Downloaded from orbit.dtu.dk on: May 08, 2024

DTU Library

Discovering timing feature interactions with timed UML 2 interactions

Störrle, Harald; Knapp, Alexander

Published in:
Proceedings of the 13th Workshop on Model-Driven Engineering, Verification and Validation

Publication date:
2016

Document Version
Publisher's PDF, also known as Version of record

Link back to DTU Orbit

Citation (APA):
Störrle, H., & Knapp, A. (2016). Discovering timing feature interactions with timed UML 2 interactions. In M. Famelis, D. Ratiu, & G. M. K. Selim (Eds.), *Proceedings of the 13th Workshop on Model-Driven Engineering, Verification and Validation* (pp. 10-19). CEUR-WS. http://ceur-ws.org/Vol-1713/

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Discovering Timing Feature Interactions with Timed UML 2 Interactions

Harald Störrle
Danmarks Tekniske Universitet
hsto@dtu.dk

Alexander Knapp
Universität Augsburg
knapp@isse.de

***Abstract*—BACKGROUND: Timing properties are hard to specify, and even harder to analyse as interactions may arise from multiple independent properties.**
OBJECTIVE: We introduce the notion of timing feature interaction (TFI), and show how to automatically detect many TFIs.
METHOD: We identify common structural patterns of timing specifications and show how they can be translated into UML 2 interactions with time constraints. We define a semantics that allows us to define and check coherence and consistency conditions of timing specifications.
RESULTS: We provide a systematic process for mapping timing requirements into timed UML interactions and algorithms for checking their coherence and consistency.
CONCLUSIONS: With our approach, it becomes easier to check and validate timing specifications. It is not our ambition to achieve complete coverage, i.e., discovering all timing specification defects. Instead, we focus on practical specifications that have numerous but comparatively simple properties.

1. Introduction

We have known for a long time that *"the hardest part of the software task is arriving at a complete and consistent specification"* [1]. Similarly, it is plain that *"errors are most frequent during the requirements and design activities and are the more expensive the later they are removed"* ("Boehm's law" [2], [3, p. 17]). Furthermore, beyond being frequent, these errors are also the most difficult (and expensive) to find and fix.[1] For instance, McConnell claims that *"for each requirement that is incorrectly specified, you will pay 50–200 times as much to correct the mistake downstream as you would pay to correct the mistake at requirements time"* [5]. Thus, it is highly relevant and worthwhile to provide methods and tools to detect defects in specifications as early as possible.

A particularly difficult form of requirements defect are so-called features interactions [6], that is, defects that arise out of the *interplay* of two or more features, each of which is correct by itself. It is important to notice that feature interactions are not implementation defects: While the fault arising from the defect will only become apparent during integration testing (or even later), the root cause is the inconsistency of requirements, which is not discovered at design time. Conventionally, only interactions of *functional* properties are considered. However, it is equally possible that defects arise out of the interplay of *timing* properties of different features. We call this phenomenon *Timing Feature Interaction (TFI)*.

In this article, we characterise TFIs, and show how they may arise from sets of straightforward, individually coherent timing property specifications. We also describe a way of detecting TFIs, and provide an efficient algorithm to help doing so. Our solution is implemented prototypically in the HUGO/RT tool [7]. Our approach is based on representing timing properties as UML 2 interactions, equipped with a suitable semantics based on *Difference Bounded Matrices* (DBMs, see [8]).

2. Expressing Timing Properties in UML 2 Interactions

Our starting point is the precise characterization of timing properties as UML 2 interactions [9]. It can be argued that the expressiveness of UML 2 interactions, Message Sequence Charts, and similar notations is somewhat limited: other, more complex formalisms such as Live Sequence Charts [10] have been proposed to express more complex modal properties like may and must scenarios. However, the given level of expressiveness seems to be adequate for practical usage in many domains and for many purposes. After all, interactions are widely used in the software and telecommunication industries, and it is common practice to specify timing requirements, service level agreements, and measurements of actual system performance using sequence and timing diagrams [11, 12]. So, UML 2 interactions seem to be a reasonable approach, striking a good balance between generality and practicality.

UML 2 interactions can be represented in different formats, though sequence and timing diagrams are the most commonly used ones. For better understanding, we generally use sequence diagrams to represent required properties, whereas timing diagrams are used to represent actual behavior. A sample UML 2 interaction is shown by the sequence

1. *"the most pernicious and subtle bugs are system bugs arising from mismatched assumptions made by the authors of various components"* [4, p. 142].

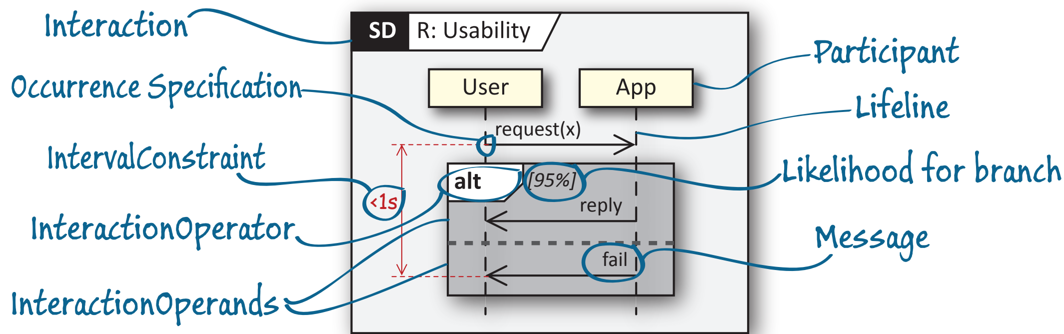


Figure 1. A first example of a UML 2 interaction specifying two timing requirements from a user perspective. Time constraints are highlighted in red, explanations of UML concepts are shown in blue.

diagram presented in Fig. 1. The basic concepts of UML 2 interactions are explained as call-outs. We describe timing constraints on traces of occurrences by visually relating the respective occurrence specifications and attaching a temporal expression.

Consider now an example where a customer uses a mobile app to manage his bank account. In a system development process, we might specify usability requirements of the app in terms of response times, and success rates. From the user's point of view, we might end up with the following requirements.

(R_1) *Server delay (success)*: For successful service calls, the overall delay experienced by a user is supposed to stay below $0.6\,\mathrm{s}$.

(R_2) *Server delay (failure)*: For failing service calls, the overall delay experienced by a user is supposed to stay below $1\,\mathrm{s}$.

(R_3) *Server success rate*: In 95% of the calls of a service, the call will be successful.

This prose description is already much more precise and detailed than what is found in the conventional textual descriptions of many software and system development projects today. Yet, it is obviously not formal enough to allow any kind of automated analysis. As a first step towards making this possible, we express these usability requirements as UML 2 interactions, see Fig. 2. Observe, that there is no straightforward way to express (R_3) in simple interactions. Thus, we are forced to combine all three requirements into one complex interaction using the alt-operator and express (R_3) as a branch probability.

