is necessary. For example, there may be one ECU for the sensor functionality in the Pedal SWC and the base braking system. This ECU is connected to a second one, which contains the ABS functionality and the SWC for the brake actuator. The connection of the two ECUs is realized over a bus-system, e.g. FlexRay. The sensor and actuator are directly connected to the ECUs.

Models like this in this process are annotated with timing information, which typically include:

- Periods, release times, offsets and deadlines of runnables and tasks: Task T1 of the Base-Brake SWC shall run once every 50ms and must be finished 30ms after its release.
- Sampling rates: The pedal angle shall be sampled once every 10ms.
- Bus configuration: The bus shall have a cycle length of 100ms.

As mentioned previously, this information allows for, for instance, detailed response time analysis, detailed schedulability analysis of both bus frames, tasks and load analysis of both ECU and bus, as well as the overall system performance. The end-to-end requirement from the design phase has been refined in the implementation phase with respect to the event chain from sensor to actuator. However, the situation in the implementation phase is not as straight forward as in the earlier phases with respect to analysis, due to the fragmentation into task periods and bus cycles and slots. Often, continuous real world entities such as the pedal angle changing over time have been discretised into sampled messages with a certain frequency. As different parts of the system may have been developed by different suppliers, there is a risk of ending up with a situation where periods of messages and receiving tasks do not match, resulting in either over- or under-sampling. If, for instance, the pedal angle sensor has a sample rate of 50ms, but at the same time it is not possible to have a repetition rate of 50ms on the bus, a decision has to be taken, whether to send duplicate sensor data or to sometimes miss sending data. This gives rise to semantical problems related to end-to-end timing.

IV. Timing Semantics

We now introduce the corner stones of the TADL semantics[2]: events, repetition-rate constraints, delay and synchronization constraints, and delay composition and decomposition.

Because TADL constraints are allowed to refer to arbitrary EAST-ADL or AUTOSAR structural models, a clear separation needs to be made between the semantics of these constraints and the semantics of the structural models. The overall goal of TADL is to allow the timing behavior of a structural model to be predicted and checked against the TADL constraints by static analysis tools. However, it is only the prediction part of such analyses that requires knowledge of EAST-ADL/AUTOSAR semantics; once a predicted behavior exists, the constraint checking part should be possible to carry out without reference to anything but the TADL constraint semantics.

[2]By the semantics of a constraint we mean the unambiguous interpretation of the conditions that must hold for the constraint to be satisfied.

A note on notation: Syntactic objects like events, constraints and time will be referenced by simple variable names in this chapter; for example, an event e, a constraint c, or a time value t. To denote attributes and associations of such an object we use a uniform functional notation, where for example $jitter(c)$ means the attribute $jitter$ of object c, and $event(c)$ denotes the object reached by following association $event$ from object c. When sequences or other groups of objects are referenced we write expressions like $\langle t_1, t_2, t_3 \rangle$ or $\langle t_1, \ldots, t_n \rangle$ for a sequence of known length, and $\langle t, \ldots \rangle$ for a sequence of unspecified length but beginning with t. We sometimes also need to express intervals of values, for which we use the $[t_1 \;..\; t_2]$ notation.

A. Events

An event is supposed to denote a distinct form of state change in a running system, taking place at distinct points in time called *occurrences* of the event. That is, a running system can be observed by identifying certain forms of state changes to watch for, and for each such observation point, noting the times when changes occur. This notion of observation also applies to a hypothetical predicted run of a system or a system model — from a timing perspective, the only information that needs to be in the output of such a prediction is a sequence of times for each observation point, indicating the times that each event is predicted to occur. To this end, the semantics of TADL only concerns the *identity* of events; that is, the particular form of state change a distinct event refers to. The actual *dynamics* of an event is merely an assumption — for a particular event the semantics cannot be more specific than to assume it occurs / have occurred / is predicted to occur at some unknown times $\langle t_1, t_2, \ldots \rangle$. Instead, the role of the semantics is to show how these assumptions must relate to each other for a particular set of timing constraints to be satisfied.

Identity: We define a semantic function $identity()$ on events, that associates every event with an identifying list of elements from the underlying structural model.

Equality: Two events e_1 and e_2 are equal $(e_1 = e_2)$ if and only if

$$identity(e_1) = identity(e_2)$$

That is, two events are equal only if they are identified by exactly the same list of structural elements; it is generally not enough that they have just some structural elements in common.

