Modeling Requirements Should be Language Agnostic!
Example of a Formal Definition of Simple Behavioral Decomposition Models

Gurvan Le Guernic
DGA Maîtrise de l'Information, 35998 Rennes Cedex 9, France

Keywords: Modeling Language, Domain Specific, Modeling Requirements, Model Validation, Formal Definition.

Abstract: This paper argues in favor of expressing modeling requirements in a modeling language agnostic way, at least whenever those requirements are part of a contracting relationship between some of the stakeholders. Contracting authorities may require from contractors specific design models in order to perform early design (functional, safety, security, etc.) analyses. However, in order to allow contractors to use the compatible modeling language of their choice with the compatible modeling guidelines of their choice, the formal definition of the model requirements must be independent from any concrete modeling language or guideline. This paper introduces, based on the example of Behavioral Decomposition Models, an approach to express such agnostic requirements. This is achieved by defining a semantic domain, some correctness constraints and, later, the necessary mappings between them and the desired concrete syntaxes.

1 INTRODUCTION

The DGA (Direction Générale de l'Armement) is the French military procurement agency. As such, it buys off-the-shelf products, but also emits requests for proposals (or tenders) regarding the development of specific products. With regard to specific products, the DGA MI (DGA Maîtrise de l'Information) is involved, among other things, with products built to secure information systems. Those products often require the development of specific hardware, have to satisfy strong non-functional requirements, and have to pass strict national or international evaluation processes before being deployed. Due to those aspects, their development does not fit well with agile processes (Cockburn, 2007), and usually follows an overall Vee model of development (Forsberg and Mooz, 1998; INCOSE, 2015). However, the development process implemented by contractors in interaction with the DGA still follows one of the pillars of agile development and IBM's Continuous Engineering (Shamieh, 2014): *early feedback*. The DGA is an active contracting authority which has accumulated knowledge on how to build a product that satisfies the strong non-functional requirements desired, and on the standards a product needs to meet in order to pass the strict national or international evaluation processes. Near the end of the development process and as could be expected, the DGA performs evaluation of the first versions of the product in order to advise the prime contractor on modifications to perform in order to meet DGA's requirements and the standards of the national or international evaluation processes. In addition, in the early phases of the development process, the DGA requires the prime contractor to deliver design documentation. Those documents are analyzed by the DGA in order to spot, as early as possible, bad design decisions which would prevent the proposed system from meeting certain requirements (either from the DGA or from the evaluation authorities).

As it is well known, natural language textual documents suffer from incoherences, ambiguities or incompleteness. In order to improve the quality of its design analyses, the DGA is moving towards a process where some of the design documentation is required to include (semi-)formal models. In order to maximize the benefits of using models, those models are required to follow strict constraints. Without such constraints (and even with them but to a lesser degree), it is still possible to produce an incomplete model full of incoherences and ambiguities. Moreover, those constraints ensure that the analysis team is able to easily identify the information it needs to perform its analyses. In addition, those constraints are also used as "smoke and sanity tests" (Mcconnell, 1996; ISTQB Glossary Working Group, 2015) before accepting the design documentations in order to ensure a minimal quality of the models and therefore an efficient use of the time and effort of the DGA's design analysis teams. However, those constraints should minimize the burden they impose on the con-

Guernic, G.
Modeling Requirements Should be Language Agnostic! - Example of a Formal Definition of Simple Behavioral Decomposition Models.
DOI: 10.5220/0005795205550562
In *Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2016)*, pages 555-562
ISBN: 978-989-758-168-7

tractors. Ideally, they should not prevent contractors from applying the development processes and tools they are used to. In particular, if a contractor is already using a particular type of modeling language supported by some specific tools, he should still be able to use those while providing to the DGA the models it needs.

One of DGA's early design analyses is focused on the specification of high level interleavings of environment solicitations and product reactions (under the form of messages emitted or operations performed). The goal is mainly to verify that user scenarios (Alexander and Maiden, 2004) are allowed by the current specification. For example, for a generic OS, user scenarios may request that it is possible to switch off a computer without logging in if no user is currently logged in (or potentially the reverse for a device with high availability requirements); other scenarios may request that a given operation is always possible. Pushing this analysis further, it is also verified that the first level of product decomposition into components allows the black box behaviors specified (for example, verifying that interactions between component behaviors can not deadlock the whole product). In order to perform those analyses a so called Behavioral Decomposition Model (BDM) is requested. BDMs are formally defined by a set of modeling requirements. Those modeling requirements are similar to well-formedness constraints (Kolovos et al., 2009); however, they are expressed independently from the concrete modeling language used, which is not usually the case for well-formedness constraints.