For the sake of our example, let us now assume, that the app interacts with a remote server to provide services to the user. Let us further assume that other requirements are present that address the acceptable handling delay between incoming and outgoing messages at the app, the delay of sending messages between the app and the server, and the processing time spent at the server while executing service requests. These properties might be defined as follows.

(P_1) *Message handling*: The handling delay at the app is below $30\,\mathrm{ms}$, for both directions.

(P_2) *Line delay*: The in-transit and handling delays of sending messages between app and server is specified to stay below $80\,\mathrm{ms}$, for both directions.

(P_3) *Service execution*: The actual execution of a service is specified to take less than $500\,\mathrm{ms}$.

(P_4) *Time-out*: If the service cannot be provided within $850 \pm 75\,\mathrm{ms}$, it aborts.

These properties exist at a lower level of abstraction than the requirements (R_1) to (R_3) defined above as they speak about characteristics of the implementation. In fact, these properties may arise from concrete measurements of existing infrastructure rather than being demanded from a client perspective. A similar situation arises, when considering service level agreements as specifications. Figure 3 expresses (P_1) to (P_4) as sequence diagrams.[2] Observe that these sequence diagrams are *patterns*, that is, they omit some details. For instance, (P_1) does not define a particular message, but a message variable (indicated by a dollar sign prefix and use of italics). So, (P_1) is a family of properties rather than a single property. In contrast, (P_3) specifies particular messages, though the parameter x turns it into a family of traces rather than a single trace.

This example illustrates the set of timing properties that UML 2 interactions allow us to express. In general, there are two kinds of timing properties, as illustrated by Fig. 4: (1) *duration* constraints between two occurrence specifications on the same lifeline, and (2) *delay* constraints between two occurrence specifications on different lifelines. Both of these can be constrained with a maximum time, minimum time, or time interval. Figure 4 summarizes these timing properties and how they may be expressed in sequence diagrams. Note that timing properties are structurally simpler than the safety and liveness properties typically expressed in temporal logic formalisms. However, practical specifications will typically express large numbers of fairly simple properties.

We will assume in the remainder that the constants used in timing constraints are always positive, that is, it is impossible

2. We take some liberty with the UML notation here: the "warp lines" in (P_1) and (P_2) allow us to condense two separate interactions into one diagram. While this is a very useful convention, it has no influence on the semantics, as such condensed diagrams can always be expanded accordingly.

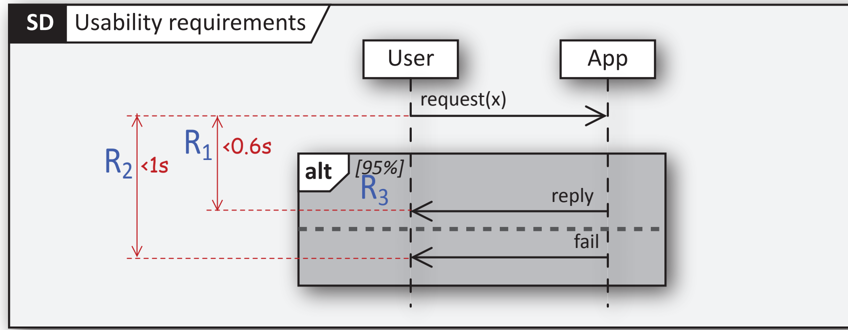


Figure 2. Usability requirements specified as a UML 2 interaction. Time constraints are highlighted in red.

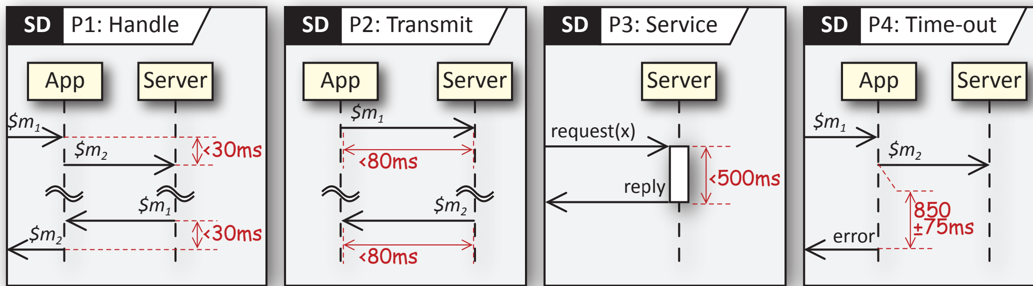


Figure 3. An example of a UML 2 interaction specifying timing requirements for the online banking app from a system perspective. Time constraints are highlighted in red. Variable names are prefixed by a dollar sign and are printed in italics. The "warp lines" are a shorthand to denote a pair of (similar) interactions.

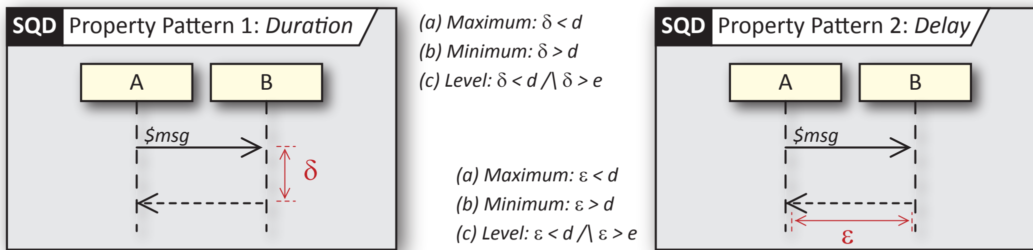


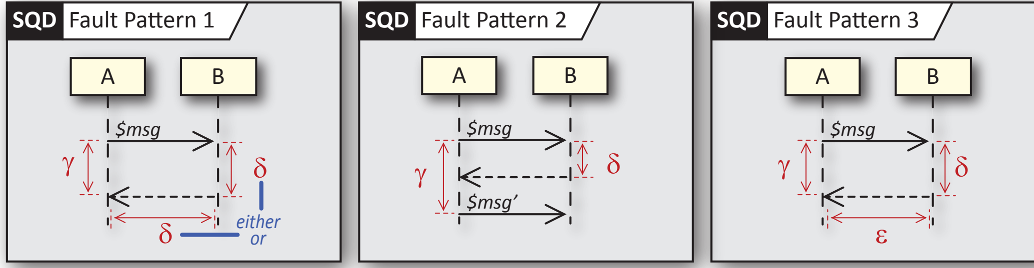
Figure 4. UML 2 interactions allow to express durations and delays with minimum and/or maximum times. Combinations of minimum and maximum constraints are called time level constraints; they may also be expressed as a given time plus/minus a deviation (cf. [11]).

to specify negative amounts of time. Obviously, this can trivially be achieved by input validation. Similarly, we shall assume that logically impossible time properties such as $\delta < x \wedge \delta > x$ for a constant x and a duration δ do not arise.

