QAML: A Multi-Paradigm DSML for Quantitative Analysis of Embedded System Architecture Models

Dominique Blouin, Eric Senn
Lab-STICC Université de Bretagne-Sud
56321 Lorient Cedex, France
+33 (0)2 97 87 45 26
{dominique.blouin, eric.senn}@univ-ubs.fr

Kevin Roussel, Olivier Zendra
Centre de recherches INRIA Nancy Grand-Est
54603 Villers-les-Nancy Cedex
+33 3 83 59 30 00
{kevin.roussel, olivier.zendra}@inria.fr

ABSTRACT
In this paper, the QAML (Quantitative Analysis Modeling Language) DSML is presented. It is a formalism for representing quantitative analysis models applied to embedded system architecture models. Issued from the need to standardize the representation of heterogeneous power consumption analysis models, QAML has been generalized to support the analysis of arbitrary physical quantities. Following a Multi-Paradigm Modeling (MPM) approach and the principle of separation of concerns, QAML combines a set of DSMLs such as the SysML QUDV annex, the W3C MathML and other custom DSMLs to favor interoperability and reuse. Using an enhanced Atlas Model Weaving language, embedded systems architecture models of arbitrary languages such as AADL can be annotated with quantitative estimation models issued from measurements campaigns, numerical simulations or other means. The complete set of models in the MPM environment is interpretable to provide analysis results in system architecture models.

Categories and Subject Descriptors
D2.2 [Design Tools and Techniques]

General Terms
Design, Languages

Keywords
Model Analysis, DSL, DSML, ADL, AADL, SysML, MPM, MathML.

1. INTRODUCTION
The purpose of Model Based Engineering (MBE) is to discover and solve system level problems early in the development cycle through analysis of various qualities of models. Many of these qualities such as resources consumption, timing, latency, etc. are often expressed in a quantitative manner with a system of units. Model analyses are typically performed with dedicated tools that extract specific properties from an input design model, perform the analysis, and optionally re-inject the analysis results into the input model.

An example of this is the Consumption Analysis Toolbox CAT [1], which integrates power consumption estimation models into AADL-based designs. These estimation models are often constructed using the FLPA (Functional Level Power Analysis) [2] method, where measurements are performed to characterize stimulated hardware components. FLPA models are typically represented as a set of mathematical laws, or multi dimensional data tables from which the estimates are interpolated. The main advantage of FLPA models is that they are accurate and fast to compute (many times faster that cycle-level accurate simulations). They are therefore much better suited for design space exploration. However, they suffer from a reduced applicability, since a model is only suitable for the specific type of the component (manufacturer / model) on which the measurements were taken. As a result, FLPA models, if adopted by the industry will be extremely numerous, and in an ideal scenario, they could even be part of components datasheets.

A problem with most analysis tools such as CAT is that the underlying analysis models are rarely represented with an adequate formalism. For instance, in CAT, the models are represented as Java or C++ code embedded in Eclipse plugins, and a qualified programmer is needed every time a new model must be integrated into the tool.

On the other hand, there is panoply of Architecture Description Languages (ADLs) available for modeling embedded systems. Well known languages are the Architecture Analysis and Design Language (AADL) [3], the Modeling and Analysis of Real-Time and Embedded Systems (MARTE) UML profile [4], the AUTomotive Open System Architecture (AUTOSAR) [5], and the Systems Modeling Language (SysML) [6]. Quantitative analyses are needed for all these languages, but the problem is that tools are often interfaced with a single language, and model transformations must be developed for interoperability. This introduces additional complexity in the analysis process, such as model synchronization that must be performed when several design models of the same system (but different languages) need to coexist.

