Using Architecture Description Languages for Software-Assisted Electronics Design

Ramón Jiménez, MSc
Santo Domingo Institute of Technology (INTEC)
Santo Domingo, Dominican Republic
ramon.jimenez@intec.edu.do; rjimenezh@gmail.com

Abstract**—**the number of people designing high-level electronic devices from pre-existing building blocks is growing rapidly thanks to more accessible prototyping platforms. Software tools currently available for this activity provide little to no assistance in terms of automated validation of several important design concerns, complicating the adoption of this activity and increasing the error rate evidenced in the assembly and testing phases of project development. Starting from the success exhibited by architectural description languages (ADLs) in supporting automated assistance and validation of software architecture, a hypothesis is made that similar techniques can help automate and validate specific aspects of electronics design. The resulting tooling and its evaluation seem to confirm the hypotheses, but there is evidence for the need of more extensive field validation and tool implementation improvements. Several extension points are also identified that, if undertaken, may provide additional insight into the approach.

Keywords**—adl, metamodeling, emf, gmf, epsilon, electronics**

Motivation

Low-level electronics design is concerned with developing discrete components (“chips”) at the logical gate or transistor level. It is complex and time-consuming, dealing with concerns such as concurrency correctness, logical correctness, efficiency, and manufacturing cost. It has very good hardware and software support by way of field-programmable grid arrays (FPGAs), hardware description languages (HDLs), simulators and analysis tools. Collectively, these tools enable a process known as electronic design automation (EDA) [1].

High-level electronics design is concerned with composing devices out of existing chips and other discrete components. It is complex because the arrangement must ensure interoperability and compatibility across several different aspects, such as voltage levels, communication protocols, availability and type of digital or analog connection points or “pins”, electrical power required or provided, cost and volume concerns, among others.

Unlike low-level electronics design, high-level electronics design has poor software support. Typical software for this task can help represent the electrical schematic that binds components together (in terms of point-to-point connectivity only), as well as assist in the mechanical construction of the overall design, i.e. printed circuit board tracing and routing. It doesn’t, however, provide higher-level semantic support in terms of voltage level compatibility, component polarity mismatch detection, semantic compatibility of communication or interconnection protocols, among others.

Mobile computing, the Internet of Things, and the Maker movement, among other trends, are driving up the demand for *ad hoc* high-level electronics design, while also opening it up to less experienced designers through platforms such as Arduino (http://arduino.cc) and Raspberry Pi (http://raspberrypi.org). The mismatched evolution of electronics capabilities and battery technology imposes an additional strain on designers to optimize their designs for constrained power configurations. And electronic design increasingly involves substantial software development, with no comprehensive tooling ensuring the proper alignment between the software and hardware layers of the system, matched up only at the designer’s minds. These three factors compound to exacerbate the lack of proper software support for high-level electronic design.

In the realm of software engineering, architectural description languages (ADLs) have a proven track record of supporting software architecture and design, especially in the embedded/real time domains [2]. They allow modeling a software system as a series of discrete components and their semantic and concrete connections, and then analyzing the ensemble in search of a number of qualities that the design must fulfill and constraints it must obey. In this manner, ADLs and their associated toolset can provide substantial assistance to software design creation and validation.

This work sought to validate whether a similar approach can support electronic design, by selecting or creating an ADL suitable of representing electronic circuits at a sufficiently high level of abstraction that, by developing proof-of-concept tooling to process said representation, the design can be automatically analyzed and validated in order to ensure compliance with a series of constraints or design goals, hence automating and assisting parts of the design process.

Approach

The work was undertook in five general stages: architectural representation metamodel definition, metamodel implementation, specification of design validation and assistance tactics, design validation and assistance automation, and evaluation of results.

Using literary review, the research team's own experiences, and consultations with electronic designers, a metamodel was designed and documented in order to represent the architecture and design of general purpose electronic devices.