This paper does not focus on BDMs, but on an example of how such an agnostic[1] model definition can be formalized. This paper presents in Sect. 2 a running example before introducing in Sect. 3 the proposed approach to formally define modeling requirements independently from the specific modeling language which will be used to concretely model the product or system. Section 4 presents an application of this approach on the example of a simplified formal definition of BDM. And possible exploitations of those formal agnostic definitions are proposed in Sect. 5 before concluding.

2 RUNNING EXAMPLE

The running example used in this paper is a simplified network pump (Moore, 2000), called NP and based on the American "Pump" architecture (Kang et al., 2005) roughly described in Fig. 1.

[1] "independent of any concrete modeling language"

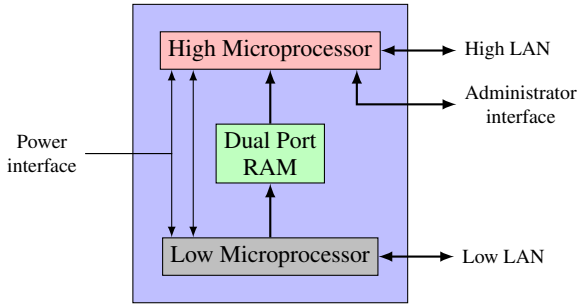

Figure 1: Simplified architecture of the "Pump" (Kang et al., 2005)

In order to save space and simplify the running example to the minimum required for this paper, NP's structural architecture is simplified to two organs[2]: a high part and a low part. The power interface is connected to both parts and its behavior is simplified to only receiving information about the state of the main power and battery. NP's behavior is reduced to: forwarding on the high LAN interface every packet received on the low LAN interface; and providing a log file on the administrator interface if requested on the same interface.

Fig. 2 provides some diagrams of a SysML (OMG, 2012b) model of the running example. Those are only "views" on the underlying SysML model. Indeed, diagrams are not part of the SysML meta-model. And, for example, the environment state machine diagram has been produced with a tool (Papyrus (Gérard et al., 2010; The Eclipse Foundation, 2015)) which, near a transition, displays its name, and not a label constructed from the trigger, guard and effect of the transition. It is the naming convention of those transitions that ensures that a transition label reflects information on its trigger, guard and effect. However, even if Papyrus does not follow strictly the graphical representation of SysML diagrams, the underlying (abstract syntax) SysML model produced by Papyrus still respects the SysML meta-model. And transitions in this underlying SysML model do have trigger, guard and effect, if needed. The agnostic requirements defining BDMs have to be checked on the (abstract syntax) underlying model, and not on the graphical (concrete syntax) diagrams.

[2] In this paper, the word "organ" is used with the following definition: "a subordinate group or organization that performs specialized functions" (Merriam-Webster)