Dynamics: We then capture the assumed dynamic behavior of structural elements by means of a semantic function $dynamics()$, which associates every list of structural elements that is an event identity according to the definition above, with a sequence of time values. That is, if $identity(e) = s$ for some event e, then we assume

$$dynamics(s) = \langle t_1, \ldots \rangle$$

where $\langle t_1, \ldots \rangle$ is a sequence of strictly increasing time values. Finally, we define the short-hand notation $dynamics(e)$ for an event e as

$$dynamics(e) = dynamics(identity(e))$$

B. Repetition-Rate Constraints

All repetition rate constraints place restrictions on the distribution of the occurrences of an event, forming a so-called event model. TADL includes the two most common examples of such models: the *periodic* event, which prescribes an even distribution of occurrences, and the *sporadic* model, whose distribution in non-deterministic but characterized by a minimum distance between occurrences (thus reducing to the periodic model in the worst case). To these TADL also adds two more elaborate forms: a *pattern* model defined by a recurring sequence of offsets between occurrences, and a model termed *arbitrary* because of its ability to put arbitrary statistical restrictions on event distributions.

However, the underlying TADL foundation for these event models is the *generic* repetition rate constraint, which allows the other forms of constraints to be defined as special cases — either by making certain attribute choices or by combining constraints in a certain way.

Syntax: A generic repetition rate constraint c is syntactically characterized by the following attributes and associations:

$event(c)$	the event that is constrained
$lower(c)$	a lower bound on the distance between occurrences
$upper(c)$	an upper bound on the distance between occurrences
$jitter(c)$	the deviation from an ideal point accepted at each occurrence
$span(c)$	a count indicating whether it is subsequent occurrences or occurrences farther apart that are constrained

All attributes except for $event$ are optional, where absence of an attribute means that the following default values apply:

$lower$	default: 0	$upper$	default: ∞
$jitter$	default: 0	$span$	default: 1

Semantics: A generic repetition constraint c is satisfied for some given event behavior $dynamics()$ if and only if

```
given $\langle t_1, t_2, \ldots \rangle = dynamics(event(c))$
there are times $\langle x_1, x_2, \ldots \rangle$ such that
for all $i > 1$,
    $x_i \leq t_i \leq x_i + jitter(c)$
and for all $i > span(c)$,
    $lower(c) \leq x_i - x_{i-span(c)} \leq upper(c)$
```

That is, a generic repetition rate constraint is satisfied if there exists ideal points in time not too far from the actual event occurrences, that are also sufficiently close (distant) according to the upper (lower) constraint attributes. Note especially that for the default span value of 1, it is any two subsequent occurrences of an event that are constrained, whereas for a higher span count, the constraint instead specifies the allowed ranges containing a larger number of occurrences.

Verification procedure: To decide whether a particular event behavior $dynamics()$ satisfies a generic repetition rate constraint c, execute the following pseudo-code program:

```
let $\langle t_1, t_2, \ldots \rangle = dynamics(event(c))$
for all $1 \leq i \leq span(c)$
    let $a_i = t_i - jitter(c)$
    let $b_i = t_i$
for all $i > span(c)$
    let $a_i = \max(t_i - jitter(c), a_{i-span(c)} + lower(c))$
    let $b_i = \min(t_i, b_{i-span(c)} + upper(c))$
    check that $a_i \leq b_i$
```

Intuitively, the a_i and b_i here constitute increasingly tight bounds of the ranges in which the unknown time points x_i must be found, taking more and more event history into account as i increases. The algorithm fails as soon as it discovers that a range is empty, in which case there cannot exist any x_i as required by the semantics. Conversely, if the algorithm succeeds, the ideal times x_i are known to exist.

C. Delay Constraints

A delay constraint restricts the occurrences of one group of event to match the occurrences on another. TADL introduces the concept of an *event-chain* for this purpose, which references two groups of events called *stimulus* and *response*. The intuition behind an event-chain is that each event in the stimulus group somehow causes, or at least affects the value of all events in the response group. TADL defines its delay constraint semantics solely in terms of the times at which the stimulus and response events occur, independently of whether there actually exists a causal connection between these events in the structural model.