The timing property templates shown in Fig. 4 also allow us to define the types of weaknesses timing property specifications may exhibit, in particular those that arise from the interaction of timing properties (see Fig. 5). There are two levels of severity of weaknesses.

At the first level, there are outright *errors*, that make the overall specification inconsistent and prevent it from being implemented altogether. For instance, on a sequence of four occurrence specifications $a_1, \ldots, a_4$, a specification demands that the time between a_2 and a_3 is larger than the time between a_1 and a_4, but a_1 comes before a_2, and a_4 comes after a_3. Clearly, such a specification cannot be satisfied.

At the second level of severity, there are *redundancies* between time specifications. For instance, for the same sequence of occurrence specifications $a_1, \ldots, a_4$ considered before, a specification might demand that the time between a_1 and a_4 is larger than a given limit, but the time between a_2 and a_3 is even larger (again, we assume that a_1 comes before a_2, and a_4 comes after a_3). Clearly, the time constraint between a_1 and a_4 is a consequence of the time constraint between a_2 and a_3. So, at best, the former constraint is redundant and causes confusion and additional work. At worst, out of this confusion and extra work, new errors



Fault Pattern	Error	Redundancy
1	$\gamma < c, \delta > d$ for $d \geq c$	$\gamma > c, \delta > d$ for $d > c$ $\gamma < c, \delta < d$ for $d < c$
2	$\gamma < c, \delta > d$ for $d \geq c$	$\gamma > c, \delta < d$ for $d + e \geq c$
3	$\gamma < c, \epsilon > e, \delta > d$ for $d + e \geq c$	$\gamma > c, \delta < d, \epsilon < e$ for $d + e < c$

Figure 5. Patterns of timing flaws in UML 2 interactions.

arise, in particular, when multiple parties collaborate over time. More importantly, however, in our experience such redundancies tend to hide misunderstandings, or specification typos (i.e., the constraint was intended to say something different than it does). Therefore, we argue that redundancies should be considered weaknesses of a specification, and eliminating them increases the quality. Observe, that both errors and redundancies arise out of the interplay between several timing properties.

3. Timing Feature Interactions

In the previous section we have described how to capture many common temporal properties in requirement specifications in a precise way. We have also described some of the issues that arise in individual timing properties. A more difficult problem arises when timing properties interact in a way that each of them is satisfied, but together they are not, which we call Timing Feature Interaction (TFI). Since this is the central notion of this paper, we explicitly define it as follows.

Definition 1. A *Timing Feature Interaction (TFI)* is a requirement defect that arises from the interaction of timing properties, each of which is correct by itself.

Many TFIs can be identified by experienced developers, when the interacting timing properties are presented together in the same sequence diagram. However, this is rarely the case in practical specifications. There, large numbers of individual properties are specified in different places. Finding those subsets of properties that, together, give rise to a defect is a major challenge. It is very easy to overlook any such situation, in particular, when a specification is created as a collaboration of many co-workers.

Consider the four different implementations A through D of our online banking app example shown in the last four columns of Tab. 1. Each implementation has slightly different performance properties, but all of them are within the individual system properties (P_1) to (P_4) defined by the interaction in Fig. 3. Together, however, the global timing requirements (R_1) and (R_2) defined in Fig. 1 may or may not be satisfied (see last two rows). Figure 6 shows the two cases answering a request with or without success. We have highlighted the two cases of implementation C in Fig. 6: the successful call violates the timing constraints, while the failing call satisfies them. This is an instance of Timing Feature Interaction.

As we have said before, finding a TFI in an interaction like Fig. 7 is feasible for engineers. It is not practical, though, as real specifications may contain hundreds and thousands of requirements, and it is simply too much effort to manually compare all individual timing properties, combine them into complex interactions, and scrutinize them for interactions. Even if this is can be done once, for smaller specifications, say, it becomes excessively tedious (and error prone) to repeat the process after each change of the requirements. This is where our approach comes in.

We are now considering how to detect Timing Feature Interactions in such specifications. Our approach consists of three steps: First, we show how interactions with timing constraints can be translated in to Difference Bound Matrices (DBMs). Second, we present an algorithm of combining sets of individual simple timing properties into a single complex timing specification. Third, we show how TFIs can be detected in DBMs, thus leading to the detection of TFIs in complex timing specifications.

4. Formalising Timing Specifications by DBMs

We now show how UML interactions may be represented as “difference bound matrices” (DBMs [8]). In its original form, a DBM describes constraints on real-valued clocks

Table 1. FOUR DIFFERENT IMPLEMENTATIONS MAY HAVE DIFFERENT TIMING PROPERTIES (A THROUGH D), ALL OF WHICH SATISFY THE INDIVIDUAL SYSTEM REQUIREMENTS DEFINED IN FIG. 3 (FIRST FOUR ROWS). YET, WHEN COMBINING THE PROPERTIES, THE SYSTEMS MAY OR MAY NOT SATISFY THE USER REQUIREMENTS DEFINED IN FIG. 1 (LAST TWO ROWS).

Property	Constraint	Implementation A	B	C	D
P_1 (handle)	< 30 ms	20 ms	25 ms	25 ms	25 ms
P_2 (transmit)	< 80 ms	70 ms	70 ms	78 ms	75 ms
P_3 (service)	< 500 ms	400 ms	400 ms	400 ms	420 ms
P_4 (time-out)	850 ± 75 ms	850 ms	920 ms	850 ms	920 ms
R_1 (succeed)	< 0.6 s	580 ms ✓	590 ms ✓	606 ms ✗	620 ms ✗
R_2 (fail)	< 1 s	940 ms ✓	1015 ms ✗	955 ms ✓	1020 ms ✗