Existing ADLs and similar tools to document and analyze electronic designs were evaluated as candidates to use or extend to implement the metamodel. In the end, a custom implementation was built using Eclipse's Modeling Framework (EMF) and Graphical Modeling Framework (GMF) [3], aided by the Epsilon project tooling [4]. This provided a visual editor where instances of the metamodel representing electronic designs could be defined and stored for further manual or automated processing.

By discussing with electronic designers, and pondering the results of the discussions with the metamodel and tooling capabilities and available time and resources, a set of key design concern aspects was drafted for which automated analysis and assistance would be provided. The metamodel tooling was augmented using Java code in order to implement these design concern aspects. The final tool was then assessed by electronic designers for suitability and fitness of use as part of ordinary high-level electronic design work.

The rest of this paper details the relevant decisions and findings of each stage of the work.

METAMODEL DEFINITION

The first step of the work was to define an architecture-based integrated electronic design language, henceforth called ABDIEL[1]. Software architecture nomenclature and guidelines as defined by the Software Engineering Institute [5], and the metamodeling concepts and guidelines from [6] (particularly the "M2/M1" metamodel/concrete model nomenclature), were used in the definition process.

A first distinction is made between types and instances. In general, electronic designs are made of *parts*, which have *pins*, electrical joints through which parts can be connected to each other. Every such element in a design must first be specified before it can be instantiated. For instance, a toy semaphore may be built with three light-emitting diodes (LEDs). Each such diode is a part in the electronic design. Before the part can be used, its type must be specified: the fact that it has two pins, one being of positive polarity (the anode) and the other of negative polarity (the cathode); the fact that it has a maximum forward voltage (a discrete property); the fact that it has a maximum forward current; and many other properties such as brightness, specific color wavelength, etc. Once the part and pin types have been specified, instances of them can be used in a particular design.

A second classification of metamodel elements is along the lines of *components* and *connectors*. *Components* are discrete elements that exhibit externally visible attributes, including *ports*, which define the semantics and protocols through which components can be connected. In order to provide sufficient semantic expressiveness, *connectors* are modeled as independent entities, possessing their own attributes and sub-components, such as *roles* and *ports*. For both components and connectors, the externally visible attributes may be complex sub-components, or simply name/value pairs, with values optionally ascribing to some typing system.

The ABDIEL metamodel (M2) is detailed first. It allows *libraries*, or collections of electronic components, to be modeled. The first basic component type in ABDIEL is the *pin specification*. A pin specification describes a type of pin, which represents an electrical joint that belongs to a part and that can be connected to other pins, potentially belonging to different parts.

The second basic ABDIEL component type is the *part specification*. A part represents a concrete bit of a high-level electronic design, whether it is passive (e.g. buttons, resistors) or active (e.g. transistors, micro-controllers). Parts contain pins, as well as *properties*, defined later ahead.

A third type of ABDIEL component is a *port specification*. A port is a logical aggregation of pins. For example, the Universal Serial Bus (USB) uses four pins on each side of the connection: the power supply pin (Vcc), the ground pin, and the symmetric data pins D+ and D-. In ABDIEL, instead of connecting USB parts by connecting four pins of one part to the corresponding four pins of the other, a single USB port may be declared on each part specification. The port specification defines the port's protocol; connected ports must have matching protocols. It also defines *port wirings*, which define aggregations of pins within the port: each wiring binds a pin specification to the port specification, and specifies the alias/external name, or role, the pin has in the port. For instance, an Atmel ATtiny85 microcontroller part specification may expose an USB port specification that aggregates four port wirings, mapping the microcontroller's PWR, GND, PB3 and PB4 pins to USB's Vcc, GND, D+ and D- aliases or roles, respectively. This allows connecting the microcontroller to a USB socket part by using a single port connection (defined later). Note well that there is potential for confusion due to semantic overload of the term "port" in both the domain model and the metamodel: both ports and pins are conceptually ports of the "part" component.