The contribution of this work is to solve these problems by providing a new DSML and toolset as means to quickly integrate new analysis into MBE design tool chains, without the need for an expert programmer. This DSML, called QAML for Quantitative Analysis Modeling Language, has been designed so that quantitative models are self contained and remain independent from any ADL. Once a model has been stored in a library, it can be shared across design models of various ADLs, since only a thin weaving model needs to be created to associate a quantitative model with a design model. User-friendly editors have been developed for this purpose, and ideally, designers should be able create by themselves the needed quantitative models and associate them with their designs. Once associated, a quantitative analysis model is automatically interpreted to update the design with the analysis results. Analysis models are automatically re-evaluated as properties of the design model on which they depend are changed, thus maintaining the analysis results consistent with the design.

QAML has been designed according to Multi-Paradigm Modeling (MPM) [7], following a separation of concerns principle to favor the reuse of existing languages such as QUDV (Quantity Units Dimensions Values) [11], MathML [12], and AMW (Atlas Model Weaver) [13]. Reusing these languages allowed saving

tremendous modeling efforts. QAML has been implemented as set of Eclipse plugins with the EMF framework [8], and tested with the Open Source AADL Tool Environment (OSATE) [9] in the frame of the Open-PEOPLE project [10].

This paper is structured as follows: section 2 introduces the language by presenting its composing DSMLs. Section 3 presents a simple use-case showing static power analysis of a video processing embedded system. In section 4, related work is compared and finally, the paper is concluded with a discussion on the assets and weaknesses of the language, and the research directions to resolve them.

2. LANGUAGE OVERVIEW

The architecture of QAML follows a separation of concerns principle in an MPM approach. MPM advocates that every aspect of a problem should be formalized with an appropriate DSML to avoid implicit information potentially leading to misinterpretation and errors. DSMLs of independent domains should remain independent of each other and controlled in size by including only the artifacts needed for representing the domain at the right level of abstraction for the problems to be solved.

Figure 1 depicts the overall architecture of the QAML language, where each ellipse represents a language covering a sub-domain of the more global quantitative analysis domain. Arrows between the languages indicate composition dependencies of various natures. A <<*use*>> dependency means that the language from which the arrow origins directly refers to concepts of the other language via its class's properties. The <<*extends*>> dependency is stronger as some classes of one language extend classes of the other language. Finally, the <<*agnostic use*>> dependency is the weakest of all. It only holds at the M1 level where models of a given language refer to instances of models of other languages through un-typed references; both languages do not have any explicit dependency in their meta-models. The next sections briefly introduce all sub-languages that compose QAML.

2.1 QUDV (Quantities and Units)

For quantitative analyses, a solid foundation of well-defined quantities, units and dimensions is crucial. Indeed, severe errors have occurred in systems just because dimensions and units had not been formalized, and mismatched those of other integrated systems. In a search for an existing language covering the domain of quantities and units, QUDV [11] was quickly identified as the best choice, due to its excellent coverage of the domain and precise semantics for unit conversions and verification of consistency.

QUDV stands for Quantity Units Dimension Values and is an annex of SysML. For this project the QUDV UML profile has been implemented as a DSML. It has the ability to represent systems of quantities and units such as the International System of Units (SI) or any other arbitrary units system. The main concepts are *Quantity*, *Quantity Kind* and *Unit*. A quantity of a given quantity kind contains a numerical value expressed in a particular measurement unit. *Simple Quantity Kinds* provide the basis for defining other quantity kinds via specialization or derivation. Each quantity kind may have an expression of its dependence in terms of base quantity kind(s) of a *System of Quantities*, so that dimension analysis can be performed and errors detected automatically. Specialized quantity kinds are variants of more general quantity kinds (thus inheriting the same units).

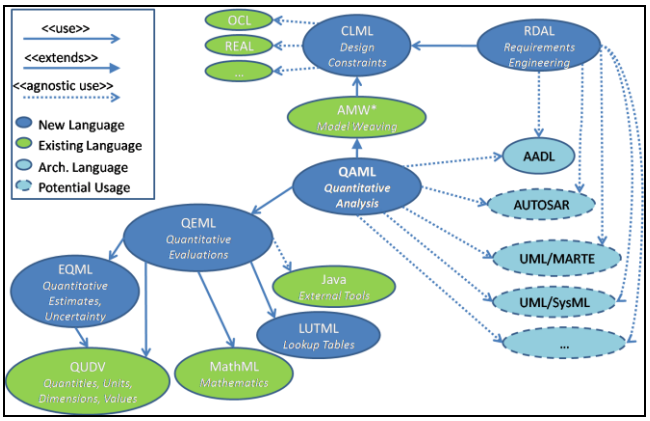

Figure 1. An overview of the architecture of QAML.