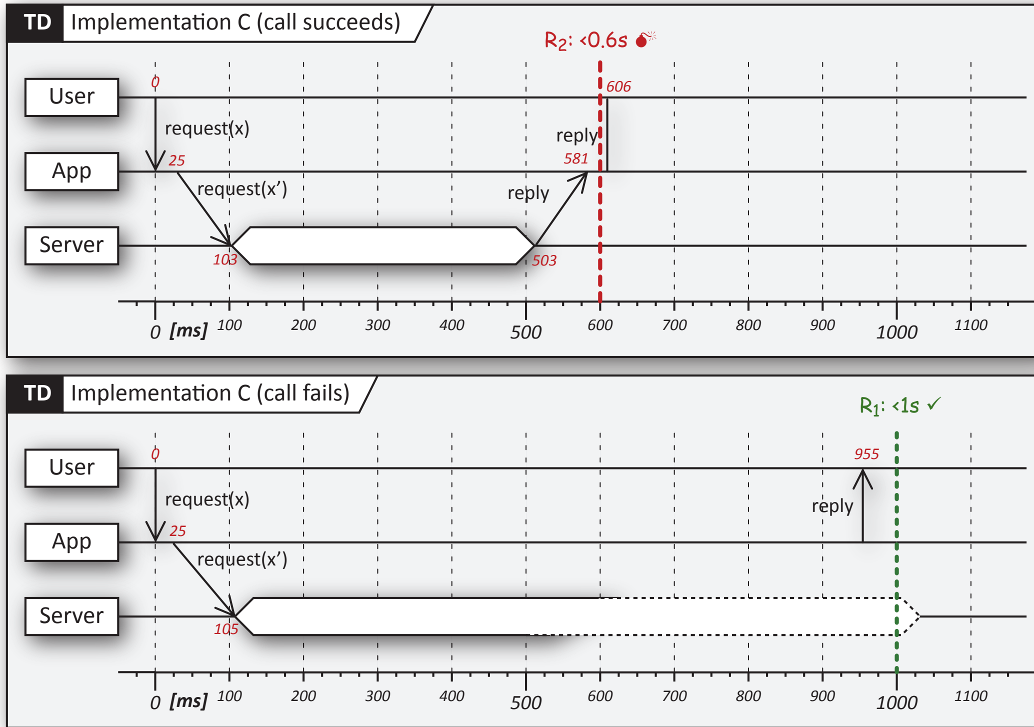


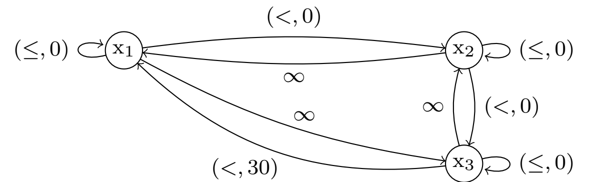
Figure 6. A UML 2 timing diagram representing a trace as it may occur in an actual implementation: The "call succeeds" timing requirement is violated while the "call fails" timing requirement is met.

by bounding the difference between each pair of clocks. If, for example, we have three clocks x_1, x_2, and x_3 with the constraints that x_1 must be strictly before x_2, that x_2 must be strictly before x_3, and, finally, that the difference between x_3 and x_1 must be less than 30, we obtain the following DBM

$$D = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \end{array} \begin{array}{c} \begin{array}{ccc} x_1 & x_2 & x_3 \end{array} \\ \begin{pmatrix} (\leq, 0) & (<, 0) & \infty \\ \infty & (\leq, 0) & (<, 0) \\ (<, 30) & \infty & (\leq, 0) \end{pmatrix} \end{array} .$$

An *entry* $D(x_i, x_j) = (\preccurlyeq, \delta)$ with $\preccurlyeq \in \{<, \leq\}$ and $\delta \in \mathbb{R}$ at row i and column j in the matrix D represents a difference bound $x_i - x_j \preccurlyeq \delta$; the entry ∞ denotes that there is no bound. In particular, the entries on the diagonal are of the form $(\leq, 0)$, expressing that the difference between some clock and itself is exactly 0.

Alternatively, a DBM may be interpreted as the adjacency matrix of a "temporal constraint graph" [13, 14]; e.g., for D



The *satisfying* clock valuations $[\![D]\!]$ of a DBM D with clocks X are those maps $\nu : X \to \mathbb{R}_{\geq 0}$ such that $\nu(x_i) - \nu(x_j) \preccurlyeq \delta$ is fulfilled for all $\{x_i, x_j\} \subseteq X$ with $D(x_i, x_j) = (\preccurlyeq, \delta)$.

We apply this formalism to UML interactions by interpreting each occurrence specification o of an interaction by a clock. Specifically, in each basic interaction fragment (not involving any interaction operators), the sending occurrence specification $\text{SND}(s, m, r)$ of a message m from a sending lifeline s to a receiving lifeline r has to occur strictly before the receiving occurrence specification $\text{RCV}(s, m, r)$ of this message, i.e., $\text{SND}(s, m, r) - \text{RCV}(s, m, r) < 0$; moreover, the ordering on each lifeline has to be obeyed, i.e., if o_1 is before o_2 on some lifeline, then $o_1 - o_2 < 0$. These requirements are complemented by the timing constraints defined explicitly in the interaction. We will, however, leave out the time units (ms for our examples) from the DBM representation, and just operate on the real numbers assuming that some common time unit has been fixed.

For example, for the first part of property 1 (Handle) defined in Fig. 3 up to the warp lines (expressing part of (P$_1$)) we obtain as representation exactly the DBM D introduced above with

$$\begin{aligned} \mathrm{x}_1 &= \text{RCV}(-, \mathsf{m}_1, \mathsf{App}) \ , \\ \mathrm{x}_2 &= \text{SND}(\mathsf{App}, \mathsf{m}_2, \mathsf{Server}) \ , \\ \mathrm{x}_3 &= \text{RCV}(\mathsf{App}, \mathsf{m}_2, \mathsf{Server}) \ . \end{aligned}$$

A DBM need not specify the tightest bounds on clock differences, however, but these have to be inferred: In our example, $\mathrm{D}(\mathrm{x}_3, \mathrm{x}_2)$ is specified as ∞, though, in fact, $\mathrm{x}_3 - \mathrm{x}_1 < 30$ and $\mathrm{x}_1 - \mathrm{x}_2 < 0$, i.e., $\mathrm{x}_3 - \mathrm{x}_2 < 30$ has to hold in every satisfying clock valuation. Indeed, in our instantiation to (P$_1$), the time for the message transfer of m_2, given by the difference $\mathrm{x}_3 - \mathrm{x}_2$ must not exceed $30\,\mathrm{ms}$, since otherwise the overall requirement of processing m_1, given by the difference $\mathrm{x}_3 - \mathrm{x}_1$, within $30\,\mathrm{ms}$ cannot be satisfied any more as x_3 occurs after x_2.

It is therefore useful to proceed to canonical DBMs where the entries cannot be tightened any more without losing any satisfying clock valuations. Such tightest bounds correspond to the shortest distance between two clocks when employing a suitable addition of entries. Formally, an entry b is *strictly tighter* than an entry b', written as $b \sqsubset b'$, if either

$$\begin{aligned} &b = (\preccurlyeq, \delta) \text{ and } b' = \infty, \text{ or} \\ &b = (\preccurlyeq, \delta),\ b' = (\preccurlyeq', \delta'), \text{ and } \delta < \delta', \text{ or} \\ &b = (<, \delta) \text{ and } b' = (\leq, \delta); \end{aligned}$$

this yields a total order $\sqsubseteq$ on entries. The (associative and commutative) *sum* of two entries, denoted by an $\oplus$, is defined by

$$\begin{aligned} &b \oplus \infty = \infty \ , \\ &(\preccurlyeq, \delta) \oplus (\preccurlyeq, \delta') = (\preccurlyeq, \delta + \delta') \ , \text{ and} \\ &(<, \delta) \oplus (\leq, \delta') = (<, \delta + \delta') \ . \end{aligned}$$

A DBM $\overline{D}$ for clocks $\{x_1, \ldots, x_n\}$ is *canonical* if

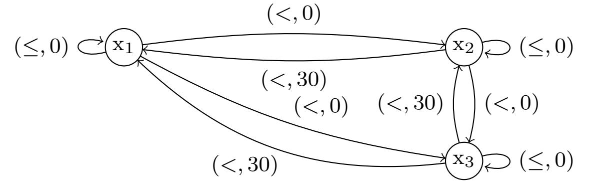
$$\overline{D}(x_i, x_j) \sqsubseteq \overline{D}(x_i, x_k) \oplus \overline{D}(x_k, x_j) \quad \text{for all } 1 \leq i, j, k \leq n.$$

The canonical form $\overline{D}$ of a given DBM D can thus be derived by applying an "all-pairs shortest path" computation w.r.t. $\oplus$ and $\sqsubseteq$, like the Floyd-Warshall-algorithm; in particular, $[\![D]\!] = [\![\overline{D}]\!]$.

