Methodological Guidelines on the Usage of MARTE VSL for Specification of Time Constraints

A. Cuccuru¹, C. Mraidha¹, A. Radermacher¹, S. Gérard¹
L. Rioux², T. Vergnaud², O. Hachet²
¹CEA, LIST, Point Courrier 94, Gif-sur-Yvette, F-91191 France
²Thales Research and Technology, 1, Avenue Auguste Fresnel, F-91767 Palaiseau, France

Abstract—One of the major benefits of Model-Based Engineering (MBE) languages for Real-Time and Embedded (RTE) systems development is their capacity to allow the specification of complex non-functional properties and constraints in a declarative manner and the refinement of these properties towards the implementation. The UML profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) provides concepts to deal with non-functional aspects at different levels of abstraction as well as a concrete syntax for their textual representation: the Value Specification Language (VSL). The usage of VSL is quite straightforward for the specification of values. In the case of constraint specifications, using VSL may appear less intuitive, especially if the constraint concerns timing aspects. In order to let users benefit from the expressivity of VSL, this paper gives some methodological guidelines on the usage of VSL for the specification of time constraints. Propositions are illustrated via a simple component-oriented model example and put into action thanks to the UML modeler Papyrus and its plugin for MARTE.

Index Terms—MARTE, VSL, Time, Constraint

I. INTRODUCTION

Specification of values for properties of model elements is a frequent need in Model-Based Engineering (MBE). While this issue exists in general purpose modeling languages such as UML[3][4] (where OCL can provide a solution), it becomes crucial in the domain of Real-Time and Embedded (RTE) systems, where one key aspect of the modeled systems concerns the non-functional properties associated with model elements. For example, if a given model captures a hardware platform, associated hardware components must be characterized with non-functional aspects such as bandwidth, throughput or energy consumption (each characterization involving both the specification of a value and a unit) such that this information can be unambiguously used for further model-based design or analysis activities.

The Value Specification Language (VSL), standardized in the context of the UML profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE)[2], provides a formal solution to this concrete modeling issue. Coupled with the Non-Functional Properties (NFP) sub-profile of MARTE (which can be used to specify the complex types behind non-functional properties of a system), VSL covers all the expressivity needs implied by a typical RTE design flow.

Beyond specification of values (which is the most common use case), other usages of VSL are also possible. The specification of constraints is such an application area, where the specification of constraints can benefit from the expressivity of VSL for non-functional values. This use case, which is mentioned in the MARTE specification, simply relies on the fact that VSL is fundamentally a typed expression language. The intuitive idea is that a VSL expression denoting a constraint must be a Boolean expression.

While the principle is simple, putting it into action is not straightforward, especially when values manipulated in a constraint are related to timing aspects. The underlying complexity is mainly due to the way modeling constructs related to constraints and time are considered in the standard UML meta-model. As Time is one of the key considerations in RTE design, simple methodological guidelines are clearly required for users who may want to benefit from the expressivity of VSL for their application domain, while not being experts of the OMG standards.

The purpose of this article is to provide such methodological guidelines on the usage of VSL for specification of time constraints. These guidelines result from a careful analysis conducted in the context of the ITEA 2 project VERDE (http://www.itea-verde.org/). In section 2, we start by providing a brief overview of the VSL syntax and semantics and illustrate its usage on some typical non-functional property examples. Section 3 then provides precise modeling guidelines on the usage of VSL for specification of time constraints. Relationships with underlying UML concepts are highlighted and focus is given to the usage of the TimeObservation concept. In section 4, we illustrate the guidelines on the basis of a simple component-oriented model example, where a UML interaction (capturing message
exchanges between components and depicted in the form of a sequence diagram) is augmented with some time constraints specified with the help of VSL. Section 5 briefly positions VSL with respect to the Clock Constraint Specification Language (CCSL, also standardized in the context of MARTE), a language dedicated to the manipulation of time. Section 6 then concludes this article and sets objectives for future works.

II. AN OVERVIEW OF VSL

VSL proposes an expression language which might be used by any other UML-based specification interested in extending the base expression infrastructure provided by UML. This MARTE expression language is an extension to the “Value specification” and “DataType” concepts provided by UML. VSL deals with the following requirements:

- How to specify parameters/variables, constants, and expressions in textual form.
- How to specify relationships between different parameters/variables, or constant values are defined.
- Arithmetic, logical, relational, and conditional expressions.
- How different time values and assertions are defined in UML.
- How to specify composite values such as collection, interval, and tuple values.

VSL expressions can be used to specify non-functional values, parameters, operations, and dependency between different values in a UML model, as illustrated in Table 1.