2.2 EQML (Estimates)

Another essential aspect of any quantitative analysis is *estimates* and their *uncertainty*, indicating how the analysis results can be trusted. In science, an estimate without uncertainty is useless, and formalizing uncertainty with a dedicated language is essential. No DSML was found covering this domain, which justified the development of the EQML language (Estimated Quantity Modeling Language). Figure 2 shows the meta-model of EQML, where QUDV is used to represent quantities and units. EQML supports the representation of estimates of various kinds from simple intervals (value ± uncertainty) to complex probability distributions functions of random variables. Functions are represented using concepts from other languages; namely LUTML and MathML that are introduced in the following sections.

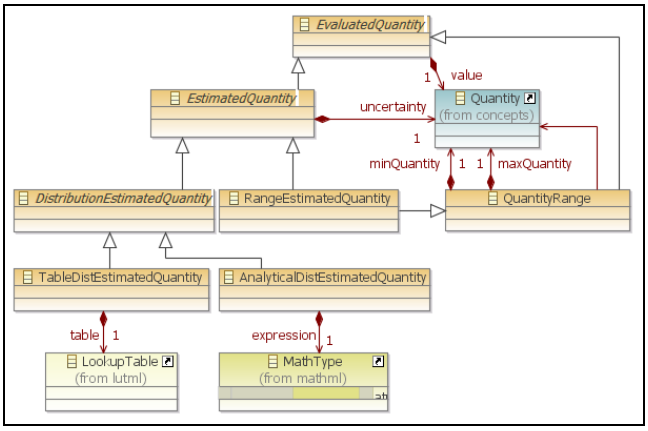

Figure 2. The meta-model of EQML.

2.3 LUTML (Lookup Tables)

Analyses of all kinds are often performed with simulation tools, which may take several hours to execute. A frequent solution to speed-up the analysis is to run several simulations for a set of input parameter values, and to store the results in data structures such as lookup tables (LUT). This advantageously replaces simulation computations with faster array indexing operations.

No existing language was found for the modeling of lookup tables, which justified the introduction of LUTML (LookUp Table Modeling Language), as shown in (Figure 3). LUTML allows the modeling of multi-dimension lookup tables in the form of a tree of nodes whose depth corresponds to the number of dimensions. The language embeds predefined enumerations representing commonly used inter/extrapolation policies.

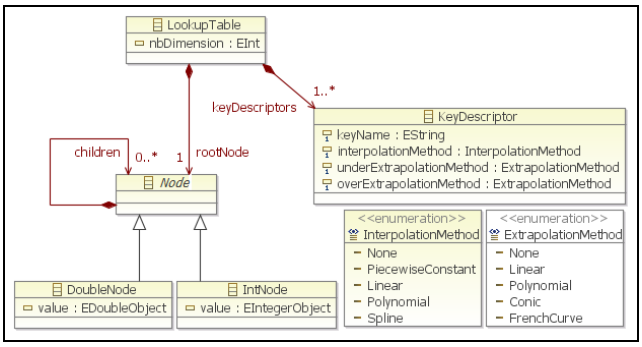

Figure 3. The meta-model of LUTML.

2.4 MathML (Mathematics)

Mathematics is at the heart of many analyses. The mathematics domain is quite large, and fortunately, good coverage is provided by the MathML [12] W3C specification. This justified the integration of MathML in QAML, despite some difficulty in converting the content MathML 3.0 XML Schema into an Ecore meta-model.