For our example, the canonization of D yields

$$\overline{\mathrm{D}} = \begin{array}{c} \\ \mathrm{x}_1 \\ \mathrm{x}_2 \\ \mathrm{x}_3 \end{array} \begin{array}{c} \begin{array}{ccc} \mathrm{x}_1 & \mathrm{x}_2 & \mathrm{x}_3 \end{array} \\ \begin{pmatrix} (\leq, 0) & (<, 0) & (<, 0) \\ (<, 30) & (\leq, 0) & (<, 0) \\ (<, 30) & (<, 30) & (\leq, 0) \end{pmatrix} \end{array} .$$

Here it is inferred that x_1 has to be strictly before x_3; that the time bound between x_3 and x_2 is less than 30, as derived above; and that the time bound between x_2 and x_1 also is less than 30, since $\mathrm{x}_3 - \mathrm{x}_1 < 30$ and $\mathrm{x}_2 - \mathrm{x}_3 < 0$. As temporal constraint graph for this DBM we obtain



A DBM is *consistent* if there is a valuation of its clocks in $\mathbb{R}_{\geq 0}$ such that all entries of the matrix are satisfied, i.e., $[\![D]\!] \neq \emptyset$. In fact, consistency can be readily read off from the canonical form: If an entry on the diagonal of the canonical form is $(\preccurlyeq, \delta)$ with $\delta < 0$, then the bounds cannot be satisfied, both in the original and the canonical DBM as they represent the same overall bounds; otherwise all diagonal entries have to be of the form $(\leq, 0)$ and both DBMs are consistent.

The canonical DBM $\overline{\mathrm{D}}$ for our example is consistent, thus the first part of the property pattern for (P$_1$) can be implemented.

5. Timing Feature Interaction Analysis

Now that we have shown how sets of timing properties can be expressed using UML 2 Interactions, and formalized as DBMs, we shall exploit this representation to analyze our running example for TFIs.

5.1. Aggregating overlapping timing properties by matching

As we have mentioned before, the largest part of the problem with realistic specifications is their size: they may contain hundreds if not thousands of requirements. Even if all of these requirements are simple in themselves, finding inconsistencies between them is usually not a practical task. The first step, thus, is to find and combine individual specifications that are overlapping, and thus potentially interacting. One could think of this as a complex model transformation not unlike UML's package merge [9, p. 240], though targeted at interactions, and instantiating templates.

Figures 2 and 3 define sets of timing properties, and timing property templates, respectively. As they are all part of the same overall requirements specification, we should expect the terminology used in the specification to be consistent:

Whenever two different parts of the specification speak about two entities with the same name, they actually refer to the *same* entity (i.e., they overlap). Conversely, using the same name for different entities amounts to inconsistent terminology, and should be considered a specification flaw. In multi-view modeling tools, this identity can be expressed directly, that is, different diagrams may refer to the same model element. We will give a more formal definition of matching in terms of conjunction of difference bound matrices in Sect. 5.2 below. In this particular context, we can simply take two interactions to be overlapping, if they contain matching lifelines.

However, there are modeling languages and tools where this is not possible: All too often, the modeling tool of choice in the real world is PowerPoint or Visio [15]. And even using "proper" UML modeling tools, we frequently see multiple model element of the same type and name ("model clones", see [16, 17]). So, we must not assume that model elements are actually identified across diagrams.

As the next step, potential overlaps are registered ("matched"). With respect to UML interactions, there are three kinds of potentially overlapping model elements across different diagrams that we must consider.

- *Lifelines*: Two lifelines in two different diagrams match, if their names match.
- *Messages*: Two messages in two different diagrams match, if the names of the messages match, and the lifelines sending and receiving the messages match.
- *OccurrenceSpecifications*: Two occurrence specifications in different diagrams match, if the lifelines and messages to which they belong match.

Two names of any two elements of the same type match, if (1) they are the same string, (2) one of them is a variable, or (3) one of them is the empty string. For simplicity, we assume that interaction operators and interaction operands are generally to be considered different from one another. Matching shall instantiate variable names consistently, similar to the PROLOG unification algorithm. This kind of matching is defined in greater detail and implemented in the Visual Model Transformation Language (VMTL, [18]). This gives rise to the following procedure for merging a set of interactions (see Algorithm 1).

Obviously, this simple fixed-point algorithm terminates for finite k (typically, $k < 5$ suffices for all realistic models). The selection of p in Line 2 is irrelevant as all operations are symmetric. In order to optimize run time, though, it is advisable to prefer large interactions with concrete names over small interactions with empty or variable names.

Applying this straightforward algorithm to the interactions defined in Figs. 2 and 3, yields a combined, single sequence diagram shown in Fig. 7. In this combined interaction, it is fairly easy to see that all of the timing constraints specified can be satisfied simultaneously. It is also clear, that it is *possible* to implement systems that satisfy each constraint individually, but not all of them together as discussed in Sect. 3.

Algorithm 1: MERGE aggregates simple interactions into complex ones. The functions MATCH and OVERLAP are explained in Sect. 5.1.

Input: A finite set S of (simple) timing properties expressed as interactions, and a positive integer k to limit the iterations.
Output: A finite set C of (complex) timing properties aggregated from simple ones

```
1  C ← ∅
2  for p ∈ S do
3      change ← true
4      i ← k
5      while change ∧ i > 0 do
6          i ← i − 1
7          change ← false
8          for r ∈ S \ {p} do
9              if OVERLAP(r, p) then
10                 p ← MATCH(p, r)
11                 change ← true
12     C ← C ∪ {p}
13 return C
```

5.2. Checking for TFIs with DBMs

Our prototypical implementation in HUGO/RT transforms UML interactions consisting of interaction fragments with occurrence specifications combined by the interaction operators seq, strict, par, and alt as well as timing specifications between the occurrence specifications into difference bound matrices. The ordering of occurrence specifications in interaction operands is translated into entries for a DBM as explained in Sect. 4. The handling of the interaction operators follows their semantics given in [19, 20]. While the sequential (seq, strict) and parallel (par) composition operators produce a single DBM from the DBMs for the operands, the interaction operator for alternatives (alt) yields several DBMs, viz. as many as it shows operands. The resulting set of DBMs is then checked for inconsistencies by computing their canonical forms and checking the diagonals (see Sect. 4).

