Specification and Design of Electrical Flight System Architectures with SysML

Mark L. McKelvin, Jr.[1] and Alejandro Jimenez[2]
Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, 91109

Modern space flight systems are required to perform more complex functions than previous generations to support space missions. This demand is driving the trend to deploy more electronics to realize system functionality. The traditional approach for the specification, design, and deployment of electrical system architectures in space flight systems includes the use of informal definitions and descriptions that are often embedded within loosely coupled but highly interdependent design documents. Traditional methods become inefficient to cope with increasing system complexity, evolving requirements, and the ability to meet project budget and time constraints. Thus, there is a need for more rigorous methods to capture the relevant information about the electrical system architecture as the design evolves. In this work, we propose a model-centric approach to support the specification and design of electrical flight system architectures using the System Modeling Language (SysML). In our approach, we develop a domain specific language for specifying electrical system architectures, and we propose a design flow for the specification and design of electrical interfaces. Our approach is applied to a practical flight system.

I. Introduction

MODERN space flight systems are typically composed of several subsystems that perform distinct tasks to support higher-level system objectives. Subsystems are developed by functionally and geographically distributed design teams. Moreover, due to the rapid increase in electronic functionality, interactions between subsystems and the amount of design information that must be managed by electrical system engineering teams is increasing. Thus, capturing a holistic view of the system under development becomes a challenge to system engineers, and in particular, to the process of electrical system development and integration.

The traditional approach to the design of electrical system architectures is characterized by an iterative process that captures baseline design decisions in a set of loosely coupled design documents. Designers capture and modify design information in a collection of documents as the spacecraft design progresses. The design information that is captured often overlaps within these documents, and they are maintained separately without an explicit reference to a central data repository. For example, some specifications of the electrical system requirements are generally captured as a set of high-level block diagrams that are used to capture the interconnectivity between subsystems at different points in the design process. These diagrams may be used to capture the types of data and power flows between subsystems of the spacecraft. Each diagram is created independently with no explicit means of traceability. This process poses a challenge to electrical system engineers because it leads to ambiguous specifications.

Over the years, the complexity of system management has influenced the use of models as a way to manage the system engineering process[1,2,3]. In general, a *model* is an abstract representation of a physical phenomenon. For years, the model of a system was embedded within the knowledge of the system engineer. This model is referred to as the *mental model*. However, to cope with increasing complexity of systems, system engineering progressed from mental models to the use of computer-based tools, along with a set of methods, processes, and tools that are referred to as *model-based system engineering* (MBSE)[4].

MBSE is a widely used approach to system engineering; however, the use of a MBSE approach to system engineering does not necessarily mean that the focus is on the model of a system. This brings up the issue of *centricity* as described by Harvey et. al.[5]. MBSE may be classified by two general approaches, *document-centric* and *model-centric*. A *document-centric* approach describes the predominant state of practice where specification and design documents are exchanged between stakeholders, such as subsystem designers, system engineers, customers,

[1] Software Systems Engineer, System Architecture and Behaviors Group, M/S 301-490.
[2] Technical Group Supervisor, Electrical System Engineering, M/S 301-490.

and manufacturers. Documents could be in a hard-copy or electronic format, such as a memorandum, word processor, diagrams, or electronic mail. Furthermore, these documents are subject to examination and review for their rigor and appropriateness through a series of formalized meetings. Although documents are useful way to communicate with stakeholders, and in some cases required, there are limitations.

One of the limitations of a document-centric approach is ensuring tracebability to the origin of design information. The relationships may be articulated in the form of cross-references, notes, and figures, just to name a few. Any information that is not expressed in the documents is captured in mental models by the engineers. Hence, traceability is only as good as the expressiveness of the relationships that are explicitly defined in the documents. In addition, the information that is stored in the model, including figures and diagrams, are static. Consequently, related information appears in multiple documents, databases, and possibly multiple places within the same document. When the design information is updated, the designer needs to identify and modify any related documents. Since the information is captured in numerous documents, even for a mildly complicated system, this can result in a time-consuming change process that becomes a major source of ambiguity, error, and inconsistency of information[6].

In contrast to a document-centric approach, a model-centric approach uses well-defined models of the system for capturing technical design information. A key distinction in the two approaches is that a document-centric approach focuses on producing *reports* of a system. The reports collectively define the system architecture, whereas the model-centric approach is a *representation* of the system in a mathematical formalism. In a model-centric approach, a model is created as a representation of the system to varying degrees of fidelity. The model is the central artifact from which other artifacts, including documents, reports, diagrams, and tables are created or generated automatically. This approach enhances traceability, consistency, and the ability to be validated by checking model completeness and correctness. Moreover, it is worth noting that a document-centric approach may be taken with some MBSE tools, but even with a MBSE tool, the document-centric approach does not necessarily guarantee traceability nor does it ensure that the focus is on the system model.

II. Problem Statement

The traditional process of designing and deploying electrical system architectures utilizes a document-centric approach. However, as the complexity of the interconnectivity between electronics in space flight systems increase, the traditional document-centric approach becomes inefficient to cope with design changes, information sharing, and consistent integration of design information across various design and test processes in the end-to-end electrical system engineering process. The traditional approach to the design of electrical system architectures relies on capturing key points in the design by the use of static documents and diagrams. This approach can no longer be sustained as the pressures to deliver a flight system on time and within budget while fulfilling system requirements increase concurrently with system complexity. As a consequence, the designer focuses more on document creation as opposed to system design. This is in part due to a lack of rigorous, system-level design methodologies and tool support.