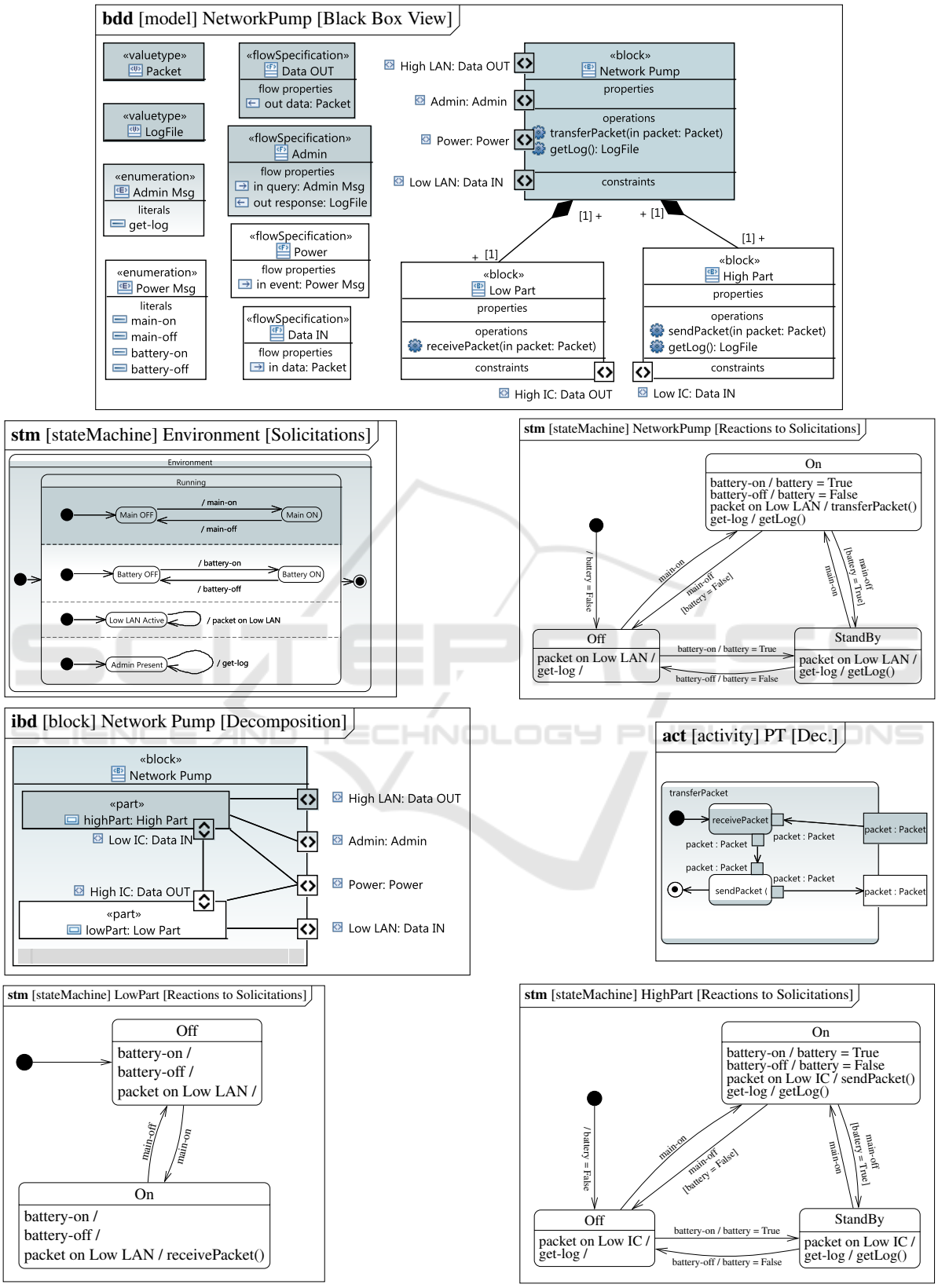

Figure 2: Diagrams of the Network Pump's SysML Model.

3 AGNOSTIC MODELING REQUIREMENTS

An obvious approach, to define requirements on models fit for a special purpose, is to choose a modeling language as generic and standard as possible, SysML (OMG, 2012b) or B (Abrial, 2010) for example, and express constraints in a compatible language, potentially in OCL (OMG, 2012a) or B itself in the previous examples. In order to cover as much use cases as possible, those general purpose modeling languages (GPMLs) often provide different means to express similar concepts, each one adapted to a particular aspect with its own subtleties. For example SysML allows to express behaviors using state machines, activity diagrams or sequence diagrams. However, this large expressiveness of GPMLs comes with an increased complexity of their abstract syntax structure and semantics. This increased complexity makes it more difficult to formally define modeling requirements directly on the abstract syntax and semantics of those GPMLs. Indeed, high level properties can often be expressed using different "means" of GPMLs. For example, possible operation interleavings can be expressed using state machines, activity diagrams or sequence diagrams. And the expression of modeling requirements have to take into consideration all those means with all their subtle and complex constructions. Moreover, this approach does not satisfy the DGA's need for the definition to be flexible regarding the concrete modeling language used in order to let contractors choose (in agreement with the DGA) the modeling language they prefer, as long as it can carry the information needed. Similarly, when requiring models fit for a special purpose, another approach commonly used is to design a new Domain Specific Modeling Language (DSML) (Luoma et al., 2004). However, with this approach, the modeling language is fixed once again; and the contractors have to use this DSML to design their product or translate their design models into this DSML.

Even if the approach proposed in this paper may seem similar in practice to the design of a DSML, the goal in this paper is not to design a new DSML, but to constrain and formalize the use of GPMLs for a specific purpose. An analogy can be made with the use of natural languages to express requirements. Natural language texts are often ambiguous, incomplete and formally incoherent. This is mainly due to the potential complexity of sentences and broad meaning of words. In order to limit the impact of those characteristics on the expression of requirements, it is usually considered good practice (ISO, 2011) to use requirement patterns and a word glossary tailored to the specific domain and type of requirements that have to be expressed. It may seem that a new language with its own grammar and vocabulary as then been created (and from a certain point of view, it is the case); however, the goal is really to still use the original natural language, but in a constrained way. This paper attempts to do the same on the use of GPMLs in a constrained way for a specific purpose (modeling BDMs for the example of this paper).

The proposed process (of formal definition of models fit for a specific purpose) bears resemblance to programming language design (Pierce, 2002). The first step consists in defining a semantic domain and some correctness constraints that are specifically tailored for the concepts that the specific model has to carry, and are as simple as possible in order to simplify the formal expression of the modeling requirements. In a second step, only skimmed through in this paper, the concrete syntax (of those special purpose models, BDMs for example) is defined by reusing the concrete syntax of a GPML and mapping its abstract syntax to the semantic domain previously defined (defining a semantics) and translating the correctness constraints to the GPML abstract syntax (defining typing rules). This mapping is done in a way that respects as much as possible the original semantics of the GPML with regard to the aspects reflected in the semantic domain of the special purpose models.

