

Consistent Design of Embedded Real-time Systems with UML-RT

Jochen M. Küster and Joachim Stroop
University of Paderborn/C-LAB
Fürstenallee 11
D-33094 Paderborn
{jkuester, jost}@c-lab.de

Abstract

Modeling embedded real-time systems consisting of different components with UML-RT leads to a design model using various diagrams. Sequence diagrams describe possible interactions between system components and may be annotated with specific real-time constraints. Statechart diagrams are used for describing each component's behavior. In order to be able to get a consistent model, a consistency concept for different diagram types is needed that takes into account real-time constraints. In this paper, a consistency concept for sequence diagrams and statechart diagrams is presented which focuses on the establishment of timing constraints. Our consistency concept distinguishes between syntactical, semantic and real-time consistency and takes into account the influence of processor allocation and scheduling. Using the consistency concept we describe a method for ensuring the consistency based on worst case execution time analysis of statecharts and schedulability analysis of tasks, thereby enabling a precise answer of the question of consistency.

1. Introduction

In many embedded real-time systems there is a considerable amount of software running on off-the-shelf microcontrollers. And the amount of software usually increases with every new generation of these systems, whereas the average productivity of software engineers is constant as long as a fixed design technology is used. This leads to a complexity problem which demands for new methods to keep pace with the increasing demands.

This problem is well known in the classical domain of software engineering. There, object-oriented and component-based methods are proposed as enabling technologies for the structuring of complex software systems and the reuse of components. In this domain, UML [12] has become the standard modeling notation. Despite of their widespread

usage, it has to be noticed that these technologies are still developing and still face some important problems, among them a universal consistency concept for models produced within the software development cycle.

Nevertheless, there is a growing interest to adapt the UML to the design of embedded software systems, e.g. by means of UML extensions such as UML-RT [15]. However, these approaches from main-stream software engineering may not easily be transferred as the design of embedded software systems faces a set of unique characteristics, expressed as non-functional requirements like real-time constraints and resource restrictions.

As a consequence, there are still a number of gaps to be filled in order to enable the wide-spread usage of UML-RT for the design of embedded real-time systems: Among them are the establishment of a real-time consistency concept for a design model consisting of different diagrams taking into account the real-time specific analysis of non-functional requirements. Such a consistency concept enables the validation of a design model with respect to important real-time aspects, leading to a greater acceptance of UML-RT in the real-time domain. Furthermore, it is a prerequisite for the reuse of components by the means of consistency tests and for applying code generation techniques.

Modeling embedded real-time systems consisting of different components with UML-RT leads to a design model using various diagrams. Sequence diagrams describe possible interactions between system components. Statechart diagrams are used for describing each component's behavior. Additionally, sequence diagrams may be annotated with timing constraints. In this paper, a consistency concept for sequence diagrams and statechart diagrams is presented which focuses on the establishment of timing constraints. Our consistency concept distinguishes between syntactical, semantic and real-time consistency and takes into account the influence of processor allocation and scheduling. Using the consistency concept we describe a method for ensuring the consistency based on worst case execution time analysis of statecharts and schedulability analysis of tasks, thereby

enabling a precise answer of the question of consistency.

The paper is structured as follows. First, an example from the automotive domain is presented to demonstrate the use of UML-RT, motivate the importance of the question of consistency and timing constraints. In Section 2.1 we introduce a general consistency concept for a UML-RT design model, with a special focus on real-time related issues. In Section 3 we show how the real-time consistency concept can be practically used. Finally, we mention related work and give some concluding remarks for future work.

2. Modeling with UML-RT

UML-RT [15] is an extension of the UML built on concepts from ROOM [16]. Making use of the built-in extension mechanisms of the UML the notions of capsules, ports, connectors, protocols and protocol roles are introduced in UML-RT in order to enable modeling of complex real-time systems. For the representation of the architecture of those systems a capsule collaboration diagram is defined depicting components of the system and how they are connected.

A *capsule* is a specialized active class and is used for modeling a self-contained component of a system. Other than ordinary classes, capsule operations can only be called from within the capsule. For communication with other capsules a capsule may have one or more ports through which it is interconnected to other capsules via *connectors*. They represent hardware connections via which capsules communicate by sending and receiving messages.

In order to illustrate concepts of UML-RT we consider an application from the automotive domain. It contains subsystems of a dashboard and an engine control of a vehicle. The conceptual design of this example is shown in Figure 1 using a capsule collaboration diagram. There we have four wheel sensors for each car. They are producing pulses with a rate proportional to the revolution speed of each wheel. These pulses are accumulated by a counter unit which derives the vehicle speed and the covered distance and sends these values to the dashboard display. Input to the motor and brake control is given by the driver through the accelerator and brake pedals. We have depicted this by a module for driver input. Driving stability and security is enhanced by an Anti Blocking System (ABS). It detects blockings of any of the four wheels and reduces the braking pressure for such a wheel.

From the point of view of behavior modeling each capsule is associated to a statechart specifying states and state transitions of the capsule. Capsule statecharts are the same as statecharts of UML which originated from Harel's statecharts [7]. They describe how capsules react to messages received via their ports and when messages are sent via their ports. State transitions of capsule statecharts may also include the calling of capsule operations. Additionally, se-

quence diagrams may be used to describe interactions of capsules and protocol statecharts may optionally be used to describe the exchange of messages within a protocol.