To address the aforementioned problems, a model-centric approach is applied to the design of electrical system architectures within a MBSE framework using the System Modeling Language (SysML)[7]. The modeling approach consists of establishing a modeling foundation through the use of metamodeling techniques to create a domain specific language for specifying electrical system architectures within a model-based environment. To support the design of electrical system architectures, a design flow is proposed. Within the design flow, a set of abstraction layers along with a set of common modeling elements that enable system composition in a stepwise refinement process is described.

III. Related Work

SysML has been used to model different aspects of systems in a variety of application domains including electrical systems, such as dynamic modeling of aircraft power systems[8]. However, our work focuses primarily on structural models that may be used for electrical system specification. For example, in Slomka et. al.[10], the authors focus on constructing models that combine the computational and physical aspects of an electrical system using a layer-based design approach that is based on a systematic refinement process. In Shah et. al.[11], the authors develop an approach to maintaining traceability between multiple data sets. The authors also introduce an approach to the integration of SysML and a commercial electronic automation design tool through the use of formal modeling and transformation techniques..

In contrast, we focus on the development of a domain specific environment for the specification of electrical flight system architectures using SysML and for managing electrical system design information. To support electrical system modeling, we develop a domain specific language that is anchored on top of a rigorous ontological

framework that enables formal analysis and the integration of design information across electrical flight system development processes and tools. The focus of the models that may be constructed is on the interfaces of the system's electrical and electronic hardware. The approach to modeling also enables the integration of electrical system models that are constructed within SysML with a commercial electrical computer aided design tool. In addition, we introduce an approach to electrical system design that is based on a successive refinement of the design through key levels of abstraction.

IV. Background

In this section, we provide a background of key technologies that are referenced and applied in this work. First, modeling with SysML is introduced. Then, the key concepts of domain specific modeling, ontologies, and metamodels are introduced. The domain of interest in this work targets the specification of electrical flight system architectures.

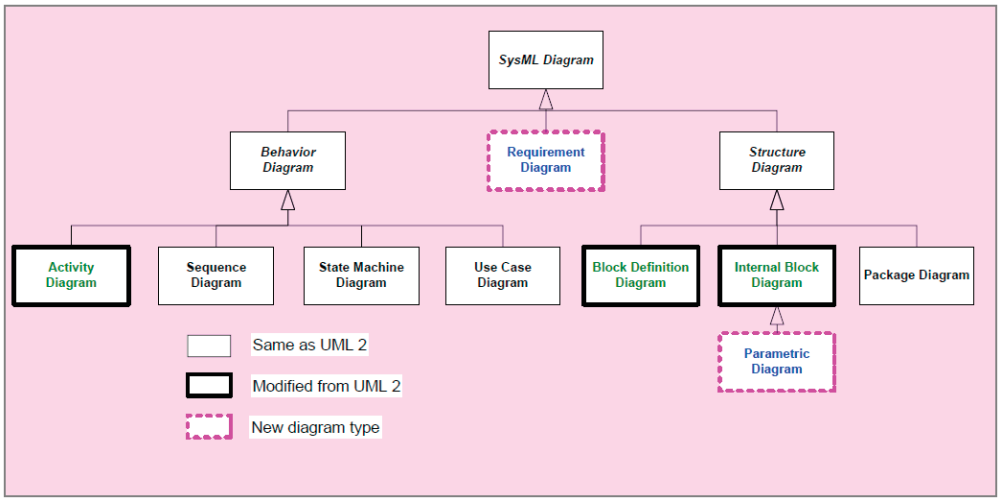

Figure 1. SysML diagram taxonomy[7]. *This figure provides a taxonomy of the diagrams that may be applied in SysML. SysML uses some diagrams from Unified Modeling Language (UML), and other diagrams are introduced in SysML specifically to target system engineers.*

A. The System Modeling Language (SysML)

SysML is a general-purpose graphical modeling language for system engineering. It is an extension of the Unified Modeling Language (UML)[12], a language that has been developed for software development. SysML is used to specify, analyze, design, and verify systems. The language provides graphical representations for modeling system requirements, behavior, structure, and parametric. As shown in the taxonomy of SysML diagrams[7] in Fig. 1, the language introduces two new diagrams to the existing set of UML diagrams that include the Parametric Diagram and Requirements Diagram. The Internal Block Diagram, Block Definition Block Diagram, and the Activity Diagram are modified to suit the system engineering domain. A key benefit to SysML is the ability to link the various diagrams. This enables an explicit traceability of model elements that are expressed within the diagrams.

The intent of the language is to provide a mechanism for realizing systems from a system engineering perspective, as opposed to its UML counterpart, which focuses on software architecting. Moreover, SysML provides a means by which system modeling may be captured within a MBSE approach without imposing a particular methodology or tool. SysML enables language extensions that support these domains through the use of *stereotypes* and *profiles*. A stereotype is a mechanism that allows the SysML language to be extended with additional concepts of a specific domain, along with the constraints and properties of the domain. Profiles are special types of packages that are used to group stereotypes. SysML provides additional extension capabilities, and the reader is referred to Friedenthal et. al. for more detail on applying SysML in system development[13]. Due to broad acceptance as a language for system engineering development, commercial vendors such as Sparx Systems[14], IBM[15], and No Magic[16] have created tools and toolkits that support SysML.