MathML was originally defined for visual rendering of mathematical formulae in web pages (presentation MathML), but evolved into a more formal language (content MathML) as it was realized it could be meaningful to many applications without regard to visual rendering. It is extensible and includes a set of predefined concepts for most of the mathematics needed up to the baccalaureate level in Europe. Reusing MathML allowed benefiting from the tremendous efforts invested by the W3C to cover the domain, and to reuse legacy MathML-based tools.

2.5 QEML (Quantitative Evaluations)

QEML (Quantitative Evaluation Modeling Language) is the DSML that composes all smaller DSMLs previously presented. It formalizes quantitatively *evaluable* models (hereafter quantity models). A quantity model has meta-data attributes for storing information such as the author of the model, its creation date, measurement campaign, etc. It specifies a relationship between a set of input model parameters of given quantity kinds and units, and an output quantity kind. This relationship must be computable in a way defined by an *Evaluation Descriptor*. Such descriptor is either based on a LUTML table, a MathML expression, or a Java class used to interface with external analysis tools. This later one allows for integrating legacy analysis tools, or to integrate more complex analyses that cannot be expressed in terms of mathematics or lookup tables. An *Estimation Descriptor* is a specialized evaluation descriptor that owns another quantity model representing how the *uncertainty* of the estimate is evaluated.

A *Quantity Model* can be specialized into a *Quantity Composition Model*, which has no uncertainty as it only specifies how a quantity shall be computed from a set of other quantities. The evaluation descriptor of a composition model is a MathML expression restricted to expressions containing a set operator such as sum, mean, product, etc. A composition model always owns an *Element Set Model Parameter* representing the set of quantities involved in the expression of the composition operator.

2.6 AADL (Design)

AADL is the first and only language for which QAML has been used so far. It is a component based language standardized by the Society of Automotive Engineers. It is separated into declarative and instance types of specifications. A *Type* declaration provides externally visible features of components such as ports, data and bus accesses. Associated *Implementation*(s) declaration(s) define the internal composition of a component through contained subcomponents declarations. Component types and implementations can be extended to support a modeling by incremental extension approach, where components are successively refined to provide a hierarchy of decreasing level of abstraction.

AADL includes a comprehensive property meta-model allowing users to define properties of their own in order to meet specific analysis needs. Properties are declared in property sets at the M1 level. The standard includes a set of predefined properties for major analysis domains such as schedulability, latency, resources allocation, etc. The AADL property meta-model supports the representation of units with prefixes, but the coverage of the domain remains limited compared to that of QUDV. An important feature of AADL properties that has inspired the semantics of QAML is their *visibility*, where a property value declared in a component type is automatically visible by all its implementations, extending components, the subcomponents of its type, etc. This nicely provides several placeholders for declaring property values at the proper level of abstraction in the components' hierarchy.

2.7 QAML (Quantitative Analysis)

One objective of this work is to be able to share libraries of quantitative models across models of various ADLs. This is why QEML has been designed so that quantity models can be represented without knowledge of any ADL. Limited effort is needed to associate a quantity model with design elements. Providing this capability is the purpose of QAML, which is just an extension of the Atlas Model Weaver language (AMW) [13] (Figure 1). QAML only provides additional semantics taking into account the meta-models of the models to be woven. AMW was chosen for the expressivity of its core weaving meta-model as it nicely captures the concepts of establishing fined-grained correspondences between model elements, without the need to modify the meta-models of the woven models. This is essential because the languages of the linked design models are often standardized that cannot be changed.

However, the core AMW language needed to be updated to increase its flexibility in establishing correspondence between model elements. This modified language (called AMW*) was added the capability to link model elements using formal language queries besides direct referencing of model elements. Query expressions make use of the Constraint Language Modeling Language (CLML) of our MPM environment (Figure 1), which declares constraints languages in an opaque manner, and provides interpreter service classes to evaluate constraint languages expressions. In the latest version of our toolset, the OCL and Lute [14] languages are available.