Before turning to connectors, it must be noted that, in order to provide higher-level connection expressiveness, the pin component is modeled as a special type of *joint*. Another special type of joint is a *net*, a named element to which many pins may be connected. If the circuit contains several nets that share the same name, it is assumed they are all connected together (this is only a conceptual convention; the metamodel itself doesn't structurally enforce this).

ABDIEL defines the base *wire* connector. A wire connects a source joint to a target joint. The other possible connector is the *port connection*, which defines a connection between a source and a target port.

At the model level (M1), ABDIEL defines a concrete *circuit* as a collection of parts, wires, nets, port wirings and port connections, whose types belong to a specific library the circuit is based on.

[1] "ABDIEL" is something of a forced backronym for "Architecture-Based Integrated Electronics Design Language". "Abdiel" also happens to be the name of the eldest son of a friend of the author.

Finally, ABDIEL allows parts to be annotated with *properties*. Part specifications include sets of properties, which are name/value pairs that can provide additional details of parts and which can be used by analysis and assistance automation developers to drive the implemented tactic. For each property specified in the part specification, each concrete part of a circuit gets a concrete instance of the property in question. Property values are subject to a simple type system such that a property's value can be a string, integer, floating number, or Boolean value.

METAMODEL IMPLEMENTATION

Once the metamodel was defined, several existing tools were considered in order to implement it. Specifically, the Architectural Analysis and Design Language (AADL) [7] and the EAST-ADL [8] were considered.

AADL is a mature, well-established ADL for mixed hardware/software architectures. Originally envisioned for avionics architecture, it has been extended for embedded systems in general. The author has some experience both using AADL and extending AADL-based tools [9]. However, the visual tools for recent releases of AADL are rather difficult to set up, and are poorly documented. The ADL itself, while covering both hardware and software, does so from a software perspective, and does not lend itself well to detailed electronics design, with "device" and "port" being the most fine-grained electronic component abstractions available.

EAST-ADL, on the other hand, is an ADL for automotive electrical and electronic design. Its metamodel does include detailed enough elements, as part of the "HardwareModeling" package within its structural constructs, to enable accurate representation of high-level electronic designs, including concepts such as hardware pins and pin groups, which maps more or less directly to ABDIEL's pins and ports. However, EAST-ADL is a Unified Modeling Language (UML) profile; modeling using this ADL would demand UML proficiency from electronic designers, which are more used to modeling their circuits after the low-level electrical schematics they map to. There were also no readily available open tools found by the author to implement or extend the required ABDIEL concepts on top of EAST-ADL in an economical fashion.

In light of these findings, a custom tool was built using EMF and GMF via Epsilon. A first EMF eCore metamodel is built with ABDIEL's M2 elements. A concrete (M1) model is then created using Eclipse's generated model editor. This model defines the specifications for parts, ports and pins that can be used to model concrete circuits. The model is then run through an Epsilon Transformation Language (ETL) script, which generates a new EMF eCore metamodel (called the ABDIEL Diagram M2), where some primitive elements are programmatically introduced (circuit, joint, pin, net, wire, etc.), and a concrete "Part" model element is created for each part specification found in the source ABDIEL M1 model, extending the abstract "Part" element. This is needed because GMF can only generate visual elements for concrete model types; the process therefore allows palette elements to be created for each part type in the library model, as well as defining concrete properties for each part type based on the properties stated in the part's specification. In order to create and associate part instances with their contained pins and ports at runtime, GMF Java initializers are also generated automatically when generating the visual GMF editor, by augmenting the Epsilon generation process through built-in hooks [10]. This makes the generated editor instantiate a part's pins and ports as part of part instantiation, which in turn makes the elements visible and properly related when creating a new part. This is akin to having a UML object diagram populate a class' structurally related objects when a new object of said class is created in the diagram.

At the end of this process, a visual editor results that can edit, save and open circuit diagram files whose contents derive from the ABDIEL M1 library model that was used to generate the ABDIEL Diagram M2 metamodel.