The interaction in Fig. 7 contains an alt with two operands and thus yields two DBMs, one showing eight occurrence specifications, and the other six. The timing constraints for HUGO/RT read as follows (`A` abbreviates "App", `S` "Server", and `U` "User" where the order of the letters indicates the direction of the underlying messages, i.e., `requestAS` denotes the request message from the app to the server):

```
snd(requestAS) - rcv(requestUA) <    30; // (P1) prop.
snd(replyAU)   - rcv(replySA)   <    30; // (P1)
rcv(requestAS) - snd(requestAS) <    80; // (P2)
rcv(replySA)   - snd(replySA)   <    80; // (P2)
snd(replySA)   - rcv(requestAS) <   500; // (P3)
rcv(failed)    - snd(requestAS) <=  925; // (P4)
snd(requestAS) - rcv(failed)    <= -775; // (P4)
rcv(replyAU)   - snd(requestUA) <   600; // (R1) req.
rcv(failed)    - snd(requestUA) <  1000; // (R2)
```

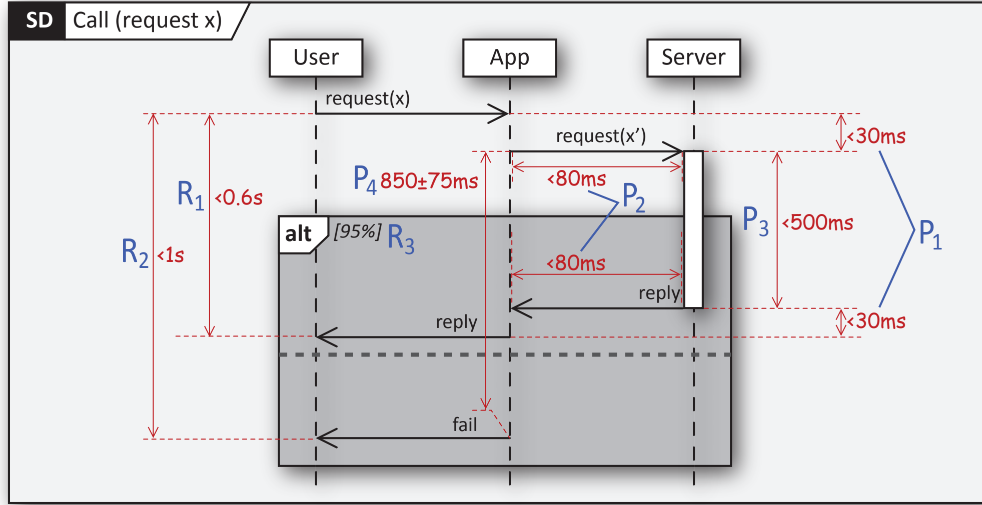


Figure 7. Combining the requirements and system properties from Figs. 1 and 3 (blue labels). Time constraints are highlighted in red.

Not surprisingly, HUGO/RT confirms that no inconsistencies occur by reporting those diagonal entries from the resulting canonical DBMs that are strictly negative:

```
Inconsistencies: { } //  upper operand
Inconsistencies: { } //  lower operand
```

In order to check the conformance of a particular implementation, say implementation C from Tab. 1, with these requirements, the timings of the implementation have to be added (in HUGO/RT, a timing constraint of the form $o_1 - o_2 = \delta$ is transformed into two DBM entries representing $o_1 - o_2 \leq \delta$ and $o_2 - o_1 \leq -\delta$):

```
snd(requestAS)  - rcv(requestUA)  ==  25; // (P1) impl.
snd(replyAU)    - rcv(replySA)    ==  25; // (P1)
rcv(requestAS)  - snd(requestAS)  ==  78; // (P2)
rcv(replySA)    - snd(replySA)    ==  78; // (P2)
snd(replySA)    - rcv(requestAS)  == 400; // (P3)
rcv(failed)     - snd(requestAS)  == 850; // (P4)
```

Combining two sets of timing requirements amounts to building the "conjunction" of their DBMs using the minima of their corresponding entries. Formally, the *greatest lower bound* (*infimum*) of two DBMs D and E over clocks X and Y, respectively, is the DBM $D \sqcap E$ over clocks $X \cup Y$ such that

$$(D \sqcap E)(z_1, z_2) = \min_{\sqsubseteq}\{D(z_1, z_2), E(z_1, z_2)\}$$

with $D(z_1, z_2) = \infty$ if $\{z_1, z_2\} \not\subseteq X$ and $E(z_1, z_2) = \infty$ if $\{z_1, z_2\} \not\subseteq Y$. The overall consistency of two DBMs can thus be read from building the canonical form of their infimum and checking the diagonal.

In fact, for implementation C, HUGO/RT reports the following inconsistencies in line with the manual calculation in Tab. 1:

```
Inconsistencies: { rcv(requestAS), snd(replySA),
                   rcv(requestUA), snd(requestAS),
                   rcv(replySA), snd(replyAU),
                   rcv(replyAU), snd(requestUA)
                 } //  upper operand
Inconsistencies: { } //  lower operand
```

More generally, the property patterns introduced in Sect. 3 can be represented as DBMs, though this time as generic matrices which have to be instantiated later on. Both the consistency of such patterns as well as their redundancy with a particular interaction, as discussed in Sect. 2, can then be checked using DBM techniques.

Consider, for example, the Duration property pattern of Fig. 4 in its "maximum" variant with annotation $\preccurlyeq \delta$ saying that the duration constraint $\text{SND}(b, m_r, a) - \text{RCV}(a, m, b) \preccurlyeq \delta$ has to hold (where m_r represents the return message for m). We can render this property pattern as the following generic DBM $\mathit{Duration}((x_1, x_2, x_3, x_4), (\preccurlyeq, \delta))$ such that

$$\mathit{Duration}((\text{SND}(a, m, b), \text{RCV}(a, m, b), \text{SND}(b, m_r, a), \text{RCV}(b, m_r, a)), (\preccurlyeq, \delta))$$

yields Duration:

$$\mathit{Duration}((x_1, x_2, x_3, x_4), (\preccurlyeq, \delta)) =$$

$$\begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{array} \begin{array}{c} \begin{array}{cccc} x_1 & x_2 & x_3 & x_4 \end{array} \\ \begin{pmatrix} (\leq, 0) & (<, 0) & \infty & (<, 0) \\ \infty & (\leq, 0) & (<, 0) & \infty \\ \infty & (\preccurlyeq, \delta) & (\leq, 0) & (<, 0) \\ \infty & \infty & \infty & (\leq, 0) \end{pmatrix} \end{array} .$$

Instantiating this pattern simply means to bind the occurrence specification variables $x_1, \ldots, x_4$ and the bound variable $(\preccurlyeq, \delta)$ to particular occurrence specifications and a particular bound of a given interaction.