2.8 SEMANTICS

The semantics of QAML is composed from the semantics of its individual sub-DSMLs. The semantics of the sub-languages is obvious. For QUDV, it consists in unit conversion and the verification of units and dimensions consistency. When the design language also declares units like AADL, values extracted from the design model are automatically converted for units expected by

the QEML model. The semantics of MathML and LUTML consists in the evaluation of a quantity value from a set of input variable values. The semantics of the weaving model composing QEML and design models is to provide information from the design needed to compute the QEML models. The information is extracted from the following weaving links:

1. A top parent link between the QEML model and a design element that defines the *context* of the association.
2. A link between the result quantity kind of the QEML model and a property of the associated context element where the analysis result will be stored.
3. A child link for every parameter of the quantity model. The link refers to two elements from the design: a model element and a property applicable to this element. This is used to retrieve, from the design model element, values substituted to the parameters of the QEML model for evaluation. The link to the design element may be a direct reference or a query of a language. This capability is needed because data required for computing a quantity model may be potentially stored in any component of the design. For example, the power consumption induced by a thread on its executing processor typically depends on properties of the processor such as its frequency. A query is then needed to retrieve the processor as identified by the AADL processor binding property.
4. For quantity composition models, an extra link is declared to retrieve the set of design model elements holding the property values to be composed.

The impact of QAML model interpretation on the system architecture model is well controlled, as linking query expressions are non-modifying. The only element modified after model interpretation is the system architecture context element, which also serves as a context for evaluating the queries.

The semantics of QAML also includes a concept of quantity model *visibility*, inspired from the AADL property visibility mechanism. A model associated with an AADL component will be visible by all its descending components. As a result, a quantity model associated at a given component of the hierarchy may only become evaluable when components lower in the hierarchy are sufficiently refined to provide all inputs needed by the model. An inherited model can also be overridden by a more accurate model estimating the same quantity kind, but requiring more detailed information from the design component to become computable.

2.8.1 Model Interpreter

The evaluation of a woven quantity model (QAML model) is achieved through model interpretation rather than code generation. This has the advantage of avoiding the need to re-compile the tool every time a new quantity model is associated with the design. Model interpreters have been coded in Java for MathML (reusing an existing math expression evaluation framework), LUTML, QEML and QAML. The QAML interpreter composes all other interpreters. It uses a model interface service class, customized for the ADL of the associated design element, and the weaving model information to extract the model parameter values from the design. These values are then passed to the QEML interpreter, which delegates the computation of the result to either the MathML interpreter, the LUTML interpreter or to an external analysis tool depending on the type of the quantity model evaluation descriptor. The QAML interpreter also uses a dependency manager to ensure that models are evaluated in a correct order. For a given design element and attached QEML model, the dependency manager builds a graph of pairs of design element / QEML model that first need to be evaluated, taking into account the fact that a property evaluated from a given quantity model may be the input of other quantity models. Evaluation is triggered as changes are detected in any of the design or QAML models, to ensure that evaluated properties are maintained consistent with the design.

3. USE CASE

To demonstrate the assets of QAML, a simple use case showing static power analysis of a video processing embedded system is presented.

3.1 Toolset

Dedicated form editors have been developed to ease the definition of units, quantity kinds, quantity models and weaving models defining the association with AADL model elements. These editors greatly help in flattening the learning curve of the language. The toolset embeds OSATE, which is used as a front-end for editing AADL models. The QAML tool can be easily adapted for other ADLs by implementing a single model interface class responsible for interacting with design models, and for providing components hierarchy trees for quantity model visibility management.

3.2 Quantitative Analyses

Pre-declared AADL property sets and classifiers have been added to the AADL environment of the QAML toolset to provide generic component declarations where generic QEML models can be associated. More precisely, Abstract *Generic_Hw* component type and implementation have been declared to be extended by all hardware components of systems modeled in the environment.