The next section provides an agnostic definition of BDMs through the definition in sect. 4.1 of a semantic domain tailored to what is to be expressed in BDMs. Sect. 4.2 relies on this previously defined semantic domain to define what are "valid BDMs" through correctness constraints. Due to space limitation, those definitions are simplified to a fragment still sufficient to support the paper's claim in favor of agnostic modeling requirements.

4 AGNOSTIC DEFINITION OF BDMS

The goal of a BDM is first to model the behavioral interactions between a product (called "system" thereafter) and its environment, i.e. the reactions of a system to the potential solicitations of its environment. In a second step, the behaviors of those internal operations are decomposed into interactions between the organs of the system. BDMs have to express relations between occurrences of events generated by the environment, the system or its organs, i.e. how the occurrence of one of those events can influence the occurrence of the other events. The events of interest

generated by the environment are the solicitation messages sent to the system. With regard to the system generated events, the interesting ones for BDMs are the messages emitted towards the environment, the system's internal messages and the internal operations performed by the system or its organs.

The approach chosen to express the possible interactions between events is to define the set of possible traces (Def. 2) of events (Def. 1) generated by the environment, the system or its organs.

Definition 1 (Event)**.**
An event*, denoted by* $\mathfrak{e}$*, is the occurrence of: an exchange of a message between two entities, or of an operation which is internal to an entity (has no observable effect outside of the entity).*

Definition 2 (Trace)**.**
A trace *on a set of events* $\mathfrak{E}$*, denoted by* $\mathfrak{t}$*, is a totally ordered sequence of events from* $\mathfrak{E}$*.*

The remainder of this paper uses the following definitions: $\emptyset$ is the empty trace (no events); $\mathfrak{e}$ is the trace of length 1 composed of the single event $\mathfrak{e}$; $\mathfrak{t}.\mathfrak{t}'$ is the trace whose sequence of events is the sequence of events of $\mathfrak{t}$ followed by the sequence of events of $\mathfrak{t}'$; $traces(\mathfrak{E})$, for $\mathfrak{E}$ a set of events, is the set of all traces containing only events in $\mathfrak{E}$.

Definition 3 (Trace germ: $\mathfrak{t} \prec_{\mathfrak{E}} \mathfrak{t}'$)**.**
Stating that a trace $\mathfrak{t}$ *is a germ*[3] *of a trace* $\mathfrak{t}'$ *over a set of events* $\mathfrak{E}$*, denoted* $\mathfrak{t} \prec_{\mathfrak{E}} \mathfrak{t}'$*, means that* $\mathfrak{t}$ *is a potential solicitation trace triggering the trace* $\mathfrak{t}'$*, when considering* $\mathfrak{t}$ *complete over* $\mathfrak{E}$*. For all traces* $\mathfrak{t}$ *and* $\mathfrak{t}'$*, event* $\mathfrak{e}$ *and set of events* $\mathfrak{E}$*,* $\prec$ *is formally defined as follows:*

- $\emptyset \prec_{\mathfrak{E}} \mathfrak{t}'$;
- $\mathfrak{e}.\mathfrak{t} \prec_{\mathfrak{E}} \mathfrak{e}.\mathfrak{t}'$ *if* $\mathfrak{t} \prec_{\mathfrak{E}} \mathfrak{t}'$;
- $\mathfrak{t} \prec_{\mathfrak{E}} \mathfrak{e}.\mathfrak{t}'$ *if* $\mathfrak{e} \notin \mathfrak{E} \wedge \mathfrak{t} \prec_{\mathfrak{E}} \mathfrak{t}'$.

4.1 Semantic Domain of BDMs

The semantic domain of BDMs defines the possible meanings of BDMs. This semantic domain is defined by a set of requirements that BDMs have to respect. Those requirements are expressed at a level of abstraction which is independent of the concrete modeling language which could be used to construct a BDM.

The definition of BDM's semantic domain in this paper relies on the implicit[4] definition by any BDM of: a set $\overline{\text{MSG}^{env}}$ of messages generated by the environment; a set $\overline{\text{MSG}^{sys}}$ of messages generated by the system; and a set $\overline{\text{OP}^{sys}}$ of internal operations that can potentially be executed by the system in reaction or not to the solicitation of the environment.

For example, considering that the underlying SysML model associated to the diagrams of Fig. 2 is a valid BDM of NP, depending on the BDM's semantics given to SysML models, $\overline{\text{MSG}^{env}}$ would likely be related to elements whose type is "Packet" `valuetype` or one of the `enumeration` appearing in the "Black Box View" bdd ("Admin Msg" or "Power Msg"); $\overline{\text{MSG}^{sys}}$ would likely be related to elements whose type is "Packet" or "LogFile" `valuetype`; $\overline{\text{OP}^{sys}}$ would most likely be the set of `operations` of the "Network Pump" `block`. However, those sets could be defined in many different ways in a SysML model.