DESIGN VALIDATION AND ASSISTANCE TACTICS SELECTION

The selection of design validation and assistance tactics to be automated was a crucial aspect of the work. The starting point for tactics selection was observing the shortcomings that current electronics design tools have to this respect, coupled with implementation complexity analysis and resource constraints.

Recalling their state of the art, current electronics design tools accommodate low-level electrical schematic modeling of circuits. This modeling is strictly limited to point-to-point connectivity, ignoring both electrical and higher-level electronic properties that must be manually considered by the designer. This means that, when considering a particular point-to-point connection, electrical properties of the intervening pins such as voltage levels, polarity, intensity and direction of electrical current expected to flow through the joint, etc.; and electronic properties such as roles (e.g. serial data, clock data), operating modes (e.g. signal input or output), digital or analog nature, among many others, are ignored by the modeling tool, which is therefore incapable of assisting the designer with these concerns.

Based on the division of electrical/electronic properties outlined above, several candidate tactics were identified, such as voltage level checks between two or more connected pins; forward voltage and/or current analysis, to prevent overloading or burning parts; substitution analysis, i.e. determine which parts can replace others in a given design in order to optimize costs or deal with availability issues; automated checking of port connections; among many others.

Beyond the inherent complexity of each candidate tactic, some issues applying to the categories as a whole informed the decision process. Specifically, analyzing several electrical properties of circuits requires representing and processing them as a set of electrical connections forming a directed cyclic graph, e.g. performing nodal analysis with Kirchhoff laws. The ABDIEL metamodel is not amenable to this representation, as it doesn't consider internal part wiring, and does not intrinsically accommodates concepts such as net interconnection (the fact that all joints connected to same-name nets are interconnected among them) and concrete pin connections between ports; these concepts would need to be programmatically implemented on top of the circuit model.

To illustrate these complexities with an example, consider the case of polarity check, e.g. ensuring that an LED's positive (anode) pin is not connected to a negative terminal. If the LED is connected in series with a current-limiting resistor (the *de facto* method of ensuring the LED doesn't allow excessive current through it and burns out), a naïve analysis considering only pins directly connected to the LED could yield a false negative, since ABDIEL's resistor pins have no polarity. The tool would need to trace back (potentially recursively) all connections that the resistor leads to, in order to determine whether a negative polarity pin is part of that network. Doing this would also require knowing that a resistor represents series impedance, with its two terminal pins connected internally, something not contemplated by the metamodel. A similar reasoning applies to a voltage level check, which would be a highly desirable validation to include, but highly complex to implement in the tooling's current state.

After considering all facts, four tactics were selected for design validation and assistance:

1. Find unused ports: flag for review any unconnected ports found in a diagram, based on the assumption that parts added to the diagram may have been intended to be connected through their ports;

2. Check port connections: validate port connections to ensure both connected ports share the same protocol and are pin-wiring-consistent, that is, that both sets of pin aliases match and that each alias on each port maps to a concrete pin of the port's containing part;

3. Suggest concrete UC: finds instances of a special part called a Generic Atmel Microcontroller, modeled as a superset of several Atmel microcontrollers. The idea is that a designer may use this generic part where the design calls for a microcontroller; this tactic analyzes the parts' pin and port connections to determine the concrete microcontrollers that satisfy the connectivity requirements and that may be substituted in the design[2];

4. Check polarity: under very specific conditions (because of the previously outlined complexities), flag for review any pin connections where a polarized pin can be traced to lead to a pin of the opposite polarity.

The "Key Findings and Future Work" section contains some notes about ways in which the metamodel could be modified to support efficient and elegant implementation of some additional types of electrical validations.