Generic Static Power Models:

Two quantity models in the form of Equation 1 and Equation 2 are defined and associated with the *Generic_Hw* component implementation. $P_{statTot}$, P_{stat} and $P_{statSubcompo}$ are specialized quantity kinds whose general quantity kind is the generic *static power* quantity kind, which itself is a specialization of an even more general *power* quantity kind. The three static power quantity kinds represent the various parts of static power of a component. P_{stat} is the component intrinsic part. The subcomponents power can be summed since static power is by definition constant over time.

$$P_{statSubcompo} = \sum_{subcomponents} P_{statTot}$$

Equation 1.

$$P_{statTot} = P_{statSubcompo} + P_{stat}$$

Equation 2.

These two models are formally represented with MathML. For Equation 1, an OCL query is added to the weaving model to retrieve all subcomponents of the associated design component.

Video Processing System:

A real image processing system that processes a 25 frames/s VGA video image stream has been modeled in AADL. The software application is executed by dedicated processors synthesized in a Xilinx Virtex5 FPGA (Field Programmable Gate Array). Image

capture and display are performed by hardware blocks respectively interfaced with a camcorder and an LCD screen. The static power consumption of the FPGA has been characterized from measurements performed for various FPGA configurations to study the variation of the consumed power as a function of parameters such as the percentage of slices (basic configurable logical elements) used by the design, the average toggle rate, which is the rate at which the output signal of a basic logical element commutes when its input commutes, and the clock frequency. Measurement data in the form of a CSV (comma separated value) file obtained from the Open-PEOPLE hardware platform have been imported to constitute a lookup table-based QEML model.

The FPGA is modeled in AADL according to a modeling by incremental extension approach as presented in [15], where FPGA components are successively refined to capture information at the proper levels of abstraction. The LUTML-based model for the total static power is associated with an AADL component type declaration representing the Xilinx XCV2P30 FPGA configurable component on which the measurements were taken (Figure 4). No specific configuration is defined at this level of abstraction yet. The advantage of associating the model at this level of abstraction is that all refined component implementations specifying a precise configuration of the FPGA will inherit the LUTML quantity model. It is also only at this lower level of abstraction that the QEML model becomes computable, since property values needed to compute the quantity model and depending on the actual configuration can be set.

While these quantitative models remain extremely simple, more complex quantitative models have been represented with QAML such as the consumption due to IPC communications services as described in [16]. A model that needed several days of eclipse plugin development to be integrated in the CAT tool could be integrated in the AADL MBE platform within a few hours by a user having knowledge of both the QAML editors and the OCL (or Lute) query language.