There are two fundamentally different perspectives on delays expressible in TADL: (a) the *reaction time* constraint, which for each occurrence of a stimulus event requires a response occurrence at some constrained point in the future, and symmetrically the (b) *age* constraint, which looks at each response event occurrence and expects to find a stimulus occurrence at some constrained point in the history.

Syntactically, a delay constraint c is characterized by the following attributes and associations:

$scope(c)$	the constrained event chain
$lower(c)$	a time offset indicating the near edge of a time window
$upper(c)$	a time offset indicating the far edge of a time window

An event chain E, in turn, is characterized by

$stimulus(E)$	the group of events acting as stimuli
$response(E)$	the group of events acting as responses
$segment(E)$	a sequence of event-chains refining E
$strand(E)$	a group of parallel event-chains refining E

A discussion on the latter two associations will be deferred until Section IV-E. Moreover, the stimulus and response of an event-chain will only be used when following a scope association from a delay constraint. We can therefore avoid the explicit mentioning of event-chains in our semantic definitions, by introducing the following notational short-hand of stimulus and response associations for a delay constraint c:

$$stimulus(c) = stimulus(scope(c))$$
$$response(c) = response(scope(c))$$

The reaction time and age constraints are both specializations of a delay, adding no attributes of their own but being defined with two different (but very symmetrical) semantics.

Syntax: A reaction time (or age constraint) c is syntactically characterized by the following attributes and associations (all inherited from the delay constraint syntax, including notational shorthand):

$stimulus(c)$	a group of events acting as stimuli
$response(c)$	a group of events acting as responses
$lower(c)$	a time offset indicating the near edge of a time window
$upper(c)$	a time offset indicating the far edge of a time window

---

Semantics: A reaction time constraint c is satisfied for some given event behavior $dynamics()$ if and only if

for all events s in $stimulus(c)$,
for all times t in $dynamics(s)$,
for all events r in $response(c)$,
there exists at least one time u in $dynamics(r)$ such that
$$t + lower(c) \leq u \leq t + upper(c)$$

Verification procedure: To decide whether a particular event behavior $dynamics()$ satisfies a reaction time constraint c, execute the following pseudo-code program:

```
for all events $s$ in $stimulus(c)$
    for all times $t$ in $dynamics(s)$
        let $T = [t + lower(c) .. t + upper(c)]$
        for all events $r$ in $response(c)$
            let $U = dynamics(r) \cap T$
            check that $U = \langle u, \ldots \rangle$
```

Semantics: An age constraint c is satisfied for some given event behavior $dynamics()$ if and only if

for all events r in $response(c)$,
for all times t in $dynamics(r)$,
for all events s in $stimulus(c)$,
there exists at least one time u in $dynamics(s)$ such that
$$t - upper(c) \leq u \leq t - lower(c)$$

Verification procedure: To decide whether a particular event behavior $dynamics()$ satisfies an age constraint c, execute the following pseudo-code program:

```
for all events $r$ in $response(c)$
    for all times $t$ in $dynamics(r)$
        let $T = [t - upper(c) .. t - lower(c)]$
        for all events $s$ in $stimulus(c)$
            let $U = dynamics(s) \cap T$
            check that $U = \langle u, \ldots \rangle$
```

D. Synchronization Constraints

TADL also defines a *synchronization* constraint as a delay constraint that adds the attribute

$width(c)$	a sliding window size

to those inherited from the delay constraint syntax. The intuition behind a synchronization constraint is that it strengthens the requirements of a delay to also impose a limit on how much the constrained occurrences can differ from one another — typically a range much more narrow than the window in which the individual events are expected to occur. Two specializations are defined by TADL: *output synchronization* strengthens a reaction constraint, and *input synchronization* is semantically a strengthening of an age constraint.