A BDM (denoted by BDM) of a system (denoted by sys) in an environment (denoted by env) defines a set of sub-models reflecting different aspects of the decomposition.

Req. 1. *BDM defines: a sub-model EBM specifying the behavior of env towards sys; a sub-model SBM specifying the global behavior (black box) of sys in reaction to the solicitations of env; a sub-model SDM specifying the decomposition of sys into organs; a set* $\overline{ODM}$ *of sub-models* ODM_i *specifying the decompositions of the internal operations of sys; a set* $\overline{OBM}$ *of sub-models* OBM_i *specifying the black box behaviors of the organs of sys.*

For Fig. 2, EBM would likely be related to the `stateMachine` element named "Environment", and be identified as such in a comment associated to this element. SBM would likely be the `stateMachine` element named "NetworkPump". SDM would likely be the `block` element named "Network Pump". The `activity` element named "PT" would likely be in the set $\overline{\text{ODM}}$. And the `stateMachine` elements named "LowPart" and "HighPart" would likely be in the set $\overline{\text{OBM}}$.

EBM defines the set of solicitation traces (sequences of messages belonging to $\overline{\text{MSG}^{env}}$ or $\overline{\text{MSG}^{sys}}$) for which SBM *must* define the black box reactions of sys. Depending on the purpose of the model, those solicitation traces could be only those that a compliant environment would generate, or it could include traces that could only be generated by an attacker.

Req. 2. *EBM defines the set* $\overline{\text{T}^{\text{SOL}}}$ *of solicitation traces taken into account in BDM. Elements of those traces belong to* $\overline{\text{MSG}^{env}}$ *or* $\overline{\text{MSG}^{sys}}$*.*

$$\overline{\text{T}^{\text{SOL}}} \subseteq traces(\overline{\text{MSG}^{env}} \cup \overline{\text{MSG}^{sys}}) \quad (1)$$

[3] "germ" is used with the meaning "something that initiates development or serves as an origin" (Merriam-Webster)

[4] It has to be implicit in this paper due to the space limitation related simplification of the original BDM's definition.

For Fig. 2, $\overline{\text{T}^{\text{SOL}}}$ would likely be all the sequences of transition triggers (related to $\overline{\text{MSG}^{sys}}$) and effects (related to $\overline{\text{MSG}^{env}}$) corresponding to at least one "execution" of the `stateMachine` element named "Environment". This trace set could also potentially be defined using *interaction* or *activity* elements.

SBM describes the possible system "reactions" (if any) to all the possible environment solicitations.

Req. 3. *SBM defines the set* $\overline{\text{T}^{\text{SYS}}}$ *of possible traces corresponding to reactions of sys to env's solicitations taken into account in BDM. Elements of those traces belong to* $\overline{\text{MSG}^{env}}$, $\overline{\text{MSG}^{sys}}$ *or* $\overline{\text{OP}^{sys}}$.

$$\overline{\text{T}^{\text{SYS}}} \subseteq traces(\overline{\text{MSG}^{env}} \cup \overline{\text{MSG}^{sys}} \cup \overline{\text{OP}^{sys}}) \quad (2)$$

For Fig. 2, $\overline{\text{T}^{\text{SYS}}}$ would likely be all the sequences of transition triggers (related to $\overline{\text{MSG}^{env}}$) and effects (related to $\overline{\text{MSG}^{sys}}$ or $\overline{\text{OP}^{sys}}$) corresponding to at least one "execution" of the `stateMachine` element named "NetworkPump".

SDM describes the organs (components) of sys and the connections among themselves and with the environment. By convention, every BDM identifies E as the environment, in the sub-models related to sys's organs.

Req. 4. *SDM defines:*

a) *the set* $\overline{\text{ORG}^{sys}}$ *of organs composing sys in the decomposition modeled in BDM.*

b) *the set* $\overline{\text{OP}^{org}}$ *of sys's organs internal operations.*

c) *the operations executable by each organ; this definition is done through a total function* OP2ORG *from* $\overline{\text{OP}^{org}}$ *to* $\overline{\text{ORG}^{sys}}$.