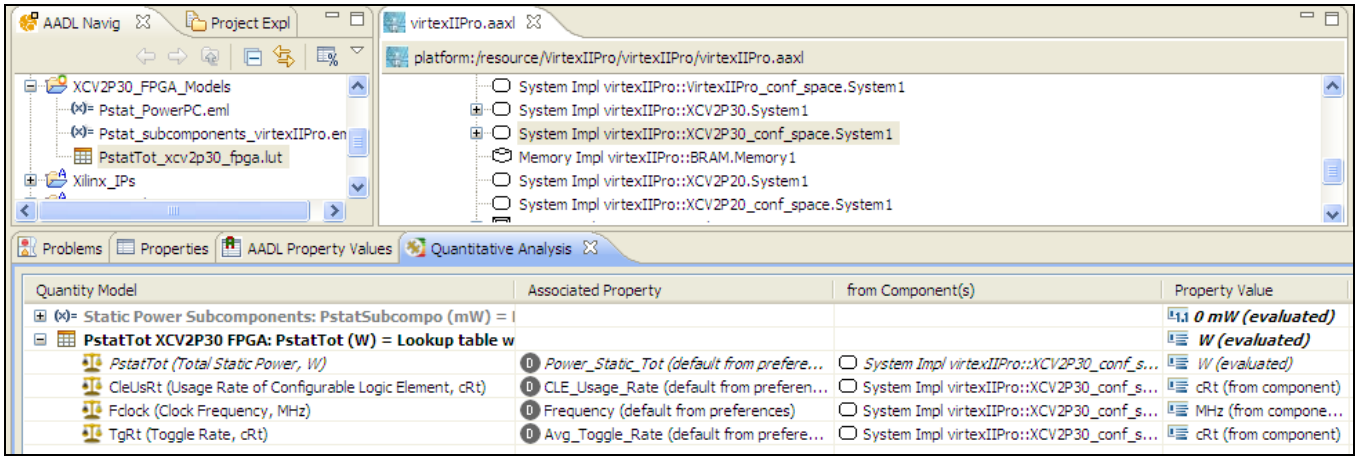

Figure 4. A lookup table quantity model associated with the XCV2P30 AADL FPGA configurable space component (selected in the AADL editor). The association is shown with the QAML weaving editor (quantitative analysis view). The grayed subcomponents power model is inherited from the generic AADL hardware component where it has been associated.

4. RELATED WORK

An interesting work related to the modeling of non-functional properties in domain specific modeling languages is found in [17]. A language has been proposed for the description of non-functional attributes in component models with properties such as *type* (e.g. power consumption, execution time, etc.), *data* including primitive and complex types such as records or tuples, meta-data to store information such as how the attribute value was obtained and its degree of confidence, validity condition, and composition in terms of other attribute values. Besides the complex data types for attributes, all these concepts are also present in our language. Attribute types, which can only be quantitative types in QAML, are represented by QUDV quantity kinds. The degree of confidence is represented by the *uncertainty* models; *attributables* is modeled through the applicability clause of the AADL properties associated with the quantity kinds. The validity condition in our language is embedded in the evaluation descriptor, but validity also depends on the actual design element with which the model is associated, as it must exhibit the features needed to compute the quantity model.

An advantage with our approach is that it uses the attributes of the ADL. Moreover, our language is interpretable while the attribute framework relies on external tools to provide analysis results. An advantage of their language however is that it is not restricted to quantitative attributes.

Another language close to QAML is ACOL [18], which is a model annotation language incorporating analysis, constraints and optimization expressions. The core language includes a set of principles that must be adapted to the system architecture language that it will be used with. ACOL analysis expressions allow defining derived properties in terms of other properties obtained from an ACOL interface plugged to the architecture model (using the annex clause for AADL). In QAML a QEML model is the equivalent of ACOL analysis clauses; the interface being replaced by the model parameter declarations. A difference is that QEML uses a standard to represent mathematical analyses (MathML), and can also evaluate results from data structures such as LUTML lookup tables besides expressions. ACOL constraint expressions guard the allowed values of properties to warn the designer when the model does not verify the guard. The equivalent of ACOL constraint in our framework is formal non-functional requirements of the RDAL language [19] (Figure 1), the advantage being that RDAL brings the assets of the requirements engineering discipline in the design flow.

An advantage of QAML over ACOL is the possibility to share QEML models across several ADLs. ACOL models embed native concepts of the system architecture language, which is good because users are already familiar with this language, but with the penalty that models are no longer shareable. We favored the definition of user-friendly editors making the language easy to learn while maintaining the shareable asset of quantity models. An asset of ACOL however is its optimization expression mechanism formulating minimization or maximization of one or several non-functional properties. QAML does not have an equivalent for this, but the intent is to implement this function with design space exploration tools using formally expressed RDAL *Goals* to trace performances of system architectures.

Finally, the MARTE Value Specification Language (VSL) is also a similar framework. However, while MARTE attribute values can also be expressed formally in terms of symbolic variables and expressions, it remains a UML profile and does not fit in our MPM approach. Furthermore, no means is provided to represent quantitative models based on data structures such as lookup tables (or to interface with external tools), and the concept of uncertainty does not seem to be explicit in the language.

5. CONCLUSION AND PERSPECTIVES

Quantitative analyses of all kinds are needed throughout the MBE design flow to help discover defects early in the development cycle. These analyses are essential whatever the ADL used to model the system may be. This paper presented the QAML language for the modeling of quantitative analysis used in the design of embedded systems with AADL. The strength of QAML is its ability to represent analyses formally so that they can be interpreted to provide analysis results stored in design models. Our trials showed that complex quantity models can be integrated in MBE designs with limited effort compared to the development of dedicated analysis tools. Reusing existing standards such as QUDV and MathML allowed covering the domains adequately, at the right level of abstraction, and to reuse tools already developed for these languages.