B. Domain Specific Modeling

Domain specific modeling[17] is a software engineering methodology for designing and developing systems within a specific area of interest. A domain specific language (DSL) is a language that is used to model and develop systems in a particular problem domain. A *domain* is a specific topic or area of concern to a set of stakeholders. DSLs provide a set of abstract notations and a meaning to each notation, called the *syntax* and *semantics*, respectively. DSLs are used to gain a better understanding and familiarity with modeling of applications in a specific domain, and they simplify commonly used constructs of a domain. The intent is to reduce the effort to construct a domain model by providing the domain expert with higher level constructs. This reduces the need to specify lower level constructs that are needed for their definition. In addition, DSLs facilitate automated model transformations since the mappings are based on language constructs. In the development of a DSL, domain experts provide the key notations and vocabulary that guide their thought process about a particular problem within their domain. Thus, to design a DSL, it is critical to have involvement from the domain experts; however, there is a consensus on some of the challenges that characterize existing DSL approaches, including interoperability with other languages and tools, formal semantics, and domain analysis[18]. Addressing these challenges is important for the successful adoption of a DSL. For example, issues such as interoperability and formal semantics motivated the use of ontology languages, such as the Web Ontology Language (OWL)[19]. To address these challenges, special attention is given to capturing the underlying concepts in a domain.

C. Ontologies

An *ontology* is a representation of the concepts and the relations between concepts in a particular domain. As pointed out by Wand et. al.[20], to create a model in a domain, a set of concepts to describe the domain is needed. This set of concepts includes the "things" and properties of those things that are of concern in a domain. The properties may be intrinsic to the thing that it characterizes or it may be mutual to a set of things. The things of the domain may also have relationships to one another, such as a type of association. The use of an ontology is for describing the precise meaning of things. This is particularly useful in denotation languages that rely on the constructs and rules that are defined for a domain to provide validation capabilities, such as the ability to check assertions using computer-based methods. An ontology describes the concepts and their relations, but a formal ontology may be constructed as a controlled vocabulary when it is expressed in an ontology representation language.

D. Metamodeling

A *metamodel* is an explicit representation of a domain's constructs and rules, such as the allowed set of model elements and their relation. A metamodel provides a specification for a domain specific model from which an instance of a model may be constructed. It also specifies the components that may be used within a model instance. An important aspect in the design of a DSL is a set of assertions on validity of a model that may be constructed within that domain. A metamodel is one approach that enables the construction of the rules that govern a valid model instance for a given domain. For example, it is likely necessary that a valid model in a domain must be fully connected in that every element must be connected to every other element. In relation to an ontology, "a metamodel is an ontology, but not all ontologies are modeled explicitly as metamodels"[21].

E. Platform Based Design Methodology

SysML does not specify any particular methodology to implement a model based design. Therefore, in this work, our design flow is based on concepts from the Platform-Based Design (PBD)[22] methodology, which has been successfully applied in the areas of consumer electronics and automotive systems[23,24,25]. The PBD methodology provides an intellectual framework where a design flow that implements a specification, proceeds through a sequence of refinement steps. An essential aspect of this framework is the separation of concerns[26]. For example, a key concern is functionality versus architecture, where functionality is a description of what the designer intends to implement, whereas the architecture expresses how the designer realizes the functionality.

A *platform* is a library of components that can be assembled to generate a design at that level of abstraction. A *platform instance* is a valid composition of library elements that are characterized by their cost and performance metrics. Each refinement step consists of selecting a platform instance that correctly implements a specification for a specific level of abstraction. Thus, a design step can be formulated as design problem whose solution is an instantiation of a platform instance that satisfies the specification. The methodology enables a systematic approach for manual, semi-automatic or automatic mapping between successive abstraction layers. This view of the design process is a generalization of a process that designers have used implicitly for years. For example, in the digital logic synthesis process Boolean logic functions represent the functionality and the platform includes a library of technology specific logic gates. The methodology not only applies to hardware design, but it has also been practiced

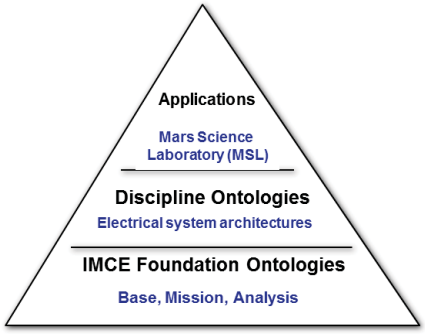

Figure 2. Ontological hierarchy. *A conceptual hierarchy of ontologies that are leveraged in the development of the electrical system ontology. The foundation ontologies, such as "Base", "Mission", and "Analysis" may be extended to support specific disciplines, which may then be used to support application specific models, such as the Mars Science Laboratory.*

in the software community, particularly in the development of the application software stack where the Open Systems Interconnection (OSI) Reference Model[27] is an instance. As referenced in the literature, the key to this methodology is the identification of appropriate abstraction layers.

V. Development of a Specification Language for Modeling Electrical System Architectures

In this section, we describe our approach to the development of a DSL for specifying electrical system architectures in space flight systems using a MBSE approach. The approach extends a hierarchy of ontologies that have been developed at the Jet Propulsion Laboratory (JPL) under the *Integrated Model Centric Engineering (IMCE)* initiative, as shown in Fig. 2. A key objective of the IMCE initiative is to define a core set of ontologies that may be applied in MBSE activities across SysML projects to establish a common semantic base. The key benefits of this approach are to:

- standardize domain terminology,
- enable a common understanding of domain terminology,
- enable reuse of domain knowledge,
- make domain concepts and assumptions explicit,
- separate domain knowledge from operational knowledge,
- analyze domain knowledge, and
- reason about the semantics for each level in the hierarchy.

A. Identifying Domain Concepts

In this section, domain concepts from the perspective of the electrical system engineer are summarized. A key responsibility of the electrical system engineer is to architect electrical interfaces. A simplified block diagram[9] of an electrical link model that features a pair of interfaces is illustrated in Fig. 3, where the transmit (TX) and receive (RX) interfaces are the end points of the communication channel. The interfaces are further characterized by a given behavior that may be represented as a high level set of functional blocks, such as those given in the figure. The functional blocks may be further refined into a connection of electrical circuit components.