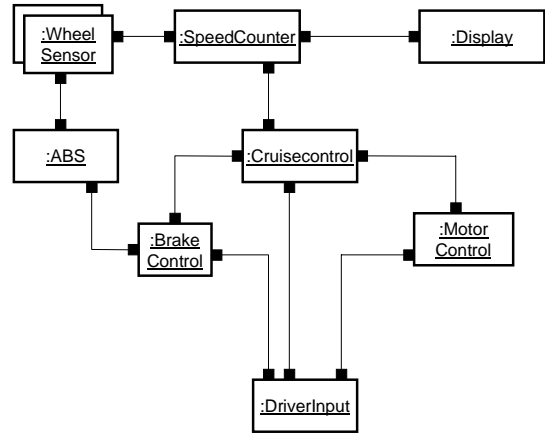


Figure 1. Architecture diagram in UML-RT

To improve driver's comfort we now assume that this application should be extended by an additional cruise control unit. We assume that the design of this unit will be provided as a self-contained component by a separate development group.

Inputs to the cruise control component are (de)activation messages from the driver and the instantaneous speed of the car. As output, the unit influences both the brake and the motor management. Therefore, it will be integrated in the system as shown in the center of Figure 1.

From the point of view of the behavior, the cruise control should behave as follows: If activated by the driver the cruise control first requests the current speed from the counter module. The result of this query is the desired speed the car has to hold without further driver interference. In this hold mode, the system periodically checks the instantaneous speed and compares it to the desired speed. If the difference between both values is greater than a certain threshold a message is sent to the motor control to increase or decrease speed. In certain driving conditions the braking systems gets involved. If the driver uses one of the accelerator or brake pedals the operation of the cruise control is suspended. Furthermore, the driver may also stop the cruise control by a deactivation message.

The two development groups have agreed on how the cruise control interacts with other components of the system and a set of sequence diagrams has been developed. One possible interaction is partly described in Figure 2. Furthermore, this diagram also contains timing requirements, e.g. the rate at which the cruise control should perform its control loop. The cruise control development group will deliver a capsule for the cruise control containing a behavioral model given as a statechart as shown in Figure 3.

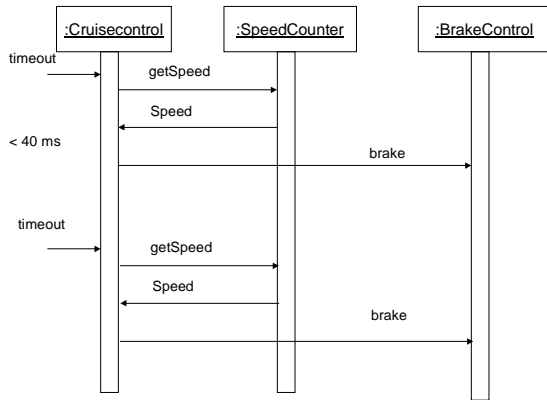


Figure 2. A scenario for the cruise control

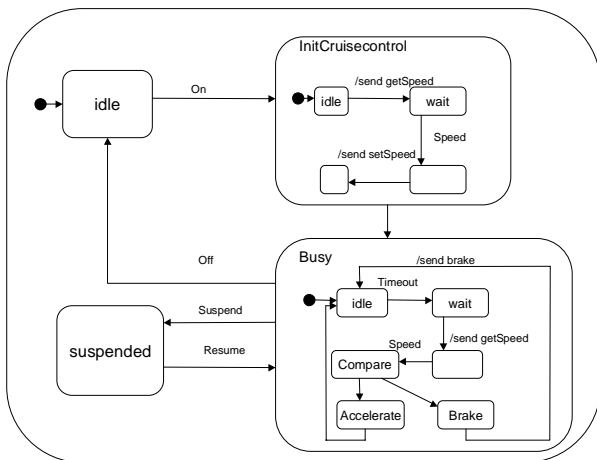


Figure 3. The statechart of the cruise control

With our sample application we have motivated that system design based on UML-RT means to compose a system out of different views expressed as separate diagrams. Among them sequence diagrams and statechart diagrams are the most important ones for the description of behavior. However, consistency between these views is not inherent within the modeling language but must be established by the software engineer. Currently, a consistency concept for UML behavior diagrams is missing.

Consistency becomes even more important if a capsule must be integrated into an existing design. In our example, the cruise control capsule is part of many sequence diagrams that are already specified. The behavior expressed by the “union” of all these sequence diagrams and the behavior expressed by the statechart diagram associated with that capsule must be consistent. Thus, it must be validated that the new component can be integrated into the overall system by performing consistency tests.

Another important aspect results from the fact that the application is an embedded real-time system. Given the

architecture of the system and a certain scheduling policy for the software tasks, can the new component be integrated such that all timing requirements are met, i.e. temporal consistency of the diagrams can be established?

2.1 Consistency of Diagrams

Our overall goal is to define a development process for building real-time systems reusing already existing components. In the previous section, we have introduced an example making use of different diagram types. It has been motivated that the consistency between the diagrams is important for the successful implementation of a model, especially when existing components have to be integrated. As a consequence, the notion of consistency deserves more attention.

In general, consistency within a specification is a mandatory requirement because it is a prerequisite for the correct execution of the system specified in different parts. Consistency is also of major importance within the development process because it facilitates by consistency tests the construction of a complex system by different developers and also supports system evolution by checking a specification for consistency after changing or replacing parts of it. With respect to our example, given a sequence diagram describing the interaction between the cruise control and other system components the cruise control can be developed independently. From the previous arguments, we conclude that consistency is the property that different parts of a specification are compatible with each other and not contradictory.

The problem of consistency arises in cases where a specification consists of different parts, each part concentrating on a specific view of the system. Then it has to be ensured that the overall specification gained from all parts does not contain consistency errors. These errors can be of syntactical or semantic nature leading to the distinction of *syntactical* and *semantic* consistency. As a consequence, consistency is closely related to the definition of a specification language which describes the syntax and semantics of the language.