Semantics: An output synchronization constraint c is satisfied for some given event behavior $dynamics()$ if and only if

for all events s in $stimulus(c)$,
for all times t in $dynamics(s)$,
there exists a time x such that
for all events r in $response(c)$,
there exists at exactly one u in $dynamics(r)$ such that
$$t + lower(c) \leq u \leq t + upper(c)$$
and moreover, for this u,
$$x \leq u \leq x + width(c)$$

Verification procedure: To decide whether a particular event behavior $dynamics()$ satisfies an output synchronization constraint c, execute the following pseudo-code program:

```
for all events $s$ in $stimulus(c)$
    for all times $t$ in $dynamics(s)$
        let $T = [t + lower(c) .. t + upper(c)]$
        for all events $r$ in $response(c)$
            let $U = dynamics(r) \cap T$
            check that $U = \langle u \rangle$
            let $a_s = u - width(c)$
            let $b_s = u$
        let $a = \max(\text{all } a_s)$
        let $b = \min(\text{all } b_s)$
        check that $a \leq b$
```

Semantics: An input synchronization constraint c is satisfied for some given event behavior $dynamics()$ if and only if

```
for all events $r$ in $response(c)$,
for all times $t$ in $dynamics(r)$,
there exists a time $x$ such that
for all events $s$ in $stimulus(c)$,
there exists at exactly one $u$ in $dynamics(s)$ such that
    $t - upper(c) \leq u \leq t - lower(c)$
and moreover, for this $u$,
    $x \leq u \leq x + width(c)$
```

Verification procedure: To decide whether a particular event behavior $dynamics()$ satisfies an input synchronization constraint c, execute the following pseudo-code program:

```
for all events $r$ in $response(c)$
    for all times $t$ in $dynamics(r)$
        let $T = [t - upper(c) .. t - lower(c)]$
        for all events $s$ in $stimulus(c)$
            let $U = dynamics(s) \cap T$
            check that $U = \langle u \rangle$
            let $a_s = u - width(c)$
            let $b_s = u$
        let $a = \max(\text{all } a_s)$
        let $b = \min(\text{all } b_s)$
        check that $a \leq b$
```

E. Delay composition and decomposition

A very strong property of TADL is that its delay concept adds up sequentially, so that a set of delay constraints on multiple interconnected event-chains can be replaced by a single constraint describing the end-to-end delay of the composed event-chain; or conversely, an end-to-end delay can be broken down into smaller intervals with preserved overall semantics. Such sequential composition and decomposition of delay constraints is an important activity in the negotiation process between automotive OEMs and their suppliers. For example, suppliers may provide their components as TADL models that are like black boxes from a functional point of view. Such a model characterized by a few timing parameters still allow the OEM to calculate the overall delay imposed by a chain of such components. Or the OEM may divide the timing constraints of a complex functionality into multiple constraints, to be supplied by separate suppliers.

Moreover, TADL supports parallel composition and decomposition of delay constraints. While this notion is intuitively simple (each part in a parallel decomposition is expected to satisfy the prescribed end-to-end delay), it is nevertheless important when a modeler wishes to express several alternative designs in one model, while resting assured that any choice taken will indeed satisfy the given overall timing requirements.

In both the sequential and the parallel case, it is meaningless to mix age and reaction time constraints in a particular constraint composition/decomposition. We therefore only consider composing or decomposing similar timing constraints either just ages or just reaction times and make this an implicit syntactic requirement in the descriptions that follow.

In the general case, it is allowed to constrain the same event-chain with several age constraints and several reaction constraints. An event chain has a number of other event chains each playing the role of a segment. Each of these segments can of course also have as well age constraints as reaction constraints. The above statement of not mixing age and reaction time, relates to whether it is possible to draw any conclusion between the delay constraint of an event chain and the sum of the delay constraints of its segments. This kind of conclusions can only be drawn when all the considered constraints are of the same kind (age or reaction).

Sequential composition and decomposition: A sequence of event chains $\langle E_1, \ldots, E_n \rangle$ is a sequential decomposition of an event chain E, and E is a sequential composition of $\langle E_1, \ldots, E_n \rangle$, if the following conditions hold:

$$stimulus(E) = stimulus(E_1)$$
$$stimulus(E_i) = response(E_{i-1}) \quad \text{for } 1 < i \leq n$$
$$response(E) = response(E_n)$$

The fact that $\langle E_1, \ldots, E_n \rangle$ is a sequential decomposition of E can be explicitly marked in E by setting

$$segment(E) = \langle E_1, \ldots, E_n \rangle$$

A sequence of age (reaction time) constraints $\langle c_1, \ldots, c_n \rangle$ is a sequential decomposition of an age (reaction time) constraint c if $\langle scope(c_1), \ldots, scope(c_n) \rangle$ is a sequential decomposition of $scope(c)$ (and conversely for composition).