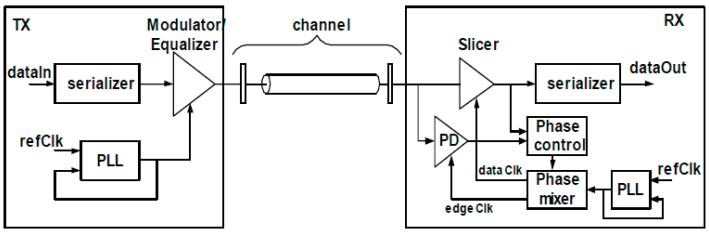

Figure 3. Electrical link model. *An illustration of a data link [9]. The link consists of a channel, a sender (TX), interface, and a receiver (RX) interface. Interfaces are intermediate circuits between electrical subsystems or components.*

To capture domain concepts, it was necessary to obtain involvement from the electrical system engineers, including conducting interviews and understanding the content of the artifacts that the engineers expect to produce, such as block diagrams, tables, and reports. Our observations resulted in a set of concepts that are summarized into several core concepts that are commonly used in the domain at different abstractions: *components*, *interfaces*, *channels*, and *signals*. A list of these concepts along with a brief description of each concept is given in Table 1. Each element may be characterized by a combination of *properties*, which may be quantified or used to further classify based on the context for which the concepts are applicable. For example, a key concern is to identify the difference between data and power interfaces since the behavior and quantitative properties of these types of interfaces differ electrically, and they are often addressed throughout the design independently. Therefore, the stereotypes in Fig. 4 may be extended to accommodate such extensions.

Table 1. Summary of key elements captured from domain knowledge. *This table highlights the key concepts from the electrical system architecture domain. The concepts are captured and summarized as a list of core concepts along with the name of its stereotype for which the concept is mapped, the SysML notation used to represent the syntax, and examples of the intended usage for each concept.*

Concept	Description	Stereotype Name	SysML Syntax	Examples
Signal	Represents discrete messages and signal flows	electrical.SignalFlow, electrical.DataFlow	Item Flow	Power flow, current flow, command message, telemetry message
Component	Represent an element that encapsulates behaviors and emits signals	electrical.Component	Block	Assembly, subsystem, end circuit, electronic board
Interface	A point of interaction between two devices where information or power is transmitted over logical channels, or media	electrical.Interface	Port	Universal Serial Bus (USB), voltage regulator circuit, Radio Frequency transceiver, data port
Channel	A logical path or physical medium	electrical.interfaceJunction	Connector (IBD)	Air, wire, logical function between interfaces
Property	A type of characterization that identifies or quantifies an element	Stereotype tag property	Block Properties and Tag Properites	Reference designator, impedance, voltage

B. Denotational Semantics

An electrical system model is an interconnection of components, interfaces, channels, and their characterizations. A component is an element that performs a set of tasks, and it may produce and consume electrical signals. Each component has at least one interface that is used as a point of interaction with other components. A pair of interfaces is related by a channel. A channel is a logical path or physical medium that is used to transmit a signal. For example, a component may contain an interface that is used to produce a data signal and transmit it over a channel to another component with a compatible receiving interface that consumes a data signal. At different levels of abstraction, components, interfaces, and channels may have different semantics. To distinguish between different types of components, interfaces, and channels, each object may be characterized with a set of properties. For example, at the highest level of abstraction, a component may be an instrument or power distribution subsystem where power flows between subsystems over a logical power channel. At a lower level of abstraction, a component may be an individual circuit board that receives power via a power converter interface circuit over a physical wire. In this example, the signal is a characterization of the channel, and power is a characterization the interfaces.

Formally, an electrical system model may be described by a labeled graph. Given a set of electrical components (C), channels (J), and interfaces (I), the semantics of an electrical system model may be defined formally as a graph $G = (V, E, \alpha)$, where $V = (V_C \cup J)$ is a set of nodes that is partitioned into two distinct sets such that $V_C = (C \cup I)$, $E \subset (C \times I) \cup (I \times J)$ is a set of undirected edges, and $\alpha : V \rightarrow L$ is a function that labels nodes by a set of types, L. Note that components are always adjacent to interfaces and interfaces are adjacent to channels. Thus, a *connection* is defined as an ordered tuple $l = ((a \times b), j, (c \times d))$ for $a, d \in C,\ b, c \in I$, and $j \in J$ such that $a \neq d$, $b \neq c$, and

$\{(a,b),(b,j),(j,c),(c,d)\} \in E$. A connection is acyclic, and for each connection, the set of adjacent vertices, $N(v) = \{u,z\}$, where $v \in (I \cup J), u \in I,$ and $z \in J$. Intuitively, this means that a connection is a distinct path that is composed of a sequence of components, interfaces, and channels where the endpoints of a channel are a pair of interfaces. A labeling function assigns properties to nodes in *G*. A signal, *s* is a property that characterizes a channel, i.e. $\alpha_j: J \rightarrow s,$ for a labeling of channels α_j. The labeling of nodes allows the user to specify different types of interfaces, such as distinguishing power interfaces from data interfaces. This technique improves the flexibility of assigning different types without altering the underlying structure of *G*. This results in a graph whose nodes may be checked or queried based on the type of nodes. Furthermore, it enables the user to distinguish nodes between abstraction levels.