The UML-RT as an extension of UML consists of a number of sublanguages and using UML-RT leads to the construction of models of different diagram types. Consequently, a UML-RT design consists of different diagrams as shown in the example described in Section 2 and thus consistency of these diagrams is an important issue. Currently, the syntax of the UML is defined using the metamodeling approach and well-formedness rules. Deplorably, the semantics of the UML is only informally defined using plain English and cannot be considered to be complete [9].

Syntactical consistency ensures that a specification conforms to the abstract syntax specified by the metamodel. Similar to syntax of programming languages, this requires that the overall model is well-formed. For example, iden-

tifier names used in one submodel must be properly defined in another submodel. Syntactical consistency constraints are expressed in UML using well-formedness rules in OCL. With respect to Figures 2 and 3, a sequence diagram describing a scenario must be compatible with the capsule statecharts such that names of messages used in the sequence diagram must correspond to the ones used in statecharts.

The notion of *semantic consistency* is rather a wide one. In general, it requires submodels of a model to be compatible semantically with regards to the aspects of the system which are specified in both submodels. Concerning Figures 2 and 3, the order of messages specified in a sequence diagram must be consistent with the one resulting from the execution of the capsule statecharts.

Real-time consistency can be considered as one specific form of *semantic consistency*. *Real-time consistency* ensures that the submodels are consistent with respect to real-time issues. Currently, there is no unique representation of real-time issues in UML. However, with regards to timing constraints, there is the possibility of annotating sequence diagrams. The validity of timing constraints is of fundamental importance for real-time systems and we will call this temporal consistency. In our example, this means that the time constraint specified in a sequence diagram is really met by the statecharts associated to the capsules occurring in that sequence diagram.

2.2 A Consistency Concept for Sequence Diagrams and Statecharts

Building on the ideas of the previous section we will now tackle the problem of defining a consistency concept for sequence diagrams and statecharts.

In the following, we will abstract from the example described above in order to get a more formal description of consistency. We will concentrate on sequence diagrams and statecharts and first introduce formal definitions for them. We define a sequence diagram as follows (taken from [11] and extended):

Definition 1 (Sequence diagram) A UML sequence diagram is a tuple $SeqD = (O, E, V, C)$, an ordering relation, a labeling function *label* and a function *object* where

- O is a finite set of objects.
- E is a finite set of events corresponding to sending or receiving a message.
- V is a finite set of arcs whose elements are of the form (e, e') where e and e' are in E and $e' \neq e$.
- C is a set of boolean expressions of the form $t(e) - t(e') \leq d$ which represents the timing constraints en-

forced on $SeqD$, where t is a mapping to a timepoint and $object(e) = object(e')$.

- order is an ordering relation on E and we assume that all events related to one object are ordered representing the timely order of events occurring in one object. We further require events e and e' ordered with $e \leq e'$ if and only if $(e, e') \in V$.
- label is a function which assigns each $v \in V$ a message name m .
- object is a function which maps each $e \in E$ to the object it belongs to. This function serves as a partitioning of the events according to the lifelines in the diagram.

We define N_{SeqD} to be the set of all message names occurring in the sequence diagram and denote $N_{SeqD,o}$ the set of all message names sent or received by the object $o \in O$ of the sequence diagram. We distinguish between events of receiving messages and events of sending messages and denote the event of receiving message m_i with $r(m_i)$ and the event of sending message m_i with $s(m_i)$.

As statecharts are currently not formally defined in UML, we will define them as simple as possible, taken and adapted from [13].

Definition 2 (Statechart diagram) A UML statechart diagram is a tuple $StatD = (S, E, A, t)$ where

- S is a non-empty, finite set of states.
- E is a finite set of events.
- A is a finite set of actions.
- t is the transition relationship with $t : S \times E \times A \rightarrow S$ or alternatively $(s_i, l_i, s_j) \in t$ where $l_i \in E \times A$ denotes the event and action of the transition.

Events may be induced by receiving a message m_i which is denoted by $r(m_i)$. Actions may be sending a message m_i which is denoted by $s(m_i)$. In UML notation, $l = (m_i, m_j)$ is abbreviated to m_i/m_j . We define $N_{StatDMessages}$ to be the finite set of messages received or sent. We define $N_{StatDEventsActions}$ to be the set of events and actions. Finally, we define a predicate $t_l(s_i, s_j) = true$ if there exists a sequence $\langle a_i \rangle = \langle (s_i, l_1, s_{i+1}), \dots, (s_n, l_n, s_j) \rangle$ with $a_i \in t$ and $l_i = l$, false otherwise.

Note that we only consider receiving messages as events and sending messages as actions although there are other possibilities for actions such as calling an operation. Further notational aspects like compound or parallel states are also not expressed by this definition, due to space limitations.

In the remainder of this paper we will use a small illustrative example. It consists of a reactive system that is

composed of three active objects A , B , and C interacting through the exchange of messages. Two such scenarios are modeled as sequence diagrams. They are depicted in Figures 4 and 5. Each scenario is initiated by the occurrence of an external event, i.e. m_0 and m_{00} , which we regard as a timer event. We assume, that both these events occur periodically. For simplicity of our further discussions we demand that no two messages are labeled the same. Note that these scenarios correspond closely to the example described in Section 2.