### Table 1. Example of VSL expressions

<table>
<thead>
<tr>
<th>Value Spec.</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real Number</td>
<td>1.2E-3</td>
</tr>
<tr>
<td></td>
<td>//scientific notation</td>
</tr>
<tr>
<td>DateTime</td>
<td>#12/01/06 12:00:00#</td>
</tr>
<tr>
<td></td>
<td>//calendar date time</td>
</tr>
<tr>
<td>Collection</td>
<td>(1, 2, 88, 5, 2)</td>
</tr>
<tr>
<td></td>
<td>//sequence, bag, ordered set..</td>
</tr>
<tr>
<td></td>
<td>((1,2,3), (3,2))</td>
</tr>
<tr>
<td></td>
<td>//collection of collections</td>
</tr>
<tr>
<td>Tuple and choice</td>
<td>(value=2.0, unit=ms)</td>
</tr>
<tr>
<td></td>
<td>//duration tuple value</td>
</tr>
<tr>
<td></td>
<td>periodic(period=2.0, jitter=3.3)</td>
</tr>
<tr>
<td></td>
<td>//arrival pattern</td>
</tr>
<tr>
<td>Interval</td>
<td>[1..251]</td>
</tr>
<tr>
<td></td>
<td>//upper opened interval between integers</td>
</tr>
<tr>
<td>Variable declaration &amp; Call</td>
<td>(s1</td>
</tr>
<tr>
<td></td>
<td>//input/output variable declaration</td>
</tr>
<tr>
<td>Arithmetic Operation Call</td>
<td>+0.1, var1</td>
</tr>
<tr>
<td></td>
<td>//add operation on Real datatypes</td>
</tr>
<tr>
<td></td>
<td>5.0*var1</td>
</tr>
<tr>
<td></td>
<td>//Infix operator notation</td>
</tr>
<tr>
<td>Conditional Expression</td>
<td>((var1&lt;6.0)?(10^6):1)</td>
</tr>
<tr>
<td></td>
<td>//if true return 10 exp 6, else 1</td>
</tr>
</tbody>
</table>

III. A MODELLING PATTERN FOR SPECIFYING TIME CONSTRAINTS WITH VSL

The following modeling pattern has been defined in the context of the ITEA 2 project VERDE, mentioned in the introduction to this article. This pattern consists in enriching a UML Interaction (which describes a set of observable communication exchanges between multiple communicating elements) by specifying time constraints related to communication events (graphically represented in a sequence diagram by the ends of messages). The expression of constraints rely on the usage of time observations (see UML Superstructure v.2.3 Section 13.3.30 TimeObservation), which are then further manipulated in a VSL expression actually constraining these time observations.

In the context of VERDE, this pattern has been modeled using the Eclipse Process Framework (EPF1), an eclipse-based tool supporting SPEM (the Software & Systems Process Engineering Metamodel OMG specification [5]). The following description therefore follows the decomposition implied by the EPF model. Section “Key considerations” identifies potential usages of stereotypes or concepts from the MARTE profile. Section “Input work products” identifies model elements considered as input of the modeling process. Section “Steps” identifies the various modeling steps implied by the modeling pattern. Section “Output work products” finally identifies the model element resulting from the modeling process.

A. Key considerations

This task may involve the following stereotypes or elements from the MARTE profile (see UML Profile for MARTE v1.1):

- <<TimedInstantObservation>> (see section 9.3.1.4), used to enrich UML time observations,
- <<TimedConstraint>> (see section 9.3.1.3), used to enrich UML constraints,
- VSL (see Annex B), used to capture formal expressions specifying the actual constraint body.

B. Input work products

- UML Interaction. A UML Interaction is a kind of UML behavior focusing on the description of exchanges between communicating entities. It is most of the time depicted as a sequence diagram.

C. Steps

- Create time observations: This step consists in creating time observations (in the context Interaction), so that they can further be referenced in VSL expressions, formally capturing timing constraints. Each time observation will typically refer to a communication event associated with a message from the interaction, or to an execution occurrence. Graphically, a time observation is represented by the symbol “@” followed by the name of the time observation. The fact that a time observation

1 http://www.eclipse.org/epf/
is actually bound to a communication event of a message will be graphically captured by having the time observation located at the corresponding end of the message.

- **Create a constraint**: In order to encapsulate an expression that will actually describe the timing constraint, a UML Constraint must be created. This constraint is typically owned by the context UML Interaction, and can additionally refer constrained elements (e.g., the time observations that will be manipulated in the VSL expression). Note that these additional references have no semantic impact on the VSL specification of the constraint, in the sense that it does not restrict the set of time observations that can be manipulated in the expression. Therefore, it can be seen as additional information making the model potentially easier to read or exploit.