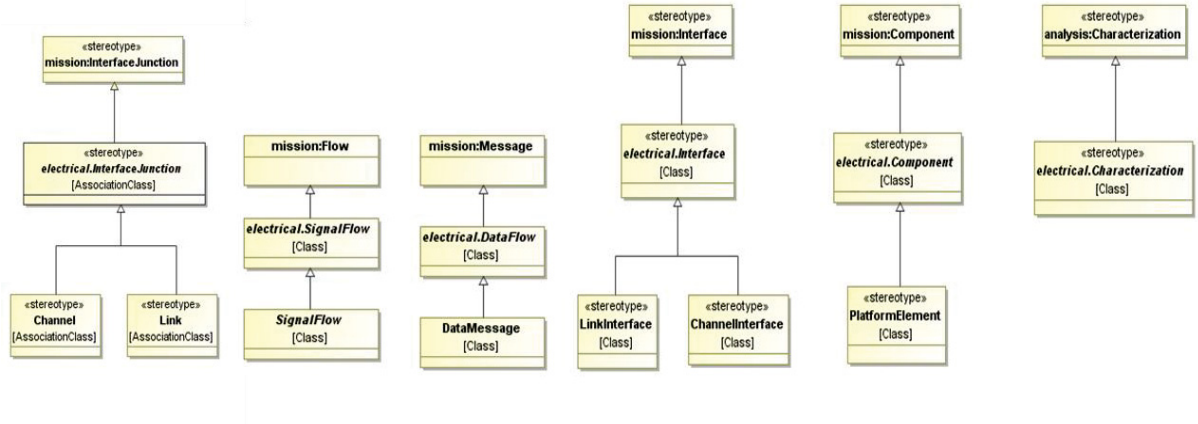

Figure 4. Application of stereotypes as a mechanism for extending SysML with electrical system domain concepts. *In this figure, the stereotypes that captures the core concepts of the electrical system domain are shown. It highlights the specialization relationship that extends the foundation ontologies, which are also encoded into SysML as a set of stereotypes.*

C. Extending the System Modeling Language (SysML) with Domain Concepts

This step within the development of our DSL maps the domain semantics into SysML. As mentioned above, SysML has mechanisms for extending its metamodel to support domain specific concepts. Domain concepts for the electrical system architecture are encoded into a set of stereotypes that are packaged within a profile to enable reuse amongst users. Instead of creating the domain stereotypes from scratch, they are derived from the hierarchy of foundation ontologies by specifying specializations of the foundation ontologies that have been incorporated by the IMCE team, as illustrated in Fig. 4. In the figure, the leaves represent specific stereotypes whose role is classified by the abstraction level for which it is intended to be used. A set of mappings between the domain concepts and stereotypes within SysML is summarized in Table 1.

D. Mapping Domain to Modeling Notations

Once the domain concepts are mapped into SysML via stereotypes, the stereotypes are associated to SysML blocks to provide an abstract model notation, also referred to as the *abstract syntax*. A SysML block is chosen as the abstract syntax because it is a general modeling concept in SysML that is used to model a variety of structural units, such as objects with physical attributes, objects with a boundary, or abstract entities. Moreover, it may be used to encapsulate other blocks, properties, stereotypes, or other modeling entities, as shown in Fig 5. In the figure, a set of interfaces are used to illustrate the abstract syntax of electrical system interfaces. With an abstract syntax identified, the stereotyped blocks may be applied to a model instance in any of the several diagrams that are provided by SysML. In this work, we have primarily utilized Block Definition Diagrams to express hierarchical relations between blocks, and we have utilized the Internal Block Diagram (IBD) extensively as the diagram of choice for representing the interconnectivity between electrical interfaces, as illustrated in Fig. 6.

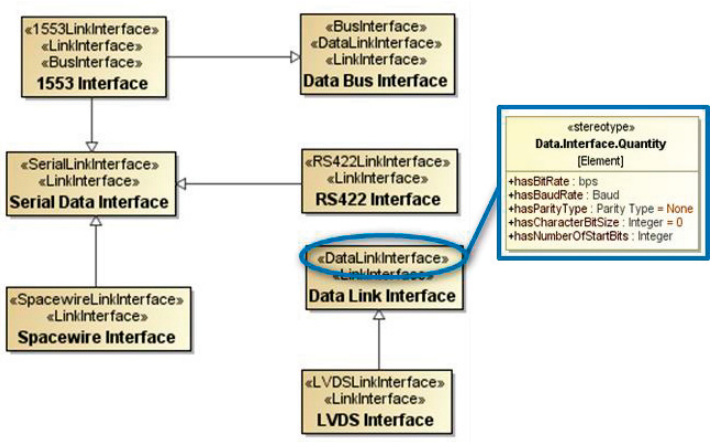

Figure 5. Example set of electrical system interfaces. *A sample set of interfaces that are captured in the metamodel for electrical system modeling. The stereotypes that are applied to the blocks capture the semantics and they are characterized by a set of properties. An example set of properties that are associated with a data*

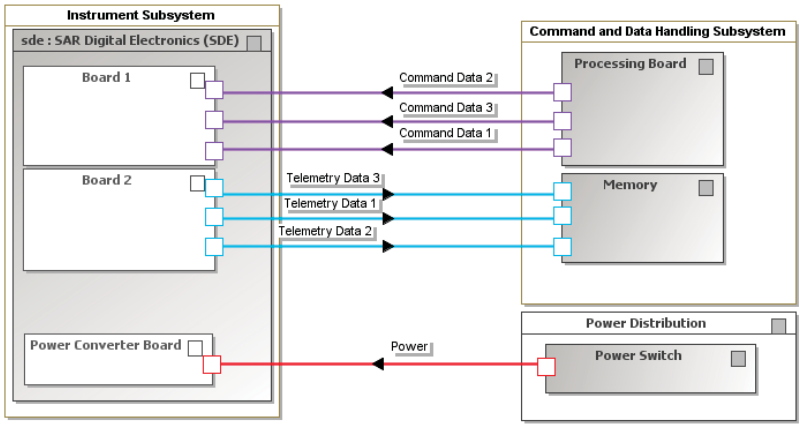

Figure 6. SysML Internal Block Diagram of an Electrical System Architecture Model. *This figure is an illustration of a specific type of view of an electrical system architecture block diagram. The diagram contains several blocks that represent components, such as electronic boards, subsystems, and assemblies. It also features electrical signals that are represented by the item flows and ports that represent the electrical interface types.*