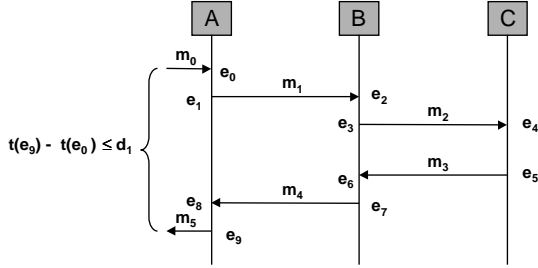


Figure 4. Scenario 1

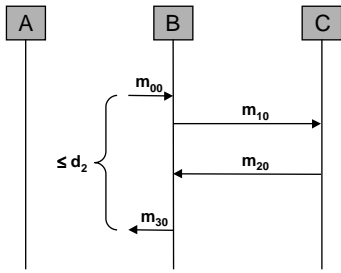


Figure 5. Scenario 2

The behavior of each object is captured by a statechart diagram as shown in Figures 6, 7 and 8. Object A enters state s_1 upon activation. When triggered by m_0 it changes state to s_2 . It then emits message m_1 etc. Note that the statechart of A also contains the compound state s_2 . The behavior of objects B and C is modeled with a parallel state. With respect to our definition, the set of messages of the statechart of A is $N_{StatAMessages} = \{m_0, m_1, m_4, m_5, m_6\}$.

We are now ready to define syntactical consistency of statecharts and sequence diagrams.

Definition 3 (Syntactical Consistency) Let $SeqD = (O, E, V, C)$ be a sequence diagram and let $StatD_1, \dots, StatD_m$ be the statechart diagrams associated to objects o_1, \dots, o_n occurring in $SeqD$. Then $SeqD$ and the statechart diagrams $StatD_1, \dots, StatD_m$ are syntactically consistent if for all $o \in O$ the set of message names $N_{SeqD,o}$ is a subset of the set of messages $N_{StatD_i,Messages}$.

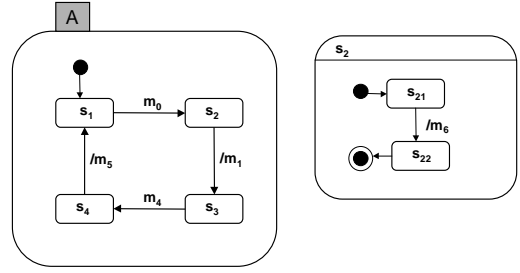


Figure 6. The statechart of object A

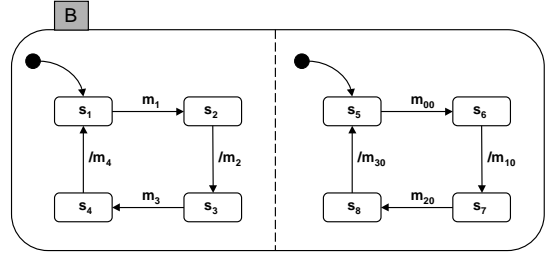


Figure 7. The statechart of object B

Likewise, we could define syntactical consistency of one sequence diagram with one statechart diagram if we require the set of message names associated to one object of the sequence diagram to be a subset of the set of messages of the statechart associated to that object. With respect to our example, the set of message names of the sequence diagram associated to A in Figure 4 is $\{m_0, m_1, m_4, m_5\}$ which is obviously a subset of the set of messages of the statechart of A .

In order to define the semantic consistency of sequence diagrams and statechart diagrams we need some further definitions. Intuitively, each sequence diagram represents one run of all the statechart diagrams associated to the objects of the sequence diagram. We now focus on one object in the sequence diagram and define the sequence of events of receiving or sending messages.

Definition 4 (Message events between two events) The sequence of message events induced by a sequence diagram $SeqD = (O, E, V, C)$ between two events e and d , $e, d \in E$ is the ordered sequence of message events between these two events:

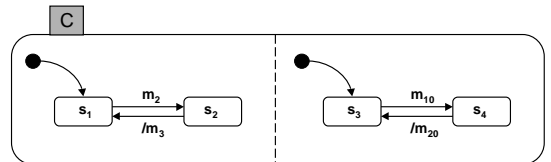


Figure 8. The statechart of object C

$sequence(e, d) = \langle a_i \rangle$ such that the following hold:

- $a_i = r(m)$ or $a_i = s(m)$ with $m = label(v)$ for one $v \in V$, all events are send or receive events of messages of the sequence diagram.
- $e \leq a_i \leq d$ and $(i \leq j \text{ implies } a_i \leq a_j)$, the sequence is ordered.

Likewise, the sequence of message events between two events e and d , $e, d \in E$ and $object(e) = object(d)$, related to one object $o \in O$ to be $sequence_o(e, d) = \langle a_i \rangle$ subject to the previous conditions, with the additional condition that $object(a_i) = o$, meaning that all events are associated to the object o .

For example, the sequence of message events induced by the first sequence diagram related to A between e_0 and e_9 is $\langle r(m_0), s(m_1), r(m_4), s(m_5) \rangle$.

For semantic consistency, the sequence of message events related to one object must now be a possible sequence of events or actions in the statechart diagram of that object. Sending of a message must correspond to a send action in the statechart diagram, receiving of a message must correspond to an event in the statechart. A statechart can be considered as a generator of sequences of events and actions. With respect to our example, one possible sequence generated by statechart A is $\langle r(m_0), s(m_1), s(m_6), r(m_4), s(m_5) \rangle$. In this case, this is also the only sequence. However, in general, a statechart usually generates several different sequences. In order to consider a sequence diagram and a statechart semantically consistent, the sequence of message events related to one object in a sequence diagram must therefore only be a subsequence of one possible sequence of events and actions generated by the statechart. We therefore define:

Definition 5 (Semantic Consistency) A sequence diagram $SeqD$ and a statechart diagram $StatD$ can only be semantically consistent if the sequence of message events related to each object between the first and the last event of that object in $SeqD$ is a subsequence of a sequence of events and actions generated by the corresponding statechart diagram $StatD$ of the object.

For further use, we now establish a more formal relationship between sequence diagrams and statecharts. Each sequence of message events related to one object is generated by a sequence of states and transitions of the statechart of that object.