A sequence of age (reaction time) constraints $\langle c_1, \ldots, c_n \rangle$ is stronger (accepts no more variations in event dynamics) than its sequential composition c if

$$upper(c_1) + \ldots + upper(c_n) \leq upper(c)$$
$$lower(c_1) + \ldots + lower(c_n) \geq lower(c)$$

An age (reaction time) constraint c is stronger than its sequential decomposition $\langle c_1, \ldots, c_n \rangle$ if

$$upper(c) \leq upper(c_1) + \ldots + upper(c_n)$$

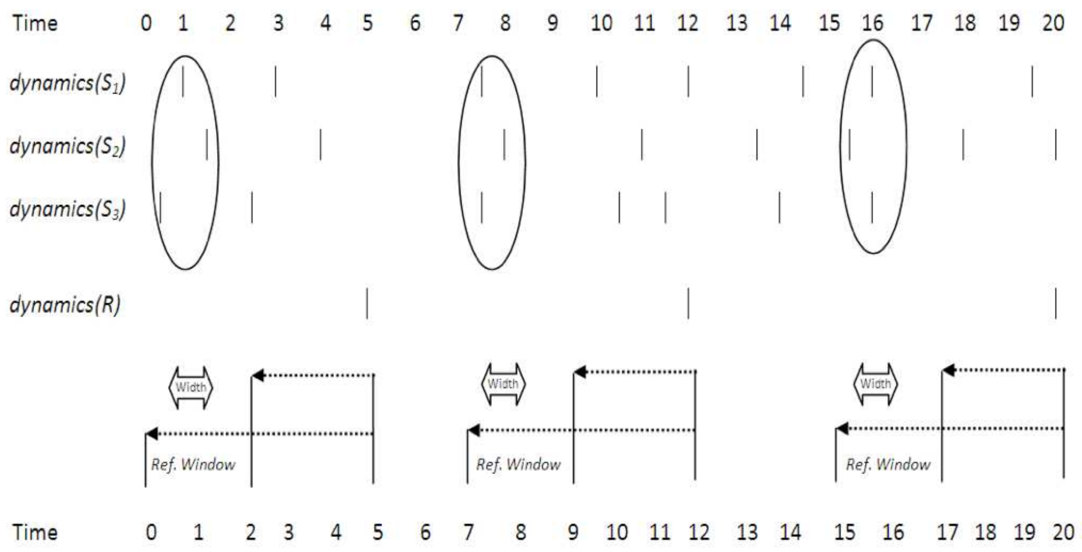

Fig. 3. Example of input synchronization

$$lower(c) \geq lower(c_1) + \ldots + lower(c_n)$$

Taken together, these relations state that an age (reaction time) constraint c and its sequential decomposition $\langle c_1, \ldots, c_n \rangle$ are equivalent (accept the same variations in event dynamics) if

$$upper(c) = upper(c_1) + \ldots + upper(c_n)$$
$$lower(c) = lower(c_1) + \ldots + lower(c_n)$$

Parallel composition and decomposition: A group of event chains $\langle E_1, \ldots, E_n \rangle$ is a parallel decomposition of an event chain E, and E is a parallel composition of $\langle E_1, \ldots, E_n \rangle$, if the following conditions hold:

$$stimulus(E) = stimulus(E_i) \quad \text{for } 1 \leq i \leq n$$
$$response(E) = response(E_i) \quad \text{for } 1 \leq i \leq n$$

The fact that $\langle E_1, \ldots, E_n \rangle$ is a parallel decomposition of E can be explicitly marked in E by setting

$$strand(E) = \langle E_1, \ldots, E_n \rangle$$

A group of age (reaction time) constraints $\langle c_1, \ldots, c_n \rangle$ is a parallel decomposition of an age (reaction time) constraint c if $\langle scope(c_1), \ldots, scope(c_n) \rangle$ is a parallel decomposition of $scope(c)$ (and conversely for composition).