For consistency of a pattern instantiation with a given interaction, assume that the interaction is represented by the DBM D and that the instantiation of the pattern yields the DBM D_p. Then the overall DBM with the pattern applied is the infimum of D and D_p, i.e., $D \sqcap D_p$. Thus the application

of a pattern is consistent with the given interaction to which it is applied if the infimum of their representing DBMs shows only diagonal entries of the form $(\leq, 0)$.

Finally, a timing constraint $x_1 - x_2 \preccurlyeq \delta$ is *redundant* for a DBM D over the clocks X if $\{x_1, x_2\} \subseteq X$ and $D(x_1, x_2) \sqsubseteq (\preccurlyeq, \delta)$, i.e., if no tightening of the difference bound between x_1 and x_2 would be achieved if the constraints were added. More generally, a complete DBM E over the clocks Y is *redundant* w.r.t. a DBM D over clocks X if $Y \subseteq X$ and $E(y_1, y_2) \sqsubseteq D(y_1, y_2)$ for all $y_1, y_2 \in Y$; this may equally be expressed as $D \sqcap E = D$, for which we shall simply write $D \sqsubseteq E$. Obviously, a check whether a pattern instantiation DBM D_p is redundant for an interaction DBM D, i.e., checking whether $D \sqsubseteq D_p$, should only be performed if the involved DBMs are canonical.

Not all property patterns, however, lend themselves easily to a representation as a DBM. For instance, requiring that two occurrence specifications have to be maximally δ time units apart, no matter, in which order they occur, leads to a pattern involving an alt-interaction operator. Its representation as a DBM hence involves two DBMs and both have to be handled, though conjointly, when checking consistency and redundancy.

6. Related Work

A comprehensive survey of UML 2 interaction semantics is found in [21]. Among other things, the transition from UML 1 to UML 2 introduced a novel semantics for interactions, which is why the first investigations of UML 2 interactions [19, 22] focused on understanding and interpreting the standard document. Since the UML specification informally suggests that the meaning of interactions are sets of sequences of so-called "interaction occurrences", this is what the first semantics defined formally. However, when the challenge is to support automated verification of properties, existing semantics are not efficient for practical cases.

The authors have recently introduced a more efficient semantics [23], but it is restricted to the efficient analysis of trace containment. Other, more general approaches based on theorem proving or model checking [24] can in principle detect TFIs in aggregated interactions, but still require the aggregation step, and also are overkill in the sense that they require high (computational) cost. It is unclear whether they are efficient enough today to solve this problem for realistic cases.

Consistency of timing properties in UML interactions has been studied in particular in the fields of embedded real-time computing and telecommunications [25, 26]; for a recent approach and overview of the literature see [27]. In fact, several efforts have been undertaken to enhance the expressiveness of UML to these fields by different profiles, like UML-RT, SPT (Schedulability, Performance, and Time), as well as MARTE (Modeling and Analysis of Real-Time Embedded Systems). The most prominent approach of ensuring timing consistency in UML models, be it in UML proper or one of the profiles, is the use of model checking [7, 27]. For "Message Sequence Charts" (MSCs), the main predecessor and competitor of UML interactions, the use of "temporal constraint graphs" mentioned in Sect. 4 has been studied extensively for timing consistency checking [13, 14]. The data structure is a bit more elaborate than DBMs, allowing also the checking of (unbounded) loops; for basic interactions, the checking process, however, similarly involves an "all-pairs shortest paths" computation. In fact, the results have been transferred to UML 1 [28] and also extensions, e.g., for time models involving clock drift, have been investigated [29]. The use of simple DBMs on the other hand makes some analyses somewhat more perspicuous, in particular when it comes to features expressed as patterns and their consistency and redundancy.

Feature interaction [6, 30] is a known problem since the 1980s, and has been the topic of its own conference series.[3] A survey of research results regarding feature interactions is presented in [6]. It is currently an open issue, what kind of feature interactions occur with what frequency, and how much of a problem feature interaction is in industry, actually [30]. To the best of our knowledge, the interaction of *timing* features has not been considered before. There is evidence, though, that (conventional) feature interaction is a considerable problem in some industries [31]. Thus, for the application scenario explored in this paper, model checking is too heavy-weight an approach: The challenge of feature interactions derives from the exponential number of potential interactions that are to be checked. Our approach, in contrast, is based on combining overlapping interactions and translating them into DBMs, a very compact data structure. This way, detecting interactions is so efficient, that it can be done as a simple background check to inform the modelers while they work on a given specification.

7. Conclusions

Summary. There is a strong incentive for finding specification flaws early. In many cases, the flaws as such are not terribly complex, but the size of the specification makes them difficult to detect. Thus, it is often not practical to try and detect them manually. In that sense, automation is much welcome, even if it covers only some potential flaws.

In this article, we characterize Timing Feature Interactions (TFIs), which are inconsistent timing properties in requirements specifications. We show how they may arise from sets of straightforward, individually coherent timing property specifications. We describe how to represent timing properties as UML 2 interactions with time constraints, and provide a simple formal semantics of systems of timing constraints as Difference Bounded Matrices (DBMs, see [8]). This representation does not aspire to cover the full meaning of interactions (see [19, 20, 23] for that), but is geared towards automatically detecting TFIs. We have implemented our approach prototypically in the HUGO/RT tool [7] and demonstrate its viability with an extended example.

3. The International Conference on Feature Interactions, see http://www27.cs.kobe-u.ac.jp/wiki/icfi/ for the latest installment.

Discussion. Our article has three main contributions. First, we introduce the notion of Timing Feature Interaction, which we believe is both novel and relevant. Second, we propose a procedure how such requirement defects may be discovered automatically. It involves the novel concept of aggregating sets of simple timing properties into more complex ones, which one may describe as a complex model transformation not unlike UML's package merge [9, p. 240], though targeted at interactions.

Limitations and future work. At current time, our approach is prototype with little validation; its practical utility remains to be demonstrated. In particular, it is not clear how many such flaws are contained in practical specifications, and how helpful it is to discover them: there are no comprehensive studies that inform us reliably about the distribution of various types of errors in realistic settings (cf. [30]). Our personal experience in industry tells us, though, that while these cases do not occur very frequently, when they do occur, their effects can be quite damaging if they go undetected.

References

[1] F. P. Brooks, "No Silver Bullet: Essence and Accidents of Software Engineering", *Computer*, vol. 20, no. 4, pp. 10–19, 1987.

[2] B. W. Boehm, R. K. McClean, and D. Urfrig, "Some Experience with Automated Aids to the Design of Large-Scale Reliable Software", *IEEE Trans. Softw. Eng.*, vol. 1, no. 1, pp. 125–133, 1975.

[3] A. Endres and H. D. Rombach, *A Handbook of Software and Systems Engineering*. Pearson/Addison-Wesley, 2003.

[4] F. P. Brooks, *The Mythical Man-Month*. Addison-Wesley, 1975.