Definition 6 (Induced states and transitions) Let $SeqD = (O, E, V, C)$ be a sequence diagram and let $StatD = (S, E, A, t)$ be the statechart associated to $o \in O$. The sequence of states and transitions induced by the sequence of

message events of the form $\langle a_1, \dots, a_n \rangle$, with $a_1 = r(m_1)$ and $a_n = r(m_n)$ and $a_i = r(m)$ or $a_i = s(m)$ and $m_1, m_n, m \in N_{SeqD, o}$, is defined to be the sequence $seq_{stattrans}(r(m_1), r(m_n)) = \langle s_1, a_1, s_2, a_2, \dots, a_{n-1}, s_n \rangle$ such that there exists $(s_n, m_n, s_{n+1}) \in t$ with $s_i \in S$ and $t_{a_i}(s_i, s_{i+1})$ for all $i = 1, \dots, n-1$.

Likewise, the sequence of states and transitions induced by the sequence of message events of the form $\langle a_1, \dots, a_n \rangle$, with $a_1 = r(m_1)$ and $a_n = s(m_n)$ and $a_i = r(m)$ or $a_i = s(m)$ and $m_1, m_n, m \in N_{SeqD, o}$ is defined to be the sequence $seq_{stattrans}(r(m_i), s(m_j)) = \langle s_1, a_1, s_2, a_2, \dots, s_n, a_n, s_{n+1} \rangle$ such that $s_i \in S$ and $t_{a_i}(s_i, s_{i+1})$ for all $i = 1, \dots, n$.

Note that the above defined sequences only exist if the sequence diagram and the statechart diagram are semantically consistent. Indeed, we could also define semantic consistency using this definition and requiring that such a sequence exists. Note further that these sequences need not be uniquely determined, i. e. there might be several different sequences with the same beginning and ending, corresponding to different traversals of the statechart diagram. We explicitly make a distinction between sequences ending with a send event and sequences ending with a receiving event. The latter induce a sequence of states and transitions ending at a state which has a transition with that particular receiving event. This is done due to the run-to-completion semantics of statecharts and important for the calculation of worst-case execution times (see below).

In order to define temporal consistency of a sequence diagram and a statechart we have to consider a concrete software architecture, i.e. a realization of the behavior for a specific execution environment. In many cases this will be a distributed hardware platform. For the realization we have to compile the behavior expressed by all statechart diagrams into executable code according to a certain execution semantics for statecharts. For run-to-completion semantics a statechart compiler has been described in [5]. Furthermore, this tool extracts worst case execution (WCET) times for the reaction of a statechart to the receipt of a message.

For temporal consistency we now have to require that each timing constraint defined in a sequence diagram is kept. Thus we are interested in all events of messages happening between two timepoints (or events in sequence diagrams). We will denote this the critical sequence of a sequence diagram.

Definition 7 (Critical sequence) The critical sequence of a sequence diagram $SeqD$ induced by the timing constraint $t(e_j) - t(e_i) \leq d$ is the sequence $sequence(e_i, e_j)$.

For example, the critical sequence with regards to our first sequence diagram in figure 4 is given by $\langle r(m_0), s(m_1), r(m_1), s(m_2), r(m_2), s(m_3), r(m_3), s(m_4) \rangle$,

$r(m_4)$). Any such critical sequence induces a series of reactions in the statecharts which are associated to the objects of the sequence diagram. Therefore the critical sequence can be split into subsequences with each subsequence associated to one object.

Definition 8 (Message events induced by a sequence)

Given a critical sequence c of a sequence diagram $SeqD=(O,E,V,C)$, the subsequence of message events related to one object $o \in O$ induced by c is the subsequence of all message events related to that object o . Thus the critical sequence can be written as

$$seq^c = subseq_1^{ok} \dots subseq_n^{om}$$

where $subseq_k^{oi} = sequence(r(m_i), r(m_j))$

or $subseq_k^{oi} = sequence(r(m_i), s(m_j))$,

not necessarily $i \neq j$.

For example, the above critical sequence induces two subsequences related to A: $\langle r(m_0), s(m_1) \rangle$ and $\langle r(m_4), s(m_5) \rangle$. Each subsequence induces a sequence of states and transitions:

Definition 9 (States and transitions induced by a sequence)

The critical sequence seq^c induces for each object several subsequences of states and transitions such that each subsequence is related to the reaction of one object O to the receipt of a message according to run-to-completion semantics. Concatenation of all these subsequences yields the induced sequence of states and transitions for a critical sequence which is defined as follows:

$$seq_{stattrans}^c = \langle seq_{stattrans}(subseq_i^{ok}) \rangle = \langle sub_{stattrans,i}^c \rangle \text{ where } seq_{stattrans}(subseq_i^{ok}) \text{ is defined as follows:}$$

We assume that $subseq_i^{ok}$ is part of a greater sequence seq^{ok} induced by c on this object, so $seq^{ok} = subseq_1^{ok} \dots subseq_i^{ok} subseq_{i+1}^{ok} \dots subseq_n^{ok}$

(i) $i < n$ and $n > 1$:

Then $subseq_i^{ok} = sequence(r(m_a), s(m_b))$ and $subseq_{i+1}^{ok} = sequence(r(m_*), s(m_c))$. In this case we define $seq_{stattrans}(subseq_i) = seq_{stattrans}(r(m_a), r(m_*))$

(ii) $i = n$ or $n = 1$:

$subseq_i^{ok}$ is the only subsequence related to this object or it is the last subsequence. Then $subseq_i^{ok} = sequence(r(m_a), s(m_b))$ and we define $seq_{stattrans}(subseq_i^{ok}) = seq_{stattrans}(r(m_a), s(m_b))$.