- **Specify the constraint body with VSL**: Once the UML constraint has been created, a VSL expression can be encapsulated in it. Encapsulating the VSL expression involves the usage of an OpaqueExpression. The property language of the opaque expression must contain the string "VSL", and the property body must contain the VSL expression. Since properties language and body are ordered collections, the indexes of "VSL" and of the VSL expression (in their respective collection) must be the same (see Section 7.3.35 OpaqueExpression from the UML superstructure).

The VSL expression must be a Boolean expression (i.e., an expression whose evaluation will produce a result of type Boolean), which will typically make reference to time observations. For example, if the context interaction defines time observations @t1 and @t2, the following VSL expressions could be specified (this list is of course not exhaustive):

- \( \@t1 < \@t2 \), which specifies that the event associated with the time observation \( \@t2 \) must occur before the event associated with the time observation \( \@t1 \) (shortly, \( \@t1 \) must occur before \( \@t2 \))
- \( \@t2 > 11:43:45 \ 2010/09/21 \), which specifies that \( \@t2 \) must occur after a date literally specified
- \( \@t2 - \@t1 < \{ \text{value} = 15.0, \text{unit} = \text{ms} \} \), which specifies that the duration between the occurrence of \( \@t1 \) and occurrence of \( \@t2 \) must be lower than 15.0 milliseconds
- \( \@t2 - \@t1 < 15.0 \), which roughly specifies the same thing as the previous constraints, without specifying the time unit (i.e., it can be implicit from the context, or can be indirectly obtained from another model element, as illustrated in the last step of this task).

- **(Optional) Refine time observations with the MARTE stereotype <<TimedInstantObservation>>**: As explained in the previous steps, in the context of an Interaction, a TimeObservation is bound to a communication event (i.e., the emission or reception of a message), and can therefore be literally interpreted as a specification of the instant where a message is emitted or received. However, this explanation only remains an interpretation.

In order to avoid any ambiguity on the interpretation of the event observed via the time observation, MARTE provides a stereotype : <<TimedInstantObservation>>. With the property obsKind : EventKind of this stereotype (possible values are start, finish, send, receive, consume), it is possible to indirectly characterize the event associated with the time observation. For example, if we have:

- \( \@t1 \) with <<TimedInstantObservation>> \{obsKind = send\}, a VSL expression such as \( \@t1 > 17:25 \) that the emission of the event underlying \( \@t1 \) must be done after a literally specified date
- \( \@t1 \) with <<TimedInstantObservation>> \{obsKind = consume\}, a VSL expression such as \( \@t1 < 17:25 \) that the event underlying \( \@t1 \) must be consumed before a literally specified date. This can be used to specify the validity date of a message.

- **(Optional) Refine constraints with the MARTE stereotype <<TimedConstraint>>**: From a given expression context, it is possible to determine if a constraint actually concerns a particular instant (e.g., \( \@t2 > 11:43:45 \ 2010/09/21 \)) or a duration (e.g., \( \@t2 - \@t1 < \{ \text{value} = 15.0, \text{unit} = \text{ms} \} \)). Determining if the constraint refers to an instant or a duration typically requires an interpretation phase (which can be automated since the VSL syntax is formally defined), with an inference mechanism exploiting the content of the expression (e.g., the time events it refers to and the operator which are manipulated) as well as the context in which it is specified.

MARTE provides a stereotype which enables to explicitly tag a constraint as an "instant" and/or a "duration" constraint: <<TimedConstraint>>, which extends the UML metaclass Constraint. By applying the stereotype on a Constraint, it is possible to specify how the constraint must be interpreted, using the property interpretation: TimeInterpretationKind (possible values are instant and duration). If interpretation is set to the enumeration literal instant, then the constraint is interpreted as a constraint on instant value. If interpretation is set to the enumeration literal duration, then the constraint is interpreted as a constraint on duration value.

Note that the stereotype <<TimedConstraint>> also inherits from stereotypes <<TimedElement>> (see Section 9.3.2.7 of the MARTE specification) and <<NfpConstraint>> (see Section 8.3.2.5 of the MARTE specification).

With the property on : Clock (inherited from TimedElement), it is possible to reference a clock, which can itself associated with a time unit (e.g., seconds, milliseconds, ticks, etc.). Considering a VSL expression such as \( \@t2 - \@t1 < 15.0 \), this can be used to indirectly specify the time unit behind the real literal "15.0".

With the property kind: ConstraintKind (inherited from NfpConstraint), it is possible to further characterize the timed constraint (typical values are required or offered). Required indicates that the constraint represents a minimum quantitative or qualitative level. Offered establishes that the constraint...
represents the space of values that the constrained elements can afford.