Design Validation And Assistance Automation

To implement the chosen validation and assistance tactics, three options were considered: OCL constraints added to the ABDIEL Diagram metamodel; Epsilon Validation Language (EVL) GMF integration [11]; and GMF popup menu action extensions to analyze the model using Java code. Due to inconsistent results using OCL and EVL, most likely due to the author's lack of experience with these tools, and lack of time to research issues properly, the tactics were implemented in Java.

[2]The generic Atmel microcontroller is not considered during unused port analysis, as by design some of its ports may go unused.

Each tactic was implemented as an Eclipse action that attaches to the diagram as a whole, providing the code access to the Java object model created by EMF from the ABDIEL Diagram model being edited. It is assumed that each action treats the model in a read-only fashion, and that it implements any required model processing independently, although common model querying and processing mechanisms were abstracted into utility classes.

In order to provide visual feedback from the analysis process, Eclipse markers [12] were generated in order to annotate circuit diagrams with informational, warning or error markers or icons. This also entailed adding Eclipse project and resource support to the editor. Three of the implemented validations use this type of visual feedback; the "Suggest Concrete UC" action provides feedback via a simple pop-up dialog that lists potential microcontroller models[3] to use in lieu of each generic microcontroller found.

The resulting tool was exported into a stand-alone application to be distributed for assessment and evaluation purposes. Exporting the tool as a GMF editor with the required Eclipse project, resource and problem marker views support was not straightforward; a detailed recollection of the reasons and required steps is available at [13].

The source code for the project is available at [14].

Evaluation

The tool was submitted to a number of electronic designers, in order to elicit feedback on its convenience and helpfulness as an auxiliary electronics design tool.

In terms of suitability and design assistance, the tool was very well received; the results seem to validate that defining and extending an ADL for electronics design analysis can certainly improve the design process. To reach a more conclusive verdict, substantial additional work must be carried out to provide support for deeper, more complex analysis and assistance, as well as improving usability, an area in which the tool's proof-of-concept status made itself evident. Adding new components to a library requires the software developers' intervention, as it entails editing the ABDIEL M1 model and re-building the ABDIEL Diagram M2 metamodel and corresponding GMF editor. When doing this, depending on the specific changes performed to the model, pre-existing diagram files can crash the editor by triggering GMF- or EMF-specific errors that require additional development work to properly handle. Despite EMF and GMF being domain-agnostic modeling and visual editor frameworks, a certain bias from software modeling is evident, and navigation of the tool is not always obvious for electronics designers, especially when compared with mainstream electronics design tools such as Eagle, Proteus or KiCad.

Key Findings And Future Work

In general, given the results of this work, the author is inclined to cautiously state that the initial hypothesis holds true,

[3] "Model" here refers to real-world part number or family variant, e.g. ATtiny85, ATmega328, etc.

namely, that an architecture-driven approach to electronic design representation provides a proper platform on which to develop electronic design assistance and validation automations. Unforeseen technical vagaries took away valuable time that was planned to be put towards more rigorous field testing and evolution of the proof-of-concept tool; more such significant real world testing and evolution is needed to validate the hypothesis.

Model-driven development proved to be very valuable to this approach, on two dimensions. On one hand, intrinsically, electronics design appears very amenable to a domain modeling approach; electronics can be construed as a series of models (electrical, component-and-connector, software) that are precisely interrelated and provide a solid basis upon which to tackle orthogonal design concerns. On the other hand, incidentally, the existence of tools such as the Eclipse Projects' EMF, GMF and Epsilon frameworks made possible the development of this project under conditions that would have otherwise been insurmountable.