In our case, the induced subsequences sequence of states and transitions for A is $\langle s_1^A, m_0, s_2^A, /m_6, s_2^A, /m_2, s_3^A \rangle$ and $\langle s_4^A, /m_5, s_1^A \rangle$. Each of these subsequences of states and transitions can now be regarded as a single unit of execution (a task), in this case denoted by TA_1 and TA_2 .

Definition 10 (Time annotated task graph) $S_{Seq} = \{SeqD_1, \dots, SeqD_n\}$ be a set of sequence diagrams. From

the corresponding critical sequences c_1, \dots, c_n we derive a task graph $G_{c_1, \dots, c_n} = (V, E, w)$. The set of nodes is the set of induced sequences of states and transitions as defined before, i.e. $V = \{T_i^c \mid c = c_1, \dots, c_n \text{ and } T_i^c = sub_{stattrans,i}^c\}$. Each element of V is denoted as a task. E is the set of edges such that $E = \{(T_i^c, T_{i+1}^c) \mid T_i^c, T_{i+1}^c \in V\}$. The function w annotates each task with its WCET value.

Figure 9 shows the resulting task graph for our example. The graph is splitted into two parts which reflect the two scenarios of the example. We have five tasks A_1, B_1, C_1, B_2, A_2 for Scenario 1 and three tasks B_3, C_2, B_4 for Scenario 2. For illustrative purposes we have assumed fictitious WCET values of all these tasks. Furthermore, we expect that every communication between tasks needs at most $t_{Comm} = 50$ time units. However, this value will only be considered, if two connected tasks are mapped onto two different processors. Communication costs between two such tasks are ignored if they run concurrently on the same processor. Intuitively, we can now decide whether a timing constraint is kept by adding up all worst-case execution time values and communication costs for one critical sequence and comparing it to the timing constraint in the sequence diagram.

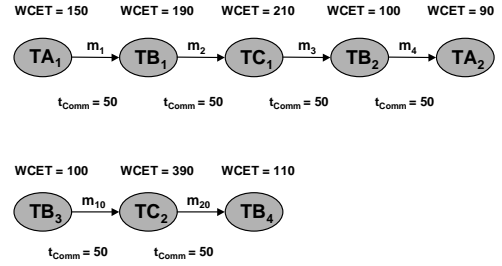


Figure 9. Time annotated task graph for the example

We therefore come to the following definition for temporal consistency:

Definition 11 (Temporal Consistency) Let $S_{Seq} = \{SeqD_1, \dots, SeqD_n\}$ be a set of sequence diagrams with associated critical sequences c_1, \dots, c_n and S_{Stat} be the set of all statechart diagrams associated to objects occurring in a $SeqD_i$. Let further be P be a set of processors and $G_{c_1, \dots, c_n} = (V, E, w)$ be the time annotated task graph induced by the critical sequences. Then S_{Seq} is temporally consistent with S_{Stat} if there exists:

- an allocation that assigns each task $T \in V$ to a processor $p_i \in P$ such that all tasks of one object are assigned to one processor.

- a schedule for all tasks $T \in V$ such that the timing constraint of each c_i is kept, meaning that the summation of all worst-execution time and communication costs along a critical sequence is less than the timing constraint of this critical sequence.

This definition is kept as general as possible, not making any assumptions on the underlying scheduling policy. On the basis of this theoretical definition we will now develop a practical approach for ensuring the temporal consistency for a set of sequence diagrams and statecharts.

3 A Practical Approach towards Temporal Consistency

In this section we will show how the notion of temporal consistency can be supported by the integration of existing tools for real-time systems. Research on real-time systems has produced a rich set of results for the analysis of such systems. We argue that these results can be used to form a seamless UML based design process for embedded real-time systems. The other two notions of consistency are inherent to a UML-RT model and should be best tackled within such an environment. With our focus on real-time systems we will restrict ourselves to temporal consistency in the following.

Our proposal consists of the following steps to validate timing constraints of a system given as a UML-RT model w.r.t. to a concrete implementation¹:

1. Build the task graph according to Definiton 10.
2. For a given execution platform perform an allocation of the tasks and a schedulability analysis.
3. Re-scale the sequence diagrams of the model in order to reflect the temporal layout of the execution trace of the objects.

The schedulability analysis of step 2 is based on the tool CHaRy that has been presented in [1, 17]. This tool checks whether a given allocation of tasks onto the available hardware platform can be realized - according the specification of the temporal constraints, the interference of tasks being mapped onto the same node and possibly servicing different messages concurrently.

To demonstrate the proposal for our example we suppose that the hardware platform consists of two identical processors P_1 and P_2 connected by a communication line.

¹Another aspect of timing analysis would be to check whether timing constraints within a model are consistent i.e. they are contradictory in themselves. We do not elaborate on this further. However, some of the methods proposed in this paper could also be used to validate this kind of consistency.

| | Task | Start | WCET | End | Load |
|-----------------|--------|-------|------|------|-------|
| Processor P_1 | TA_1 | 0 | 150 | 150 | 20.8% |
| | TA_2 | 1246 | 100 | 1346 | |
| Processor P_2 | TB_3 | 0 | 100 | 100 | 90.7% |
| | TB_1 | 201 | 190 | 391 | |
| | TC_2 | 392 | 390 | 782 | |
| | TC_1 | 783 | 210 | 993 | |
| | TB_4 | 994 | 110 | 1104 | |
| | TB_2 | 1105 | 90 | 1195 | |

| | Task | Start | WCET | End | Load |
|-----------------|--------|-------|------|------|-------|
| Processor P_1 | TA_1 | 0 | 150 | 150 | 70.8% |
| | TC_2 | 153 | 390 | 543 | |
| | TC_1 | 546 | 210 | 756 | |
| | TA_2 | 952 | 100 | 1052 | |
| Processor P_2 | TB_3 | 0 | 100 | 100 | 38.0% |
| | TB_1 | 203 | 190 | 393 | |
| | TB_4 | 596 | 110 | 706 | |
| | TB_2 | 809 | 90 | 899 | |