A group of age (reaction time) constraints $\langle c_1, \ldots, c_n \rangle$ is stronger (accepts no more variations in event dynamics) than its parallel composition c if

$$upper(c_i) \leq upper(c) \quad \text{for } 1 \leq i \leq n$$
$$lower(c_i) \geq lower(c) \quad \text{for } 1 \leq i \leq n$$

An age (reaction time) constraint c is stronger than its parallel decomposition $\langle c_1, \ldots, c_n \rangle$ if

$$upper(c) \leq upper(c_i) \quad \text{for } 1 \leq i \leq n$$
$$lower(c) \geq lower(c_i) \quad \text{for } 1 \leq i \leq n$$

Taken together, these relations state that an age (reaction time) constraint c and its parallel decomposition $\langle c_1, \ldots, c_n \rangle$ are equivalent (accept the same variations in event dynamics) if

$$upper(c) = upper(c_i) \quad \text{for } 1 \leq i \leq n$$
$$lower(c) = lower(c_i) \quad \text{for } 1 \leq i \leq n$$

F. Verification example

Consider the input synchronization constraint defined by

$$stimulus(c) = \langle \mathrm{S}_1, \mathrm{S}_2, \mathrm{S}_3 \rangle$$
$$response(c) = \langle \mathrm{R} \rangle$$
$$lower(c) = 3$$
$$upper(c) = 5$$
$$width(c) = 1$$

Assume that the actual occurrences of the events S and R are as follows:

$$dynamics(\mathrm{S}_1) = \langle 1, 3, 7.5, 10, 12, 14.5, 16, 19.5 \rangle$$
$$dynamics(\mathrm{S}_2) = \langle 1.5, 4, 8, 11, 13.5, 15.5, 18, 20 \rangle$$
$$dynamics(\mathrm{S}_3) = \langle 0.5, 2.5, 7.5, 10.5, 11.5, 14, 16 \rangle$$
$$dynamics(\mathrm{R}) = \langle 5, 12, 20 \rangle$$

The actual occurrences of the events in the synchronization and reference groups are given in Figure 3, together with the reference window for each occurrence of the response event R.

In this example, there are three events in $stimulus(c)$ namely event S_1, S_2 and S_3. And there is one event in $response(c)$ namely event R. In Figure 3, for each occurrence of the response event R, there is a reference window backward in time in which exactly each one of the three stimulus events must occur. For the response event R occurred at time instant t, we must have a unique occurrence of each of the stimulus events S_1, S_2 and S_3 within $t - upper(c)$ and $t - lower(c)$. Moreover, the distance between any two of the three stimulus events must not be separated by a distance greater than $width(c) = 1$. There are three occurrences of the event R.

For the response event R that occurs at time $t = 5$, there are unique occurrences of the three stimulus events between time instants $5 - upper(c) = 5 - 5 = 0$ and $5 - lower(c) = 5 - 3 = 2$. The time interval $[0 .. 2]$ is the reference window for the first occurrence of event R. The events S_1, S_2 and S_3 occur uniquely within the reference window at time 1, 1.5, and 0.5, respectively (left-most oval in Figure 3). These occurrences of the synchronization events are within a sliding window of length 1.

For the response event R that occurs at time $t = 12$, there are unique occurrences of the three stimulus events between time instants $12 - upper(c) = 12 - 5 = 7$ and $12 - lower = 12 - 3 = 9$. The time interval $[7 .. 9]$ is the reference window for the third occurrence of event R. The events S_1, S_2 and S_3 occur uniquely within reference window at time 7.5, 8, and 7.5, respectively (middle oval in Figure 3)). These occurrences of the synchronization events are within a sliding window of length 1.

For the response event R that occurs at time $t = 20$, there are unique occurrences of the three stimulus events between time instants $20 - upper(c) = 20 - 5 = 15$ and $20 - lower(c) = 20 - 3 = 17$. The time interval $[15 .. 17]$ is the reference window for the first occurrence of event R. The events S_1, S_2 and S_3 occur uniquely within reference window at time 16, 15.5, and 16, respectively (right-most oval in Figure 3). These occurrences of the synchronization events are within a sliding window of length 1.

Since, for all three occurrences of response event R, there are unique occurrences of each of the three stimulus events within the sliding window which is within the reference window, the input synchronization constraint is satisfied.

V. Related Work

While single-processor scheduling analysis has been studied extensively, end-to-end timing has received much less attention. Task-offset analysis [4], [5], [6] calculates response times of *transactions* comprised of several chained tasks. Richter *et al.* [7] introduced a compositional scheduling model that uses such response-time calculations for local transactions and then characterize the external interaction of these components through event stream models. Chakraborty and Thiele [8] followed another compositional approach based on real-time calculus with local service and arrival curves to calculate delays and backlogs. The work mentioned above focused mostly on FIFO queuing on non-multi-rate systems.