A SysML IBD is chosen to describe and represent the *concrete syntax* of specific electrical system architecture views. A concrete syntax describes the appearance of the domain modeling elements as presented visually to the end user. In this case, the IBD was a logical choice because within the domain, information and different views of that information, are centered primarily on the structural, interconnection of blocks that are captured in block diagrams. The notations within an IBD provide a natural way for the domain experts to construct and express their models. Moreover, to support modeling varying levels of abstraction, abstract components and interfaces are refined by more detailed model elements. For example, in Fig. 6, the Instrument Subsystem is a SysML block that is refined by SAR Digital Electronics assembly, which is further refined by a set of boards. The ports are typed by data, telemetry, and power interface. Item flows denote the type of signals that flow across the connector. Note that the assembly serves as a placeholder until the information on its refinement is available. An illustration of an electrical system model is illustrated in Fig. 6 where signals are represented by item flows, interfaces are represented by block ports, and components are represented by blocks in the IBD.

VI. Proposed Design Flow for Electrical System Architecture Design

In addition to managing large data sets, a key concern of the electrical system architecture domain is the ability to capture a design from specification through physical implementation of electrical interfaces. The PBD methodology is used to guide the design flow from specification to implementation of the electrical system architecture. Several key abstraction levels are identified in our design flow, as illustrated in Fig. 7.

Within each level, there may be additional levels of refinement, but ideally, the difference in abstraction levels is driven by a difference in communication semantics. It is observed that the levels of abstraction within the traditional design process are driven by the documents. This results in abstraction levels that are not always semantically correct with respect with one another. This could potentially result in ambiguous and inconsistent representations. In contrast, the approach in this paper allows the engineer to construct the design as a model that is stored within a central repository. The design model may be refined as design information becomes available, and at any point within the design process, specific views of the system may be created or generated. Each view is constructed from a set of rules that specify the elements and levels of detail that may be shown within that view. For example, at the highest level of abstraction within the design flow, abstract communication and power channels are used for message passing. In contrast to the lowest level of abstraction, wires are used as the medium for which current flows. The interaction of these two types of signals is semantically different. Therefore, the levels that are captured in Fig. 7 are to be representative of different semantics. We have intentionally separated these two concerns – the design model and views of the design model. Therefore, we refer to the views to represent the different components that may be referenced, or queried, by the domain expert as visual artifacts in the form of block diagrams, reports, and tables. These views may be used in formal documentation procedures or as references to the current state of the system design as the system is developed.

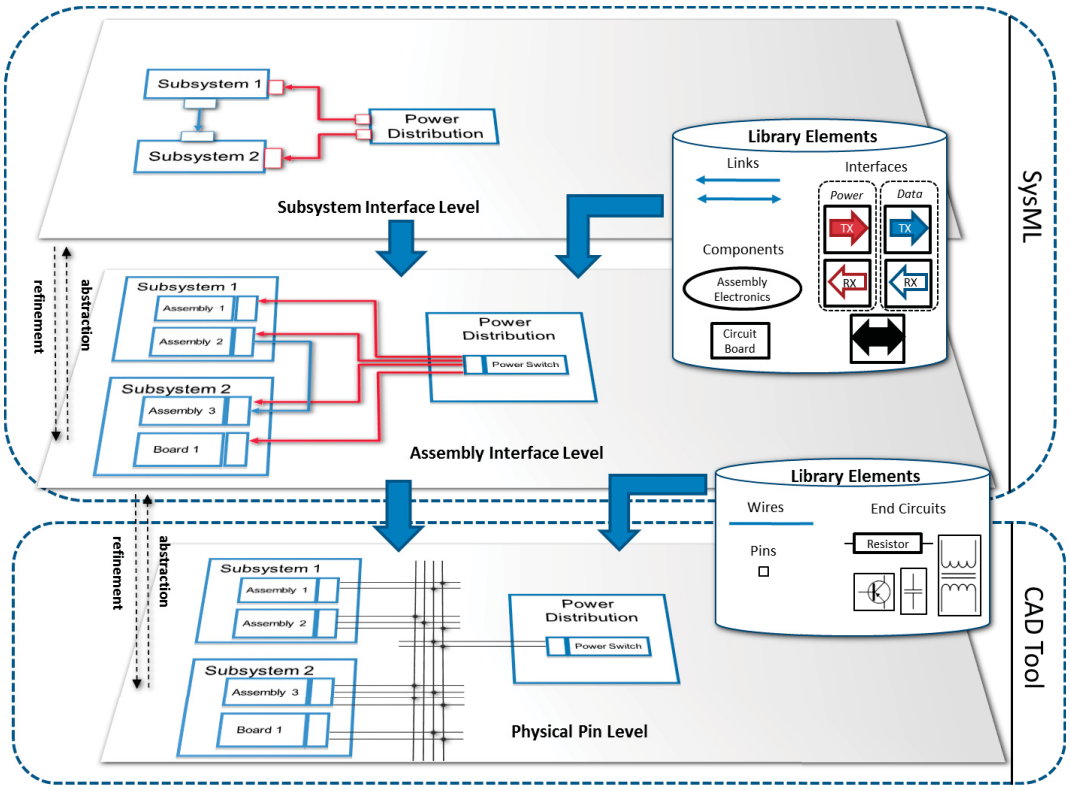

Figure 7. Electrical system abstraction levels. *This figure illustrates the abstraction levels that are identified in the proposed design flow. Each abstraction level is constructed by selecting an element from a library of characterized model elements. The top two levels are captured completely within SysML. The results of an Assembly Interface Level design is translated to a computer aided design (CAD) tool automatically.*

A. Abstraction Levels

The different levels of abstraction within our modeling environment include the Subsystem Interface Level, Assembly Interface Level, and the Physical Pin Level. For adjacent levels of abstractions, the upper level provides a specification to the lower level. At each level, a library of model elements that are characterized by a set of quantities are selected to refine the elements of the upper level of abstraction. With appropriately defined levels of abstraction, the designer can successively refine the design sytematically. Each level is described below.