$$\text{OP2ORG} : \overline{\text{OP}^{org}} \rightarrow \overline{\text{ORG}^{sys}} \quad (3)$$

d) *the set* $\overline{\text{MSG}^{org}}$ *of messages that sys's organs can send and receive.*

e) *the direct unidirectional connections among sys's organs themselves and with the environment (bidirectional connections of the architecture are modeled by two unidirectional connections); this definition is done through a set* $\overline{\text{CON}^{org}}$ *of triplets made of a message belonging to* $\overline{\text{MSG}^{org}}$ *and two elements among sys's organs and env.*

$$\begin{aligned}\overline{\text{CON}^{org}} \subseteq \big(&(\overline{\text{ORG}^{sys}} \times \overline{\text{MSG}^{org}} \times \overline{\text{ORG}^{sys}}) \\ &\cup (\{\text{E}\} \times \overline{\text{MSG}^{org}} \times \overline{\text{ORG}^{sys}}) \\ &\cup (\overline{\text{ORG}^{sys}} \times \overline{\text{MSG}^{org}} \times \{\text{E}\})\big)\end{aligned} \quad (4)$$

For Fig. 2, $\overline{\text{ORG}^{sys}}$ would likely contain only the "highPart" and "lowPart" `parts`; $\overline{\text{OP}^{org}}$ would likely contain only the "receivePacket", "sendPacket" and "getLog" `operations`; OP2ORG would map "receivePacket" to "lowPart" and "sendPacket" and "getLog" to "highPart". $\overline{\text{MSG}^{org}}$ would likely be the union of $\overline{\text{MSG}^{env}}$ and $\overline{\text{MSG}^{sys}}$ with the addition of messages related to values of type "Packet" transiting on the internal connection between "lowPart" and "highPart".

For all sys's internal operations, BDM contains an ODM_i in $\overline{\text{ODM}}$ which describes the functional decomposition of this internal operation in a sequence of organs internal operations. BDM also implicitly defines a total function OP2ODM from $\overline{\text{OP}^{sys}}$ to $\overline{\text{ODM}}$ which maps every sys's operation to its functional decomposition.

Req. 5. *Each* ODM_i *of* $\overline{\text{ODM}}$ *defines the call graph of a sys's operation o (*$o = \text{OP2ODM}^{-1}(\text{ODM}_i)$*). This call graph is defined by a labeled transition system; itself defined by a set of edges* $\overline{\text{CG}_o^{\text{OP}}}$ *whose associated nodes belong to* $\overline{\text{OP}^{org}}$ *or* $\{\text{E}\}$ *and labels to* $\overline{\text{MSG}^{org}}$.

$$\begin{aligned}\overline{\text{CG}_o^{\text{OP}}} \subseteq \Big(&(\overline{\text{OP}^{org}} \times \overline{\text{MSG}^{org}} \times \overline{\text{OP}^{org}}) \\ &\cup (\{\text{E}\} \times \overline{\text{MSG}^{org}} \times \overline{\text{OP}^{org}}) \\ &\cup (\overline{\text{OP}^{org}} \times \overline{\text{MSG}^{org}} \times \{\text{E}\})\Big)\end{aligned} \quad (5)$$

For Fig. 2, the ODM_i corresponding to the `operation` "transferPacket" would likely be related to the `activity` "PT". This information could also be embedded in an `interaction` element.

For every organ identified in SDM, BDM contains an OBM_i in $\overline{\text{OBM}}$ describing the possible "reactions" of this organ to the solicitations of its environment (env and other organs). BDM also implicitly defines a total function OR2OBM from $\overline{\text{ORG}^{sys}}$ to $\overline{\text{OBM}}$ which maps every sys's organ to its behavior.

Req. 6. *Each* OBM_i *defines the set* $\overline{\text{T}_o^{\text{ORG}}}$ *of possible traces corresponding to reactions of sys's organ o (*$o = \text{OR2OBM}^{-1}(\text{OBM}_i)$*) to its environment solicitations. Elements of those traces belong to* $\overline{\text{MSG}^{env}}$, $\overline{\text{MSG}^{sys}}$, $\overline{\text{MSG}^{org}}$ *or* $\text{OP2ORG}^{-1}(o)$.

For Fig. 2, $\overline{\text{T}^{\text{ORG}}_{\texttt{lowPart}}}$ would likely be all the sequences of transition triggers and effects corresponding to at least one "execution" of the `stateMachine` element named `LowPart`, and similarly for $\overline{\text{T}^{\text{ORG}}_{\texttt{highPart}}}$. Those trace sets could also potentially be defined using *interaction* or *activity* elements.

4.2 Correctness Constraints of BDMs

Every entity belonging to the semantic domain of BDMs is not necessarily a valid (complete, unambiguous and coherent) BDM. Additional constraints are expressed to refine the definition of "correct" BDMs.