Mangeruca, Beneviste, and others [9], [10] proposed a relaxation that supports multi-rate systems. The proposed mechanisms ensure “semantics preservation” by deliberately constraining the way the consumer must access the FIFO buffers. In a similar way, Matic and Henzinger [11] proposed slight changes to the way task schedules are built during system generation by model-based design tools. Both approaches require that certain platform mechanisms are used in a way that preserves the semantics of the communication. Such restrictions compromise to some extent the broad applicability of the approach.

Gerber et. al. studied multi-rate communication [12] without requiring special semantic preservation mechanisms. The main drawback with this approach is that it requires that task periods can be changed freely, something that is very uncommon in automotive design. Guermazi and George [13] really analyze periodic systems by calculating response times locally for each involved component, and then create the worst-case sequencing of them by composition. Unfortunately, the approach assumes “worst-case asynchronicity” and does not account for offsets, which means that the results are overly pessimistic and therefore of limited relevance in automotive systems.

VI. Conclusions

TADL introduces two distinct forms of delay constraints, age and reaction time, and a very general form of repetition rate constraint that can be specialized to capture the standard periodic/sporadic event models as well as recurring event patterns and arbitrary event distributions. The semantics of the different constraint forms has been formalized, and algorithmic verification procedures have also been defined.

Furthermore, the TIMMO methodology enables the identification of timing information and validation of timing behaviour throughout the development process. Timing constraints and properties can systematically be controlled and validated during all design stages and also across collaborating design teams. This leads to a better understanding of a system’s timing behaviour already at very early phases in the design.

References

[1] *TIMMO – Timing Model*, The TIMMO Consortium, 2008, URL: http://www.timmo.org.

[2] The AUTOSAR Development Partnership, “Automotive Open System Architecture (AUTOSAR),” 2003, URL: http://www.autosar.org.

[3] The ATESST Consortium, “Advancing Traffic Efficiency and Safety through Software Technology,” 2009, URL: http://www.atesst.org.

[4] K. Tindell, “Adding time-offsets to schedulability analysis,” Dept. of Computer Science, Univ. of York, UK, Tech. Rep. YCS 221, 1994.

[5] J. C. Palencia and M. G. Harbour, “Exploiting precedence relations in the schedulability analysis of distributed real-time systems,” in *Proc. of the IEEE Real-Time Systems Symp.*, 1999, pp. 328–399.

[6] O. Redell, “Accounting for precedence constraints in the analysis of tree-shaped transactions in distributed real time systems,” Royal Inst. of Tech., Sweden, Tech. Rep., 2003.

[7] K. Richter, D. Ziegenbein, M. Jersak, and R. Ernst, “Model composition for scheduling analysis in platform design,” in *Proceeding 39th Design Automation Conference*, New Orleans, USA, Jun. 2002.

[8] L. Thiele, S. Chakraborty, and M. Naedele, “Real-time calculus for scheduling hard real-time systems,” in *Proc. Int’l Symp. on Circuits and Systems (ISCAS)*, Geneva, Switzerland, 2000.

[9] L. Mangeruca *et al.*, “Semantics-preserving design of embedded control software from synchronous models,” *IEEE Trans. Software Eng.*, vol. 33, no. 8, pp. 497–509, 2007.

[10] A. Benveniste *et al.*, “Loosely time-triggered architectures based on communication-by-sampling,” in *Proc. of the 7th ACM & IEEE Int’l Conf. on Embedded Software*, New York, USA, 2007, pp. 231–239.

[11] S. Matic and T. A. Henzinger, “Trading end-to-end latency for composability,” in *Proc. of the IEEE Real-Time Systems Symp.*, Washington, DC, USA, 2005, pp. 99–110.

[12] R. Gerber, S. Hong, and M. Saksena, “Guaranteeing end-to-end timing constraints by calibrating intermediate processes,” in *Proc. of the IEEE Real-Time Systems Symp.*, 1994, pp. 192–203.

[13] R. Guermazi and L. George, “Worst case end-to-end response times of periodic tasks with an AUTOSAR/FlexRay infrastructure,” in *7th International Workshop on Real-Time Networks RTN’08*, 2008.