**D. Output work products**

- **UML Interaction with timing constraints**

The next section illustrates the usage of this pattern in the context of a component-based architecture design. In addition, it is important to notice that the same modeling pattern can be applied using UML DurationObservation. In VSL expressions as well as in diagrams, duration observations are denoted using the symbol ‘&’ followed by the name of the observation. Note also that VSL allows the expression of instant or duration observations with an occurrence index. For example, we can express the i-th occurrence of the \( t_1 \) instant observation with the following syntax: \( @t_1(i) \). VSL also introduces an additional syntax for expressing jitters, i.e. a special duration expression that specifies an unwanted variation in the instants when periodic events should occur.

**IV. SPECIFYING TIME CONSTRAINTS WITH VSL: EXAMPLE**

This section illustrates the use of the MARTE VSL to specify timed constraints in a component-based architecture design. We consider a very simplified architecture of an on-board satellite software (OBSW).

The architecture consists of three high-level components, as represented in the composite structure diagram of Figure 1: one component for the ground station, and two components for the satellite software itself. The first component of the satellite software, swBus, deals with communications with the ground station (and possibly with other software satellite components that are not represented). The second component, swCore, implements the on-board software itself (altitude and orbit control, energy management, management of the on-board payload, etc.).

Communications between ground and satellite are done through telecommands (commands from the ground) and telemesures (data from the satellite) that conform to space communication standards. They can be considered as one way messages. The communications between the software bus and the software core are mostly operation invocations.

Such software runs on real-time critical systems (as it is sent into space, where maintenance is very complex). Therefore, it is very important to be able to specify execution times for each operation in the workflow in order to correctly allocate the threads that will carry the execution of these operations.

Scenarios can be defined to describe communications between the different components, using sequence diagrams. Time constraints and VSL notations can be used in these diagrams to specify execution and communication times, as illustrated in Figure 3.

![Figure 1: Composite structure diagram of example](image)

In this scenario, the ground first emits a command. The instant when the command is emitted is identified by the time observation \( @t_{\text{Emit}} \). swBus propagates the command to swCore, by a synchronous invocation of method `processCommand()`. The time observation \( @t_{\text{EndOfProcessing}} \) identifies the moment when the processing of the command by swCore is finished. The constraint illustrated in the bottom of Figure 3 specifies that the duration between the moment when the command is emitted by the ground station and the moment when the corresponding processing is performed must be lower than 20 milliseconds.

This expression first relies on the fact that in VSL, a time observation expression (such as \( @t_{\text{Emit}} \)) is of type `DateTime`, a primitive data type defined in the TimePackage of VSL representing a date. This data type defines operator “-”, whose return value is of type `NFP_Duration`. The result of \( @t_{\text{EndOfProcessing}} - @t_{\text{Emit}} \) can therefore be compared to \{unit = ms, value = 20.0\}, a literal specification of an `NFP_Duration` specified using the standard VSL syntax for tuples. The constraint finally relies on the usage of the MARTE stereotype TimedConstraint to specify that the constraint is related to a duration (property interpretation) and represents a required QoS (property kind).

Such specifications can then be used to refine the model of the system under design, for example by setting thread

![Figure 3: Time Constraint Example](image)
configurations (periods, etc.) that match the architecture requirements with respect to scheduling analysis or execution simulations.

V. VSL VS. CCSL

Users interested in the description of more complex time relationships are encouraged to use CCSL[1] (Clock Constraint Specification Language). In CCSL, time is considered as a set of instances of time values making reference to Clocks. The language then provides mechanisms to describe relationships between instants or clocks. The VSL is actually one brick that must be combined with other elements (e.g. CCSL) to address the complete range of problematic of a realistic application architecture model.

We can use CCSL to enhance our architecture example. If we suppose that the satellite we modeled is not geostationary, then communications with earth can only occur when the satellite is above the ground station. This depends on the revolution period of the satellite. The Satellite has to transmit its data only when its position is vertical above the GroundStation (+/- 5 degree, as illustrated in Figure 4), otherwise the satellite is collecting data.

By providing a brief positioning of VSL with respect to CCSL, we have highlighted the fact that VSL is meant to be used together with other UML and / or MARTE elements in order to provide a complete modeling solution. As a consequence, the use of the VSL must be integrated in a general methodological approach. In the VERDE project, we define such a methodological approach that focuses on the resolution of particular modeling issues. Thus, VERDE defines process sub-elements that can be integrated within a design process, just like design patterns provide solutions to address well identified programming problems.

VI. CONCLUSION

We have proposed a modeling pattern for expression of timing constraints using VSL. Relying on an EPF model of the pattern, we have precisely identified the various modeling steps involved as well as relationships with underlying UML concepts.

REFERENCES

[1] Time modeling in MARTE, Charles André, Frédéric Mallet, Robert de Simone, I3S, Aoste Project, I3S/INRIA.