Table 1. Results of the Scheduling Analysis

Furthermore, we will assume a very simple non-preemptive scheduling strategy. All tasks belonging to one of the object A , B and C will be mapped onto one processor, thus we have three possible allocations of the tasks. For the purpose of illustration we have set the deadline and the period to the same value of 1200 time units, i.e. $d_1 = 1200$ in Scenario 1 and $d_2 = 1200$ in Scenario 2.

The following tables summarize the results of the schedulability analysis for two of these allocations. Each table shows the order in which the subtasks are scheduled onto both processors. Furthermore, for each task three time values are given. The start time indicates when the task will be scheduled. The entry in the WCET column corresponds to the value in Figure 9. And finally, the end time indicates when the subtask is expected to finish at latest². The last column gives the calculated load for each processor.

The allocation that corresponds to the first table does not yield a feasible schedule. Task TA_2 might finish 1346 time units after the start of the period and thus would violate the deadline requirement of 1200 time units. The second table shows the results for a feasible schedule. We propose to reflect this information also in the sequence diagrams themselves - a visual means and back-annotation for the developer. Introducing a time scale on the vertical time lines and scaling and positioning the active execution phases of each task on that time axis would allow to visualize the calculated

²The given values slightly deviate from what would be expected by simply adding them. This is caused by small latency values for communication start-up, that have been taken into account

scheduling. Currently, any kind of scheduling is not represented at all in UML tools. In Figures 10 and 11 we have depicted such re-scaled sequence diagrams for the schedule and allocation related to the second table. The shaded boxes show the tasks that are active within each sequence. In each diagram, the “complementary tasks” are also depicted.

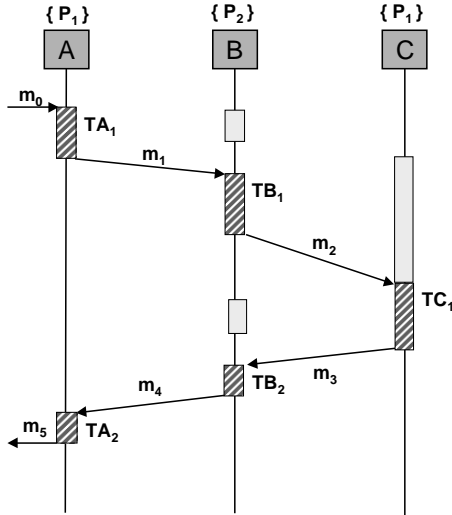


Figure 10. Re-scaled Scenario 1

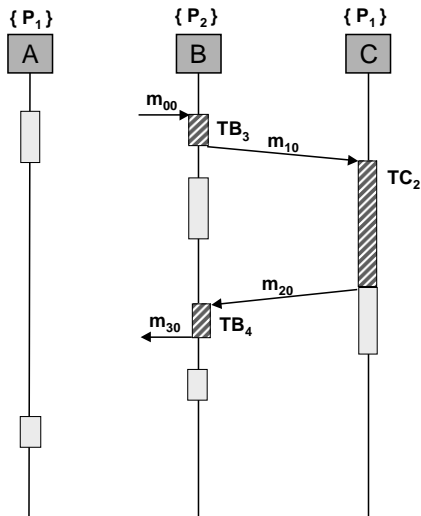


Figure 11. Re-scaled Scenario 2

4. Related Work

The UML and its real-time extension UML-RT provide different models for specifying the dynamics of a system from different points of view. Although the question of consistency is naturally arising it has not been much covered

by existing work [3], possibly due to the missing formal semantics of UML. Ideas of syntactical, semantic and temporal consistency have been developed in [10] which served as a basis for this work. An approach to the consistency of protocol descriptions given as protocol statecharts and consistency of capsule statecharts with the given protocol statechart is described in [4] which can be considered as another aspect of consistency, both syntactical and semantic, within UML-RT.

So far, there are mainly approaches focusing on the synthesis of statecharts from sequence diagrams (or a variant of them). Harel et al. [8] describe an approach to synthesize statecharts for objects from a set of Live Sequence Charts (LSCs, an extension of message sequence charts) describing different scenarios. After establishing consistency conditions for a set of LSCs a method for synthesizing statecharts for objects from such a specification is described. Briefly, the method relies on constructing a global system automaton from which statecharts for single objects are constructed. On the contrary to the work described in this paper, their approach does not deal with timing constraints. Furthermore, their approach has limitations in scenarios where existing statecharts and sequence diagrams have to be checked for consistency.

Alur et al. [2] study possibilities to derive implied scenarios from a set of scenarios described by MSCs. They apply a synthesis algorithm which generates statecharts for an object given a set of scenario descriptions in a form of MSCs. However, their focus is mainly on detecting implied scenarios and in avoiding deadlocks.

With respect to timing constraints in sequence diagrams, Li et al. [11] describe an algorithm based on linear programming that analyzes whether several timing constraints within a sequence diagram are consistent with each other. They extend their approach to compositions of sequence diagrams. On the contrary to our work, they do not deal with the consistency of sequence diagrams and statecharts and execution times derived from them. Other approaches [6] use model checking and timed automata for the verification of timing constraints.

The integration schedulability analysis with the design model has also been inspired by Saksena et al. [14]. They describe how schedulability analysis can be combined with object-oriented design distinguishing between single-threaded and multi-threaded implementations.