1. *Subsystem Interface Level*

At the highest level of abstraction for the electrical system platform, the subsystems that will be needed to support the mission are identified and captured. It is assumed that a specification of the behavior has been determined at the system level which is used to drive the hardware and software development. This aids the electrical system engineer in estimating the general types of interfaces that will be needed to interface electronics from the subsystems. For example, in Fig. 7, three subsystems are identified. Subsystems 1 and 2 will require a set of power and data interfaces, and according to the illustration, the power subsystem will be the source of power to the other subsystems. The general types of interfaces that will be needed by the subsystems then provides a specification to the components that may be selected to realize the interface requirements of the Subsystem Interface Level components.

2. Assembly Interface Level

This is the next level of refinement. It is a representation of the electronic assemblies and boards that may be used to realize the subsystem components, and if the information about the specific types of interfaces is known, then that information is also captured. For example, assume that Assembly 1 of Subsystem 1 is only capable of receiving 7 Volts, and the Power Switch component in the Power Subsystem distributes 28 Volts. Then the power interface to Assembly 1 will more than likely be used to convert 28 Volts down to 7 Volts. The circuitry that does this conversion is the end circuit for Assembly 1. The signals at this level of abstraction may be assigned to an *interface function*. An interface function is a logical signal that is assigned to the junction of a pair of interfaces. Later in the design process, it is realized at the physical design level by a set of wires, connectors, and pins.

3. Physical Pin Level

This level of abstraction represents the lowest level of detail that is of concern to the electrical system engineer. It is at this level that interface functions are realized by physical connections between electrical circuits. At this level of abstraction, the components are end circuits that are realized by electrical circuit components, such as resistors, capacitors, transformers, etc., and the junctions are wires. The interfaces are conductors, such as pins, that connect wires between circuits. Connectors and cable harnesses may be assigned to elements in the physical design model. To accommodate this part of the design flow that is carried out in a tool that is external to SysML, we generate a list of interface functions that may be imported into the CAD tool.

B. Application to a Space Flight System

In this section, an example of a space flight system is briefly highlighted to demonstrate the applicability of our approach to any existing space flight system. The example flight system is a moderately sized spacecraft that targets an Earth observatory mission. The electrical system architecture is composed of approximately 100 electrical boards and assemblies, and the electrical interfaces may be decomposed into approximately 700 interface functions. The interface functions are a combination of data and power signals. Data signals may be further classified by their role in a spacecraft as a command or telemetry data signal. An example set of interfaces and components that are expressed in the application appears in Fig. 8. The example was constructed using No Magic's MagicDraw with the SysML plugin. To demonstrate applicability to a flight system, the framework we developed needed to be capable of expressing each type of interface in the flight system.

In addition, the electrical system engineer must be able to capture all three abstraction levels with traceability between each adjacent level of abstraction, and generate a set of existing documentation including block diagrams, a list of interface functions that are deployed in the design, and a list of identification labels called *reference designators* for each component. The abstraction levels provide a template for progressing the design. So, as the design progresses, more detailed components, interfaces, and signals are added to refine the design. This is done using encapsulation. However, encapsulation with SysML connectors is not intuitive. Thus, the connectors are explicitly related by directed relationships that denotes a set of signals that refine a more abstract signal. The design is captured and maintained within the design data repository of the tool, therefore traceability between any type of view of the design model is always maintained within the system model. To produce the different views, we apply a

transformation that captures different views based on a set of user defined rules that determine which elements are applicable within a particular view. By this method, we are able to separate the user defined views from the system design model. As a result, the documentation and block diagram views were created from the design repository with traceability back to the system model. Furthermore, we were able to provide a list of interface functions in a format that may be imported into the commercial CAD tool for physical pin level design.

VII. Conclusion

In this paper, a model-centric approach to the specification and design of electrical system architectures for space flight systems is introduced and described. A domain specific specification language is developed that allows electrical system engineers to construct electrical system models. In addition, an outline of a design flow that is guided by the principles of the Platform Based Design methodology is highlighted. The approach that is described in this paper is applied to documents and diagrams that are commonly used to document the electrical system design from a central repository. We were able to produce a set of documents and diagrams that are commonly used to document the electrical system design of a practical space flight system as proof of concept.

Future work will refine our approach and focus our efforts on formalizing the design within the Platform Based Design framework. In particular, we plan to capture electrical system requirements using SysML, and integrate requirement models into our modeling environment. We also plan to add additional characterizations and properties to our existing model elements. The model element library is planned to be expanded to cover pre-characterized sets of hardware platforms to enable reuse.

Acknowledgments

This work was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. The authors would like to thank Jose Gomez-Mustafa for thoughtful discussions and initial work on developing application models using the method presented in this paper. The authors acknowledge the assistance with advanced tool support from Nicolas Rouquette and feedback on this work from Steven Jenkins.

References

[1]Wymore, A. W., *A Mathematical Theory of Systems Engineering and Analysis,* John Wiley and Sons, New York, 1967.