[5] S. McConnell, *Software Project Survival Guide: How to Be Sure Your First Important Project Isn't Your Last*. Microsoft Press, 1997.

[6] M. Calder, M. Kolberg, E. H. Magill, and S. Reiff-Marganiec, "Feature Interaction: A Critical Review and Considered Forecast", *Computer Networks*, vol. 41, no. 1, pp. 115–141, 2003.

[7] A. Knapp and J. Wuttke, "Model Checking of UML 2.0 Interactions", in *Reports Rev. Sel. Papers Ws.s Symp.s MoDELS 2006*, ser. Lect. Notes Comp. Sci., no. 4364, T. Kühne, Ed., Springer, 2007, pp. 42–51.

[8] J. Bengtsson and W. Yi, "Timed Automata: Semantics, Algorithms and Tools", J. Desel, W. Reisig, and G. Rozenberg, Eds., ser. Lect. Notes Comp. Sci., no. 3098, Springer, 2004, pp. 87–124.

[9] *OMG Unified Modeling Language. Version 2.5*, formal/2015-03-01. http://www.omg.org/spec/UML/2.5/, 2015.

[10] W. Damm and D. Harel, "LSCs: Breathing Life into Message Sequence Charts", *Formal Methods in System Design*, vol. 19, no. 1, pp. 45–80, 2001.

[11] B. P. Douglass, *Real Time UML: Advances in the UML for Real-Time Systems*, 3rd. Addison-Wesley, 2004.

[12] C. Buckl, I. Gaponova, M. Geisinger, A. Knoll, and E. A. Lee, "Model-based Specification of Timing Requirements", in *Proc. 10th ACM Intl. Conf. Embedded Software (EMSOFT'10)*, ACM, 2010, pp. 239–248.

[13] H. Ben-Abdallah and S. Leue, "Expressing and Analyzing Timing Constraints in Message Sequence Chart Specifications", University of Waterloo, Techn. Rep. 97-04, 1997.

[14] ——, "Timing Constraints in Message Sequence Chart Specifications", in *Proc. IFIP TC6 WG6.1 Joint Intl. Conf. Formal Description Techniques and Protocol Specification, Testing and Verification (FORTE X, PSTV XVII)*, A. Togashi, T. Mizuno, N. Shiratori, and T. Higashino, Eds., ser. IFIP Conf. Proc., no. 107, Chapman & Hall, 1998, pp. 91–106.

[15] S. Baltes and S. Diehl, "Sketches and Diagrams in Practice", in *Proc. 22nd ACM SIGSOFT Intl. Symp. Foundations of Software Engineering (FSE'14)*, ACM, 2014, pp. 530–541.

[16] H. Störrle, "Towards Clone Detection in UML Domain Models", *Softw. Syst. Model.*, vol. 12, no. 2, pp. 307–329, 2013.

[17] ——, "Effective and Efficient Model Clone Detection", in *Software, Services and Systems. Essays Dedicated to Martin Wirsing on the Occasion of His Emeritation*, R. De Nicola and R. Hennicker, Eds., ser. Lect. Notes Comp. Sci., no. 8950, Springer, 2014, pp. 307–329.

[18] V. Acretoaie, H. Störrle, and D. Strüber, "VMTL: A Language for End-User Model Transformation", *Softw. Syst. Model.*, pp. 1–29, 2016.

[19] H. Störrle, "Semantics of Interactions in UML 2.0", in *Proc. IEEE Symp. Human Centric Computing Languages and Environments (HCC'03)*, J. Hosking and P. Cox, Eds., IEEE, 2003, pp. 129–136.

[20] M. V. Cengarle, A. Knapp, and H. Mühlberger, "Interactions", in *UML 2 — Semantics and Applications*, K. Lano, Ed., Wiley, 2009, ch. 9, pp. 205–248.

[21] Z. Micskei and H. Waeselynck, "The Many Meanings of UML 2 Sequence Diagrams: A Survey", *Softw. Syst. Model.*, vol. 10, no. 4, pp. 489–514, 2011.

[22] H. Störrle, "Assert, Negate and Refinement in UML-2 Interactions", in *Proc. Ws. Critical Systems Development with UML*, J. Jürjens, B. Rumpe, R. France, and E. B. Fernandey, Eds., 2003, pp. 79–94.

[23] A. Knapp and H. Störrle, "Efficient Representation of Timed UML 2 Interactions", in *Proc. 8th System Analysis and Modeling Conf. (SAM'14)*, D. Amyot, P. Fonseca i Casas, G. Mussbacher, R. Gotzhein, and J. González Huerta, Eds., ser. Lect. Notes Comp. Sci., no. 8769, Springer, 2014, pp. 110–125.

[24] A. Knapp and S. Merz, "Model Checking and Code Generation for UML State Machines and Collaborations", in *Proc. 5th Ws. Tools for System Design and Verification, Techn. Rep. 2002-11*, D. Haneberg, G. Schellhorn, and W. Reif, Eds., Institut für Informatik, Universität Augsburg, 2002, pp. 59–64.

[25] J. Küster and J. Stroop, "Consistent Design of Embedded Real-time Systems with UML-RT", in *Proc. 4th IEEE Intl. Symp. Object-oriented Real-time Distributed Computing (ISORC'01)*, IEEE, 2001, pp. 31–40.

[26] L. Lavagno, G. Martin, and B. Selic, Eds., *UML for Real — Design of Embedded Real-Time Systems*. Kluwer Academic Publ., 2003.

[27] J. Choi, E. Jee, and D.-H. Bae, "Timing Consistency Checking for UML/MARTE Behavioral Models", *Software Qual. J.*, vol. 24, pp. 835–876, 2016.

[28] X. Li and J. Lilius, "Timing Analysis of UML Sequence Diagrams", in *Proc. 2nd Intl. Conf. Unified Modeling Language (UML'99)*, R. France and B. Rumpe, Eds., ser. Lect. Notes Comp. Sci., no. 1723, Springer, 1999, pp. 661–674.

[29] S. Akshay, B. Genest, L. Hélouët, and S. Yang, "Symbolically Bounding the Drift in Time-Constrained MSC Graphs", in *Proc. 9th Intl. Coll. Theoretical Aspects of Computing (ICTAC'12)*, A. Roychoudhury and M. D'Souza, Eds., ser. Lect. Notes Comp. Sci., no. 7521, Springer, 2012, pp. 1–15.

[30] S. Apel, S. Kolesnikov, N. Siegmund, C. Kästner, and B. Garvin, "Exploring Feature Interactions in the Wild: The New Feature-interaction Challenge", in *Proc. 5th Intl. Ws. Feature-Oriented Software Development (FOSD'13)*, ACM, 2013, pp. 1–8.

[31] M. Weiss and B. Esfandiari, "On Feature Interactions Among Web Services", in *Proc. Intl. Conf. Web Services*, IEEE, 2004, pp. 88–95.