A related approach is realized by the tool “TimeWiz for Rational Rose RT” from TimeSys Corporation. UML-RT models are exported to this tool which then analyses timing constraints. We argue that the integration of analysis tools needs to be tighter, e.g. by considering results of architecture specific code generation and WCET analysis as mentioned in our proposal such as the back-annotation of sequence diagrams.

5. Conclusion and Future Work

In this paper we have presented a method to ensure the consistency of a design model in UML-RT. Motivated by a practical example we have first discussed the concept of consistency and explained the notions of syntactical, semantic and real-time consistency. Then we have focused on the consistency of sequence diagrams and statecharts and formalized the previous notions of consistency for these sublanguages of UML-RT. In particular, we have given a general definition for temporal consistency between a set of sequence diagrams and statecharts, showing that the allocation and scheduling is also of major importance to temporal consistency. On the basis of these discussions, we have then described a method for ensuring the temporal consistency, reusing existing well-known real-time analysis techniques. First, WCETs must be computed. Then the timing constraints together with the results of the WCET analysis are given to a scheduling tool which computes a schedule. Given such a schedule, we can either validate or invalidate the timing constraints and finally the results are described using an annotated sequence diagram.

There is plenty of room for future work. Concerning our definitions, these must be extended to more complex statecharts. Furthermore, the consistency of timing constraints within a sequence diagram should also be considered which we assume as given. With respect to the practical analysis of timing constraints, we make the prerequisite that all scenarios are modeled in sequence diagrams. With respect to tool support, tools for syntactical, semantic and temporal consistency should be developed which also support the back-annotation of sequence diagrams.

6. Acknowledgements

This research has been performed in the scope of the ITEA project 99012 DESS that is nationally funded by the German Federal Ministry of Education and Research under contract number 01 IS 903C. The authors would like to thank their colleague Reiko Heckel for his suggestions on the formal definitions of consistency and their colleague Peter Altenbernd for his support in applying the CHaRy toolset.

References

- [1] P. Altenbernd. CHaRy: The C-LAB Hard Real-Time System to Support Mechatronical Design. In *Proceedings of ECBS'97*, Monterey, CA, 1997.
- [2] R. Alur, K. Etessami, and M. Yannakakis. Inference of message sequence charts. In *22nd International Conference on Software Engineering*, 2000.
- [3] G. Engels and L. Groenewegen. Object-oriented modeling: A roadmap. In A. Finkelstein, editor, *Future Of Software Engineering 2000*, pages 105–116. ACM, June 2000.
- [4] G. Engels, L. Groenewegen, and J. M. Küster. Modelling concurrent behaviour through consistent statechart views. In G. Reggio, A. Knapp, B. Rumpe, B. Selic, and R. Wieringa, editors, *Proceedings of the Workshop Dynamic Behaviour in UML Models: Semantic Questions*, pages 44–49. LMU München, TR-0006, Oct. 2000.
- [5] E. Erpenbach, F. Stappert, and J. Stroop. Timing Analysis of Statecharts Models for Embedded Systems. In *The Second International Workshop on Compiler and Architecture Support for Embedded Systems (CASES'99)*, Washington, Oktober 1999.
- [6] T. Firley, M. Huhn, K. Diethers, T. Gehrke, and U. Goltz. Timed sequence diagrams and tool-based analysis A case study. In R. France and B. Rumpe, editors, *UML'99 - The Unified Modeling Language. Beyond the Standard. Second International Conference, Fort Collins, CO, USA, October 28-30. 1999, Proceedings*, volume 1723 of LNCS, pages 645–660. Springer, 1999.
- [7] D. Harel. Statecharts: A visual formulation for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.
- [8] D. Harel and H. Kugler. Synthesizing state-based object systems from LSC specifications. Technical Report MCS99-20, Weizmann Institute of Science, Oct. 1999.
- [9] D. Harel and B. Rumpe. Modeling languages: Syntax, semantics and all that stuff. Technical Report MCS00-16, Weizmann Institute of Science, 2000.
- [10] J. M. Küster and J. Stroop. Towards consistency of dynamic models and analysis of timing constraints. *1st Workshop on Formal Design Techniques for Real-Time Systems at UML'2000*, Oct. 2000.
- [11] X. Li and J. Lilius. Timing analysis of UML sequence diagrams. In R. France and B. Rumpe, editors, *UML'99 - The Unified Modeling Language. Beyond the Standard. Second International Conference, Fort Collins, CO, USA, October 28-30. 1999, Proceedings*, volume 1723 of LNCS, pages 661–674. Springer, 1999.
- [12] Object Modeling Group. *Unified Modelling Language Specification, version 1.3*, June 1999. URL: uml.shl.com:80/docs/UML1.3/99-06-08.pdf.
- [13] A. Peron and A. Maggiolo-Schettini. Transitions as interrupts: A new semantics for timed statecharts. *Lecture Notes in Computer Science*, 789:806–824, 1994.
- [14] M. Saksena and P. Karvelas. Designing for schedulability integrating schedulability analysis with object-oriented design. In *Proceedings of the 12th Euromicro Conference on Real-Time Systems*, 2000.
- [15] B. Selic. Using UML for modeling complex real-time systems. *Lecture Notes in Computer Science*, 1474:250–262, 1998.
- [16] B. Selic, G. Gullekson, and P. Ward. *Real-Time Object Oriented Modeling*. John Wiley & Sons, 1994.
- [17] F. Stappert and P. Altenbernd. Complete Worst-Case Execution Time Analysis of Straight-line Hard Real-Time Programs. *Journal of Systems Architecture*, 46(4):339–355, 2000.