[2]Wymore, A. W., *Model-Based Systems Engineering*, CRC Press, Inc., Boca Raton, FL, 1993.

[3]Ogren, I., "On Principles for Model-Based Systems Engineering," *Systems Engineering*, Vol. 3, No. 1, 2000, pp. 38-49.

[4]Estefan, J. A., "Survey of Model-Based Systems Engineering (MBSE) Methodologies", Rev B., INCOSE MBSE Focus Group, 2008.

[5]Harvey, D., Waite, M., Logan, P., and Liddy, T., "Document the Model, Don't Model the Document," *SETE APCOSE Conference*, 2012.

[6]Metcalfe, R., "Packet Communication", Massachusetts Institute of Technology (MIT) Project MAC Technical Report MAC TR-114, Cambridge, MA, Dec. 1973.

[7]Object Management Group (OMG)*, "*OMG Systems Modeling Language (OMG SysML): version 1.2"*,* OMG document number formal/2010-06-02*,* URL: http://www.omg.org/docs/formal/08-11-02.pdf [cited 17 May 2012]*,* 2008.

[8]Derler, P., Lee, E. A., and Sangiovanni-Vincentelli, A. L., "Addressing Modeling Challenges in Cyber-Physical Systems", Technical Report UCB/EECS-2011-17, EECS Department, University of California, Berkeley, CA, Mar. 2011.

[9]Stojanovic, V. and Horowitz, M., "Modeling and Analysis of High-Speed Links", *IEEE Custom Integrated Circuits Conference*, Sept. 2003, pp. 589-594.

[10]Slomka, F., Kollmann, S., Moser, S., and Kempf, K., "A Multidisciplinary Design Methodology for Cyber-physical Systems", *MoDELS 2011 ADES-MB Workshop Proceedings*, Oct. 2011, pp. 23-37.

[11]Shah, A. A., Schaefer, D., and Paredis, C. J. J., "Enabling Multi-View Modeling With SysML Profiles and Model Transformations", *6th International Product Lifecycle Management Conference (PLM09)*, Bath, U.K., 2009.

[12]Booch, G., Jacobson, I., and Rumbaugh, J., *The Unified Modeling Language User Guide*, Addison-Wesley, 1998.

[13]Friedenthal, S., Moore, A., and Steiner, R., *A Practical Guide to SysML: Systems Modeling Language*, Morgan Kaufmann, San Francisco, CA, 2008.

[14]Sparx Enterprise Architect, Software Package, Ver. 9.3, Sparx Systems, Creswick, Victoria, Australia, Apr. 2012, URL: http://www.sparxsystems.com.au/products [cited 17 May 2012].

[15]IBM Rational Rhapsody, Software Package, IBM, Armonk, New York, URL: http://www.sparxsystems.com.au/products [cited 17 May 2012].

[16]MagicDraw SysML Plugin, Software Package, Ver. 17.0.1, No Magic, Allen, TX, 2012, URL: http://www.nomagic.com [cited 17 May 2012].

[17]Tolvanen, J. P. and Kelly, S., "Defining Domain-Specific Modeling Languages to Automate Product Derivation: Collected Experiences", *Lecture Notes in Computer Science*, No. 3714, 2005, pp. 198-209.

[18]Gray, J., Fisher, K., Consel, C., Karsai, G., Mernik, M., and Tolvanen, J. P., "Panel – DSLs: The Good, The Bad, and The Ugly", *Object Oriented Programming, System, Languages, and Applications (OOPSLA) Companion*, New York, 2008.
[19]OWL 2 Web Ontology Language Overview, W3C Recommendation, URL: http://www.w3.org/TR/owl-features/ [cited 17 May 2012], Nov. 2009.
[20]Wand, Y., Monarchi, D. E., Parson, J., and Woo, C. C., "Theoretical Foundations for Conceptual Modelling in Information Systems," *Decision Support Systems Journal – Special Issue on WITS*, Vol. 15, No. 4, 1995, pp. 285-304.
[21]Gonzalez-Perez, C. and Henderson-Seller, B., *Metamodelling for Software Engineering*. Chichester (UK), Wiley and Sons, 2008.
[22]Sangiovanni-Vincentelli, A. L., Carloni, L., De Bernardinis, F., and Sgroi, M., "Benefits and Challenges for Platform-Based Design", *Proceedings of the Design Automation Conference (DAC)*, San Diego, CA, June 2004.
[23]Carloni, L., Bernardinis, F. D., Pinello, C., Sangiovanni-Vincentelli, A. L., and Sgroi, M., "Platform-Based Design for Embedded Systems", *The Embedded Systems Handbook*, CRC Press, 2005.
[24]Davare, A., Zhu, Q., Moondanos, J., and Sangiovanni-Vincentelli, A. L., "JPEG Encoding on the Intel MXP5800: A Platform-Based Design Case Study", *ESTIMedia 2005: 3rd Workshop on Embedded Systems for Real-time Multimedia*, Sept. 2005, pp. 89-94.
[25]Zeng, H., Davare, A., Sangiovanni-Vincentelli, A. L., Sonalkar, S., Kanajan, S., and Pinello, C., "Design Space Exploration of Automotive Platforms in Metropolis", *Society of Automotive Engineers World Congress*, April, 2006.
[26]Keutzer, K., Malik, S., Newton, A. R., Rabaey, J. M., and Sangiovanni-Vincentelli, A. L., "System-Level Design: Orthogonalization of Concerns and Platform-Based Design", *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, Vol. 19, No. 12, 2000, pp. 1523-1543.
[27]Zimmermann, H., "OSI Reference Model", IEEE Transactions on Communications, Vol. 28, No. 4, Apr. 1980, p. 425.