Areas of further possible work include:

- Declarative metamodel extensibility. This would allow electronics designers to declaratively create their own parts, something which currently requires involvement of a software engineer. This may require choosing a new framework to generate the visual editor, entailing substantial development work to duplicate GMF's capabilities while allowing declarative editor extensions; alternatively, a much deeper understanding of GMF may allow creating an editor that can be declaratively re-configured, but GMF is not intended for this and simply establishing the feasibility of this approach implies a significant development effort. Another concern to be addressed is that currently, circuit diagrams containing parts that have been renamed or modified in the model cause the editor to crash; these failures would need to be handled cleanly

- Integration with the rest of the electronics design process, including but not limited to generation of schematic files to carry out detailed electrical and mechanical refinement of designs with external tools, and generation of skeletal source code to drive electronic designs. Model to text (m2t) transformations [6] could provide an economical way to achieve this

- Deeper electrical and electronic analysis. As explained earlier, the ABDIEL Diagram model is not suitable for this, but a model transformation to produce an equivalent lower-level electrical model may allow more precise and efficient analysis of some desirable circuit properties, by modeling the circuit as a directed cyclic graph as it concerns voltage and current. Another approach could be transforming the model to a representation that can be directly fed to an analog circuit simulation tool such as SPICE (http://bwrcs.eecs.berkeley.edu/Classes/IcBook/SPICE/), and reading back the output of said tool into the editor in order to provide integrated analysis

ACKNOWLEDGMENT

The author would like to thank Dimitris Kolovos, Ph. D., lecturer at The University of York and main committer/maintainer of the Eclipse Epsilon Project. Dr. Kolovos kindly answered questions and provided pointers about Epsilon usage that were highly relevant to the completion of this work.

The author would also like to thank Jimmy Pol, MSc, adjunct lecturer of Software Architecture at INTEC, for patiently listening and warmly encouraging the author throughout the execution of the work, and for proof-reading and helping correct drafts of this paper.

REFERENCES

[1] Wikipedia. "Electronic Design Automation". [Online] Available: http://en.wikipedia.org/wiki/Electronic_Design_Automation (February 12th 2015) .

[2] J. Delange. "Introduction to the Architecture Analysis and Design Language". SEI Blog. [Online] Available: http://blog.sei.cmu.edu/post.cfm/introduction-to-the-architecture-analysis-design-languag (February 12th 2015).

[3] The Eclipse Foundation. "Graphical Modeling Framework Tutorial". [Online] Available: https://wiki.eclipse.org/Graphical_Modeling_Framework/Tutorial/Part_1 (February 15th 2015).

[4] D. Kolovos, L. Rose, A. García-Domínguez, R. Paige. "The Epsilon Book". [Online] Available: http://eclipse.org/epsilon/doc/book/ (December 2013)

[5] L. Bass, P. Clements, R. Kazman. "Software Architecture in Practice", 3rd Ed. Boston: Addison-Wesley, 2012.

[6] M. Brambilla, J. Cabot, M. Wimmer. "Model-Driven Software Engineering in Practice". Morgan & Claypool, 2012.

[7] SAE International. "Architecture Analysis & Design Language (AADL) Annex Volume 2". SAE International Standards: AS5506/2, 2011.

[8] The ATESST2 Consortium. "EAST-ADL Domain Model Specification, version 2.1". [Online] Available: http://www.east-adl.info/ (2010).

[9] R. Jiménez. "Extending an open source, Eclipse-based AADL toolset". York:University of York, 2004.

[10] Epsilon Blog. "Customizing a GMF editor generated by EuGENia" . [Online] Available: http://eclipse.org/epsilon/doc/articles/eugenia-polishing/ (February 12th 2015).

[11] Epsilon Blog. "Live validation and quick-fixes in GMF-based editors with EVL". [Online] Available: http://www.eclipse.org/epsilon/doc/articles/evl-gmf-integration/ (February 12th 2015).

[12] D. Glozic, J. McAffer. "Mark My Words. Using markers to tell users about problems and tasks". [Online] Available: https://www.eclipse.org/articles/Article-Mark%20My%20Words/mark-my-words.html (April 1 2001)

[13] R. Jiménez. "Exporting a GMF Editor as an Eclipse Product". [Online] Available: http://www.modelesis.com/?p=204 (February 12th 2015).

[14] R. Jiménez. Github repositories. https://github.com/rjimenezh