Req. 7. *For every trace* t_{env} *of* $\overline{T^{SOL}}$, $\overline{T^{SYS}}$ *contains at least one trace that is a potential description of the reaction of* sys *to the solicitation* t_{env}.

$$\forall t_{env} \in \overline{T^{SOL}},\ \exists t_{sys} \in \overline{T^{SYS}}\ s.t.\ \ t_{env} \prec_{\overline{MSG^{env}}} t_{sys} \quad (6)$$

This requirement does not ensure that the modeling of $\overline{T^{SYS}}$ is correct. However, if this requirement does not hold, then the modeling of $\overline{T^{SYS}}$ is incomplete with regard to $\overline{T^{SOL}}$; and BDM must be corrected.

Req. 8. *Each* ODM_i *is such that every (data or control) message exchanged between two organs' operations or with* env *is physically allowed by the organic architecture.*

$$\begin{aligned} &\forall\ (S,M,D) \in \overline{CG^{OP}_{OP2ODM^{-1}(ODM_i)}},\\ &\quad \exists\ (S',M,D') \in \overline{CON^{org}}\ s.t.\\ &\qquad (S = S' \ \vee\ OP2ORG(S) = S')\\ &\qquad \wedge\quad (D = D' \ \vee\ OP2ORG(D) = D') \end{aligned} \quad (7)$$

This requirement ensures that message exchange in the functional decomposition of operations is compatible with the physical connections between organs.

Section 4 formally defines some modeling requirements for BDMs. Expressing the same constraints directly on SysML would have been more difficult due to the inherent complexity coming with the expressive power of SysML.

5 USE OF THE AGNOSTIC DEFINITION

A seemingly obvious way to use the above agnostic modeling requirements for BDMs would be to define: an abstract syntax for BDMs mapping closely the elements in BDM's semantic domain; and a translation from the abstract syntax of a GPML to this newly defined BDM's abstract syntax. However, this approach is not practical in general as some elements in the semantic domain are infinite (traces and trace sets for BDMs). Moreover, it would go against the philosophy of the proposed approach which is to tailor the use of a GPML, and not to create a new DSML.

Figure 3 illustrates how to exploit such agnostic modeling requirements to tailor the use of GPMLs. Sections 4 exposed the definition process of the top part of the figure (semantic domain and correctness constraints). For every GPML L (SysML, AADL, B, etc.) that is to be used to describe a BDM, what remains to be done is to define: a semantics of L on the semantic domain of BDM which respects the original semantics of L with regard to the concepts featured in BDM (this semantics describes how a contractor

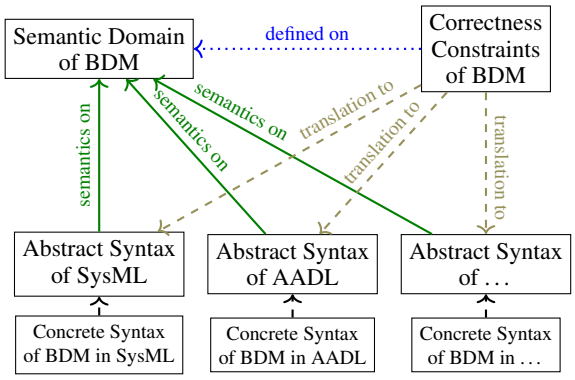

Figure 3: Associating concrete syntaxes.

can express a BDM in L, and how a model written in L is to be interpreted as a BDM by the contracting authority); a translation of the agnostic correctness constraints to the abstract syntax of L that is coherent to the previously defined semantics (those new constraints allow one to directly verify that a model expressed in L is a valid BDM). Once this is done, contractors are able to write BDM in the modeling language of their choice (*CS*). The DGA is able to verify directly that the delivered *CS* models are indeed valid BDMs (some of the constraints could even be verified automatically), and to interpret the delivered *CS* models as BDMs in order to perform efficient early design analyses. Moreover, as all the modeling languages for which a BDM's semantics has been defined share the same BDM's semantic domain, it could be envisioned to write BDMs mixing different modeling languages.

6 CONCLUSION

This paper addresses the problem of defining modeling requirements from a contracting authority (CA) to a contractor (Co), while still providing flexibility to the contractor regarding the concrete modeling language used.

Using a well-formedness constraints approach, this task would require: *a*) the CA to pick a General Purpose Modeling Language (GPML) expressive enough for the information to carry in the models; *b*) the CA to define well-formedness constraints on the GPML; and *c*) the Co to use this GPML while respecting the constraints defined in *b*).

Using a DSML approach, this task would require: *a*) the CA to build a DSML whose expressiveness maps closely the one needed for the modeling requirements; *b*) the CA to define correctness constraints on the DSML defined in *a*); and *c*) the Co to use the DSML defined in *a*) and respect the constraints of *b*).