QAML is well suited for the representation of complex quantitative models and would nicely replace Excel data tables typically used within hardware component data sheets. Having formal data sheets would greatly help in reducing errors during MBE design.

Using QAML with other ADLs is also an interesting perspective, although the language has been designed so that little work is needed for this. It only requires the development of another extension of AMW for the ADL, and the implementation of a Java class used by the tool to interact with design models in an agnostic manner. All other elements of the tool (editors and model interpreters) can be reuse as is. This is possible thanks to MPM that drove the design of QAML.

Finally, generalizing QAML to support non numerical properties is also an interesting perspective. In the actual version of the language, quantity model parameters can only be of numerical type, and parameters expressed as non numerical data types such as enumerations must be mapped to numerical values.

6. REFERENCES

[1] D. Blouin and E. Senn. CAT: An extensible system-level power consumption analysis toolbox for model-driven design. In NEWCAS Conference (NEWCAS), 2010 8th IEEE International, pages 33 –36, 2010.

[2] J. Laurent, N. Julien, E. Senn, and E. Martin, Functional Level Power Analysis: An efficient approach for modeling the power consumption of complex processors, in Proceedings of the DATE Conference, Munich, 2004.

[3] Architecture analysis & design language (AADL), version 2, January 2010, http://standards.sae.org/as5506a/.

[4] A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems V1.1, OMG Adopted Specification, OMG Document Number: formal/2011-06-02, http://www.omg.org/spec/MARTE/1.1/PDF/.

[5] AUTomotive Open System ARchitecture, http://www.autosar.org.

[6] SYStems Modeling Language, OMG," 2008, http://www.omgsysml.org/.

[7] Vangheluwe H, de Lara J, Mosterman P (2002) An Introduction to Multi-Paradigm Modeling and Simulation. In: Proceedings of AI, Simulation and Planning – AIS'2002. Lisbon. SCS International, pp: 9–20.

[8] The Eclipse Modeling Framework (EMF), http://www.eclipse.org/modeling/emf/.

[9] Open Source AADL Tool Environment (OSATE), http://www.aadl.info/aadl/currentsite/tool/osate-down.html.

[10] The Open-PEOPLE Project Website, 2009, http://www.open-people.fr/.

[11] Quantity, Units, Dimensions, Values (QUDV), http://www.omgwiki.org/OMGSysML/.

[12] Mathematics Markup Language (MathML), http://www.w3.org/Math/.

[13] M. D. Del Fabro, J. Bézivin, F. Jouault, E. Breton, G. Gueltas. AMW: a Generic Model Weaver. In Procs. of IDM05. 2005.

[14] Lute Constraints Checker, https://wiki.sei.cmu.edu/aadl/index.php/RC_META

[15] D. Blouin, D. Chillet, E. Senn, S. Bilavarn, R. Bonamy, and C. Samoyeau, AADL Extension to Model Classical FPGA and FPGA Embedded within a SoC, International Journal of Reconfigurable Computing, 2011.

[16] Saadia Dhouib, Jean-Philippe Diguet, Eric Senn, Johann Laurent. Energy models of real time operating systems on FPGA. Euromicro 4th Int. Work. on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT) 2008.

[17] Séverine Sentilles, Petr Štěpán, Jan Carlson and Ivica Crnković. Integration of Extra-Functional Properties in Component Models. In Proceedings of the 12th International Symposium on Component-Based Software Engineering (CBSE '09), 2009.

[18] Dries Langsweirdt, Nelis Bouck and Yolande Berbers. Architecture-Driven Development of Embedded Systems with ACOL. In Proceedings of the 13th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops, 2010.

[19] Blouin, D. Senn, E. Turki, S. Defining an annex language to the architecture analysis and design language for requirements engineering activities support, Model-Driven Requirements Engineering Workshop (MoDRE).