This paper introduces, based on an example, an

approach to tailor the use of General Purpose Modeling Languages (GPMLs) to a specific purpose, instead of defining a new Domain Specific Modeling Language (DSML). Our approach is based on the formal definition of Domain Specific Semantic Domains (DSSDs) with associated correctness constraints. The tailoring of the GPML is achieved by defining a specific semantics from the GPML syntax to the DSSD. Using this approach requires: *a*) the CA to build a DSSD D whose expressiveness maps closely the one needed to carry the information of interest; *b*) the CA to define correctness constraints C on D; *c*) the CA to define a semantics S_{L_i} on D for every GPML L_i of interest; *d*) the CA to translate C to C_{L_i} for every GPML L_i while respecting S_{L_i} ($C_{L_i} = S_{L_i}^{-1}(C)$); and *e*) the Co to use any GPML L_i of its choice while respecting S_{L_i} and C_{L_i}.

This approach allows contractors to use the modeling language of their choice, while still providing the contracting authorities with the information they need in order to perform early design analyses. In addition, by providing a simplified purpose-fit semantics to a GPML, such as SysML, it simplifies the task to verify the validity and interpret a model written in this GPML for a given precise purpose.

The author experience on the specific subject of this paper is mainly experimental and industrial. Which can explain why, to the author's knowledge, there is no other work directly related to the approach proposed in this paper. There is a vast amount of work on programming language design (Pierce, 2002), DSML design (Luoma et al., 2004) or model validation (Debbabi et al., 2010). However, the author does not know of a closely related work which would also be using the DSSD approach.

Future work, already started or planned, include the definition of the mapping from the abstract syntax of SysML to the semantic domain of BDM, and the development of an integrated environment for the development and verification of valid BDM SysML models.

ACKNOWLEDGEMENTS

The author wishes to acknowledge the authors of the insightful reviews which helped improve and clarify this manuscript.

REFERENCES

Abrial, J.-R. (2010). *Modeling in Event-B: System and Software Engineering*. Cambridge University Press.

Alexander, I. F. and Maiden, N. (2004). *Scenarios, Stories, Use Cases: Through the Systems Development Life-Cycle*. John Wiley & Sons, Inc., 1st edition.

Cockburn, A. (2007). *Agile Software Development: The Cooperative Game*. Pearson Education, 2nd edition.

Debbabi, M., Hassaïne, F., Jarraya, Y., Soeanu, A., and Alawneh, L. (2010). *Verification and Validation in Systems Engineering: Assessing UML/SysML Design Models*. Springer-Verlag, Berlin Heidelberg, 1st edition.

Forsberg, K. and Mooz, H. (1998). System Engineering for Faster, Cheaper, Better. *INCOSE International Symposium*, 8(1):917–927.

Gérard, S., Dumoulin, C., Tessier, P., and Selic, B. (2010). Papyrus: A UML2 Tool for Domain-specific Language Modeling. In *Proc. 2007 Model-based Engineering of Embedded Real-time Systems*, volume 6100 of *LNCS*, pages 361–368. Springer.

INCOSE, editor (2015). *Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities*. John Wiley and Sons, 4th edition.

ISO (2011). Systems and software engineering – Life cycle processes –Requirements engineering. Standard ISO/IEC/IEEE 29148:2011(E), International Organization for Standardization.

ISTQB Glossary Working Group (2015). Standard Glossary of Terms Used in Software Testing. Technical report, International Software Testing Qualifications Board. Version 3.01.

Kang, M. H., Moskowitz, I. S., and Chincheck, S. (2005). The Pump: A Decade of Covert Fun. In *Proc. Annual Computer Security Applications Conference*, pages 352–360. IEEE Computer Society.

Kolovos, D. S., Paige, R. F., and Polack, F. A. C. (2009). On the Evolution of OCL for Capturing Structural Constraints in Modelling Languages. volume 5115 of *Lecture Notes in Computer Science*, pages 204–218.

Luoma, J., Kelly, S., and Tolvanen, J.-P. (2004). Defining Domain-Specific Modeling Languages: Collected Experiences. In *Workshop on Domain-Specific Modeling*.

Mcconnell, S. (1996). Daily Build and Smoke Test. *IEEE Software*, 13(04):144,143.

Moore, A. P. (2000). Network Pump (NP) Security Target. Common Criteria's Security Target NRL/MR/5540–00-8459, Naval Research Laboratory.

OMG (2012a). Object Constraint Language (OCL). Standard ISO/IEC 19507:2012(E), International Organization for Standardization. Version 2.3.1.

OMG (2012b). OMG Systems Modeling Language (OMG SysML™). Standard, Object Management Group. Version 1.3.

Pierce, B. C. (2002). *Types and Programming Languages*. MIT Press, Cambridge, MA, USA.

Shamieh, C. (2014). *Continuous Engineering For Dummies®*. IBM Limited Edition. John Wiley & Sons, Inc.

The Eclipse Foundation (2015). Papyrus. https://eclipse.org/papyrus/. [Online; accessed 9-December-2015].